OASIS3-MCT User Guide, OASIS3-MCT 5.0

S. Valcke, Anthony Craig, Éric Maisonnave, Laure Coquart

OASIS3-MCT User Guide

OASIS3-MCT_5.0

Edited by:
S. Valcke, T. Craig, E. Maisonnave, L. Coquart
CECI, Université de Toulouse, CERFACS

CERFACS TR/CMGC/21/161

15/12/2021

How to get assistance?

Assistance can be obtained by sending a mail to oasishelp(at)cerfacs.fr

How to get documentation ?

The documentation can be downloaded from the OASIS web site under the URL :
https://oasis.cerfacs.fr/en/

Contents

.

ACKNOWLEDGMENTS

The development of this new version of OASIS, OASIS3-MCT_5.0 has been possible thanks the folllowing fundings:

- EU Centre of Excellence ESiWACE2 , GA #823988
- EU IS-ENES3 - Infrastructure for the European Network for Earth System modelling - Phase 3 project (GA # 824084)

We would like to thank the main past or present developers of OASIS (in alphabetical order, with the name of their institution at the time of their contribution to OASIS):

Arnaud Caubel (LSCE/IPSL & FECIT/Fujitsu), Laure Coquart (CNRS/CERFACS) Anthony Craig (CERFACS - consultant), Damien Declat (CERFACS), Italo Epicoco (CMCC), Rupert Ford (STFC), Philippe Gambron (STFC), Veronika Gayler (MPI-M&D), Josefine Ghattas (CERFACS), Christopher Goodyer (NAG) Jean Latour (CERFACS & Fujitsu-Fecit), Eric Maisonnave (CERFACS), Silvia Mocavero (CMCC), Andrea Piacentini (CERFACS - consultant) Elodie Rapaport (CERFACS), Sami Saarinen (ECMWF), Eric Sevault (Météo-France), Laurent Terray (CERFACS), Olivier Thual (CERFACS), Sophie Valcke (CERFACS), Reiner Vogelsang (SGI Germany), Li Yan (CERFACS).

We also would like to thank the following people for their help and suggestions in the design of the OASIS software (in alphabetical order, with the name of their institution at the time of their contribution to OASIS):

Dominique Astruc (IMFT), Chandan Basu (NSC, Sweden), Sophie Belamari (Météo-France), Dominique Bielli (Météo-France), Yamina Boumediene (CERFACS), Gilles Bourhis (IDRIS), Pascale Braconnot (IPSL/LSCE), Sandro Calmanti (Météo-France), Christophe Cassou (CERFACS), Yves Chartier (RPN), Jalel Chergui (IDRIS), Philippe Courtier (Météo-France), Philippe Dandin (Météo-France), Michel Déqué (Météo-France), Ralph Doescher (SMHI), Jean-Louis Dufresne (LMD), Jean-Marie Epitalon (CERFACS), Laurent Fairhead (LMD), Uwe Fladrich (SMHI), Marie-Alice Foujols (IPSL), Gilles Garric (CERFACS), Christopher Goodyer (NAG), Eric Guilyardi (CERFACS), Charles Henriet (CRAY France), Pierre Herchuelz (ACCRI), Maurice Imbard (Météo-France), Luis Kornblueh (MPI-M), Stephanie Legutke (MPI-M&D), Claire Lévy (LODYC), Yann Meurdesoif (IPSL/LSCE) Olivier Marti (IPSL/LSCE), Sébastien Masson (IPSL/LOCEAN) Claude Mercier (IDRIS), Pascale Noyret (EDF), Marc Pontaud (Météo-France), Adam Ralph (ICHEC), René Redler (MPI-M), Hubert Ritzdorf (CCRLE-NEC), Tim Stockdale (ECMWF), Rowan Sutton (UGAMP), Véronique Taverne (CERFACS), Jean-Christophe Thil (UKMO), Nils Wedi (ECMWF).

Chapter 1

Introduction

In 1991, CERFACS started the development of a software interface to couple existing ocean and atmosphere numerical General Circulation Models. Today, different versions of the OASIS3-MCT coupler are used by at least 65 modelling groups all around the world to couple more than 80 applications on different computing platforms[1]. OASIS3-MCT sustained development is ensured by a collaboration between CERFACS and the Centre National de la Recherche Scientifique (CNRS) and its maintainance and user support is regularly reinforced with additional resources coming from European and national projects.

The current OASIS3-MCT internally uses MCT, the Model Coupling Toolkit[2] [Larson et al 2005] [Jacob et al 2005], developed by the Argonne National Laboratory in the USA. MCT implements fully parallel remapping, as a parallel matrix vector multiplication, and parallel distributed exchanges of the coupling fields, based on pre-computed remapping weights and addresses. Its design philosophy, based on flexibility and minimal invasiveness, is close to the OASIS3-MCT approach. MCT has proven parallel performance and is, most notably, the underlying coupling software used in National Center for Atmospheric Research Community Earth System Model (NCAR CESM).

OASIS3-MCT is a portable set of Fortran 77, Fortran 90 and C routines. Low-intrusiveness, portability and flexibility are OASIS3-MCT key design concepts. After compilation OASIS3-MCT is a coupling library to be linked to the component models, and which main function is to interpolate and exchange the coupling fields to form a coupled system. OASIS3-MCT supports coupling of 2D logically-rectangular fields but 3D fields and 1D fields expressed on unstructured grids are also supported using a one dimension degeneration of the structures. Thanks to MCT, all transformations, including remapping, are performed in parallel on the set of source or target component processes and all coupling exchanges are now executed in parallel directly between the component processes via Message Passing Interface (MPI). OASIS3-MCT also supports file I/O using NetCDF and has python and C language bindings.

The developments realised in the different versions of OASIS3-MCT are described in Appendix B. To communicate with another component, or to perform I/O actions, a component model needs to include few specific calls to OASIS3-MCT Application Programming Interface (API). The *namcouple* configuration file is also largely unchanged, although several options are either added, deprecated, not used or not supported.

Results obtained with IS-ENES2 coupling technology benchmarks show that OASIS3-MCT performs as well as, and even better at very high number of cores, than other coupling technologies, at least for up to O(10000) cores. It is therefore very likely that OASIS3-MCT will provide an efficient and easy-to-use coupling solution for many climate modelling groups in the few years to come.

[1] A list of coupled models realised with OASIS3-MCT can be found at https://oasis.cerfacs.fr/en/oasis-coupled-models/

[2] www.mcs.anl.gov/research/projects/mct/

1.1 Step-by-step use of OASIS3-MCT

To use OASIS3-MCT for coupling codes, one has to follow these steps:

1. Obtain OASIS3-MCT source code (see chapter 1.2).
2. Get familiar with OASIS3-MCT, either by following the Short Private Online Course (SPOC, see https://cerfacs.fr/online-training/) or by going through the tutorial steps. Tutorial sources are available in directory `examples/tutorial_communication` and all explanations are provided in the document `tutorial_communication.pdf` therein.
3. Identify the coupling or I/O fields and adapt the codes to implement the coupling exchanges with the OASIS3-MCT coupling library based on MPI message passing. The OASIS3-MCT coupling library uses NetCDF and therefore can also be used to perform I/O actions from/to disk files. For more detail on how to use the OASIS3-MCT API in the codes, see chapter 2.
4. Define all coupling and I/O parameters and the transformations required to adapt each coupling field from the source model grid to the target model grid. On this basis, prepare OASIS3-MCT configuring file *namcouple*. OASIS3-MCT supports different interpolation algorithms as described in chapter 4. Remapping files can be computed online using the SCRIP options, or offline using either the SCRIP, ESMF or XIOS (see `examples/regrid_environment`, section 6.3.3) and read in during the run using the MAPPING transformation.

   We strongly recommend to tests off-line the quality of the chosen transformations and remappings using the environment available in `examples/regrid_environment` and explanations provided in the document `regrid_environment.pdf` therein.
5. Generate required auxiliary data files (see chapter 5).
6. Compile OASIS3-MCT, the component models and start the coupled experiment. For details on how to compile and run a coupled model with OASIS3-MCT, see section 6.

If you need extra help, do not hesitate to contact us at oasishelp(at)cerfacs.fr .

1.2 OASIS3-MCT sources

OASIS3-MCT sources are available from CERFACS git server. To obtain more detail on downloading the sources, please fill in the registration form at https://oasis.cerfacs.fr/en/downloads/ .

OASIS3-MCT directory structure is the following one:

```
- oasis3-mct/lib/cbindings        C language bindings source code
                /mct              Model Coupling Toolkit Coupling Software
                /psmile           OASIS3-MCT coupling library
                /scrip            SCRIP interpolation library

- oasis3-mct/doc                  OASIS3-MCT User Guide

- oasis3-mct/util/make_dir        Utilities to compile OASIS3-MCT
                 /load_balancing  Tool for load balancing analysis

- oasis3-mct/pyoasis              Python wrapper source code

- oasis3-mct/examples             Environment to compile, run and use
                                  different toy coupled models.
```

1.3 Licenses and Copyrights

1.3.1 OASIS3-MCT license and copyright statement

1.3.2 MCT copyright statement

FRINGEMENT, (2) DO NOT ASSUME ANY LEGAL LIABILITY OR RESPONSIBILITY FOR THE ACCURACY, COMPLETENESS, OR USEFULNESS OF THE SOFTWARE, (3) DO NOT REPRESENT THAT USE OF THE SOFTWARE WOULD NOT INFRINGE PRIVATELY OWNED RIGHTS, (4) DO NOT WARRANT THAT THE SOFTWARE WILL FUNCTION UNINTERRUPTED, THAT IT IS ERROR-FREE OR THAT ANY ERRORS WILL BE CORRECTED.

5. LIMITATION OF LIABILITY. IN NO EVENT WILL THE COPYRIGHT HOLDER, THE UNITED STATES, THE UNITED STATES DEPARTMENT OF ENERGY, OR THEIR EMPLOYEES: BE LIABLE FOR ANY INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL OR PUNITIVE DAMAGES OF ANY KIND OR NATURE, INCLUDING BUT NOT LIMITED TO LOSS OF PROFITS OR LOSS OF DATA, FOR ANY REASON WHATSOEVER, WHETHER SUCH LIABILITY IS ASSERTED ON THE BASIS OF CONTRACT, TORT (INCLUDING NEGLIGENCE OR STRICT LIABILITY), OR OTHERWISE, EVEN IF ANY OF SAID PARTIES HAS BEEN WARNED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGES.

1.3.3 The SCRIP 1.4 license copyright statement

The SCRIP 1.4 copyright statement reads as follows:

“Copyright © 1997, 1998 the Regents of the University of California. This software and ancillary information (herein called SOFTWARE) called SCRIP is made available under the terms described here. The SOFTWARE has been approved for release with associated LA-CC Number 98-45. Unless otherwise indicated, this SOFTWARE has been authored by an employee or employees of the University of California, operator of Los Alamos National Laboratory under Contract No. W-7405-ENG-36 with the United States Department of Energy. The United States Government has rights to use, reproduce, and distribute this SOFTWARE. The public may copy, distribute, prepare derivative works and publicly display this SOFTWARE without charge, provided that this Notice and any statement of authorship are reproduced on all copies. Neither the Government nor the University makes any warranty, express or implied, or assumes any liability or responsibility for the use of this SOFTWARE. If SOFTWARE is modified to produce derivative works, such modified SOFTWARE should be clearly marked, so as not to confuse it with the version available from Los Alamos National Laboratory.”

Chapter 2

Interfacing a component code with OASIS3-MCT

At run-time, OASIS3-MCT performs parallel exchange of coupling data between parallel components and sub-components and allows regridding (also called remapping or interpolation), time integration or accumulation and other transformations of these coupling fields.

This chapter describes how to adapt the component codes to couple them through OASIS3-MCT.

OASIS3-MCT supports coupling exchanges between parallel components and sub-components deployed in diverse configurations; the different possibilities and how to use the OASIS3-MCT library accordingly are described in section 2.1.

The OASIS3-MCT Application Programming Interface (API) includes different classes of modules or routines that are described in detail in sections 2.2, 2.3 and 2.4 for Fortran, C and python codes respectively. In section 2.5, the reader will also find an overview of the MCT library (see 2.5.1) and additional notes on how to exchange scalars (see 2.5.2), how to to reproduce different coupling algorithms with OASIS3-MCT using the `LAG` index (see 2.5.3), and on the `SEQ` index (see 2.5.4).

2.1 Configurations of components supported

Since OASIS3-MCT_3.0 release, coupling exchanges between components and sub-components deployed in a much larger diversity of configurations than before are supported. This is illustrated on figure 2.1 and how to use the OASIS3-MCT library accordingly is detailed on figure 2.2. All OASIS3-MCT API routines are also described in details in sections 2.2, 2.3 and 2.4.

We call here an "executable" a compiled code forming a part of or the whole coupled system. A "component" is the ensemble of processes, or tasks, within the coupled system calling `oasis_init_comp` with the same `comp_name` argument (see section 2.2.2). A "sub-component" is the subset of tasks within a component sending or receiving coupling fields on a specific grid; of course, a component may have only one sub-component that gathers all its tasks.

In `examples/tutoria_communication` and `examples/regrid_environment` one finds practical examples of how to use the OASIS3-MCT library (see sections 6.3.1 and 6.3.3).

With OASIS3-MCT, it is currently possible to (the text between [and] refers to figure 2.1) :

- to implement coupling exchanges between two components or sub-components running concurrently on separate sets of tasks within two different executables [A, D, E, J];
- to implement coupling exchanges between two components or sub-components running concurrently on separate sets of tasks within one same executable [B, F, I];
- to implement coupling exchanges within one executable between two concurrent sub-components [C]

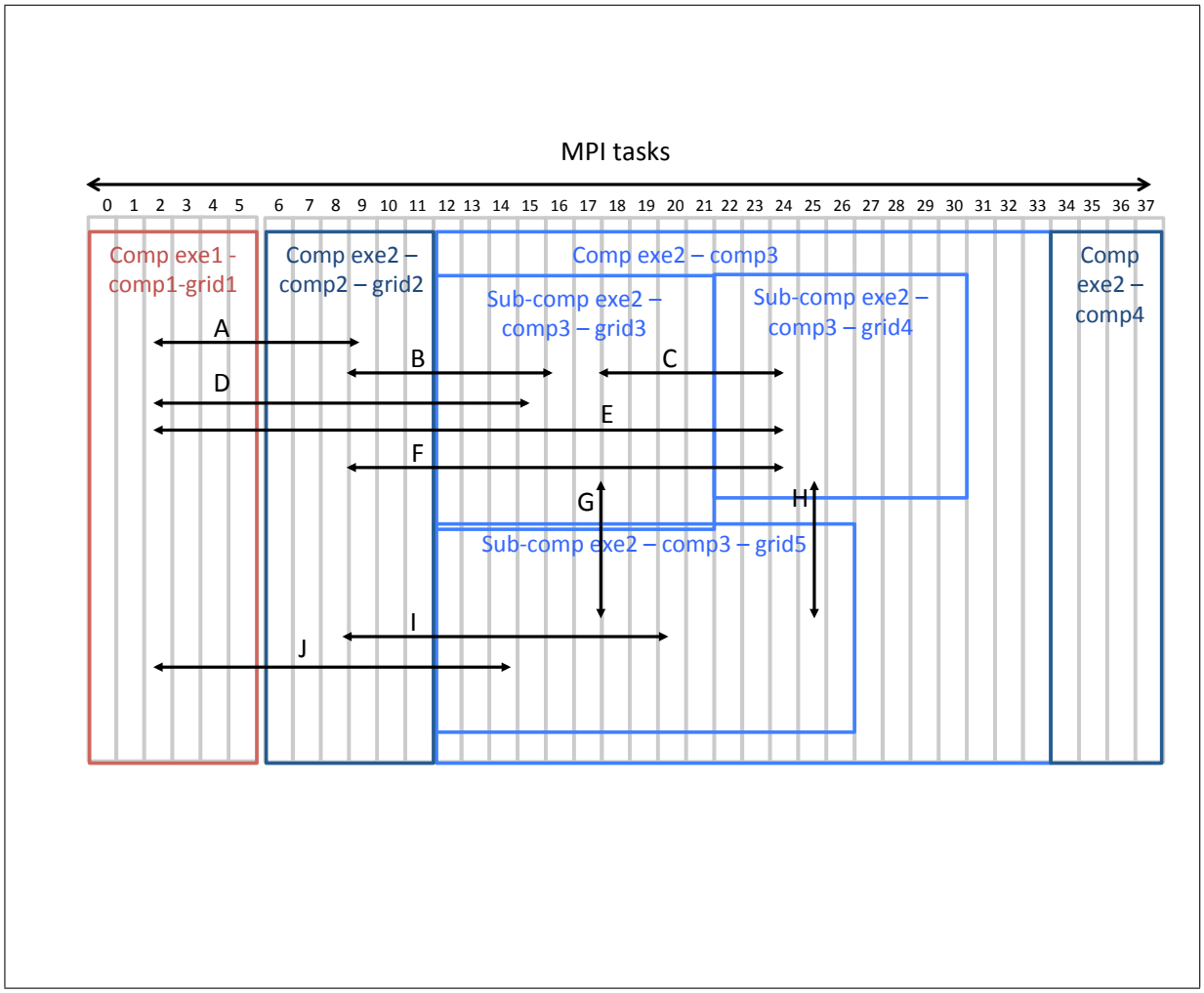

Figure 2.1: The different configuration of components supported by OASIS3-MCT. Two executables `exe1` and `exe2` are running concurrently on separate sets of MPI tasks (0-5 for `exe1` and 6-37 for `exe2`). Executable `exe1` includes only one component `comp1` that has coupling fields defined on only one grid `grid1` (decomposed on all of its 6 tasks). Executable `exe2` includes 3 components, `comp2`, `comp3`, and `comp4` running concurrently respectively on tasks 6-11, 12-33 and 34-37. Component `comp2` participates in the coupling with fields defined on only one coupling grid `grid2` (decomposed on all of its 5 tasks) while `comp4` does not participate at all in the coupling. Component `comp3` has 3 sub-components, respectively exchanging coupling fields defined on `grid3` (tasks 12-21), `grid4` (tasks 22-30) and `grid5` (tasks 12-26, therefore overlaping with both `grid3` and `grid4`); finally, `comp3` has 3 tasks (31-33) not involved in the coupling. Sub-components`exe2-comp3-grid3` and `exe2-comp3-grid5`, or sub-components `exe2-comp3-grid4` and `exe2-comp3-grid5` are examples of coupling between sub-components running sequentially on overlapping sets of tasks.

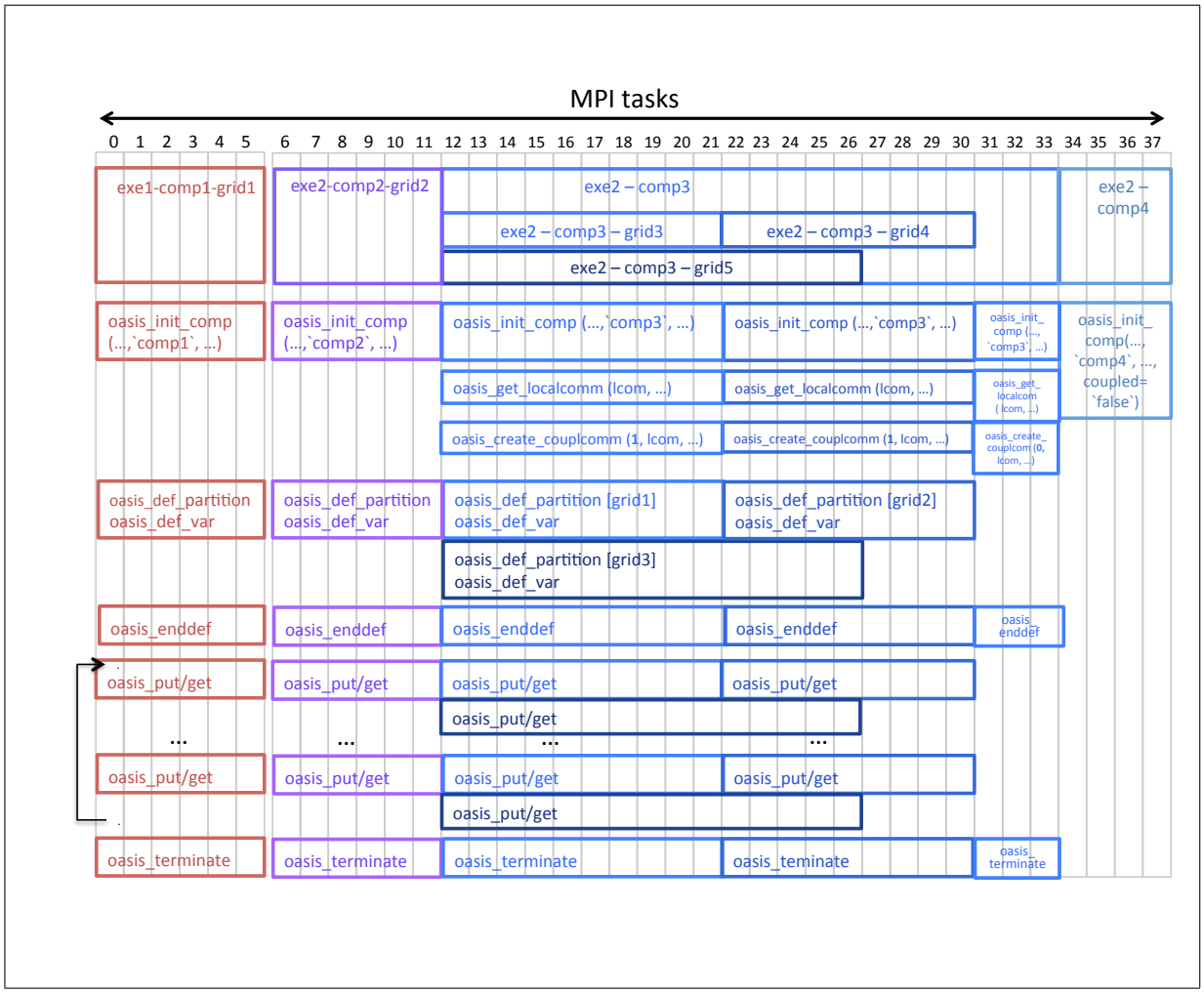

Figure 2.2: The sequence of OASIS3-MCT calls that have to be implemented in the codes so to allow the configuration of components described on figure 2.1. Each MPI tasks has to call `oasis_init_comp` once with the name of its component as 2^{nd} argument. As none of `comp4` tasks is participating to the coupling, `comp4` tasks calls `oasis_init_comp` with `coupled=.false."` as 4^{th} argument and does not call any other OASIS3-MCT routine. As some of `comp3` tasks are participating in the coupling, all `comp3` tasks have to call `oasis_init_comp`, `oasis_get_localcomm`, `oasis_create_couplcomm`, `oasis_enddef` and `oasis_terminate` (these are the only routine to be called by `comp3` tasks 31-33 not participating to the coupling). To initialise the coupling exchanges, the tasks of a sub-component holding a field decomposed on a specific grid have to call the `oasis_def_partition` to express the decomposition of the grid, `oasis_def_var` to declare the coupling field and `oasis_enddef`. Finally, the tasks of a sub-component exchanging coupling fields have to call `oasis_put` and `oasis_get` accordingly.

- to implement coupling exchanges within one executable between two sub-components running sequentially on overlapping sets of tasks (i.e. a task can be coupled to itself calling both the "put" and the "get" of the exchange) [G, H]
- to have some tasks of a component not participating to the coupling exchanges [tasks 31-33 of `exe2-comp3`]
- to have all processes of a component not participating to the coupling exchanges [`exe2-comp4`, tasks 34-37]

The sequence of OASS3-MCT API routines that have to be called in the different cases is detailed on figure 2.2. These routines are also described in detail in the next section.

2.2 OASIS3-MCT Fortran API

To interact with the rest of the coupled system, few calls of the OASIS3-MCT library routines, which sources can be found in `oasis3-mct/lib/psmile` directory, have to be implemented in component Fortran codes. They belong to the following classes:

1. Module to use (section 2.2.1)
2. Initialisation (section 2.2.2)
3. Partition definition (section 2.2.3)
4. Grid data file definition (section 2.2.4)
5. Coupling-I/O field declaration (section 2.2.5)
6. End of definition phase (section 2.2.6)
7. Coupling-I/O field sending and receiving (section 2.2.7)
8. Termination (section 2.2.8)
9. Optional auxiliary routines (section 2.2.9)

2.2.1 Module to use in the code

To use OASIS3-MCT library, a user needs to add in his code:

- `USE mod_oasis`
  or
- `USE mod_prism`

Both use statements are valid but only one needs to be used in a particular component. This single use statement provides all methods required. The methods, datatypes, and capabilities are identical for both the `mod_prism` or `mod_oasis` interfaces, the only difference being the name of the interface. The interface in module `mod_prism` is provided for backwards compatability with prior versions of OASIS3. Use of module `mod_oasis` is recommended and provides access to a set of updated routine names that will continue to evolve in the future, always ensuring backward compatibility. In the following sections, both the `mod_prism` and `mod_oasis` interface names are defined and a single description of the interface arguments is provided.

2.2.2 Initialisation

Coupling initialisation

- `CALL oasis_init_comp (compid, comp_name, kinfo, coupled, commworld)`
- `CALL prism_init_comp_proto (compid, comp_name, kinfo, coupled, commworld)`

- `compid [INTEGER; OUT]`: returned component ID
- `comp_name [CHARACTER; IN]`: component name; maximum length of 80 characters
- `kinfo [INTEGER; OUT]`: returned error code
- `coupled [LOGICAL, OPTIONAL; IN]`: flag to specify if the calling task is participating or not to the coupling (`.true.` by default).
- `commworld [INTEGER, OPTIONAL; IN]` : optional argument to specify the global communicator gathering the components of the coupled model. If not specified, `MPI_COMM_WORLD` will be used as the default communicator to startup. All components of the coupled model must specify the same `commworld` argument.

This routine must called by all tasks of all components whether or not they are involved in the coupling [1].

A component is defined as the ensemble of tasks calling `oasis_init_comp` with the same `comp_name` argument. If and only if all tasks of a component are excluded from the coupling, the logical `coupled` can be set to `.false.` for this component tasks; in this case, `oasis_init_comp` is the only API routine that needs to be called by the component tasks. If at least one tasks of a component is participating to the coupling, all component tasks have to call `oasis_init_comp` with `coupled=.true.` (which is the default); in this case, the component tasks not participating to the coupling will also have to call `oasis_get_localcomm`, `oasis_create_couplcomm`, `oasis_enddef` and `oasis_terminate`.

Communicator for internal parallelisation

- `CALL oasis_get_localcomm (local_comm, kinfo )`
- `CALL prism_get_localcomm_proto (local_comm, kinfo )`
  - `local_comm [INTEGER; OUT]`: value of local communicator
  - `kinfo [INTEGER; OUT]`: returned error code.

  This routine returns the value of a local communicator gathering only the tasks of the component (i.e. the tasks that called `oasis_init_comp` with the same `comp_name` argument).

  This may be needed as all executables of the coupled system are started in a pseudo-MPMD mode with MPI1 and therefore share automatically the same MPI_COMM_WORLD communicator. Another communicator has to be used for the internal parallelisation of each component. OASIS3-MCT creates this local communicator `local_comm` based on the value of the `comp_name` argument in the `oasis_init_comp` call.

  Retrieving a local communicator `local_comm` is also needed if `oasis_create_couplcomm` is called, as `local_comm` is an argument of this routine (see below).

- `CALL oasis_create_couplcomm(icpl, local_comm, coupl_comm, kinfo)`
- `CALL prism_create_couplcomm(icpl, local_comm, coupl_comm, kinfo)`
  - `icpl [INTEGER; IN]`: coupling process flag
  - `local_comm [INTEGER; IN]`: MPI communicator with all processes of the component
  - `coupl_comm [INTEGER; OUT]`: returned MPI communicator gathering only component processes participating in the coupling
  - `kinfo [INTEGER; OUT; OPTIONAL]`: returned error code

  This routine creates a coupling communicator for a subset of processes. It is mandatory to call this routine if only a subset of the component processes participate in the coupling (e.g. `comp3` in figure 2.2); in that case, the processes involved in the coupling have to call it with icpl=1 while the other

[1] The component may also call MPI_Init explicitly, but if so, has to call it before calling `oasis_init_comp`; in this case, the component also has to call MPI_Finalize explicitly, but only after calling `oasis_terminate`.

have to call it with icpl = MPI_UNDEFINED. Argument local_comm is the MPI communicator associated with all processes of the component returned by oasis_get_localcomm. The new coupling communicator is returned in coupl_comm.

If this communicator already exists in the code, the component should simply provide it to OASIS3-MCT with:

- CALL oasis_set_couplcomm(coupl_comm, kinfo)
- CALL prism_set_couplcomm(coupl_comm, kinfo)
  - coupl_comm [INTEGER; IN]: MPI communicator gathering only component processes participating in the coupling
  - kinfo [INTEGER; OUT; OPTIONAL]: returned error code

  This routine allows users to provide a local coupling communicator to OASIS3-MCT, given that it already exists in the code. The value of coupl_comm must be the value of this local coupling communicator for the processes participating to the coupling and it must be MPI_COMM_NULL for processes not involved in the coupling.

2.2.3 Partition definition

The coupling fields sent or received by a component are usually scattered among the different component processes. With OASIS3-MCT, all processes exchanging coupling data have to describe, in a global index space, the local partitioning of the different grids onto which the data is expressed (see 2.2.4 for the grid definition). The processes not implied in the coupling do not have to call this routine (for backward compatibility with OASIS3-MCT_2.0, they may still call it describing a null partition, i.e. with ig_paral(:)=0).

- CALL oasis_def_partition (il_part_id, ig_paral, kinfo, ig_size, name) or
- CALL prism_def_partition_proto (il_part_id, ig_paral, kinfo, ig_size, name)
  - il_part_id [INTEGER; OUT]: partition ID
  - ig_paral [INTEGER, DIMENSION(:), IN]: vector of integers describing the local grid partition in the global index space; has a different expression depending on the type of the partition; in OASIS3-MCT, 5 types of partition are supported: Serial (no partition), Apple, Box, Orange, and Points (see below).
  - kinfo [INTEGER; OUT]: returned error code.
  - ig_size [INTEGER, OPTIONAL, IN]: Optional argument, mandatory if the coupling data is exchanged for only a subdomain of the global grid; in this case, ig_size must give the total number of grid points.
  - name [CHARACTER, OPTIONAL, IN]: Optional argument associating a name to the partition, mandatory if oasis_def_partition is called either for a grid decomposed not across all the processes of a component or if the related oasis_def_partition are not called in the same order on the different component processes; this argument is new since OASIS3-MCT_3.0 release and is linked to the greater flexibility in the configuration of components supported (see 2.1); it has a maximum length of 120 characters.

Serial (no partition)

This is the choice for a grid entirely supported by only one process . In this case, we have ig_paral(1:3):

- ig_paral(1) = 0 (indicates a Serial “partition”)

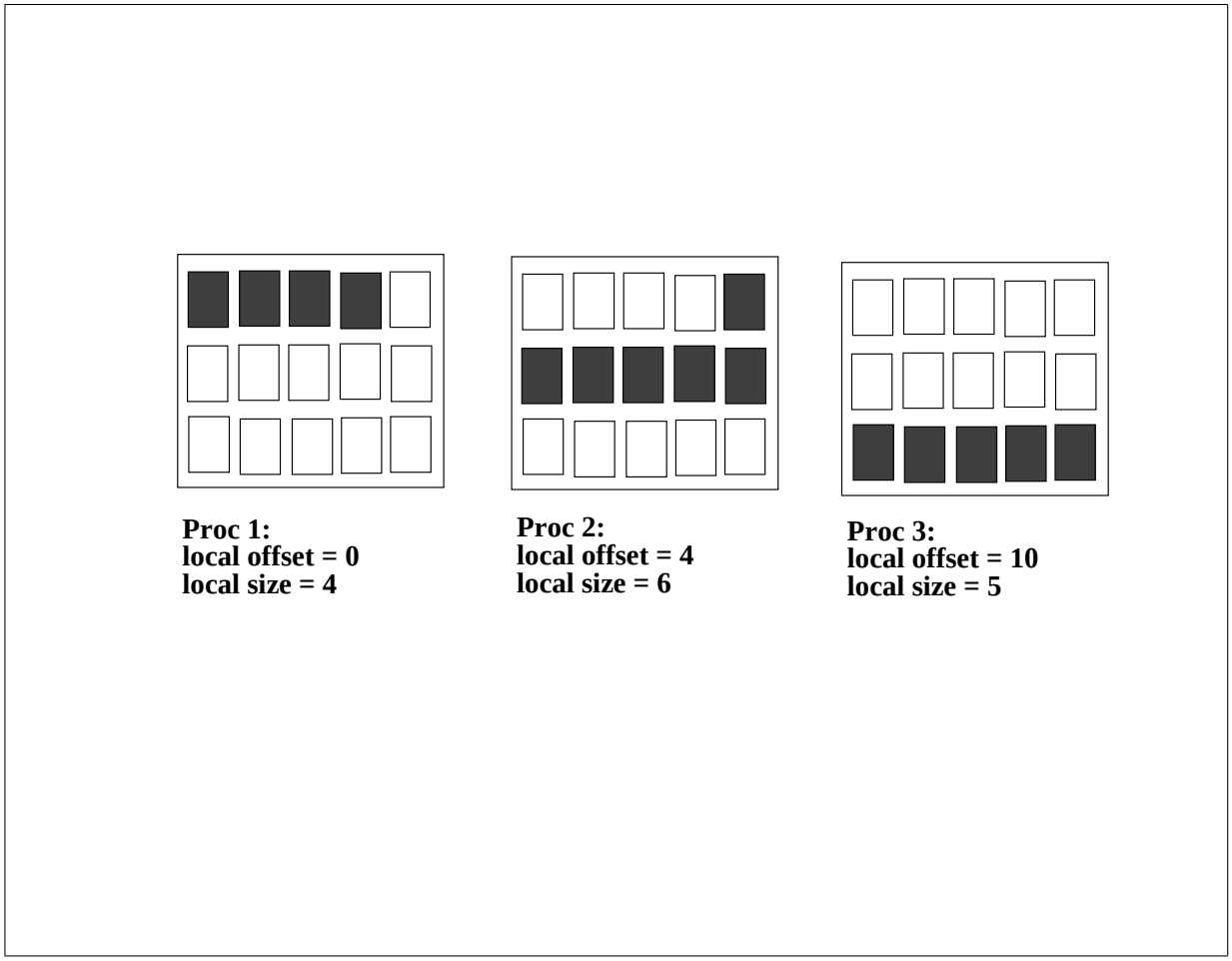

Figure 2.3: Apple partition. It is assumed here that the global index starts at 0 in the upper left corner.

- `ig_paral(2)` = 0
- `ig_paral(3)` = the total grid size.

Apple partition

Each partition is a segment of the global domain, described by its global offset and its local size. In this case, we have `ig_paral(1:3)`:

- `ig_paral(1)` = 1 (indicates an Apple partition)
- `ig_paral(2)` = the segment global offset
- `ig_paral(3)` = the segment local size

Figure 2.3 illustrates an Apple partition over 3 processes.

Box partition

Each partition is a rectangular region of the global domain, described by the global offset of its upper left corner, and its local extents in the X and Y dimensions. The global extent in the X dimension must also be given. In this case, we have `ig_paral(1:5)`:

- `ig_paral(1)` = 2 (indicates a Box partition)
- `ig_paral(2)` = the upper left corner global offset
- `ig_paral(3)` = the local extent in x
- `ig_paral(4)` = the local extent in y

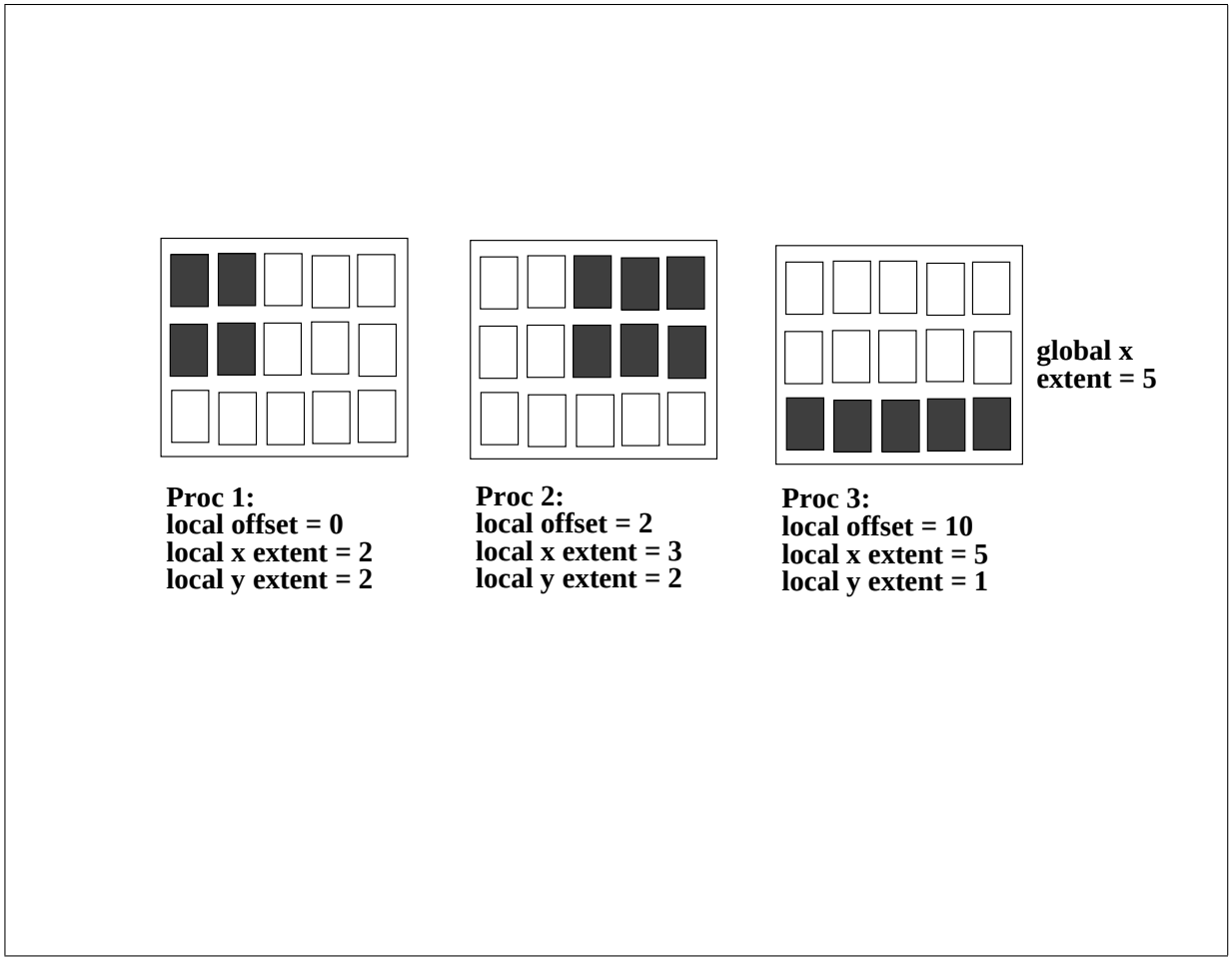

Figure 2.4: Box partition. It is assumed here that the global index starts at 0 in the upper left corner.

- `ig_paral(5)` = the global extent in x.

Figure 2.4 illustrates a Box partition over 3 processes.

Orange partition

Each partition is an ensemble of segments in the global domain. Each segment is described by its global offset and its local extent. In this case, we have `ig_paral(1:N)` where `N = 2 + 2*number of segments`

- `ig_paral(1)` = 3 (indicates a Orange partition)
- `ig_paral(2)` = the total number of segments for the partition (limited to 200 presently, see note for ig_paral(4) for Box partition above)
- `ig_paral(3)` = the first segment global offset
- `ig_paral(4)` = the first segment local extent
- `ig_paral(5)` = the second segment global offset
- `ig_paral(6)` = the second segment local extent
- ...
- `ig_paral(N-1)` = the last segment global offset
- `ig_paral(N)` = the last segment local extent

Figure 2.5 illustrates an Orange partition with 3 segments for one process. The other process partitions are not illustrated.

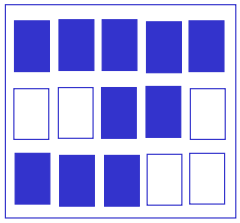

Figure 2.5: Orange partition for one process. It is assumed here that the global index starts at 0 in the upper left corner.

Points partition

This partition is a list of global indices associated with each process. The index naming is arbitrary but must be consistent between all processes involved in the partition description. In this case, we have `ig_paral(1:N)` where `N = 2 + number of points`

- `ig_paral(1)` = 4 (indicates a Points partition)
- `ig_paral(2)` = number of points in the partition
- `ig_paral(3)` = the first global index
- `ig_paral(4)` = the second global index
- ...
- `ig_paral(N)` = the last global index

2.2.4 Grid data file definition

Grid data files are required by OASIS3-MCT for specific operations, see sections 4 and 5.1. These grid data files can be created by the user before the run or can be written directly at run time by the components with the following routines. If a grid data files does not exist, the corresponding routine will create it; if the grid data file exists, the routine can be used to **add** grid definition fields but it will not **overwrite** grid definition fields already existing in the file with the same grid name.

These routines can be called only by one component process to write the whole grid or by each process holding a part of a grid. In the former case, optional argument `il_part_id` is not needed and the arrays handling the longitudes of the grid points or corners (`lon`, `clon`), the latitudes of the grid points or corners (`lat`, `clat`), the masks (`mask`), fracs (`frac`), and areas (`area`) of the grid cells need to cover the whole grid; in the later case, the `il_part_id` returned by `oasis_def_partition` needs to be provided as input argument and the arrays need to cover only the local partition of the grid.

The field names in the *grids.nc*, *masks.nc*, and *areas.nc* follow a well-defined convention. The fields are normally two-dimensional, and each field name consists of a grid acronym followed by a string that identifies the field. For instance, the center latitudes for the grid torc will be called torc.lat and the center longitudes will be called torc.lon in the netcdf file. The *grids.nc* file contains the center latitudes (.lat) and longitudes (.lat) as well as the corner latitudes (.cla) and corner longitudes (.clo). The corner fields have a third dimension associated with the number of corners per gridcell. The *area.nc* file constains the area field (.srf). The *masks.nc* file contains the mask (.msk) and frac (.frc) fields.

- `CALL oasis_start_grids_writing (flag)` or
- `CALL prism_start_grids_writing (flag)`
  - `flag [INTEGER; OUT]`: always 1

  Must be called to start the grid writing process.

- `CALL oasis_write_grid (cgrid, nx_global, ny_global, lon, lat, il_part_id)`
- `CALL prism_write_grid (cgrid, nx_global, ny_global, lon, lat, il_part_id)`
  - `cgrid [CHARACTER; IN]`: grid name prefix (see 3.3 and 5.1); maximum length of 64 characters (4 are usually used for historical reasons)
  - `nx_global [INTEGER; IN]` : first dimension of the global grid
  - `ny_global [INTEGER; IN]` : second dimension of the global grid (`=1` if the grid is expressed as a 1D vector)
  - `lon [REAL, DIMENSION(nx,ny); IN]` : single or double real array of longitudes covering the whole grid (`nx=nx_global`, `ny=ny_global`) or only the local partition (degrees East).

  - lat [REAL, DIMENSION(nx,ny); IN] : single or double real array of latitudes covering the whole grid (nx=nx_global, ny=ny_global) or only the local partition (degrees North)
  - il_part_id [INTEGER, OPTIONAL; IN]: partition ID returned by oasis_def_partition, see 2.2.3; needed if each component task holding a part of a decomposed grid writes its own part of the grid.

  Writes the component grid longitudes and latitudes. Longitudes must be given in degrees East in the interval -360.0 to 720.0. Latitudes must be given in degrees North in the interval -90.0 to 90.0. Note that if some grid points overlap, it is recommended to define those points with the same number (e.g. 90.0 for both, not 450.0 for one and 90.0 for the other) to ensure automatic detection of overlap by OASIS3-MCT (which is essential to have a correct conservative remapping SCRIPR/CONSERV, see section 4.3).

- CALL oasis_write_corner (cgrid, nx_global, ny_global, nc, clon, clat, il_part_id))
- CALL prism_write_corner (cgrid, nx_global, ny_global, nc, clon, clat, il_part_id))

  - cgrid , nx_global , ny_global , il_part_id : as for oasis_write_grid
  - nc [INTEGER; IN] : number of corners per grid cell (can be any number)
  - clon [REAL, DIMENSION (nx,ny,nc);IN] : single or double real array of corner longitudes covering the whole grid (nx=nx_global, ny=ny_global) or only the local partition (in degrees_East)
  - clat [REAL, DIMENSION (nx,ny,nc);IN] : single or double real array of corner latitudes covering the whole grid (nx=nx_global, ny=ny_global) or only the local partition (in degrees_North)

  Writes the grid cell corner longitudes and latitudes (counterclockwise sense). Longitudes must be given in degrees East in the interval -360.0 to 720.0. Latitudes must be given in degrees North in the interval -90.0 to 90.0. Note also that cells larger than 180.0 degrees in longitude are not supported. Writing of corners is optional as corner information is needed only for SCRIPR/CONSERV (see section 4.3). If called, needs to be called after oasis/prism_write_grid.

- CALL oasis_write_mask (cgrid, nx_global, ny_global, mask, il_part_id, companion)
- CALL prism_write_mask (cgrid, nx_global, ny_global, mask, il_part_id, companion)

  - cgrid , nx_global , ny_global , il_part_id : as for oasis_write_grid
  - mask [INTEGER, DIMENSION(nx,ny) ;IN] : mask array covering the whole grid (nx=nx_global, ny=ny_global) or only the local partition. Be careful about OASIS3-MCT historical convention (!): 0 = not masked (i.e. active), 1 = masked (i.e. not active).
  - companion [CHARACTER ;IN; OPTIONAL] : the character string value associated with the mask field attribute coherent_with_grid'. This will be written to the *masks.nc* netcdf file with the mask field. It is purely informational and used in cases where the mask field is derived from or consistent with another grid.

  Writes the component grid mask. The mask field should be consistent with the frac field (see below) and will define the 0/1 mask of the grid cell. The mask field is used by both the SCRIPR map generation function and in the global CONSERV operations if defined. The mask field is written to the *masks.nc* file.

- CALL oasis_write_frac (cgrid, nx_global, ny_global, frac, il_part_id, companion)
- CALL prism_write_frac (cgrid, nx_global, ny_global, frac, il_part_id, companion)
  - cgrid , nx_global , ny_global , il_part_id : as for oasis_write_grid
  - frac [REAL, DIMENSION(nx,ny) ;IN] : single or double real frac array covering the whole grid (nx=nx_global, ny=ny_global) or only the local partition.
  - companion [CHARACTER ;IN; OPTIONAL] : the character string value associated with the frac field attribute "coherent_with_grid". It should refer to the acronym of the grid which mask was used to define the fraction of the current grid (see section 4.4). This will be written to the *masks.nc* NetCDF file with the fraction field. It is purely informational and used in cases where the frac field is derived from or consistent with another grid mask.

  Writes the component grid cell fractions. This should be consistent with the mask field and defines the fraction of the grid cell that is active (i.e. not masked). The fraction field is only used in the global CONSERV operations. Either the mask or fractions must be defined for that operation. If both are defined, they must be consistent; OASIS3-MCT will abort if they are not coherent or if both are missing. Note that by OASIS3-MCT conventions for the mask, a gridcell with mask=0 (active) should have a fractions greater than 0 and a gridcell with mask=1 (inactive) should have a fractions equal to 0. The fraction field is written to the *masks.nc* file.

- CALL oasis_write_area (cgrid, nx_global, ny_global, area, il_part_id)
- CALL prism_write_area (cgrid, nx_global, ny_global, area, il_part_id)
  - cgrid , nx_global , ny_global , il_part_id : as for oasis_write_grid
  - area [REAL, DIMENSION(nx,ny); IN] : single or double real array of grid cell areas covering the whole grid (nx=nx_global, ny=ny_global) or only the local partition

  Writes of the component grid cell areas. Needed for some SCRIPR options and for the CONSERV operation (see section 4.4). The area field is written to the *areas.nc* file. The surfaces may be given in any units but they must be the same on the source and target sides. Furthermore they must be in square radians if the True Area (TR) correction is activated, see section 4.3.

- CALL oasis_write_angle (cgrid, nx_global, ny_global, angle, il_part_id)
- CALL prism_write_angle (cgrid, nx_global, ny_global, angle, il_part_id)
  - cgrid , nx_global , ny_global , il_part_id : as for oasis_write_grid
  - angle [REAL, DIMENSION(nx,ny); IN] : single or double real array of grid cell angles covering the whole grid (nx=nx_global, ny=ny_global) or only the local partition

  Writes of the component grid cell angles. The angle field is written to the *grids.nc* file. This field does not play a role in OASIS3-MCT implementation and is never needed.

- CALL prism_terminate_grids_writing() or
- CALL oasis_terminate_grids_writing()

The creation of the different grid data files is completed in the routine oasis_enddef.

2.2.5 Coupling field declaration

All processes of a component that send or receive a coupling field, or a part of it, needs to declare the coupling field.

Processes not implied in the coupling do not have to call this routine at all (for backward compatibility with OASIS3-MCT_2.0, they may still call it with any name and il_part_id).

- `CALL oasis_def_var (var_id, name, il_part_id, var_nodims, kinout, var_type, kinfo)` or
- `CALL oasis_def_var (var_id, name, il_part_id, var_nodims, kinout, var_actual_shape, var_type, kinfo)` or
- `CALL prism_def_var_proto(var_id, name, il_part_id, var_nodims, kinout, var_type, kinfo)` or
- `CALL prism_def_var_proto(var_id, name, il_part_id, var_nodims, kinout, var_actual_shape, var_type, kinfo)`
  - `var_id [INTEGER; OUT]`: coupling field ID. Note that all coupling fields appearing in the *namcouple* must be defined with a call to `oasis_def_var`; not doing so would lead to an abort. But all fields defined with a call to `oasis_def_var` must not necessarily appear in the *namcouple*. If a field does not appear in the *namcouple*, the `var_id` returned by the `oasis_def_var` will be equal to -1; the value of the `var_id` should be tested and the corresponding `oasis_put` and `oasis_get` should not be called if `var_id` equals -1. These constraints are imposed to avoid that a typo in the *namcouple* would lead to coupling exchanges not corresponding to what the user intends to activate.
  - `name [CHARACTER; IN]`: field symbolic name (as in the *namcouple*); maximum length of 80 characters
  - `il_part_id [INTEGER; IN]`: partition ID returned from `oasis_def_partition` (see section 2.2.3)
  - `var_nodims [INTEGER, DIMENSION(2); IN]`: is an integer array of size two. The first element, var_nodims(1), is not used anymore in OASIS3-MCT, so its value can be anything; The second element, var_nodims(2), is the number of fields in a bundle (this will be 1 for unbundle fields or greater than 1 for fields that are bundled; note that if var_nodims(2)=0, it will be automatically reset to 1 in the routine, to ensure backward compatibility).
  - `kinout [INTEGER; IN]`: `OASIS_In` or `PRISM_In` (i.e. = 21) for fields received by the component; `OASIS_Out`, `PRISM_Out` (i.e. = 20) for fields sent by the component [2].
  - `var_actual_shape [INTEGER, DIMENSION(2*id_var_nodims(1)), IN]`: is not used anymore. The interface has recently been overloaded, and this argument is no longer required. But for backwards compatibility, it can still be passed; if so, it has to be a vector of integers of any length (for simplicity we advise to pass a vector of length 1).
  - `var_type [INTEGER; IN]`: type of coupling field array; put `OASIS_Real` or `PRISM_Real` (i.e. = 4) for single or double precision real arrays. All coupling data is treated as double precision in the coupling layer, but conversion to or from single precision data is supported in the interface.
  - `kinfo [INTEGER; OUT]`: returned error code.

2.2.6 End of definition phase

All processes of components at least partly involved in the coupling (e.g. `comp3` in figure 2.2) have to close the definition phase. Different configurations of components and corresponding use of `oasis_enddef` are described in section 2.1 and on figures 2.1 and 2.2.

- `CALL oasis_enddef (kinfo)`
- `CALL prism_enddef_proto(kinfo)`
  - kinfo [INTEGER; OUT]: returned error code.

[2] Parameters OASIS_In, PRISM_In, OASIS_Out, PRISM_Out are defined in oasis3-mct/lib/psmile/src/mod_oasis_parameters.F90

2.2.7 Sending "put" and receiving "get" actions

This section describes how to send (put) and receive (get) fields through OASIS-MCT API. This coupling interface supports several ranks and types of coupling fields. First, the fields passed to the interface can be 4 byte or 8 byte reals. The field decomposition must be consistent with the decomposition defined by the grid partition (see 2.2.3)[3] and the fields can be bundled (i.e. have an extra non-spatial dimension for something like different ice categories). The bundle dimension is always the last dimension in the field passed to the get and put routines. And the size of the bundle dimension must match the value defined for the variable in var_nodims(2) in the `oasis_def_var` interface (see section 2.2.5).

So in general, the fields passed into the get and put interface can have rank 1, 2, or 3 and include the following possible options where fld can be a 4 byte or 8 byte real array.

- 1D, fld(:) = a single, unbundle field of decomposition rank 1.
- 2D, fld(:,:) = a single, unbundle field of decomposition rank 2.
- 1D bundle, fld(:,:) = a bundle set of fields of decomposition rank 1. The size of the second dimension must equal the number of fields in the bundle, defined by var_nodims(2) in the `oasis_def_var` interface.
- 2D bundle, fld(:,:,:) = a bundle set of fields of decomposition rank 2. The size of the third dimension must equal the number of fields in the bundle, defined by var_nodims(2) in the `oasis_def_var` interface.

Different bundle fields can have different numbers of fields, but for a given bundle field, the number of fields must match on the send and receive side. This is explicitly checked within the coupling layer and will lead to an abort if not done correctly. It is possible to define a 1D bundle or 2D bundle field with a bundle dimension of 1, for a bundle that contains only one single field.

Finally, the bundle field option can be used to bundle together multi-level variables, multiple related fields, and other types of fields. The fields must share a common partition and common namcouple settings (e.g. interpolation) to be bundle. While this is a useful feature for multi-level fields, **this does not mean that 3D interpolation is supported**. Each field in the bundle is treated internally as a separate field in the coupling layer without any information about the relationship between the fields in the bundle. In fact, the bundle field variables are internally renamed and a field number is appended to the variable name to keep track of the distinct fields in the bundle. That updated variable name will appear in restart and output files.

Sending a coupling (or I/O) field or writing a coupling restart file

In the component time step loop, each process sends its part of the coupling (or I/O) field.

- `CALL oasis_put (var_id,date,fld1,info,fld2,fld3,fld4, fld5,write_restart)`
- `CALL prism_put_proto(var_id, date, fld1, info, fld2, fld3, fld4, fld5, write_restart)`
  - `var_id [INTEGER; IN]`: field ID (returned from corresponding `oasis_def_var`, see section 2.2.5)
  - `date [INTEGER; IN]`: number of seconds (or any other time units as long as the same are used in all components and in the *namcouple*) at the time of the call (by convention at the beginning of the timestep)
  - `fld1 [REAL, IN]`: coupling (or I/O) field array; can be 1D, 2D, bundle 1D, or bundle 2D, see above.
  - `info [INTEGER; OUT]`: returned info code:
    - `OASIS_Sent` (=4) if the field was sent to another component

[3] But the decomposition of a field does not necesseraly have to match the rank of the grid partition description (i.e. it can be expressed in either 1D or 2D).

  - OASIS_LocTrans (=5) if the field was only used in a time transformation (not sent, not output)
  - OASIS_ToRest (=6) if the field was written to a restart file only
  - OASIS_Output (=7) if the field was written to an output file only
  - OASIS_SentOut (=8) if the field was both written to an output file and sent to another component
  - OASIS_ToRestOut (=9) if the field was written both to a restart file and to an output file.
  - OASIS_WaitGroup (=14) if the field was not sent because it is part of a group. It will be sent only when the oasis_put of the last field in the group will be called; however, the field is buffered and therefore the field array can be modified in the component code when returning from the oasis_put call.
  - OASIS_Ok (=0) otherwise and no error occurred.
- fld2 [REAL, IN, OPTIONAL]: optional 2^{nd} coupling field array; can be 1D, 2D, bundle 1D, or bundle 2D. Rank and size must match fld1.
- fld3 [REAL, IN, OPTIONAL]: optional 3^{rd} coupling field array; can be 1D, 2D, bundle 1D, or bundle 2D. Rank and size must match fld1.
- fld4 [REAL, IN, OPTIONAL]: optional 4^{th} coupling field array; can be 1D, 2D, bundle 1D, or bundle 2D. Rank and size must match fld1.
- fld5 [REAL, IN, OPTIONAL]: optional 5^{th} coupling field array; can be 1D, 2D, bundle 1D, or bundle 2D. Rank and size must match fld1.
- write_restart [LOGICAL, IN, OPTIONAL]: optional argument to write an intermediate restart file associated with the variable var_id at the current timestep (see below).

To ensure a proper use of the oasis_put, one has to take care of the following aspects:

- A 2^{nd}, 3^{rd}, 4^{th} and 5^{th} source field can be passed as optional arguments. If so, the 2^{nd}, 3^{rd}, 4^{th} and 5^{th} set of weights present in the remapping file will be applied, respectively (see section 5.4 for the remapping file format). This will be used for example for the SCRIPR/BICUBIC remapping for which a 1^{st}, 2^{nd}, 3^{rd}, 4^{th} set of weights should be respectively applied to the field value, its gradient in the first dimension, its gradient in the second dimension, and its cross-gradient in that order. For SCRIPR/BICUBIC, fld2, fld3 and fld4 are therefore mandatory.

  This will be used also for the CONSERV/SECOND for which a 1st, 2nd, 3rd set of weights should be respectively applied to the field value, its gradient with respect to the latitude (θ) $\frac{\delta f}{\delta \theta}$ and its gradient with respect to the longitude (ϕ) $\frac{1}{cos\theta}\frac{\delta f}{\delta \phi}$ in that order. For CONSERV/SECOND, fld2 and fld3 are therefore mandatory.

  Bicubic and higher order remapping are therefore supported given that the higher order fields are provided at each time step as oasis_put arguments. Note that if fld3, or fld4, or fld5 are passed, fld2, or fld3 and fld2, or fld4 and fld3 and fld2 must also be passed respectively.
- Note that from OASIS3-MCT_4.0 onwards, the number of weights in the remapping file and the number of fields in the coupling restart file (when such a file is needed) must strictly match the number of source fields passed to the oasis_put .
- This routine may be called by the component at each timestep. The convention for the date argument is to indicate the time at the beginning of the timestep. The sending is actually performed if the time obtained by adding the field lag (LAG in the *namcouple*, if any, with LAG=0 by default) to the date corresponds to a time at which it should be activated, given the coupling or I/O period indicated by the user in the *namcouple* (see section 3).
- By convention, the first coupling of a run occurs at date=0.

- For a coupling field with a positive lag, the coupling restart file (see section 5.2) is automatically overwritten by the `oasis_put` when the `date+LAG=runtime`.
- The total run time should match an integer number of coupling periods.
- If a local time transformation is indicated for the field by the user in the *namcouple* (INSTANT, AVERAGE, ACCUMUL, T_MIN or T_MAX, see section 4), it is automatically performed and the resulting field is finally sent at the coupling or I/O frequency. For non-instantaneous transformations, partially transformed fields will be written to the restart file at the end of the run for use on the next component startup, when needed.
- A coupling field sent by a source component can be associated with more than one target field and component, with different entries in the *namcouple* configuration file. In that case, the source component needs to send the field only once and the corresponding data will arrive at multiple targets as specified in the *namcouple*. Different coupling frequencies and transformations are allowed for different coupling exchanges of the same field. If coupling restart files are required (either if a `LAG` or if a `LOCTRANS` transformation is specified), it is mandatory to specify different files for the different fields.
- Trying to send with `oasis_put` a field declared with a `oasis_def_var` but not defined in the configuration file *namcouple* will lead to an abort. When a field is not defined in the *namcouple*, the field ID returned by the `oasis_def_var` is equal to -1 and the corresponding `oasis_put` should not be called.
- Coupling multiple fields via a single communication is supported through colon delimited field lists in the namcouple (see 3.3.1). All fields will use the namcouple settings for that entry. In the component model codes, these fields are still sent (**"put"**) one at a time. Inside OASIS3-MCT, the fields are stored and a single mapping and send instruction is executed for all fields. This is useful to reduce communication costs by aggregating multiple fields into a single communication when multiple fields have the same coupling transformations.

  This option does not put any constraint on the order of the related `oasis_put` and `oasis_get` in the codes.

  As they appear in one single entry line, these fields must share the same coupling restart file but this restart file may contain other fields.
- The optional argument, `write_restart`, in the `oasis_put` routine is **false** by default. If a user sets that argument to true, an "intermediate" restart is written for that field **only for that timestep**. The `write_restart` saves the data that exists at the time of the call, taking into account LOCTRANS operations. In cases where multiple fields are coupled as a single operation in the model (indicated via a list of colon delimited fields in the *namcouple*, see 3.3.1), users are encouraged to specify the `write_restart` flag on ALL `oasis_put` calls at a given time for this set of fields. Restarts are created with a 9 digit timestamp in their filename, corresponding to the time in seconds of the oasis_put call, e.g. TA000003600_rst4.nc or TC000014400_rst4.nc. A restart file that starts with TA is a restart file associated with LOCTRANS operations; a restart file that starts with TC is a restart file associated with coupling operations. The coupling restart filename defined in the namcouple (e.g. rst4.nc) is used to generate the filename of these intermediate restart files.

Receiving a coupling (or I/O) field

In the component time stepping loop, each process receives its part of the coupling field.

- `CALL oasis_get (var_id, date, fld, info)`
- `CALL prism_get_proto(var_id, date, fld, info)`
  - `var_id [INTEGER; IN]`: field ID (returned by the corresponding `oasis_def_var`)
  - `date [INTEGER; IN]`: number of seconds (or any other time units as long as the same are used in all components and in the *namcouple*) at the time of the call (by convention at the

beginning of the timestep)

- `fld [REAL, OUT]`: I/O or coupling field array; can be 1D, 2D, bundle 1D, or bundle 2D.
- `info [INTEGER; OUT]`: returned info code:
  - OASIS_Recvd(=3) if the field was received from another component
  - OASIS_FromRest (=10) if the field was read from a restart file only
  - OASIS_Input (=11) if the field was read from an input file only
  - OASIS_RecvOut (=12) if the field was both received from another component and written to an output file
  - OASIS_FromRestOut (=13) if the field was both read from a restart file and written to an output file
  - OASIS_Ok (=0) otherwise and no error occurred.

To ensure a proper use of the `oasis_get`, one has to take care of the following aspects:

- This routine may be called by the component at each timestep. The `date` argument is automatically analysed and the receiving action is actually performed only if `date` corresponds to a time for which it should be activated, given the period indicated by the user in the *namcouple*. An exchange at the beginning of the run at time=0 is expected.
- Trying to receive with `oasis_get` a field declared with a `oasis_def_var` but not defined in the configuration file *namcouple* will lead to an abort. In this case, the field ID returned by the `oasis_def_var` is equal to -1 and the corresponding `oasis_get` should not be called.
- If a coupling field has a positive lag, the coupling field that matches the `oasis_get` at time=0 is automatically read in from a coupling restart file and sent to match that `oasis_get` under the `oasis_enddef` of the source model (see section 2.5.3).
- Coupling multiple fields via a single communication is supported through colon delimited field lists in the namcouple (see 3.3.1). All fields will use the namcouple settings for that entry. In the component model codes, these fields are still received (via an `oasis_get`) one at a time. Inside OASIS3-MCT, the fields are stored and a single mapping and receive instruction is executed for all fields. This is useful in cases where multiple fields have the same coupling transformations and to reduce communication costs by aggregating multiple fields into a single communication. If a LOCTRANS transformation is needed for these multiple fields, it is necessary to define a restart file for these multiple fields.

2.2.8 Termination

- `CALL oasis_terminate (kinfo)`
- `CALL prism_terminate_proto(kinfo)`
  - `kinfo [INTEGER; OUT]`: returned error code.

  All processes of components at least partly involved in the coupling (e.g. `comp3` in figure 2.2) have to terminate the coupling by calling this routine[4](normal termination). Different configurations of components and corresponding use of `oasis_terminate` are described in section 2.1 and on figures 2.1 and 2.2.

2.2.9 Auxiliary routines

The following auxiliary routines are currently available.

- `CALL oasis_abort (compid, routine_name, abort_message, file, line, rcode)`

[4]If the process called `MPI_Init` (before calling `oasis_init_comp`), it must also call `MPI_Finalize` explicitly, but only after calling `oasis_terminate_proto`.

- CALL prism_abort_proto(compid, routine_name, abort_message, file, line, rcode)
  - compid [INTEGER; IN]: component ID (from oasis_init_comp)
  - routine_name [CHARACTER*; IN]: name of calling routine
  - abort_message [CHARACTER*; IN]: message to be written out
  - file [CHARACTER*; OPTIONAL; IN]: file from which oasis_abort is called from
  - line [INTEGER, OPTIONAL; IN]: line in file from which oasis_abort is called from
  - rcode [INTEGER, OPTIONAL; IN]: Optional argument. When OASIS3-MCT aborts, it returns rcode if it is present, else it returns 1

  If a process needs to abort voluntarily, it should do so by calling oasis_abort. This will ensure a proper termination of all processes in the coupled model communicator. This routine writes the name of the calling component, the name of the calling routine, and the message to the process debug file (see $NLOGPRT in section 3.2). This routine cannot be called before oasis_init_comp.

- CALL oasis_get_debug(debug, kinfo)
- CALL prism_get_debug(debug, kinfo)
  - debug [INTEGER; OUT]: output debug value
  - kinfo [INTEGER; OUT; OPTIONAL]: returned error code

  This routine may be called at any time to retrieve the current OASIS3-MCT internal debug level (see $NLOGPRT in section 3.2). This is useful if the user wants to return the original debug value after changing it.

- CALL oasis_set_debug(debug, kinfo)
- CALL prism_set_debug(debug, kinfo)
  - debug [INTEGER; IN]: input debug value
  - kinfo [INTEGER; OUT; OPTIONAL]: returned error code

  This routine may be called at any time to change the debug level in OASIS3-MCT. This method allows users to vary the debug level at different points in the component integration.

- CALL oasis_get_intercomm(new_comm, cdnam, kinfo)
- CALL prism_get_intercomm(new_comm, cdnam, kinfo)
  - new_comm [INTEGER; OUT]: MPI inter-communicator
  - cdnam [CHARACTER*; IN]: other component name (i.e. 2nd argument of the call to oasis_init_comp in that component)
  - kinfo [INTEGER; OUT; OPTIONAL]: returned error code

  This routine sets up an MPI inter-communicator between two components, the local component and the component associated with cdnam. This call is collective across the tasks of the two components only. An MPI inter-communicator preserves the rank of the original communicators and does not allow collective communication within the communicator. It provides point to point communication between two non-overlapping MPI groups. This method must be called synchronously across all components involved to minimize the chance of a deadlock, and it should be called only after oasis_enddef is called. See oasis_get_intracomm below to create an intra-communicator.

- CALL oasis_get_intracomm(new_comm, cdnam, kinfo)
- CALL prism_get_intracomm(new_comm, cdnam, kinfo)
  - new_comm [INTEGER; OUT]: MPI intra-communicator
  - cdnam [CHARACTER*; IN]: other component name (i.e. 2nd argument of the call to oasis_init_comp in that component). This argument is a single string.

  - `kinfo [INTEGER; OUT; OPTIONAL]`: returned error code

  This routine sets up an MPI intra-communicator between two components, the local component and the component defined by the string `cdnam`. This call is collective across the tasks of the two components creating the intra-communicator only, and it must be called synchronously across all tasks of the two components to minimize the chance of a deadlock. It should be called only after `oasis_enddef` is called. This method creates a new communicator consisting of a new collective group of tasks with new ranks. This communicator supports collective communications and is more typically used in MPI applications than inter-communicators (see `oasis_get_intercomm` above). See also `oasis_get_multi_intracomm` for another method that supports creating an MPI intra-communicator between two or more components.

- `CALL oasis_get_multi_intracomm(new_comm, cdnam, root_ranks, kinfo)`
- `CALL prism_get_multi_intracomm(new_comm, cdnam, root_ranks, kinfo)`
  - `new_comm [INTEGER; OUT]`: MPI intra-communicator
  - `cdnam [CHARACTER*; IN]`: array of component names (i.e. 2nd argument of the call to `oasis_init_comp` in that component). This argument is a 1d array of character strings (i.e. cdnam(:)).
  - `root_ranks [INTEGER; OUT]`: array of root ranks. This argument is a 1d integer array (i.e. root_ranks(:)) of the same size as `cdnam`.
  - `kinfo [INTEGER; OUT]`: returned error code

  This routine sets up an MPI intra-communicator between two or more components defined by the component names passed in the `cdnam` array argument. The local model name MUST BE one of the models defined in the `cdnam` array. The component names must be valid names, but empty strings are allowed and ignored. This call is collective across all the tasks of the components defined in `cdnam`, and it must be called synchronously and consistently across all tasks of those components to minimize the chance of a deadlock. It should be called only after `oasis_enddef` is called. This method creates a new communicator consisting of a new collective group of tasks with new ranks. The root ranks of the individual components relative to the new communicator is output in the `root_ranks` argument. The size of `cdnam` and `root_ranks` should be identical, and the values of `root_ranks` are consistent with the order of `cdnam`.. This communicator supports collective communications and is more typically used in MPI applications than inter-communicators (see `oasis_get_intercomm` above). See also `oasis_get_intracomm` for another method that supports creating an MPI intra-communicator between two components.

- `CALL oasis_put_inquire(var_id, date, kinfo)`
- `CALL prism_put_inquire_proto(var_id, date, kinfo)`
  - `var_id [INTEGER; IN]`: field ID (from corresponding oasis_def_var)
  - `date [INTEGER; IN]`: as in `oasis_put`, number of seconds (or any other time units as long as the same are used in all components and in the *namcouple*) in the run at the time of the call
  - `kinfo [INTEGER; OUT]`: returned info code
    - OASIS_Sent(=4) if the field would be sent to another component
    - OASIS_LocTrans (=5) if the field would be only used in a time transformation (not sent, not output)
    - OASIS_ToRest (=6) if the field would be written to a restart file only
    - OASIS_Output (=7) if the field would be written to an output file only
    - OASIS_SentOut (=8) if the field would be both written to an output file and sent to another component (directly or via OASIS3 main process)

  - OASIS_ToRestOut (=9) if the field would be written both to a restart file and to an output file.
  - OASIS_Ok (=0) otherwise and no error occurred.

  This routine may be called at any time to inquire what would happen to the field corresponding to that `var_id` if it was sent with an `oasis_put` at that same `date`). This maybe useful if, for example, the calculation of a coupling field is costly and if one wants to compute it only when it is really sent out.

- `CALL oasis_get_ncpl(var_id, ncpl, kinfo)`
- `CALL prism_get_ncpl_proto(var_id, ncpl, kinfo)`
  - `var_id [INTEGER; IN]`: field ID (from corresponding oasis_def_var)
  - `ncpl [INTEGER; OUT]`: number of coupling exchanges in which the field is involved (i.e. when a field is sent to multiple targets)
  - `kinfo [INTEGER; OUT]`: returned info code

  This routine returns the number of coupling exchanges in which the field with that `var_id` is involved. This number is needed to get the coupling frequencies with the routine oasis_get_freqs, see below.

- `CALL oasis_get_freqs(var_id, mop, ncpl, cpl_freqs, kinfo)`
- `CALL prism_get_freqs_proto(var_id, mop, ncpl, cpl_freqs, kinfo)`
  - `var_id [INTEGER; IN]`: field ID (from corresponding oasis_def_var)
  - `mop [INTEGER; IN]`: OASIS_Out or OASIS_In
  - `ncpl [INTEGER; IN]`: number of couplings in which the field is involved (i.e. when a field is sent to multiple targets)
  - `cpl_freqs [INTEGER; DIMENSION(ncpl); OUT]`: coupling period(s) (in number of seconds) of field var_id. There is one coupling period for each coupling exchange in which the field is involved
  - `kinfo [INTEGER; OUT]`: returned info code

  This routine can be used to retrieve the coupling period(s) of field with corresponding `var_id`, as defined in the *namcouple*

2.3 OASIS3-MCT C API

OASIS3-MCT is distributed with C bindings and can be called from models written in C and in C++. These bindings leverage the Fortran ISO_C_BINDING standard. The C bindings can be compiled into static or shared libraries by the OASIS3-MCT TopMakefileOasis3 as documented in 6.1.

The C interfaces largely match up with equivalent interfaces in Fortran. An interface named `oasis_interface` in Fortran can be expected to be named `oasis_c_interface` in C.

All of the C interfaces return an integer error code which can be tested against the `OASIS_Ok` (or the equivalent `OASIS_Success`) constant. The `OASIS_CHECK_ERR` macro aborts OASIS3-MCT with a meaningful message in the debug files in case of failure. For most of the functions, the return code is consistent with the `kinfo` argument in the Fortran interfaces (see section 2.2). However, for the put and get communication functions, the return error code only indicates success or failure, while the detailed status is returned by the `kinfo` argument (equivalent to the returned value of `info` argument for the Fortran `oasis_put` and `oasis_get` routines, see section 2.2.7).

For example, the following constructs are equivalent in the two languages:

```
call oasis_get_localcomm(localcomm, kinfo)
if (kinfo .ne. OASIS_Ok)  &
   &  call OASIS_Abort(comp_id, "oasis_get_localcomm", &
```

```
   &  "Runtime error", __FILE__, __LINE__)
```

and

```
OASIS_CHECK_ERR(oasis_get_localcomm(localcomm))
```

For convenience a similar macro `OASIS_CHECK_MPI_ERR` has been defined for testing the return code of any MPI function against `MPI_SUCCESS` and cleanly aborting OASIS3-MCT in case of failure.

Use of these C bindings are illustrated in practical examples in directories `/C` in the different subdirectories in `pyoasis/examples/`. Notes and deviations from the Fortran standard are noted below.

- To use the OASIS3-MCT C bindings, the statement

  ```
  #include "oasis_c.h"
  ```

  needs to be added to the C model source code. All available parameters macros and interfaces are defined there.
- `int oasis_c_init_comp(int* compid, const char* comp_name, const bool coupled)`
  - This `oasis_c_init_comp` interface includes the `coupled` flag but NO communicator argument; use `oasis_c_init_comp_with_comm` below to indicate a communicator argument.
  - The `coupled` argument can receive the predefined mnemonic boolean constants `OASIS_Coupled` and `OASIS_Not_Coupled`
- `int oasis_c_init_comp_with_comm(int* compid, const char* comp_name, const bool coupled, const MPI_Comm commworld)`
  - This alternative `init_comp` interface has the `coupled` flag and a C `MPI_Comm` communicator
- `int oasis_c_get_localcomm(MPI_Comm* local_comm)`
  - Returns the local communicator by reference in a C `MPI_Comm*` type argument
- `int oasis_c_create_couplcomm(const int icpl, const MPI_Comm local_comm, MPI_Comm* coupl_comm)`
  - `icpl` is a C `int` type input argument (see `oasis_create_couplcomm` in section 2.2.2)
  - `loca_comm` is a C `MPI_Comm` type input argument (by value)
  - `coupl_comm` is a C `MPI_Comm*` type output argument (by reference)
- `int oasis_c_set_couplcomm(const MPI_Comm coupl_comm)`
  - Takes a C `MPI_Comm` type as an input argument (by value)
- `int oasis_c_def_partition(int* il_part_id, const int ig_paral_size, const int* ig_paral, const int ig_size, const char* name)`
  - `ig_paral_size` is the size of the array `ig_paral`: the `oasis_c.h` header provides a set of constants and macros for the size of every partition strategy, namely `OASIS_Serial_Params`, `OASIS_Apple_Params`, `OASIS_Box_Params`, `OASIS_Orange_Params(n_segments)`, `OASIS_Points_Params(n_points)`
  - the `oasis_c.h` header also provides a set of predefined constants for the storages positions in `ig_paral`, namely `OASIS_Strategy`, `OASIS_Segments`, `OASIS_Npoints`, `OASIS_Offset`, `OASIS_Length`, `OASIS_SizeX`, `OASIS_SizeY`, `OASIS_LdX`
  - the partition strategy, to be stored in the `OASIS_Strategy` position of `ig_paral`, can take one of the following predefined constants values: `OASIS_Serial`, `OASIS_Apple`, `OASIS_Box`, `OASIS_Orange`, `OASIS_Points`

- for the cases in which the ig_size and name arguments are not relevant for the partition definition, the placeholders OASIS_No_Gsize and OASIS_No_Name can be used instead

Here an example for the OASIS_Part_Apple strategy

```
int part_params[OASIS_Apple_Params];
part_params[OASIS_Strategy] = OASIS_Apple;
part_params[OASIS_Offset] = offset;
part_params[OASIS_Length] = local_size;
int part_id;
OASIS_CHECK_ERR(oasis_c_def_partition(&part_id, OASIS_Apple_Params,
                                      part_params, OASIS_No_Gsize,
                                      OASIS_No_Name));
```

- int oasis_c_start_grids_writing()
- int oasis_c_write_grid(const char* cgrid, const int nx_global, const int ny_global, const int nx_loc, const int ny_loc, const double* lon, const double* lat, const int il_partid)

  /local
  - nx_global and ny_global are the first and second dimensions of the global grid
  - nx_loc and ny_loc are the two dimensions of the local arrays lon and lat
  - lon and lat are stored with a Fortran compatible (column major) ordering. For example, if they are declared as double pointers, they have to be lon[ny_loc][nx_loc]
  - il_partid is relevant only for parallel writing; in case of single proc invocations, the constant OASIS_No_Part can be passed as a placeholder
- int oasis_c_write_corner(const char* cgrid, const int nx_global, const int ny_global, const int nc, const int nx_loc, const int ny_loc, const double* clon, const double* clat, const int il_partid)
  - nx_global and ny_global are the first and second dimensions of the global grid
  - nc is the maximum number of corners per cell
  - nx_loc and ny_loc are the two dimensions of the local arrays clon and clat
  - clon and clat are stored with a Fortran compatible (column major) ordering. If they are declared as triple pointers, they have to be clon[nc][ny_loc][nx_loc]
  - il_partid is relevant only for parallel writing. In case of single proc invocations, the constant OASIS_No_Part can be passed as a placeholder
- int oasis_c_write_mask(const char* cgrid, const int nx_global, const int ny_global, const int nx_loc, const int ny_loc, const int* mask, const int il_partid, const char* companion)
  - nx_global and ny_global are the first and second dimensions of the global grid
  - nx_loc and ny_loc are the two dimensions of the local array mask
  - mask is stored with a Fortran compatible (column major) ordering. If it is declared as a double pointer, it has to be mask[ny_loc][nx_loc]
  - il_partid is relevant only for parallel writing. In case of single proc invocations, the constant OASIS_No_Part can be passed as a placeholder
  - if no companion grid attribute is needed, the constant OASIS_No_Companion can be passed as a placeholder
- int oasis_c_write_frac(const char* cgrid, const int nx_global, const int ny_global, const int nx_loc, const int ny_loc, const double* frac,

`const int il_partid, const char* companion)`

- `nx_global` and `ny_global` are the first and second dimensions of the global grid
- `nx_loc` and `ny_loc` are the two dimensions of the local array `frac`
- `frac` is stored with a Fortran compatible (column major) ordering. If it is declared as a double pointer, it has to be `frac[ny_loc][nx_loc]`
- `il_partid` is relevant only for parallel writing. In case of single proc invocations, the constant `OASIS_No_Part` can be passed as a placeholder
- if no `companion` grid attribute is needed, the constant `OASIS_No_Companion` can be passed as a placeholder

- `int oasis_c_write_area(const char* cgrid, const int nx_global, const int ny_global, const int nx_loc, const int ny_loc, const double* area, const int il_partid)`
  - `nx_global` and `ny_global` are the first and second dimensions of the global grid
  - `nx_loc` and `ny_loc` are the two dimensions of the local array `area`
  - `area` is stored with a Fortran compatible (column major) ordering. If it is declared as a double pointer, it has to be `area[ny_loc][nx_loc]`
  - `il_partid` is relevant only for parallel writing. In case of single proc invocations, the constant `OASIS_No_Part` can be passed as a placeholder

- `int oasis_c_write_angle(const char* cgrid, const int nx_global, const int ny_global, const int nx_loc, const int ny_loc, const double* angle, const int il_partid)`
  - `nx_global` and `ny_global` are the first and second dimensions of the global grid
  - `nx_loc` and `ny_loc` are the two dimensions of the local array `angle`
  - `angle` is stored with a Fortran compatible (column major) ordering. If it is declared as a double pointer, it has to be `angle[ny_loc][nx_loc]`
  - `il_partid` is relevant only for parallel writing. In case of single proc invocations, the constant `OASIS_No_Part` can be passed as a placeholder

- `int oasis_c_terminate_grids_writing()`

- `int oasis_c_def_var(int* var_id, const char* name, const int il_part_id, const bundle_size, const int kinout, const int var_type)`
  - `bundle_size` is a scalar integer, corresponding to the second entry of the Fortran `var_nodims` array
  - `kinout` can receive one of the two predefined constants `OASIS_In` or `OASIS_Out` (also in the form `OASIS_IN` or `OASIS_OUT`)
  - `var_type` can receive one of the two predefined constants `OASIS_Real` or `OASIS_Double` (also in the form `OASIS_REAL` or `OASIS_DOUBLE`)

- `int oasis_c_enddef()`

- `int oasis_c_put(const int var_id, const int date, const int x_size, const int y_size, const int bundle_size, const int fkind, const int storage, const void* fld1, const bool write_restart, int* kinfo)`
  - This interface does not support higher order mapping through optional fields at this time.
  - `x_size` and `y_size` are the dimensions of the local portion of the domain (i.e. fld1). The order of these dimensions must be the same than the order of the dimensions of the arrays in the `grids.nc` file, which corresponds to the storage order in the corresponding internal

OASIS3-MCT Fortran work arrays.
Notice that for unstructured grids `y_size=1`

- `bundle_size` should be coherent with the argument of same name in the corresponding `oasis_c_def_var`
- `fkind` can take one of the two predefined constants `OASIS_Real` or `OASIS_Double` (also in the form `OASIS_REAL` or `OASIS_DOUBLE`) should be coherent with the argument `ktype` in `oasis_c_def_var`
- `storage` can take one of the two predefined constants `OASIS_COL_MAJOR` or `OASIS_ROW_MAJOR`.
  In the first case, the array containing the field has to be declared as `field[bundle_size][y_size][x_size]` (or any equivalent storage of total size `bundle_size*y_size*x_size` stored in column major order.
  In the second case, the field has to be declared as `field[x_size][y_size][bundle_size]` (or any equivalent row major storage).
  Notice that this choice implies an extra memory copy performed internally by OASIS3-MCT before acting on the field and is therefore to be avoided whenever possible.
- `write_restart` can take one of the two predefined constants `OASIS_Write_Restart` or `OASIS_No_Restart`
- `kinfo` contains a return status to be compared against the values of the `return_codes` enumeration in `oasis_c.h` (equivalent to the returned value of `info` argument for the Fortran `oasis_put` routine, see section 2.2.7)

- `int oasis_c_get(const int var_id, const int date, const int x_size, const int y_size, const int bundle_size, const int fkind, const int storage, void* fld1, int* kinfo)`
  - arguments are the same than for `oasis_c_put`, except that `kinfo` return status is equivalent to the returned value of `info` argument for the Fortran `oasis_get` routine, see section 2.2.7.

- `int oasis_c_terminate()`

- `int oasis_c_abort(const int compid, const char* routine_name, const char* abort_message, const char* file, const int line)`

- `int oasis_c_get_debug(int* debug)`

- `int oasis_c_set_debug(const int debug)`

- `int oasis_c_get_intercomm(MPI_Comm* new_comm, char* cdnam)`
  - Returns a C MPI_Comm type by reference

- `int oasis_c_get_intracomm(MPI_Comm* new_comm, char* cdnam)`
  - Returns a C MPI_Comm type as by reference

- `int oasis_c_get_multi_intracomm(MPI_Comm* new_comm, const int cdnam_size, char** cdnam, int* root_ranks)`
  - Returns a C MPI_Comm type as by reference
  - cdnam_size is the size of the array cdnam

- `int oasis_c_put_inquire(int var_id, int date, int* kinfo)`
  - `kinfo` contains a return status to be compared against the values of the `return_codes` enumeration in `oasis_c.h`

- `int oasis_c_get_ncpl(const int var_id, int* ncpl)`

- `int oasis_c_get_freqs(const int var_id, const int mop, const int ncpl, int* cpl_freqs)`

2.4 OASIS3-MCT python API

The source code of pyOASIS is in the directory `pyoasis/src`. Complete documentation is available in `pyoasis/pyoasis.pdf`. The pyoasis interface ultimately call the Fortran version. This is done by wrapping the Fortran in ISO-C bindings (see `lib/cbindings/fortran_isoc`), then wrapping the Fortran ISO-C bindings in C (see `lib/cbindings/c`), then wrapping the C bindings in python (see `pyoasis/src`). This method provides both the C and python bindings. Examples on how to use pyOASIS are provided in `pyoasis/examples`. To run all tests including python, C, and Fortran examples, use `make test`. The python wrapper functions are briefly described next.

- Creating a component using MPI

  In pyOASIS, components are instances of the `Component` class. To initialise a component, its name has to be supplied. It is also possible to provide an optional `coupling_flag` argument which defaults to "True", which means the component is coupled through OASIS3_MCT.

  OASIS3_MCT couples models which communicate using MPI. If the global communicator at the start of the run is different from the default `MPI_COMM_WORLD` communicator, the global communicator has to be passed to the `Component` class through the third optional argument. By default, the `Component` class will set up MPI internally and provides methods to get access to information such as rank and number of processes in the local communicator gathering only the component processes.

```
import pyoasis
[...]
comm = my_global_comm
component_name = "component"
coupling_flag = True
comp = pyoasis.Component(component_name, coupling_flag, comm)
print("Hello world from process " + str(comp.localcomm.rank)
         + " of " + str(comp.localcomm.size))
```

  To create a coupling communicator for a subset of processes, one can use the method `create_couplcomm`, with a flag being `True` for all these processes; see `pyoasis/examples/4-orange/python` for a practical example.

  If such a communicator already exists in the code, it should simply be provided to OASIS3_MCT with the method `set_couplcomm`; as in `pyoasis/examples/6-apple_and_orange/python`.

  To set up an MPI intra communicator or inter communicator between the local component and another component, one can use the methods `get_intracomm` or `get_intercomm`; as in `pyoasis/examples/3-box/python` .

  To set up an MPI intra-communicator among some of the coupled components, listed in the `comp_list` list, one can use the method `get_multi_intracomm`, as in `9-python_fortran_C-multi_intracomm`.

  Also, the current OASIS3-MCT internal debug level (`$NLOGPRT` value in the *namcouple*), can be retrieved as a property of a component, namely `debug_level`, and can be changed by directly modifying this property, as in `pyoasis/examples/7-multiple-puts/python`.

- Creating a partition

  The data can be partitioned in various ways. These correspond to the `SerialPartition`, `ApplePartition`, `BoxPartition`, `OrangePartition` and `PointsPartition` classes

which are inherited from the `Partition` abstract class. For details on the different ways to describe the partitions, see OASIS3_MCT User Guide, section 2.2.3 and examples `1_serial`, `2_apple`, `3_box`, `4_orange`, `5_points` in `pyoasis/examples`.

The simplest situation is the serial partitioning where all the data is held by a single process and only the number of points has to be specified (see example `1_serial`)

In the case of the Apple partitioning, each process contains a segment of a linear domain. To initialise such a partitioning, an offset has to be supplied for each rank as well as the number of data points that will be stored locally (see example `2_apple`).

When we use the Box partitioning, a two-dimensional domain is split into several rectangles. The global offset, local extents in the x and y directions and the global extent in the x direction have to be supplied to the constructor. The global offset is the index of the corner of the local rectangle (see example in `3_box`) .

The Orange partitioning consists of several segments of a linear domain (see an example with only one segment per process in `4_orange`.)

The last type of partitioning is Points, where we have to specify, in a list, the global indices of the points stored by the process (see example in `5_points`).

- Defining the coupling grids

  The grid data files, containing the definition of the grids onto which the coupling data is defined, can be created by the user before the run or can be written directly at run time by the components, either by one component process to write the whole grid or by each process holding a part of a grid. Details about the grid definition can be found in section 2.2.4 of OASIS3_MCT User Guide. A full example of writing a grid in sequential and parallel models can be found in `examples/10_grid` .

  To initialise a grid and write the grid longitudes and latitudes, one has to create an instance of the `Grid` class. Then to write the grid cell corner longitudes and latitudes, areas, mask, cell valid fraction, angle, the `set_corners`, `set_area`, `set_mask`, `set_frac`, and `set_angle` methods can be used respectively.

- Declaring the coupling data

  The coupling data is handled by the class `Var`. Its constructor requires its symbolic name, as it appears in the *namcouple* file, the partition and a flag indicating whether the data is incoming or outgoing. The latter is an enumerated type and can have the values `pyoasis.OasisParameters.OASIS_OUT` or `pyoasis.OasisParameters.OASIS_IN`.

  The property `is_active` can be tested to check if the variable is activated in the *namcouple* configuring file (see `example 3-box/python`).

  The coupling period(s) of the data, as defined in the *namcouple*, can be accessed with the property `cpl_freqs` and the number of coupling exchanges in which the data is involved by `len(cpl_freqs)` (see example `7-multiple-puts/python`).

  The property `put_inquire` of the variable tells what would happen to the corresponding data at that date below the corresponding send action. This maybe useful if, for example, the calculation of a coupling field is costly and if one wants to compute it only when it is really sent out. The different possible return codes are listed in section 2.2.9 of OASIS3_MCT User Guide.

- Ending the definition phase

  Then the definition of the component must be ended by calling the `enddef()` method. This must be done only once the partitioning and the variable data have been initalised.

- Sending and receiving data

  pyOASIS expects data to be provided as a `pyoasis.asarray` object:

```
field = pyoasis.asarray(range(n_points))
```

This is a numpy array but ordered in the Fortran way. In C, multidimensional arrays store data in row_major order where contiguous elements are accessed by incrementing the rightmost index while varying the other indices will correspond to increasing strides in memory as we use indices further towards the left. By default, numpy arrays use that ordering as well. Fortran, on the other hand, uses column_major order. In that case, contiguous elements are accessed by incrementing the leftmost index. `pyoasis.asarray` objects use the same ordering as Fortran. As a consequence, it is not necessary to transform data in order to use it in the OASIS3_MCT Fortran library.

The sending and receiving actions may be called by the component at each timestep. The date argument is automatically analysed and actions are actually performed only if date corresponds to a time for which it should be activated, given the period indicated by the user in the namcouple. See OASIS3_MCT User Guide section 2.2.7 for details.

The data is sent with the `put` function.

```
date = int(0)
variable.put(date, field)
```

Conversely, it is received with the `get` function, which fills the `pyoasis.asarray` object.

```
variable.get(date, field)
```

- Termination

  Finally, the coupling is terminated with the destruction of the component:

```
del comp
```

- Exceptions and aborting

  When an error occurs in OASIS3_MCT, the code coupler returns an error code and an `OasisException` is raised. In practice, OASIS3_MCT will internally handle the error, write an error message in its debug log files and to the screen, and abort before the exception is raised. It may also happen that the code aborts before the error message appears on the screen.

  When an error is caught by the pyOASIS wrapper, such as an incorrect parameter or a wrong argument type, a `PyOasisException` is raised.

  In the following example, where we attempt to initialise a component, a `PyOasisException` will be raised as the user supplies an empty name :

```
try:
      comp = pyoasis.Component("")
except (pyoasis.PyOasisException) as exception:
        pyoasis.pyoasis_abort(exception)
```

  The function `pyoasis.pyoasis_abort` takes an exception as argument. It stops the execution of all the processes after having displayed an error message and written information in the log files about the error and the context in which it took place.

  Another function is available, `pyoasis.oasis_abort`, for the cases where a voluntary abort is needed in the code where or not an exception has been raised. Its interface mimics the corresponding OASIS3_MCT functio `oasis_abort`.

2.4.1 Fortran python API correspondence

Figures 2.6 , 2.7 and 2.8 show the Fortran python API correspondence for different parts of the API. Different examples implementing the different parts of the API with the Fortran, C and python interfaces are also provided as practical illustrations in directory `pyoasis/examples` and are described in section 6.3.4.

Fortran	python	Notes
use mod_oasis	import pyoasis from pyoasis import OASIS	The second line is optional and gives access to the short constant syntax, e.g. `OASIS.OUT` instead of `pyoasis.OasisParameters.OASIS_OUT`
CALL oasis_init_comp (compid, comp_name, ierror)	comp = pyoasis.Component(name)	Standard form. Notice that *comp* is a user chosen variable name for an object of the Component class. In the following you can replace *comp* by the name of the object of your choice
CALL oasis_init_comp (compid, comp_name, ierror, coupled)	comp = pyoasis.Component(name, coupled=[True/False])	Optional argument for actual coupling, default to True
CALL oasis_init_comp (compid, comp_name, ierror, commworld=mycomm)	comp = pyoasis.Component(name, communicator=mycomm)	Optional argument for global communicator, default to None, i.e. communicator provided by OASIS. The two optional arguments can be combined
CALL oasis_get_localcomm (local_comm, ierror)	comp.localcomm	Local comm accessed as a property of the comp object Notice that upon creation the comp.couplcomm is a duplicate of comp.localcomm
CALL oasis_create_couplcomm(icpl, local_comm, coupl_comm, kinfo)	comp.create_couplcomm(icpl)	Process not involved in the coupling set icpl to `MPI.UNDEFINED` It returns an optional error code, but it sets the comp.couplcomm property. Cf example 4
CALL oasis_set_couplcomm(coupl_comm, kinfo)	comp.set_couplcomm(couplcomm)	Sets the comp.couplcomm property. Cf example 6
CALL oasis def partition (il_part_id, [0,ig_paral(2:)], ierror, ig_size, name)	part = pyoasis.SerialPartition(n_points[, global_size, name])	ig_paral(1) = 0 Optional arguments global_size and name. Cf. sect 2.2.3 of the OASIS3-MCT user guide. Notice that *part* is a user chosen variable name for an object of the Partition superclass.
CALL oasis def partition (il_part_id, [1,ig_paral(2:)], ierror, ig_size, name)	part = pyoasis.ApplePartition(offset, size[, global_size, name])	ig_paral(1) = 1 Optional arguments global_size and name
CALL oasis def partition (il_part_id, [2,ig_paral(2:)], ierror, ig_size, name)	part = pyoasis.BoxPartition(global_offset, local_extent_x, local_extent_y, global_extent_x[, global_size, name])	ig_paral(1) = 2 Optional arguments global_size and name
CALL oasis def partition (il_part_id, [3,ig_paral(2:)], ierror, ig_size, name)	part = pyoasis.OrangePartition(offsets, extents[, global_size, name])	ig_paral(1) = 3 Optional arguments global_size and name
CALL oasis def partition (il_part_id, [4,ig_paral(2:)], ierror, ig_size, name)	part = pyoasis.PointsPartition(global_indices[, global_size, name])	ig_paral(1) = 4 Optional arguments global_size and name

Figure 2.6: Fortran python AP correspondence for the initialisation, communication and partition definition.

Fortran	python	Notes
CALL oasis_start_grids_writing (flag)		Implicitly in the pyoasis.Grid class constructor for the first instantiated grid
CALL oasis_write_grid (cgrid, nx_global, ny_global, lon, lat, il_part_id)	grid = pyoasis.Grid(cgrid, nx_global, ny_global, lon, lat[, partition])	Notice that *grid* is a user chosen variable name for each object of the Grid class lon and lat are entered as 2D arrays - for unstructured grids the second dimension has to be 1 - of the same shape (nx_loc, ny_loc) where nx_loc, ny_loc are the number of points in each dimension on the calling process. The optional partition argument is a member of the pyoasis.Partition class.
CALL oasis_write_corner (cgrid, nx_global, ny_global, nc, clon, clat, il_part_id))	grid.set_corners(clo, cla)	clo, cla are 3d arrays: the last dimension is the max. number of corners per cell, the two others are nx_loc, ny_loc.
CALL oasis_write_mask (cgrid, nx_global, ny_global, mask, il_part_id, companion)	grid.set_mask(mask[, companion])	mask is a 2d integer array of size nx_loc, ny_loc. Oasis convention: 1 for masked points, 0 elsewhere. The optional argument companion is the name of a companion grid.
CALL oasis_write_frac (cgrid, nx_global, ny_global, frac, il_part_id, companion)	grid.set_frac(frac[, companion])	frac is a 2d array of size nx_loc, ny_loc. The optional argument companion is the name of a companion grid.
CALL oasis_write_area (cgrid, nx_global, ny_global, area, il_part_id)	grid.set_area(area)	area is a 2d array of size nx_loc, ny_loc.
	grid.write()	Terminates the declarations for the grid *grid*
CALL oasis_terminate_grids_writing ()		Implicitly activated by the grid.write() of the last grid instance to call it
CALL oasis_def_var (var_id, name, il_part_id, [1,1], kinout, var_type, ierror)	var = pyoasis.Var(name, partition, inout)	partition is a pyoasis.Partition object as in previous section. inout is an enumerate parameter, either in the long form `pyoasis.OasisParameters.OASIS_OUT/OASIS_IN` or in the short form (cf Module) `OASIS.OUT/OASIS.IN` Notice that the var_type is left to the dtype of the exchanged field. Once again *var* is a user chosen variable name for an object of the Var class.
CALL oasis_def_var (var_id, name, il_part_id, [1,n], kinout, var_type, ierror)	var = pyoasis.Var(name, partition, inout[, bundle_size=n])	The size of the bundle is set via the optional argument bundle_size
CALL oasis_enddef (ierror)	comp.enddef()	Notice that it is a method of the comp object, comp being the user chosen variable name of the component

Figure 2.7: Fortran python AP correspondence for the grid definition, variable declaration and end of definition phase.

2.5 Additional notes on coupling functionality

2.5.1 A brief overview of MCT

As described elsewhere, OASIS3-MCT leverages the MCT 2.11 coupling infrastructure developed at Argonne National Laboratory. That infrastructure is designed to couple fields on static grids between model components. The fields can be decomposed using MPI across mutiple processes, and the decomposition and number of proceses involved can be arbitrary in each component. MCT supports both communication of data between unique MPI non-overlapping communicators and within a single MPI communicator, although these two operations are functionally independent within MCT.

MCT also provides the ability to map (i.e. interpolate or regrid) data between grids as long as the interpolation is linear and can be computed by a linear sparse matrix multiply. MCT does not compute interpolation weights, but it has interfaces that allow those weights to be passed into MCT. Within MCT, mapping and communication are also treated as independent features.

OASIS3-MCT supports both mapping and communication of data through MCT. As a results, mapping and coupling are implemented, in many ways, as separate steps in the underlying implementation. A coupling field that also requires mapping will carry out the mapping on either the source or destination component on the associated processes, while coupling between components will be done either before or after mapping. In other words, a coupling field can be interpolated to the destination grid on the source processes then communicated to the destination component OR a coupling field can be communicated to the destination component on the source grid then mapped to the destination grid on the destination

Fortran	python	Notes
field=pyoasis.asarray(field[, dtype])	Make sure that field is a numpy asfortran array of the required dtype (optional argument defaulting to `numpy.float64`) The call can be skipped if field is already so	
CALL oasis_put (var_id, date, fld1, info, fld2, fld3, fld4, fld5, write_restart)	var.put(time, field[, write_restart])	var is a previously defined object of the Var class field has been checked by pyoasis.asarray Bundle are supported (bundle index is the last index) Bicubic and conservative 2nd extra fields not supported. write_restart boolean optional argument for triggering a NetCDF dump of the sent field. Returned value is to be compared against the Oasis Parameters enumeration (in long or short syntax)
CALL oasis_get (var_id, date, fld, info)	var.get(time, field)	field has to be preallocated and checked by pyoasis.asarray Returned value is to be compared against the Oasis Parameters enumeration (in long or short syntax)
CALL oasis_terminate (ierror)	del comp / nothing	oasis terminates at the comp object destruction, either explicitly by del comp or automatically at garbage collection on exit
CALL oasis get_debug(debug_value)	comp.debug_level pyoasis.get_debug()	Coded as a property of a Component object or as a standalone function
CALL oasis set_debug(debug_value)	comp.debug_level = debug_value pyoasis.set_debug(debug_value)	Coded as a setter for a Component object property or as a standalone function
CALL oasis put_inquire(var_id, date, kinfo)	var.put_inquire(time)	var is a Var object. Returned value is to be compared against the Oasis Parameters enumeration (in long or short syntax)
CALL oasis_get_ncpl(var_id, ncpl, kinfo) CALL oasis_get_freqs(var_id, mop, ncpl, cpl_freqs, kinfo)	var.cpl_freqs	Coded as a property of a Var object. Returns an iterable list
CALL oasis_get_intercomm(new_comm, cdnam, kinfo)	comp.get_intercomm(comp_name)	Coded as a method of a Component object.
CALL oasis_get_intracomm(new_comm, cdnam, kinfo)	comp.get_intracomm(comp_name)	Coded as a method of a Component object.
CALL oasis_get_multi_intracomm(new_comm, cdnam, root_ranks, kinfo)	comp.get_multi_intracomm(complist)	Coded as a method of a Component object. Returns a pair of values: the new comm and a dictionary associating root ranks (values) to component names (keys)
CALL oasis_abort (compid, routine_name, abort_message, file, line, rcode)	raise PyOasisException(text) raise OasisException(text, error)	Preferred use of exceptions over an abort function

Figure 2.8: Fortran python AP correspondence for the coupling field exchanges, termination and auxialiary routines.

component. At the present time, it is not possible to map the field as part of the communication rearrangement, although in theory, that capability should be possible to implement in the future, and OASIS3-MCT developers are considering it. The separation of mapping and communication is handled by the OASIS layer. Users only need to be aware of a few options that can be set to fine tune the performance of these operations.

OASIS3-MCT also imposes a few other constraints on the usage. MCT does NOT support haloed communication. There must be a 1-to-1 relationship between grid point values on the source grid and on the destination grid. A user cannot send a single grid point value to multiple destination gridcells or processes via OASIS3-MCT. The partitions defined by the user that define the field decomposition cannot reference the same global gridcell more than once.

OASIS3-MCT does not support dynamically varying grids nor dynamically varying decompositions at the present time. MCT is NOT currently setup to support those features. In the future, it is possible that some dynamic grid capabilities might be supported through OASIS3-MCT but requirements and implementation are still being assessed.

Since the OASIS3-MCT_4.0 release, OASIS3-MCT includes a mixed MPI+OpenMP parallel version of the SCRIP library for the calculation of the remapping weights (see section 4.3). But besides this aspect,

MCT itself, and therefore the communication layer in OASIS, has only minimal OpenMP support at the present time. Users are discouraged from calling OASIS3-MCT `oasis_put` and `oasis_get` from a threaded region. And it is not possible to define a partition (decomposition) across multiple active threads. Most of the work that OASIS3-MCT does related to decomposition rearrangement for communication or mapping per se is MPI-based. There are some floating point operations in the sparse matrix multiple as well as in some of the OASIS3-MCT transforms; but generally, these are highly parallel and would often not benefit from use of OpenMP. OASIS3-MCT is designed primarily to support large memory parallel applications that require efficient and scalable coupling and mapping capabilities.

MCT only supports coupling of integer or floating point data. Fields based on logicals and character strings cannot be coupled.

2.5.2 Coupling scalar values

OASIS3-MCT does not have a distinct feature to support scalar coupling. By scalar coupling, we mean scalar variables such as date and time, logical flags, integer or real parameters, or other scalar data that might be defined identically across all MPI tasks in a component or even just on the root task or a subset of tasks. It is often desirable to communicate scalar data between components to coordinate scientific or technical features. There isn't a feature that supports this type of coupling specifically, but it is possible to do so using available interfaces. The most robust implementation is probably to setup root to root coupling of scalars between components. To do so :

- Determine the number and type of scalars to couple
- Allocate an array in each component of that size
- Initialize a partition using the POINTS approach (see section 2.2.3) with the size of the array assigned to the root process in each component and no portion of the partition allocated to other processes.
- Define coupling field names for use in both the model components and the namcouple file and declare variables as usual. Create a set of namcouple inputs. You will not need any mapping.
- Use the `oasis_get` and `oasis_put` interfaces to communicate data between components.
- If required, broadcast the scalars after being received from the root process to the other processes, outside the OASIS3-MCT layer.

This can be extended as needed to send or receive from non root processes. But you cannot send the scalar data to multiple processes as this violates the halo restriction in MCT. Users need to be aware which processes contain valid scalar data both on the source and destination side and to manage data synchronization between processes within a component outside the OASIS3-MCT layer.

2.5.3 The lag concept

Using the OASIS3-MCT coupling library, the user has the flexibility to reproduce different coupling algorithms defining `LAG` values for the different coupling fields . In the components, the sending and receiving routines, respectively `oasis_put` and `oasis_get`, can be called at each component timestep, with the appropriate `date` argument giving the actual time (at the beginning of the timestep), expressed in number of seconds since the start of the run, or in any other time units as long as the same are used in all components and in the *namcouple* (see section 2.2.7). This `date` argument is automatically analysed by the coupling library and depending on the coupling period and the lag chosen by the user for the coupling field in the *namcouple* (`LAG`), different coupling algorithms can be reproduced without modifying the component codes themselves.

The lag (`LAG`) must be expressed in the time unit used (that must be the same in the components and in the *namcouple*, see section 2.2.7) and can be positive or negative but should never be larger (in absolute magnitude) than the coupling period of any field due to problems with restartability and dead-locking. When a component calls a `oasis_put`, the value of the lag is automatically added to the value of the

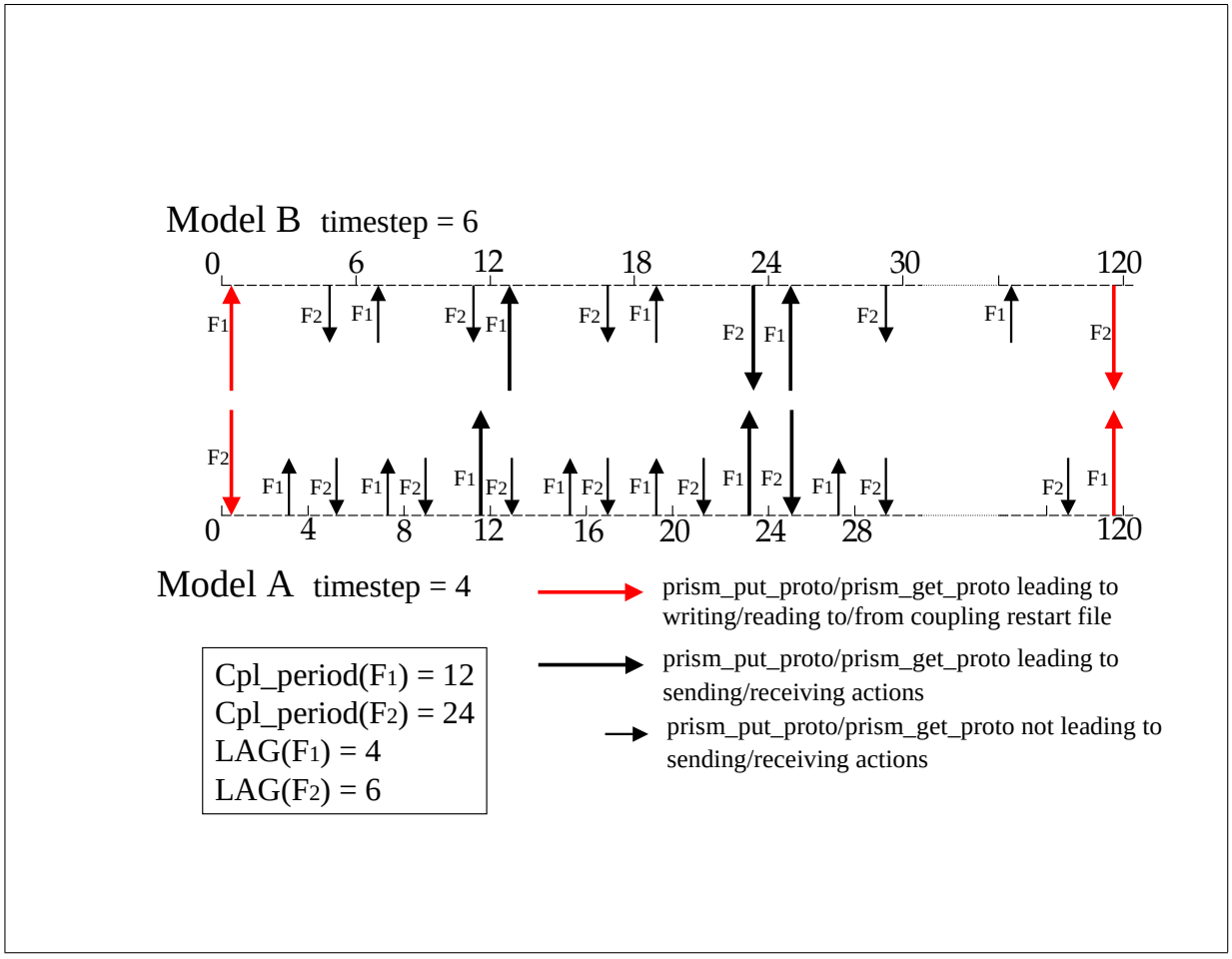

Figure 2.9: LAG concept first example

`date` argument and the "put" is actually performed when the sum `date+lag` is a coupling time; in the target component, this "put" will match a `oasis_get` for which the `date` argument is the same coupling time. So the lag only shifts the time at which the data is sent but not the time at which the data is received.

When the lag is positive, a restart file must be available to initiate the coupling. For a field with positive lag, the source component automatically reads the field in the restart file during the coupling initialization phase (below the `oasis_enddef`) and send the data to match the `oasis_get` performed at time=0 in the target component. The final coupling data on the source side will then be automatically written to the restart file for use in the next run[5].

On the 4 figures in this section, short black arrows correspond to `oasis_put` or `oasis_get` called in the component that do not lead to any "put" or receiving action; long black arrows correspond to `oasis_put` or `oasis_get` called in the components that do actually lead to a "put" or "get" action; long red arrows correspond to `oasis_put` or `oasis_get` called in the component models that lead to a reading or writing of the coupling field from or to a coupling restart file.

1. LAG concept first example

   A first coupling algorithm, exploiting the LAG concept, is illustrated on figure 2.9.

   During a coupling timestep, model A receives F_2 and then sends F_1; its timestep length is 4. During a coupling timestep, model B receives F_1 and then sends F_2; its timestep length is 6. F_1 and F_2 coupling periods are respectively 12 and 24. If F_1/F_2 "put" action by model A/B was used at a coupling timestep to match the model B/A "get" action, a deadlock would occur as both models

[5] When there is a lag, the first and last instance of the source field in the debug netCDF file (EXPOUT fields, see section 3.3) always correspond respectively to the field read from and written to the restart file.

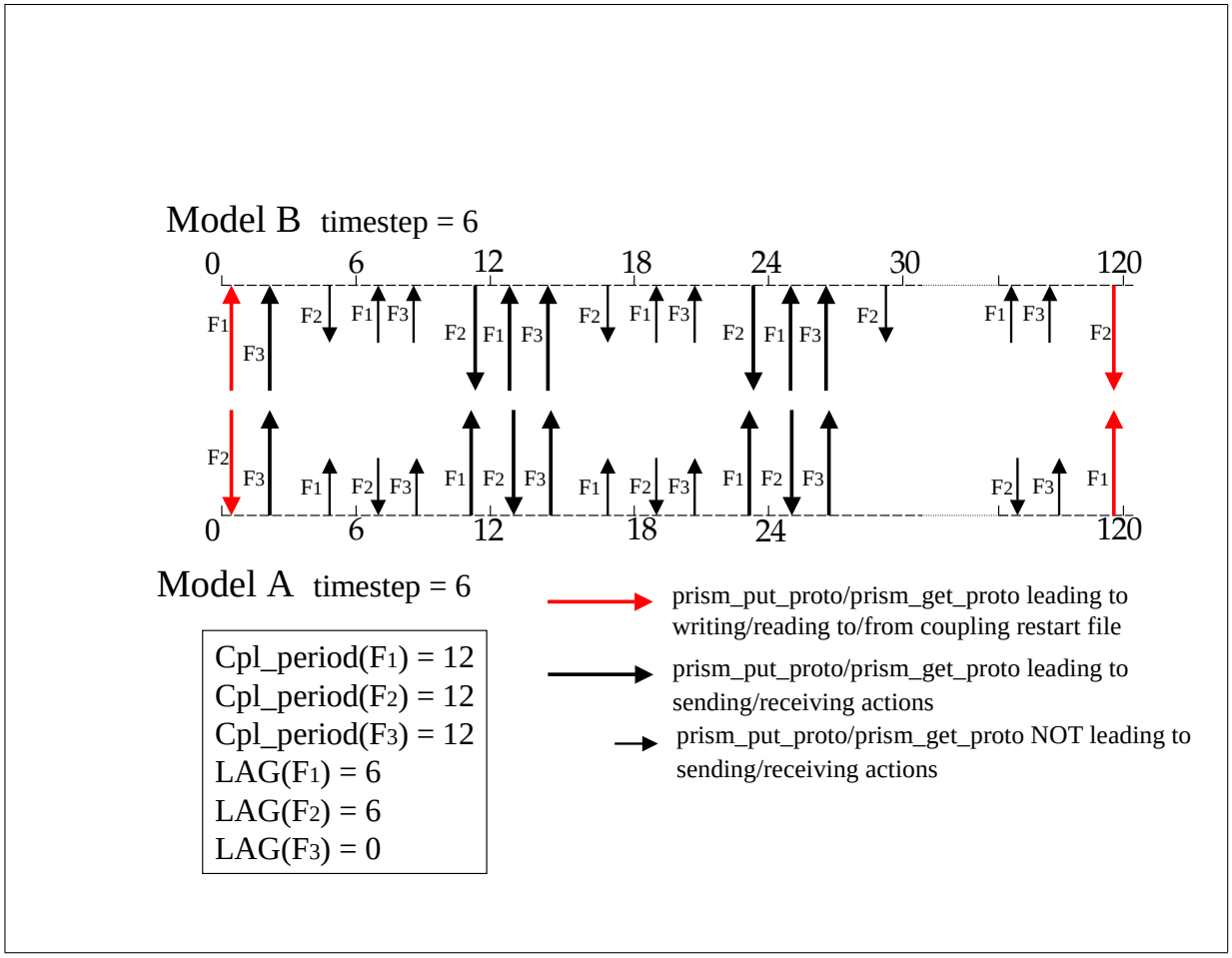

Figure 2.10: LAG concept second example

would be initially waiting on a "get" action. To prevent this, F_1 and F_2 produced at the timestep before have to be used to match respectively the model B and model A "get" actions.

This implies that a lag of respectively 4 and 6 seconds must be defined for F_1 and F_2. For F_1, the `oasis_put` performed at time 8 and 20 by model A will then lead to "put" actions (as 8 + 4 = 12 and 20 + 4 = 24 which are coupling periods) that match the "get" actions performed by `oasis_get` called by model B at times 12 and 24. For F_2, the `oasis_put` performed at time 18 by model B then leads to a "put" action (as 18 + 6 = 24 which is a coupling period) that matches the "get" action performed at time 24 by the `oasis_get` called by model A.

At the beginning of the run, as their LAG index is greater than 0, the first `oasis_get` of F_1 and F_2 will automatically be fulfilled with fields read from their respective coupling restart files. The user therefore has to create those coupling restart files before the first run in the experiment. At the end of the run, F_1 having a lag greater than 0, is automatically written to its coupling restart file below the last F_1 `oasis_put` as the `date+lag` equals the total run time. The analogue is true for F_2. These coupling restart fields will automatically be read in at the beginning of the next run below the respective `oasis_get`.

2. LAG concept second example

   A second coupling algorithm exploiting the LAG concept is illustrated on figure 2.10. During its timestep, model A receives F_2, sends F_3 and then F_1; its timestep length is 6. During its timestep, model B receives F_1, receives F_3 and then sends F_2; its timestep length is also 6. F_1, F_2 and F_3 coupling periods are both supposed to be equal to 12.

   For F_1 and F_2 the situation is similar to the first example. Without any lag specified and without any restart file, a deadlock would occur as both models would be waiting on a "get" action. To prevent

this, F_1 and F_2 produced at the timestep before have to be used to match the model A and model B "get" actions, which means that a lag of 6 must be defined for both F_1 and F_2. For both coupling fields, the `oasis_put` performed at times 6 and 18 by the source model then lead to "put" actions (as 6 + 6 = 12 and 18 + 6 = 24 which are coupling periods) that match the "get" action performed at time 12 and 24 by the `oasis_get` called by the target model.

For F_3, sent by model A and received by model B, no lag needs to be defined: the coupling field produced by model A at the coupling timestep can be "consumed" by model B without causing a deadlock situation.

As in the first example, the `oasis_get` performed at the beginning of the run for F_1 and F_2, will automatically receive data read from their coupling restart files, and the last `oasis_put` performed at the end of the run automatically write them to their coupling restart file. For F_3, no coupling restart file is needed.

We see here how the introduction of appropriate LAG indices results in receiving in the target component the coupling fields produced by the source component the time step before; this is, in some coupling configurations, essential to avoid deadlock situations.

2.5.4 The sequence concept

The order of coupling operations in the system is determined solely by the order of calls to send (`oasis_put` or "put") and receive (`oasis_get` or "get") data in the components in conjunction with the setting of the lag in the *namcouple*. Data that is received is always blocking while data that is sent is non-blocking with respect to the component making that call. It is possible to deadlock the system if the relative orders of puts and gets in different components are not compatible.

With OASIS3-MCT, the sequence (SEQ) index in the *namcouple* file now provides the coupling layer with an ability to detect a deadlock before it happens and exit. It does this by tracking the order of get and put calls in components compared to the SEQ specified in the *namcouple*. If there are any inconsistencies, the component will abort gracefully with a useable error message before the system deadlocks. If there are any coupling dependencies in the system, use of the SEQ index is recommended for diagnosis but has no impact on the ultimate solution and is NOT required.

In the following two examples, there are two models, each "put" a field to the other at every coupling period without any lags. In the first case, there is no dependency as each model first sends and then receives some data.

```
 model1         model2
 ------         ------
put(fld1)      put(fld2)
get(fld2)      get(fld1)
```

In this case, there is no sequencing dependency and the value of SEQ must be identical (or unset) in the *namcouple* description of the fld1 and fld2 coupling. If by mistake, SEQ is set to 1 for fld1 and 2 for fld2, then the coupled model will abort because at runtime, the coupler will detect in model 2 that fld2 was sent before fld1 was received which is out of sequence as defined by the SEQ settings.

In the next example, there is a dependency in the sequencing.

```
 model1         model2
 ------         ------
put(fld1)      get(fld1)
               fld2=g(fld1)
get(fld2)      put(fld2)
```

In model2, fld2 depends on fld1. If SEQ is not used and if, for example, model1 does not have the consistent ordering of the put and get shown above (required by model2), then the models would deadlock and hang. If this dependency is known, there is a benefit in setting SEQ=1 for fld1 and SEQ=2 for fld2;

at runtime, if the sequencing of model1 or model2 does not match the above diagram, then the coupling layer will detect it and will exit gracefully with an error message.

Again, the SEQ namecouple setting is only diagnostic and is not required.

Chapter 3

The configuration file *namcouple*

The OASIS3-MCT configuration file *namcouple* contains, below pre-defined keywords, all user-defined information necessary to configure a particular coupled run.

The *namcouple* is a text file with the following characteristics:

- the keywords used to separate the information can appear in any order;
- the number of blanks between two character strings is non-significant;
- all lines beginning with # are ignored and considered as comments;
- blank lines are supported, but only since OASIS3-MCT_4.0 version.

The first part of *namcouple* is devoted to configuration of general parameters such as the total run time or the desired debug level. The second part gathers specific information on each coupling (or I/O) field, e.g. their coupling period, the list of transformations or remapping to be performed by OASIS3-MCT and associated configuring lines (described in more details in chapter 4).

In OASIS3-MCT, several *namcouple* inputs have been deprecated but, for backwards compatibility, they are still allowed. These inputs will be noted in the following text using the notation "UNUSED" and not fully described. Information below these keywords is obsolete; they will not be read and will not be used.

In the next sections, a *namcouple* example is given and all configuring parameters are described. Additional lines containing different parameters for each transformation are described in section 4. A realistic *namcouple* can be found in `oasis3-mct/examples/tutorial/data_oasis3/` directory.

3.1 An example of a simple *namcouple*

The following simple *namcouple* configures a run into which e.g. an ocean, an atmosphere and an atmospheric chemistry components are coupled. The ocean running on grid `toce` provides only the SOSSTSST field to the atmosphere (grid `atmo`), which in return provides the field CONSFTOT to the ocean. One field COSENHFL is exchanged from the atmosphere to the atmospheric chemistry (also running on grid `atmo`), and one field SOALBEDO is read from a file by the ocean.

```
########## First section ##############################################
 $NFIELDS
    4
#
 $RUNTIME
    432000
#
 $NLOGPRT
   2     1    0
```

```
#
 $NCDFTYP
   cdf1
#
 $NUNITNO
   901     920
#
 $NMAPDEC
   decomp_wghtfile
#
 $NMATXRD
   ceg
#
 $NWGTOPT
   ignore_bad_index
#
 $SEQMODE
 $CHANNEL
 $JOBNAME
 $NBMODEL
 $INIDATE
 $MODINFO
 $CALTYPE
#
########## Second section ##############################################
#
 $STRINGS
#
# Field 1
 SOSSTSST SISUTESU 1 86400  5  sstoc.nc  EXPOUT
 182  149  128  64  toce  atmo   LAG=+14400  SEQ=+1
 P 2 P 0
 LOCTRANS CHECKIN MAPPING  BLASNEW CHECKOUT
#
  AVERAGE
  INT=1
  map_toce_atmo_120315.nc src opt
  1.0  1
  CONSTANT     273.15
  INT=1
#
# Field 2
 CONSFTOT SOHEFLDO 6 86400  4   flxat.nc  EXPORTED
 atmo   toce  LAG=+14400  SEQ=+2
 P 0 P 2
 LOCTRANS  CHECKIN  SCRIPR CHECKOUT
#
  ACCUMUL
  INT=1
  BILINEAR LR SCALAR LATLON 1
  INT=1
```

```
#
# Field 3
 CONSFTOT  CONSFTOT 1  86400   1  flda3.nc  OUTPUT
 128  64  128  64 atmo   atmo
 LOCTRANS
 AVERAGE
#
# Field 4
 SOALBEDO SOALBEDO  17  86400  0  SOALBEDO.nc  INPUT
```

3.2 First section of *namcouple* file

The first section of *namcouple* uses some predefined keywords prefixed by the $ sign to locate the related information. The $ sign must be the first non-blank character on the line but can be in any column. 8 keywords are used by OASIS3-MCT and 5 of these are optional :

- `$NFIELDS`: On the line below this keyword, put a number equal (or greater) to the total number of field entries in the second part of the *namcouple*. If more than one field are described on the same line, this counts as only one entry.
- `$RUNTIME`: On the line below this keyword, put the total simulated time of the run, expressed in seconds (or any other time units as long as the same are used in all components and in the *namcouple*, see 2.2.7). Note that by convention the first coupling of a run occurs at the beginning of the run and all other couplings occur at a time strictly smaller than `$RUNTIME`.
- `$NLOGPRT`: The first, second and third numbers on the line below this keyword refer to (i) the debug verbosity, (ii) internal timing statistics and (iii) component load balancing analysis. The information is written by OASIS3-MCT for each component and (optionnally) for each process.

  The first number (that can be modified at runtime with the `oasis_set_debug` routine, see section 2.2.9) may be:
  - 0 : production mode. One file debug.root.xx is open by the master process of each component and one file debug_notroot.xx is open for all the other processes of each component to write only error information.
  - 1 : one file debug.root.xx is open by the master process of each component to write information equivalent to level 10 (see below) and also to write memory usage information; one file debug_notroot.xx is open for all the other processes of each component to write error information.
  - 2 : one file debug.yy.xxxxxx is open by each process of each component (with "yy" being the component number and "xxxxxx" the process number) to write normal production diagnostics and memory usage information
  - 5 : as for 2 with in addition some initial debug info
  - 10: as for 5 with in addition the routine calling tree
  - 12: as for 10 with in addition some routine calling notes
  - 15: as for 12 with even more debug diagnostics
  - 20: as for 15 with in addition some extra runtime analysis
  - 30: full debug information

  The second number defines how time statistics are written out to file *comp_name*.timers_xxxx (with *comp_name* being the component name, see section 2.2.2); it can be:
  - 0 : nothing is calculated or written.

  - 1 : some time statistics are calculated and written in a single file by the processor 0 as well as the min and the max times over all the processors.
  - 2 : some time statistics are calculated and each processor writes its own file ; processor 0 also writes the min and the max times over all the processors in its file.
  - 3 : some time statistics are calculated and each processor writes its own file ; processor 0 also writes in its file the min and the max times over all processors and also writes in its file all the results for each processor.

  For more information on the time statistics written out, see section 6.4.2.

  The third number (new in OASIS3-MCT_5.0) can be set to 1 to activate a load balancing diagnostic. An efficient use of the allocated computing resources in a coupled system requires the harmonisation of the components speed. This operation, called load balancing, is often neglected, either because of the apparent resource abundance and practical difficulties. To facilitates this work, OASIS3-MCT can output the full timeline of all coupling related events, for any of the allocated resources. This timeline is saved in one netCDF file per coupled component (`timeline_XXX_component.nc`). It provides the comprehensive sequence of any operations related to the coupling (field exchange through MPI, field output on disk, field interpolation and mapping, field reading on disk, restart writing, initialisation and termination phase of the OASIS3-MCT setup) so that any simulation slow down in link with the use of the OASIS library can be identified. The analysis of the coupling field exchanges, amongst all the coupling events, allows not only to identify the resources waste of components which are recurrently waiting for their coupling fields but it also reveals other bottlenecks such as disk access, OS interruptions or model internal load imbalance. The full picture of these events makes possible an optimum load balancing, even for the most complex configurations. For a detailed information on load balancing analysis and timeline visualisation see respectively [Maisonnave et al 2020] and in [Piacentini and Maisonnave 2020]. In addition to the timeline, computing information (time to solution, speed, cost) and a synthesis of the time spent on MPI routines for each coupled component can also help, in the simpler cases, to allocate resources in a balanced way (see file `load_balancing_info.txt`).

- `$NCDFTYP`: Optional (new in OASIS3-MCT_5.0); on the line below this keyword is a character string that indicates the NetCDF file type for all (i.e. mapping, restart, output) new NetCDF files generated by OASIS3. The options are `cdf1`, `cdf2`, and `cdf5`. The mode `cdf1` is also known as classic mode, `cdf2` as large file format or 64bit_offset and supports larger files, `cdf5` as 64bit_data and supports both larger files and larger variables. More information about these file types can be found in NetCDF documentation. Because `cdf5` may not be generally available in all NetCDF installations, use of this option requires that the C preprocessor directive `CDF_64BIT_DATA` be used when compiling OASIS3-MCT. If that preprocessor is not used, `cdf5` is not a valid option. The file format for any NetCDF file can be diagnosed by using "ncdump -k filename". $NCDFTYP only affects new files created by OASIS3-MCT. NetCDF will read and/or overwrite existing files of any NetCDF file type, and the file type will remain unchanged in that case.
- `$NUNITNO`: Optional (new in OASIS3-MCT_4.0); on the line below this keyword are two integers that indicate the minimum and maximum unit numbers to be used for input and output files in the coupling layer. The user should choose values that will NOT conflict or overlap with unit numbers in use in any of the component models. The defaults are 1024 for the minimum and 9999 for the maximum unit number if not explicitly set by the user.
- `$NMAPDEC`: Optional (new in OASIS3-MCT_4.0); on the line below this keyword is a character string that indicates the mapping decomposition value to be used during local mapping. The options are `decomp_1d` and `decomp_wghtfile`. Option `decomp_1d` decomposes the grid in a simple one dimensional way while `decomp_wghtfile` decomposes the grid using the information in the remapping weight file to reduced mapping communication. Option `decomp_wghtfile` will take some extra time in initialization but it should result in faster mapping. The default is `decomp_1d` but it is recommended to test `decomp_wghtfile` to see if that option improves performance.

More details can be found in [Craig et al 2018] and in [Valcke et al 2018].

- `$NMATXRD`: Optional (new in OASIS3-MCT_4.0); on the line below this keyword is a character string that indicates the method used to read remapping weights. There are two options, `orig` and `ceg`. In both, the weights are read in chunks by the root process. In the `orig` option, the weights are then broadcasted to all processes and each process then saves the weights needed in order to be consistent with the mapping decomposition. In the `ceg` option, the root process reads the weights and then decides which process each weight should be assigned to. A series of exchanges are then done and just the weights needed on each process are sent. The `orig` method sends much more data but is more parallel. The `ceg` method does most of the work on the root process but less data is communicated. The default option is `ceg`. More details can be found in [Craig et al 2018].
- `$NWGTOPT` : Optional (new in OASIS3-MCT_4.0); on the line below this keyword is a character string that indicates how to handle bad remapping weights. There are four options `abort_on_bad_index`, `ignore_bad_index`, `ignore_bad_index_silently`, and `use_bad_index`. Bad weights are defined as weights in the mapping file for which either the source or destination index are out of bounds relative to the number of grid cells in the grid; in that case, the weight is referencing a gridcell that does not physically exist. Note that an index equal to zero will not be considered as a bad index if the associated weight is also zero. There are other situations where the value of the actual mapping weight is scientifically incorrect, but this is not easy to detect and is not dealt with in OASIS3-MCT.
  - `abort_on_bad_index` will write error messages to the log files and abort if a bad weight index is detected. This is the default option.
  - `ignore_bad_index` will write an error message and then remove bad weights internally before continuing.
  - `ignore_bad_index_silently` will remove bad weights and continue without writing an error message.
  - `use_bad_index` will attempt to keep bad weights in the interpolation computation, but this can result in memory corruption, silent dropping of weights, and incorrect results ; this is not recommended.

  Note that the ability to check mapping files at runtime in OASIS3-MCT is limited. It is always recommended that mapping files be analyzed offline before long production runs are carried out. Checks can be done to make sure the source and destination indices are valid, that weights values are reasonable (for instance, between 0 and 1, although this will depend on the mapping method), and that the sum of weights on the destination cells are reasonable (for instance, 1, in many cases). In addition, offline tests can be run with analytical functions to verify conservation, gradient preserving features and other characteristics associated with the particular mapping approach.
- `$NNOREST`: Optional (new in OASIS3-MCT_4.0); on the line below this keyword is a character string that can override the requirement that restart files must exist if they are needed. If the character string value starts with T, t, .T, or .t (as in true), then OASIS3-MCT will initialise with zero any variable that normally requires a restart (for instance, variables with LAG > 0) if the restart file does not exist. By default, missing restart files will cause the model to abort. It is strongly recommended that this keyword NOT be used in production runs. It exists to provide a quick shortcut for running technical tests. Note that if `$NNOREST` is true but the restart file nonetheless exists, it will be used.
- `Keywords $SEQMODE, $CHANNEL, $JOBNAME, $NBMODEL, $INIDATE, $MODINFO, $CALTYPE` are not used anymore.

3.3 Second section of *namcouple* file

The second part of the *namcouple*, starting after the keyword `$STRINGS`, contains coupling information for each coupling (or I/O) field. Its format depends on the field status given by the last entry on the field

first line (EXPORTED, IGNOUT or INPUT in the example above). The field may be (status AUXILARY is now UNUSED) :

- EXPORTED: exchanged between components and transformed by OASIS3-MCT
- EXPOUT: exchanged, transformed and also written to two debug NetCDF files, one before the sending action in the source component below the oasis_put call (after local transformations LOCTRANS and BLASOLD if present), and one after the receiving action in the target component below the oasis_get call (after all transformations). EXPOUT should be used only when debugging the coupled model. The name of the debug NetCDF file (one per field) is automatically defined based on the field and component names.
- IGNORED: with OASIS3-MCT, this setting is equivalent to and converted to EXPORTED
- IGNOUT: with OASIS3-MCT, this setting is equivalent to and converted to EXPOUT
- INPUT: read in from the input file by the target component below the oasis_get call at appropriate times corresponding to the input period indicated by the user in the *namcouple*. See section 5.3 for the format of the input file.
- OUTPUT: written out to an output debug NetCDF file by the source component below the oasis_put call, after local transformations LOCTRANS and BLASOLD, at appropriate times corresponding to the output period indicated by the user in the *namcouple*.

3.3.1 Second section of *namcouple* for EXPORTED and EXPOUT fields

The first 3 lines for fields with status EXPORTED and EXPOUT are as follows:

```
SOSSTSST SISUTESU 1 86400  5  sstoc.nc  EXPOUT
182  149    128  64  toce  atmo   LAG=+14400 SEQ=+1
P 2 P 0
```

where the different entries are:

- Field first line:
  - SOSSTSST : symbolic name for the field in the source component (80 characters maximum). It has to match the argument name of the corresponding field declaration in the source component; see oasis_def_var in section 2.2.5
  - SISUTESU : symbolic name for the field in the target component (80 characters maximum). It has to match the argument name of the corresponding field declaration in the target component; see oasis_def_var in section 2.2.5
  - 1 : UNUSED but still required for parsing
  - 86400 : coupling and/or I/O period for the field, in seconds
  - 5 : number of transformations to be performed by OASIS3 on this field
  - sstoc.nc : name of the coupling restart file for the field (32 characters maximum); mandatory even if no coupling restart file is effectively used (for more detail, see section 5.2)
  - EXPOUT : field status
- Field second line:
  - 182 : number of points for the source grid first dimension (optional)
  - 149 : number of points for the source grid second dimension (optional)[1]
  - 128 : number of points for the target grid first dimension (optional)
  - 64 : number of points for the target grid second dimension (optional)[1]

    These source and target grid dimensions are optional but note that in order to have 2D fields written as 2D arrays in the debug files, these dimensions must be provided in the *namcouple*; otherwise, the fields will be written out as 1D arrays.

[1] For 1D field, put 1 as the second dimension

  - toce : prefix of the source grid name in grid data files (see section 5.1) (80 characters maximum)
  - atmo : prefix of the target grid name in grid data files (80 characters maximum)
  - `LAG=+14400`: optional lag index for the field (see section 2.5.3)
  - `SEQ=+1`: optional sequence index for the field (see section 2.5.4)
- Field third line
  - P : source grid first dimension characteristic ('P': periodical; 'R': regional).
  - 2 : source grid first dimension number of overlapping grid points.
  - P : target grid first dimension characteristic ('P': periodical; 'R': regional).
  - 0 : target grid first dimension number of overlapping grid points.
- The fourth line gives the list of transformations to be performed for this field. In addition, there is one or more configuring lines describing some parameters for each transformation. These additional lines are described in more details in the chapter 4.

Support to couple multiple fields via a single communication

With OASIS3-MCT, it is possible to couple mutiple fields via a single communication. To activate this option, the user must list the related fields on a single entry line (with a maximum of 5000 characters on one line) through a colon delimited list in the *namcouple*, for example:

```
ATMTAUX:ATMTAUY:ATMHFLUX TAUX:TAUY:HEATFLUX 1 3600 3 rstrt.nc EXPORTED
```

All fields will then use the same *namcouple* settings (source and target grids, transformations, etc.) for that entry. In the component model codes, these fields are still apparently sent or received one at a time through individual `oasis_put` and `oasis_get`. Inside OASIS3-MCT, the fields are stored and a single mapping and send or receive instruction is executed for all fields. This is useful in cases where multiple fields have the same coupling transformations and to reduce communication costs by aggregating multiple fields into a single communication.

This option does not put any constraint on the order of the related `oasis_put` and `oasis_get` in the codes.

As they appear in one single entry line, these fields must share the same coupling restart file but this restart file may contain other fields.

3.3.2 Second section of *namcouple* for `OUTPUT` fields

The first 3 lines for fields with status `OUTPUT` are as follows:

```
CONSFTOT  CONSFTOT 1  86400   1  flda3.nc  OUTPUT
128  64  128  64 atmo   atmo
LOCTRANS
```

where the different entries are:

- Field first line:
  - the source symbolic name must be repeated twice on the field first line
  - 1 : UNUSED but still required for parsing
  - 86400 : output period for the field, in seconds
  - 1 : number of transformations to be performed by OASIS3-MCT on this field
  - flda3.nc : name of the coupling restart file for the field (32 characters maximum); needed only if a `LOCTRANS` transformation is present
  - `OUTPUT` : field status
- Field second line:
  - 128 : number of points for the source grid first dimension (optional)

  - 64 : number of points for the source grid second dimension (optional)[2]
  - 128 : number of points for the target grid first dimension (optional)
  - 64 : number of points for the target grid second dimension (optional)[1]
  - atmo : prefix of the source grid name in grid data files repeated twice (80 characters maximum)
- The third line is `LOCTRANS` if this transformation is chosen for the field. Note that `LOCTRANS` is the only transformation supported for `OUTPUT` fields.

3.3.3 Second section of *namcouple* for `INPUT` fields

The first and only line for fields with status `INPUT` is:

```
SOALBEDO SOALBEDO  1  86400  0  SOALBEDO.nc  INPUT
```

where the different entries are:

- `SOALBEDO`: symbolic name for the field in the target component (80 characters maximum, repeated twice)
- 1: UNUSED but still required for parsing (as for EXPORTED fields above)
- 86400: input period in seconds
- 0: number of transformations (always 0 for `INPUT` fields)
- `SOALBEDO.nc`: the input file name (32 characters maximum) (for more detail on its format, see section 5.3)
- `INPUT`: field status.

[2]For 1D field, put `1` as the second dimension

Chapter 4

Transformations and interpolations

Different transformations and 2D interpolations are available in OASIS3-MCT to adapt the coupling fields from a source model grid to a target model grid. In the following paragraphs, a description of each transformation with its corresponding configuration lines that the user has to write in the *namcouple* file is given. Features that are now deprecated (non functional) compared to prior versions will be noted with the string UNUSED and not described.

4.1 Time transformations

- **LOCTRANS**:

  LOCTRANS requires one configuring line on which a time transformation, automatically performed below the call to oasis_put, should be indicated:

  ```
  # LOCTRANS operation
    $TRANSFORM
  ```

  where $TRANSFORM can be

  - INSTANT: no time transformation, the instantaneous field is transferred;
  - ACCUMUL: the field accumulated over the previous coupling period is exchanged (the accumulation is simply done over the arrays field_array provided as third argument to the oasis_put calls without any specific weighting);
  - AVERAGE: the field averaged over the previous coupling period is transferred (the average is simply done over the arrays field_array provided as third argument to the oasis_put calls without any specific weighting);
  - T_MIN: the minimum value of the field for each source grid point over the previous coupling period is transferred;
  - T_MAX: the maximum value of the field for each source grid point over the previous coupling period is transferred;
  - ONCE: UNUSED, not supported in OASIS3-MCT.

  Time transformations are now supported more generally than with previous versions of the coupler with use of the coupling restart file. The coupling restart file allows the partial time transformation at the end of the run, if any (e.g. if the AVERAGE period specified is longer than the run), to be saved at the end of a run for exact restart at the start of the next run. When LOCTRANS transformations are specified, the initial coupling restart file should not contain any LOCTRANS restart fields. For the following runs, it is mandatory that the coupling restart file contains LOCTRANS restart fields coherent with the current namcouple entries. For example, it will not be possible to restart a run with a multiple field entry in the namcouple with a coupling restart file created by a run not activating this multiple file option. This is the reason why it is now possible to specify a restart file name on

the OUTPUT *namcouple* input line. Note that if there is no partial transformation to be saved, the restart file will still contain a restart field with 0 everywhere.

4.2 The pre-processing transformations

- **REDGLO** UNUSED
- **INVERT**: UNUSED
- **MASK**: UNUSED
- **EXTRAP**: UNUSED
- **CHECKIN**:

  CHECKIN calculates the global minimum, maximum, mean, and sum of the source field values taking the mask into consideration. If a grid area or fraction field is also available, (respectively in the file *areas.nc* or *masks.nc*), then the area and/or fraction weighted mean and sum are also diagnosed and written. Information about masking and weighting is written to the output file. All diagnotics are written to the master process OASIS3-MCT debug file (under the attribute "CHECK* diags"). This operation does not transform the field. CHECKIN operations can slow down the simulation and should not be used in production mode. For backward compatibility, CHECKIN has one generic input line that is no longer used but is still required and can contain anything. See also CHECKOUT.

  The generic input line is as follows:

  ```
  # CHECKIN operation
      INT = 1
  ```

- **CORRECT**: UNUSED
- **BLASOLD**:

  BLASOLD allows the source field to be scaled and allows a scalar to be added to the field. The prior ability to perform a linear combination of the current coupling field with other coupling fields has been deprecated in OASIS3-MCT. This transformation occurs before the interpolation *per se*.

  This transformation requires at least one configuring line with two parameters:

  ```
  # BLASOLD operation
       $XMULT   $NBFIELDS
  ```

  where $XMULT is the multiplicative coefficient of the source field. $NBFIELDS must be 0 if no scalar needs to be added or 1 if a scalar needs to be added. In this last case, an additional input line is required where $AVALUE is the scalar to be added to the field, which must also be given as a REAL value :

  ```
       CONSTANT  $AVALUE
  ```

4.3 The remapping (or interpolation or regridding)

- **MAPPING**:

  The MAPPING keyword is used to specify an input file to be read and used for remapping. The MAPPING file must follow the SCRIPR format (see section 5.4) but can be generated by another library, for example ESMF or XIOS (see details in section 6.3.3). As for the other transformations and interpolations, different mappings can be specified for the different coupling fields. Grid data files *grids.nc*, *masks.nc*, and *areas.nc* are not needed for MAPPING.

  Since OASIS3-MCT_2.0, MAPPING can be used for higher order remapping. Up to 5 different sets of weights (see section 5.4 for the weight file format) can be applied to up to 5 different fields transfered through the oasis_put argument (see section 2.2.7).

This transformation requires at least one configuring line with one filename and two optional string values:

```
$MAPNAME  $MAPLOC  $MAPSTRATEGY
```

- `$MAPNAME` is the name of the mapping file to read. This is a NetCDF file consistent with the SCRIPR map file format (see section 5.4).
- `$MAPLOC` is optional and can be either `src` or `dst`. With `src`, the mapping will be done in parallel on the source processors before communication to the destination model processes; this is the default. With `dst`, the mapping is done on the destination processes after the source grid data is sent from the source model.
- `$MAPSTRATEGY` is optional and can be either `bfb`, `sum`, or `opt`. In `bfb` mode, the mapping is done using a strategy that produces bit-for-bit identical results regardless of the grid decompositions; this is the default. With `sum`, the transform is done using the partial sum approach which generally introduces roundoff level changes in the results on different processor counts. Option `opt` allows the coupling layer to choose either approach based on an analysis of which strategy is likely to run faster. Usually, partial sums will be used if the source grid has a higher resolution than the target grid as this should reduce the overall communication (in particular for conservative remapping). By default `$MAPSTRATEGY` = `bfb`.

Note that if `SCRIPR` (see below) is used to calculate the remapping file, `MAPPING` can still be listed in the `namcouple`, for example to use it with a specific `$MAPLOC` or `$MAPSTRATEGY` option.

- **SCRIPR**:

  `SCRIPR` gathers the interpolation techniques offered by Los Alamos National Laboratory SCRIP 1.4 library [Jones 1999][1]. `SCRIPR` routines are in `oasis3-mct/lib/scrip`. See the SCRIP 1.4 documentation in `oasis3/doc/SCRIPusers.pdf` for more details on the interpolation algorithms.

  Since OASIS3-MCT_4.0 release, a hybrid MPI+OpenMP parallel version of the SCRIP library is available. It relies on the MPI parallel layout of the calling model but only uses one MPI process per node. The number of OpenMP threads per MPI process used by SCRIP is set by a dedicated environment variable OASIS_OMP_NUM_THREADS, which can be different from the default number of threads per MPI process used elsewhere in the application, set by the environment variable OMP_NUM_THREADS ; in practice, OASIS_OMP_NUM_THREADS has to be smaller or equal to OMP_NUM_THREADS. For optimum performance, it is recommended to set OASIS_OMP_NUM_THREADS to the number of cores on the node. The `regrid_environment` directory (see section 6.3.3) in `oasis3-mct/examples/regrid_environment` gives a practical example on how to use the SCRIP library[2] in parallel to calculate regridding weight-and-address files. The details of the SCRIP parallelisation can be found in [Piacentini et al 2018] and [Valcke et al 2018][3] .

  When the SCRIP library performs a remapping, it first checks if the file containing the corresponding remapping weights and addresses exists; if it exists, it reads them from the file; if not, it calculates them and store them in a file. In the later case, SCRIP needs the grid data files `grids.nc` and `masks.nc` (see section 5.1). The weight-and-address file is created in the working directory and is by default called `rmp_`*src*`_to_`*tgt*`_INTTYPE_NORMAOPT.nc`, `rmp_`*src*`_to_`*tgt*`_INTTYPE_NBR.nc`,

[1] See also http://climate.lanl.gov/Software/SCRIP/ and the copyright statement in appendix 1.3.3.

[2] The `regrid_environment` directory provides a practical example on how to calculate the regridding weight-and-address files with the SCRIP library but also with ESMF and XIOS, see section 6.3.3

[3] A few bugs were fixed in the SCRIP library available since OASIS3-MCT_4.0 release, in particular in the bounding box definition of the grid cells. This solves an important bug observed in the Pacific near the equator for the bilinear and bicubic interpolations for Cartesian grids. However, given these modifications, one cannot expect to get exactly the same results for the weight-and-address remapping files with this new parallel SCRIP version as compared to the previous SCRIP version in OASIS3-MCT_3.0. We checked in many different cases that the interpolation error is smaller or of the same order than before. We also observed that the parallelisation does not ensure bit reproducible results when varying the number of processes or threads.

or rmp_*src*_to_*tgt*_*INTTYPE*.nc. *src* and *tgt* are the acronyms of respectively the source and the target grids, *INTTYPE* is the interpolation type (i.e. DISTWGT, GAUSWGT, GAUSWGTNF, DISTWGTNF, LOCCUNIF, LOCCDIST, LOCCGAUS, BILINEAR, - **not BILINEA as in OASIS3.3**, BILINEARNF, BICUBIC, BICUBICNF, or CONSERV). Then *NORMAOPT* is the normalization option, for SCRIPR/CONSERV only (i.e. DESTAREA, DESTARTR, DESTNNEI, DESTNNTR, FRACAREA, FRACARTR, FRACNNEI or FRACNNTR, see below). *NBR* is the number of neighbors used for DISTWGT, DISTWGTNF, GAUSWGT, GAUSWGTNF LOCCUNIF, LOCCDIST or LOCCGAUS only. One has to take care that the remapping file will have the same name even if other details, like the grid masks or the $MAPLOC or $MAPSTRATEGY options, are changed. When reusing a remapping file, one has to be sure that it was generated in exactly the same conditions than the ones it is used for.

The following types of interpolations are available:

- DISTWGT performs a distance weighted nearest-neighbour interpolation (N neighbours) with a nearest neighbor fill for non-masked target points that do not receive a value. All types of grids are supported.
- DISTWGTNF performs a distance weighted nearest-neighbour interpolation (N neighbours) without the nearest neighbor fill for non-masked target points. All types of grids are supported.

  The configuring line is:

  ```
  # SCRIPR (for DISTWGT or DISTWGTNF)
      $CMETH $CGRS $CFTYP $REST $NBIN $NV $ASSCMP $PROJCART
  ```

  where:

  * $CMETH = DISTWGT or DISTWGTNF.
  * $CGRS is the source grid type (LR, D or U)- see appendix A.
  * $CFTYP is the field type: SCALAR. The option VECTOR, which in fact leads to a scalar treatment of the field (as in the previous versions), is still accepted. **VECTOR_I or VECTOR_J (i.e. vector fields) are not supported anymore in OASIS3-MCT**. See "Support of vector fields with the SCRIPR remappings" below.
  * $REST is a bin search restriction type: LATLON or LATITUDE, see SCRIP 1.4 documentation SCRIPusers.pdf.
  * $NBIN the number of restriction bins that must be equal to 1 for DISTWGT, DISTWGTNF, GAUSWGT, GAUSWGTNF, BILINEAR, BILINEARNF, BICUBIC or BICUBICNF (i.e. the bin restriction is not allowed[4]; for details, see [Piacentini et al 2018]).
  * $NV is the number of neighbours used.
  * $ASSCMP, $PROJCART: UNUSED; **vector fields are not supported anymore in OASIS3-MCT.** See "Support of vector fields with the SCRIPR remappings" below.
- GAUSWGT performs a N nearest-neighbour interpolation weighted by their distance and a gaussian function with a nearest neighbor fill for non-masked target points that do not receive a value. All grid types are supported.
- GAUSWGTNF performs a N nearest-neighbour interpolation weighted by their distance and a gaussian function without the nearest neighbor fill for non-masked target points. All grid types are supported.

  The configuring line is:

[4]The only exceptions are for Gaussian Reduced (D) grids for (BILINEAR, BILINEARNF, BICUBIC and BICUBICNF; in that case, **if the Gaussian-reduced grid is stored from North to South** the number of bins is the number of latitude circles of the grid (minus one, to be precise), independently of $NBIN; **for Gaussian-reduced grid stored from South to North to South, the bin definition will not work and the interpolation will become a 4 distance-weighted nearest-neighbour for all target points.**

```
# SCRIPR (for GAUSWGT or GAUSWGTNF)
    $CMETH  $CGRS  $CFTYP  $REST  $NBIN  $NV $VAR
```

where all entries are as for DISTWGT, except that:

* $CMETH = GAUSWGT or GAUSWGTNF
* $VAR defines the weight given to a neighbour source grid point as proportional to $exp(-1/2 \cdot d^2/\sigma^2)$ where d is the distance between the source and target grid points, and $\sigma^2 = \$VAR \cdot \overline{D}^2$ where $\overline{D}^2$ is, for each target grid point, the square of average distance between its source grid points (calculated automatically by OASIS3-MCT).

– LOCCUNIF, LOCCDIST and LOCCGAUS perform a locally conservative interpolation by associating N target nearest neighbours to every SOURCE grid point, and applying a weight normalization taking into account the source/target mesh area ratio. Interpolation weights can additionally be modulated by the source/target distances (LOCCDIST) or by the source/target distances and a gaussian function (LOCCGAUS). All types of grids are supported. These interpolations are convenient to transfer coupling fields from disjoints areas, such as the river runoff flux, as studied in [Voldoire 2020].

  The configuring line is:

```
 # SCRIPR (for LOCCUNIF or LOCCDIST)
$CMETH $CGRS $CFTYP $REST $NBIN $NV
```

  or

```
# SCRIPR (for LOCCGAUS)
$CMETH $CGRS $CFTYP $REST $NBIN $NV $VAR
```

  where:

  * $CMETH $CGRS $CFTYP $REST $NBIN $NV entries are as for DISTWGT and $VAR is as for GAUSWGT. Note that for these interpolations, $NV represents the number of *target* (and not source) neighbours. For details, see [Maisonnave 2020].

– BILINEAR performs an interpolation based on a local bilinear approximation (see details in chapter 4 of SCRIP 1.4 documentation SCRIPusers.pdf). Logically-Rectangular (LR) and Reduced (D) source grid types are supported. It also generates regridded values with a nearest neighbor fill for non-masked target points that do not receive any value with the original algorithm.

– BILINEARNF is identical to BILINEAR but without the nearest neighbor fill.

– BICUBIC performs an interpolation based on a local bicubic approximation for Logically-Rectangular (LR) grids (see details in chapter 5 of SCRIP 1.4 documentation SCRIPusers.pdf), and on a 16-point stencil for Gaussian Reduced (D) grids. Note that for Logically-Rectangular grids, 4 weights for each of the 4 enclosing source neighbours are required corresponding to the field value at the point, the gradients of the field with respect to *i* and *j*, and the cross gradient with respect to *i* and *j* in that order. OASIS3-MCT will calculate the remapping weights and addresses (if they are not already provided) but will not, at run time, calculate the two gradients and the cross-gradient of the source field (as was the case with OASIS3.3). These 3 extra fields need to be calculated by the source code and transfered as extra arguments of the oasis_put (see fld2, fld3, fld4 in section 2.2.7). Finally, this interpolation will fill non-masked target points that do not receive a value with a nearest neighbor fill.

– BICUBICNF is identical to BICUBIC but without the nearest neighbor fill.

  For BILINEAR, BILINEARNF, BICUBIC, and BICUBICNF, the configuring line is:

```
 # SCRIPR  (for BILINEAR, BILINEARNF, BICUBIC, or BICUBICNF)
$CMETH  $CGRS  $CFTYP  $REST  $NBIN
```

  where:

  * $CMETH = BILINEAR, BILINEARNF, BICUBIC, or BICUBICNF

  * $CGRS is the source grid type: LR or D.
  * $CFTYP, $REST, $NBIN are as for DISTWGT.

  Note that for DISTWGT, GAUSWGT, BILINEAR and BICUBIC:

  * Masked target grid points: the zero value is associated to masked target grid points.
  * Non-masked target grid points having some of the source points normally used in the interpolation masked: a N nearest neighbour algorithm using the remaining non masked source points is applied.
  * Non-masked target grid points having all source points normally used in the interpolation masked: by default, the nearest non-masked source neighbour is used (ll_nnei is set to .true. in the SCRIP run).

  Note that for DISTWGTNF, GAUSWGTNF, BILINEARNF and BICUBICNF:

  * Identical behavior to non-NF options but no nearest non-masked source neighbour fill is done (ll_nnei is set to .false in the SCRIP run). Non-masked target grid points that do not receive any interpolated value are set to 0 .

– CONSERV performs 1st or 2nd order conservative remapping, which means that the weight of a source cell is proportional to area intersected by the target cell (plus some other terms proportional to the gradient of the field in the longitudinal and latitudinal directions for the second order).

  The configuring line is:

```
# SCRIPR (for CONSERV)
```

  where: $CMETH $CGRS $CFTYP $REST $NBIN $NORM $ORDER $NTHRESH $STHRESH

  * $CMETH = CONSERV
  * $CGRS is the source grid type: LR, D and U. Note that the grid corners have to given by the user in the grid data file grids.nc or by the code itself in the initialisation phase by calling routine oasis_write_corner (see section 2.2.4) ; OASIS3-MCT will not attempt to automatically calculate them as OASIS3.3.
  * $CFTYP, $REST are as for DISTWGT.
  * $NBIN is the number of restriction bins that can be more than 1 as bin restriction is effectively allowed for CONSERV; for details, see [Piacentini et al 2018].
  * $NORM is the normalization option:
    · FRACAREA or FRACARTR: The sum of the non-masked source cell intersected areas is used to normalise each target cell field value: the flux is not locally conserved, but the flux value itself is reasonable. With FRACARTR, an additional correction involving the "true" area of the grid cells provided in the file areas.nc is also applied, see details in the paragraph on the "True Area" (TR) correction below.
    · FRACNNEI or FRACNNTR: as FRACAREA or FRACARTR, except that an additional unmasked source nearest neighbour is used for unmasked target cells that intersect only masked source cells.
    · DESTAREA or DESTARTR: The total target cell area is used to normalise each target cell field value even if it only partly intersects non-masked source grid cells: local flux conservation is ensured, but unreasonable flux values may result. With DESTARTR, an additional correction involving the "true" area of the grid cells provided in the file areas.nc is also applied, see details in the paragraph on the "True Area" (TR) correction below.
    · DESTNNEI or DESTNNTR: as DESTAREA or DESTARTR, except that an additional unmasked source nearest neighbour is used for unmasked target cells that intersect only masked source cells.

* $ORDER: FIRST or SECOND for first or second order conservative remapping respectively (see SCRIP 1.4 documentation).

  For CONSERV/SECOND, 3 weights are needed; OASIS3-MCT will calculate these weights and corresponding addresses (if they are not already provided) but will not, at run time, calculate the two extra terms to which the second and third weights should be applied. These terms, respectively the gradient of the field with respect to the latitude (θ) $\frac{\delta f}{\delta \theta}$ and the gradient of the field with respect to the longitude (ϕ) $\frac{1}{cos\theta}\frac{\delta f}{\delta \phi}$ need to be calculated by the source code and transferred as extra arguments of the oasis_put as fld2 and fld3 respectively (see section 2.2.7). Note that CONSERV/SECOND is not positive definite.

* $NTHRESH is the value of the northern latitude threshold in radians where conservative area computation switches from linear boundaries in longitude and latitude at the equator to a Lambert equivalent azimuthal projection toward the north pole. This value is optional and the default is 2.0 radians (greater than 90 degrees) which means the Lambert projection is never invoked. If this value is set, $STHRESH must be set as well.

* $STHRESH is the value of the southern latitude threshold in radians where conservative area computation switches from linear boundaries in longitude and latitude at the equator to a Lambert equivalent azimuthal projection toward the south pole. This value is optional and the default is -2.0 radians (less than -90 degrees) which means the Lambert projection is never invoked. If this value is set, $NTHRESH must be set as well.

Precautions related to the use of the SCRIPR/CONSERV remapping

1. Lambert projection above/below north_thresh/south_thresh

   For the 1st order conservative remapping, the weight of a source cell is proportional to area of the source cell intersected by target cell. Using the divergence theorem, the SCRIP library evaluates this area with the line integral along the cell borders enclosing the area. As the real shape of the borders is not known (only the location of the corners of each cell is known), the library assumes that the borders are linear in latitude and longitude between two corners. This assumption becomes less valid closer to the pole.

   Therefore for latitudes above the north_thresh or below the south_thresh values (see oasis3-mct/lib/scrip/remap_conserv.F90), the library evaluates the intersection between two border segments using a Lambert equivalent azimuthal projection. **However, by default, north_thresh and south_thresh are set to 2.0 and -2.0 radians respectively and the Lambert projection is not activated.** (Note that north_thresh was set to 1.45 in previous versions prior to OASIS3-MCT_4.0.)

   The value of the north and south threshhold can be defined in the namcouple file via the $NTHRESH and $STHRESH optional settings on the SCRIPR input line.

   [Valcke and Piacentini 2019] analyses the impact of the Lambert projection for specific grids.

2. Another limitation of the SCRIP 1st order conservative remapping algorithm is that it assumes, for line integral calculation, that $sin(latitude)$ is linear in longitude on the cell borders which again is in general not valid close to the pole.

3. For a proper consevative remapping, the corners of a cell have to coincide with the corners of its neighbour cell, with no "holes" between the cells.

4. Overlying cells

   If two cells of one same grid overlay, the one with the greater numerical index must be masked in *masks.nc* for a proper conservative remapping. For example, if the grid cells with *i=1* overlays the grid cells with *i=imax*, the latter must be masked. If none of overlying cells is masked (given the original mask defined in *masks.nc*), OASIS3-MCT must be compiled with the CPP key TREAT_OVERLAY which will ensure that these rules are respected. This CPP key was introduced in OASIS3.3.

5. Masked (non-active) target grid cells are set to 0.
6. If a target grid cell intersects only masked source cells, or falls outside the source grid domain, it will get a zero value unless the DESTNNEI, DESTNNTR, FRACNNEI, or FRACNNTR normalisation options are used, in which case it will get the source nearest non masked neighbour value. Note that the option of having the value 1.0E+20 assigned to these target grid cell intersecting only masked source cells (for easier identification) is not yet availble in OASIS3-MCT.

The "True Area" (TR) correction: DESTARTR, DESTNNTR, FRACARTR, or FRACNNTR

The approximations and hypotheses adopted by the SCRIP impact its estimation of the grid cell areas. Therefore, to have an exact conservation of the field surface-integrated values, a correction based on the "TRue" (TR) area of the cells can be applied by choosing DESTARTR, DESTNNTR, FRACARTR or FRACNNTR options. These are based respectively on DESTAREA, DESTNNEI, FRACAREA and FRACNNEI normalisations adding the so-called "TR correction". The true area of the cell, i.e. the one considered by the component model itself, must be provided in the auxiliary file *areas.nc* in **square radians**. Equations from [Chavas et al 2013] (eqn. (27) in particular) are implemented.

Special care is taken in the implementation for "polar" cells in the SCRIP sense. The SCRIP detects cells encompassing a pole or cells with a side going through a pole as "polar" cells and a specific treatment is applied for those cells modifying its area. The resulting estimated area serves as a normalisation factor but its value is not representative of the surface of the cell anymore. For this reason, the TR correction is not activated for those "polar" cells.

More details can be found in section 6 of [Valcke and Piacentini 2019]. A full validation of the TR correction can be found in that report. It is noted there that the TR correction always improves the results even if, in the cases tested, the misfit between the true area and the are evaluated by the SCRIP is always small (except for "polar" cells of course).

Support of vector fields with the SCRIPR remappings

Vector mapping is NOT supported and will not be supported by OASIS3-MCT. For proper treatment of vector fields, the source code has to send the 3 components of the vector projected in a Cartesian coordinate system as separate fields. The target code has to receive the 3 interpolated Cartesian components and recombine them to get the proper vector field.

- **INTERP**: UNUSED
- **MOZAIC**: UNUSED; note that MAPPING (see above) is the NetCDF equivalent to MOZAIC.
- **NOINTERP**: UNUSED
- **FILLING**: UNUSED

4.4 The post-processing stage

- **CONSERV**:

  CONSERV performs a global modification of the coupling field such that the area integrated fields are conserved.

  This analysis requires the source and target grid mesh areas be defined in file *areas.nc* file and the source and target grid mask be defined in file *masks.nc*.

  The areas must be expressed in matching units on the source and destination sides. Futhermore, these must be square radians if the TR correction is activated (see paragraph The "True Area" (TR) correction in section 4.3).

  A grid cell mask must be defined for that operation in the *masks.nc* file. If grid cell fractions are also defined in that file, the mask and fractions are considered and must be consistent; OASIS3-MCT will abort if they are not or if both are missing. Note that by OASIS3-MCT conventions for

the mask, a gridcell with mask=0 (active) should have fractions greater than 0 and a gridcell with mask=1 (inactive) should have fractions equal to 0.

Best practice for fraction definition in ocean-atmosphere coupling

In principle, the fractions can be defined for both the source and target grids. But for ocean-atmosphere coupling, we strongly encourage the following best practice for a consistent ocean-atmosphere coupled system. Indeed, to have a well-posed coupled problem, the ocean and the atmospheric total surfaces must be the same allowing global conservation of integrated quantities. To do so, the original ocean mask should be taken as it is from the ocean model. For the atmosphere, cell fractions should be defined by the conservative remapping of [1 - ocean mask] on the atmospheric grid, retaining fractions above a certain threshold, to be fixed by the user. These atmospheric cell fractions should be used in the atmospheric model to define the % of ocean (water) subsurface to be considered. Then the atmospheric coupling mask should be adapted associating a non-masked index (i.e. 0) to all cells with a water fraction above the chosen threshold. The global water surface as seen by the atmosphere model is then the sum of its cell areas multiplied by its respective cell fractions. Note that invalid masked atmospheric cells should have null ocean fractions. If we follow this best practice, the atmospheric cell fractions and mask will be specific to the coupling with each particular ocean grid. A specific attribute named `coherent_with_grid` indicating the grid prefix of the `companion` grid may be defined for mask and fractions. If the OASIS API is used to define the mask and fractions, this can be done via the optional argument `companion` indicating the grid prefix of the companion grid (see section 2.2.4).

In the *namcouple*, `CONSERV` requires one input line with one argument and one optional argument:

```
# CONSERV operation
```
where: `$CMETH  $CONSOPT`

- `$CMETH` is the method desired with the following choices; a detailed analysis of these choices can be found in [Craig 2019] :
  - with `$CMETH = GLOBAL`, the field is integrated on both source and target grids, without considering values of masked points, and the residual (target - source) is uniformly distributed on the target grid as an additive term; this option ensures global conservation of the field;
  - with `$CMETH = GLBPOS`, the same operation as `GLOBAL` is performed except that the residual is distributed proportionally to the value of the original field as a multiplicative term; this option ensures the global conservation of the field, and it does not change the sign of the field if the field is well behaved; if the field is well behaved, multiplication factors close to 1 are expected.
  - with `$CMETH = GSSPOS`, the same operation as `GLBPOS` is performed except that the multiplicative term is computed separately for positive and negative values of the field; this option ensures the global conservation of the field and, as `$GLBPOS`, does not change the sign of the field, but it is more expensive because many extra global sums are required; this should be used in cases where the area averaged field value tends to 0 as these cases can generate poor corrections, even leading to changes of sign, when carried out with the `GLBPOS` option.
  - with `$CMETH = BASBAL`, the operation is analogous to `GLOBAL` as an additive term except the non masked active surface of the source and the target grids are taken into account in the calculation of the residual; this option does not ensure global conservation of the field but ensures that the energy received is proportional to the non masked active surface of the target grid;
  - with `$CMETH = BASPOS`, the operation is analgous to `GLBPOS` as a multiplicative term except the non masked surface of the source and the target grids are taken into account and the residual is distributed proportionally to the value of the original field; this option

does not ensure global conservation of the field but ensures that the energy received is proportional to the non masked surface of the target grid and it does not change the sign of the field if the field is well behaved.

* with `$CMETH = BSSPOS`, the same operation as `BASPOS` is performed except that the multiplicative term is computed separately for positive and negative values of the field; this option has the same characteristics as `BASPOS` except it does not change the sign of the field, but is more expensive because many extra global sums are required; this should be used in cases where the area averaged field value tends to 0 as these cases can generate poor corrections, even leading to changes of sign, when carried out with the `BASPOS` option.

– `$CONSOPT` is an optional argument specifying the algorithm. `$CONSOPT` can be `bfb`, `gather`, `lsum16`, `lsum8`, `ddpdd`, `reprosum` or `opt`. Details on the perfromance of these different options can also be found in [Craig et al 2017] and references there in.

* The `bfb` option is the default for `CONSOPT` and uses the `reprosum` option (see below).
* The `gather` option computes global sums by gathering a decomposed array onto the root process before doing an index ordered sum. This is guaranteed to produce identical results for different numbers of processors and decompositions but is expensive both with respect to performance and memory use. This is equivalent to the `bfb` option in previous OASIS3-MCT versions.
* The `lsum16` option computes a local sum at quadruple precision before doing an MPI reduction on the local sums at quadruple precision. This is likely to be bit-for-bit for different numbers of processors and decompostions but that's not guaranteed. This is just like `lsum8` but at quadruple precision and a little slower.
* The `lsum8` option computes a local sum at double precision before doing an MPI reduction on the local sums at double precision. This is NOT likely to be bit-for-bit for different numbers of processors and decompostions. This is just like `lsum16` but at double precision and faster.
* The `ddpdd` option is a parallel double-double algorithm using a single scalar reduction. It should behave between `lsum8` and `lsum16` with respect to performance and reproducibility. See [He and Ding 2001].
* The `reprosum` option is a fixed point method based on ordered double integer sums that requires two scalar reductions per global sum. The cost of reprosum will be higher than some of the other methods but it will be bit-for-bit for different processor counts or different decompostions except in extremely rare cases and the cost is significantly less than the `gather` option. See [Mirin and Worley 2012].
* The `opt` option carries out the global sum using the fastest algorithm generally available. Currently, this is set to `lsum8`.

- **SUBGRID**: UNUSED
- **BLASNEW**:

  `BLASNEW` performs a scalar multiply or scalar add to any destination field. This is the equivalent of `BLASOLD` on the destination side. In addition, unlike `BLASOLD`, other fields on the destination side can be added with a multiplier and addition weight.

  This transformation requires at least one configuring line with two parameters:

```
# BLASNEW operation
     $XMULT   $NBFIELDS
```

  where `$XMULT` is the multiplicative coefficient of the destination field. `$NBFIELDS` will be 0 if no additional fields or scalars are needed, 1 if a single scalar needs to be added, and greater than 1

if additional fields are to be added. The number of $NBFIELDS indicates the number of additional lines. If $NBFIELDS is greater than 0, the first additional input line must be the string CONSTANT and then a real value, $AVALUE, which will be added to the field. Even if $AVALUE is zero, this line must still be included if $NBFIELDS is greater than 0. If $NBFIELDS is greater than 1, then additional input lines have the format $FNAME $XMULT $AVALUE where $FNAME is the name of a field received in OASIS3-MCT in the same model and $XMULT and $AVALUE are the multipliers and additive terms to be applied to $FNAME :

```
CONSTANT  $AVALUE
$FNAME1 $XMULT1 $AVALUE1
$FNAME2 $XMULT2 $AVALUE2
```

For example :

```
2.0  3
CONSTANT  0.0
FLD001 1.0 0.0
FLD002 5.0 -100.
```

will multiply the destination field by 2.0 and then add (FLD001 + FLD002*5.0 - 100) to that destination field. All combined fields must be received by the same model component in OASIS3-MCT (either via coupling or input), and the field size and decomposition must be consistent across all fields being combined. The value of the field being combined is associated with the last valid coupled value. This allows fields to be combined that are not coupled at the same frequency by using valid lagged values. The order of the receive calls is also important. If a field to be combined is received after the destination field, then the values used are from an earlier timestep. Note that while this feature is supported in OASIS3-MCT, a more transparent implementation might be to combine fields in the model (not in OASIS3-MCT) after they are received independently.

- **MASKP**: UNUSED
- **REVERSE**: UNUSED
- **CHECKOUT**:

  CHECKOUT calculates the global minimum, maximum, mean, and sum of the destination field values taking the mask into consideration. If a grid area or fraction field is also available, (respectively in the file *areas.nc* or *masks.nc*), then the area and/or fraction weighted mean and sum are also diagnosed and written. Information about masking and weighting is written to the output file. All diagnotics are written to the master process OASIS3-MCT debug file (under the attribute "CHECK* diags"). This operation does not transform the field. CHECKOUT operations can slow down the simulation and should not be used in production mode. For backward compatibility, CHECKOUT has one generic input line that is no longer used but is still required and can contain anything. See also CHECKIN.

  The generic input line is as follows:

  ```
  # CHECKOUT operation
      INT = 1
  ```

- **GLORED**: UNUSED

Chapter 5

OASIS3-MCT auxiliary data files

OASIS3-MCT uses auxiliary data files, e.g. defining the grids of the models being coupled, containing the field coupling restart values or input data values, or the remapping weights and addresses.

5.1 Grid data files

With OASIS3-MCT, the grid data files *grids.nc, masks.nc* and *areas.nc* are required for certain operations:

- *grids.nc* and *masks.nc* for all `SCRIPR` regriddings (see section 4.3)
- in addition, *areas.nc* for `SCRIPR` conservative remapping for which the normalisation by the true area of the cells is activated (i.e. `DESTARTR`, `DESTNNTR`, `FRACARTR`, or `FRACNNTR`
- *masks.nc* and *areas.nc* for global `CONSERV`, see section 4.4).

These NetCDF files can be created by the user before the run or can be written directly at run time by the processes of each component model using the grid data definition routines (see section 2.2.4).

The arrays containing the grid information are dimensioned `(nx, ny)`, where `nx` and `ny` are the grid first and second dimension. Unstructured grids or other grids expressed with 1D vectors are supported by setting `nx` to the total number of grid points and `ny` to 1.

1. *grids.nc*: contains the model grid longitudes and latitudes in double precision `REAL` arrays. The array names must be composed of a prefix (4 characters), given by the user in the *namcouple* on the second line of each field (see section 3.3), and of a suffix, `.lon` or `.lat`, for respectively the grid point longitudes or latitudes.

   If the `SCRIPR/CONSERV` remapping is specified, longitudes and latitudes for the source and target grid **corners** must also be available in the *grids.nc* file as double precision `REAL` arrays dimensioned `(nx,ny,nc)` where `nc` is is the maximum number of corners (in the counterclockwize sense, starting by any corner) over all cells; `nc` can be any number. For cells that do not have the maximum number of distinct corners, we recommend to repeat the last corner as many times as needed to describe `nc` corners. The names of the arrays must be composed of the grid prefix and the suffix `.clo` or `.cla` for respectively the grid corner longitudes or latitudes. As for the other grid information, the corners can be provided in *grids.nc* before the run by the user or directly by the component code through specific calls (see section 2.2.4).

   Longitudes must be given in degrees East in the interval -360.0 to 720.0. Latitudes must be given in degrees North in the interval -90.0 to 90.0. Note that if some grid points overlap, it is recommended to define those points with the same number (e.g. 360.0 for both, not 450.0 for one and 90.0 for the other) to ensure automatic detection of overlap by OASIS3-MCT.

   The corners of a cell cannot be defined modulo 360 degrees. For example, a cell located over Greenwich will have to be defined with corners at -1.0 deg and 1.0 deg but not with corners at 359.0 deg and 1.0 deg.

Cells larger than 180.0 degrees in longitude are not supported.

Longitudes and latitudes of grid points or corners can be defined through the `oasis_write_grid` or `oasis_write_corner` API routines respectively, see section 2.2.4.

2. *masks.nc*: contains the masks for all component model grids in `INTEGER` arrays. **Be careful to use the historical OASIS3-MCT convention: 0 = not masked (i.e. active), 1 = masked (i.e. not active) for each grid point**. The array names must be composed of the grid prefix and the suffix ".msk". Mask can be defined through the `oasis_write_mask` API routine, see section 2.2.4.

   File *masks.nc* may also contain the grid cell fractions that defines the active (i.e. not masked) part of the cells. Fractions should be consistent with the mask field. The fraction fields, if present, are only used in the global `CONSERV`, `CHECKIN` and `CHECKOUT` operations. If both a mask and fractions are defined for a grid, they must be consistent; OASIS3-MCT will abort if they are not coherent or if both are missing. Note that by OASIS3-MCT convention for the mask, a gridcell with mask=0 (active) should have a fractions greater than 0 and a gridcell with mask=1 (inactive) should have a fractions equal to 0. The fraction array name must be composed of the grid prefix and the suffix `.frc`. Fractions can be defined through the `oasis_write_frac` API routine, see section 2.2.4.

3. *areas.nc*: this file contains mesh surfaces for the component model grids in double precision `REAL` arrays. The array names must be composed of the grid prefix and the suffix `.srf`. The surfaces may be given in any units but they must be the same on the source and target sides; furthermore they must be in square radians if the True Area (TR) correction is activated, see section 4.3.

   Surfaces can be defined through the `oasis_write_area` API routine, see section 2.2.4.

   This file *areas.nc* is mandatory for the global `CONSERV` post-processing operation; it is not required otherwise.

5.2 Coupling restart files

At the beginning of a coupled run, some coupling fields may have to be initially read from their coupling restart file on their source grid (see the LAG concept in section 2.5.3). When needed, these files are also automatically updated by the last active `oasis_put` or `prism_put_proto` call of the run (see section 2.2.7) . **Warning**: the date is not written or read to/from the restart file; therefore, the user has to make sure that the appropriate coupling restart file is present in the working directory. The coupling restart files must follow the NetCDF format.

The name of the coupling restart file is given by the 6th character string on the first configuring line for each field in the *namcouple* (see section 3.3). Coupling fields coming from different models cannot be in the same coupling restart files, but for each model, there can be an arbitrary number of fields written in one coupling restart file. One exception is when a coupling field sent by a source component model is associated with more than one target field and model; in that case, if coupling restart files are required, it is mandatory to specify different files for the different fields.

The coupling restart files are also used automatically by OASIS3-MCT to allow partial LOCTRANS time transformation to be saved at the end of a run for exact restart at the start of the next run. When LOCTRANS transformations are specified, the initial coupling restart file should not contain any LOCTRANS restart fields. For the following runs, it is mandatory that the coupling restart file contains LOCTRANS restart fields coherent with the current namcouple entries. For example, it will not be possible to restart a run with a multiple field entry in the namcouple with a coupling restart file created by a run not activating this multiple file option.

In the coupling restart files, the fields must be provided on the source grid in single or double precision REAL arrays and, as the grid data files, must be dimensioned `(nx, ny)`, where `nx` and `ny` are the grid first and second dimension (see section 5.1 above). The shape and orientation of each restart field (and of the corresponding coupling fields exchanged during the simulation) must be coherent with the shape of its grid data arrays.

5.3 Input data files

Fields with status INPUT in the *namcouple* will, at runtime, simply be read in from a NetCDF input file by the target model below the oasis_get call, at appropriate times corresponding to the input period indicated by the user in the *namcouple*.

The name of the file must be the one given on the field first configuring line in the *namcouple* (see section 3.3.3). There must be one input file per INPUT field, containing a time sequence of the field in a single or double precision REAL array named with the same field symbolic name as in the *namcouple* and dimensioned (nx,ny,time). The time variable has to be an array time(time) expressed in "seconds since beginning of run" (or any other time units as long as the same are used in all components and in the *namcouple*) . The "time" dimension has to be the unlimited dimension.

5.4 Transformation auxiliary data files

The mapping files to be used for the MAPPING option must be consistent with the files generated by the SCRIP library to be used for the SCRIPR transformations (see also section 4.3). The files are NetCDF containing the following fields:

```
dimensions:
      src_grid_size = xxx ;
      dst_grid_size = xxx ;
      num_links = xxx ;
      num_wgts = xxx ;
variables:
      int src_address(num_links)
      int dst_address(num_links)
      double remap_matrix(num_links, num_wgts)
```

where

- src_grid_size is a scalar integer indicating the total number of points in the source grid. This field is a netCDF dimension.
- dst_grid_size is a scalar integer indicating the total number of points in the target grid. This field is a netCDF dimension.
- num_links is a scalar integer indicating the total number of associated source and target grid point pairs in the file. This field is a netCDF dimension.
- num_wgts is a scalar integer indicating the number of weights per associated grid point pair (up to 5 are supported, see sections 2.2.7 and 4.3 for BICUBIC and CONSERV/SECOND). This field is a netCDF dimension.
- src_address is a one dimensional array of size num_links. It contains the integer source address associated with each weight. This field is a netCDF variable.
- dst_address is a one dimensional array of size num_links. It contains the integer destination address associated with each weight. This field is a netCDF variable.
- remap_matrix is a two dimensional array of size (num_links, num_wgts). It contains the real weight value(s) associated with the source and destination address. For each link, up to 5 weights are supported, see sections 2.2.7 and 4.3 especially for BICUBIC and CONSERV/SECOND. This field is a netCDF variable.

Chapter 6

Compiling, running, debugging, load balancing

6.1 Compiling OASIS3-MCT

OASIS3-MCT is a mixed MPI-OpenMP parallel code. Compiling OASIS3-MCT libraries can be done from the `oasis3-MCT/util/make_dir` directory with the makefile `TopMakefileOasis3`.

`TopMakefileOasis3` includes the header file `make.inc` which should then point to (include) your own `make.your_platform` file. That file is specific to the hardware and compiling platform used.

Several header files are distributed with the release and can by used as a template to create a custom file for your machine. The root of the OASIS3-MCT tree can be anywhere, but it must be defined by the variable `COUPLE`. Similarly, the variable `ARCHDIR` defines the location of the compilation directory. Finally, the OASIS3-MCT library should be compiled with the same compilers and system software as any coupled model component using it. After successful compilation, resulting libraries are found in the directory in `$ARCHDIR/lib` while object and module files are found in `$ARCHDIR/build-static` and `$ARCHDIR/build-shared`.

OASIS3-MCT has historically created static libraries for use in Fortran source codes. However, C language bindings are now available, and python codes are now fully supported. Therefore, the OASIS3-MCT makefile `TopMakefileOasis3` supports compilation of both static and shared libraries.

`TopMakefileOasis3` has several targets including:

- `oasis3-psmile` = static-libs-fortran (for backwards compatibility)
- `static-libs-fortran` = static OASIS3-MCT libraries for Fortran only (default)
- `shared-libs-fortran` = shared (dynamic) OASIS3-MCT libraries for Fortran only
- `static-libs` = static OASIS3-MCT libraries including Fortran and c-bindings
- `shared-libs` = shared (dynamic) OASIS3-MCT libraries including Fortran and c-bindings
- `pyoasis` = builds and installs shared-libs plus higher and intermediate python classes
- `realclean` = cleans and resets the build

The names of the libraries produced are *mct*, *mpeu*, *scrip*, *psmile.MPI1*, and *oasis.cbind* with standard prefixes (*lib*) and suffixes (*.a* or *.so*).

The following targets have been used historically to compile OASIS3-MCT for Fortran codes and they are all still supported:

- `make -f TopMakefileOasis3 help`
  provides a current list of available targets.
- `make -f TopMakefileOasis3 realclean`

removes all OASIS3-MCT compiled sources and librairies.

- `make -f TopMakefileOasis3` or

  `make -f TopMakefileOasis3 oasis3_psmile`

  compiles static versions of OASIS3-MCT Fortran libraries *mct*, *mpeu*, *scrip* and *psmile*;

Log and error messages from compilation are normally saved in the directory `/util/make_dir` in the files `COMP.log` and `COMP.err` or similar. The `TopMakefileOasis3` output will direct users to the compile output files.

To interface a component code with OASIS3-MCT and use the module `mod_oasis` (see section 2.2.1), it is required to include OASIS3-MCT modules from `$ARCHDIR/include` and link with appropriate libraries in `$ARCHDIR/lib` during the compilation and linking.

Exchange of coupling fields in single and double precision is now supported directly through the interface (see section 2.2.5). Single precision fields are converted to double precision fields internally and temporarily. For double precision coupling fields, there is no need to promote REAL variables to DOUBLE PRECISION at compilation; this is done automatically within the OASIS3-MCT library.

6.2 CPP keys

The following OASIS3-MCT CPP keys can be specified in `CPPDEF` in `make.`*your_platform* file:

- `TREAT_OVERLAY`: ensures, in `SCRIPR/CONSERV` remapping (see section 4.3), that if two cells of the source grid overlay and none is masked a priori, the one with the greater numerical index will not be considered (they also can be both masked); this is mandatory for this remapping. For example, if the grid line with *i=1* overlaps the grid line with *i=imax*, it is the latter that must be masked; when this is not the case with the mask defined in *masks.nc*, this CPP key forces these rules to be respected.
- `__NO_16BYTE_REALS`: **must** be specified if you compile with **PGF90**.

6.3 Examples on how to run OASIS3-MCT

The following examples of running environments are provided with the sources in the `oasis3-mct/examples` directory.

6.3.1 tutorial_communication

The directory `oasis3-mct/examples/tutorial_communication` contains the files of a tutorial to learn how to instrument codes with calls to the OASIS3-MCT library in order to couple them together. The tutorial involves two toy model codes, `ocean.F90` and `atmos.F90`, to be instrumented with calls to OASIS3-MCT API (Application Program Interface) routines. Toy models are skeleton programs that do not contain any real physics or dynamics but that can reproduce real exchanges of coupling fields. Instrumenting those toy models gives a practical experience of using the OASIS3-MCT library. All information about this tutorial is provided in the document `tutorial_communication.pdf` therein.

This tutorial is extracted from the Short Online Private Course (SPOC) on "Code Coupling with OASIS3-MCT" shortly described in the next section.

6.3.2 spoc

This directory contains the sources used in the Short Online Private Course (SPOC) on "Code Coupling with OASIS3-MCT" developed in the framework of the ESiWACE Centre of Excellence. This SPOC is

composed of videos, quizzes and hands-on. The goal is to instrument two toy models to set-up a real coupled model exchanging coupling fields (directory /spoc_communication) and to learn more about OASIS3-MCT regridding functionality (directory `/spoc_regridding`). If you are interested in attending the SPOC, please visit the online training section of CERFACS web site at https://cerfacs.fr/online-training/.

Videos and quizzes extracted from the SPOC are also available as Open Education Resources (OER) material at https://www.oercommons.org/courseware/lesson/85340 .

6.3.3 regrid_environment

The `regrid_environment` directory offers a scripting environment to calculate the regridding weights and the regridding error for specific couple of grids and specific regridding algorithms with either the SCRIP library, ESMF or XIOS. The document `regrid_environment_documentation.pdf` therein contains all instructions on how to run this tutorial.

6.3.4 Fortran, C and python equivalent examples

Different examples implementing the different parts of the API with the Fortran, C and python interfaces are provided as practical illustrations in directory `/pyoasis/examples` :

- `1-serial`: one coupling exchange between a serial sender and a serial receiver.
- `2-apple`: one coupling exchange between an Apple-parallel sender and a serial receiver; an additonal component, not part of the coupling, is also started and the example shows how to use the `commworld` argument, in Fortran and C, and the communicator optional argument when setting the component in python.
- `3-box`: one coupling exchange between an Box-parallel sender and a serial receiver; it shows also how to check if a coupling field declared in the code is activated in the configuration file *namcouple*.
- `4-orange`: one coupling exchange between an Orange-parallel sender and a serial receiver; not all processes of the sender participate in the coupling and this example shows how to use `create_couplcomm`.
- `5-points`: one coupling exchange between a Point-parallel sender and a serial receiver.
- `6-apple_and_orange`: one coupling exchange between an Apple-parallel sender and an Orange-parallel receiver; not all processes of the sender participate in the coupling and this example shows how to use `set_couplcomm`.
- `7-multiple-puts`: two coupling fields are both sent from a serial sender to two different serial receivers; this example also sets up an intra communicator between the sender and one receiver and an inter communicator between the sender and the other receiver.
- `8-interoperability/fortran_and_C`: implements a coupling of a bundle field, with two bundle elements, between a Fortran Apple-parallel sender and a C component. This C component is Orange-parallel for the reception of the bundle field; it also defines another partition of type Box onto which a second coupling field is defined and sent to a third Fortran serial receiver. The sum of the Box partitions in the C component does not cover the global grid, hence the fourth argument `ig_size` is used to specify the grid global size. The C component also illustrates how the order of the partition definition does not need to be the same for the different processes but that, in that case, a meaningful `name` fifth argument must be used.
- `8-interoperability/fortran_and_python`: implements the same coupling exchanges than `8-interoperability/fortran_and_C` but with the C component replaced by a python component.
- `9-python_fortran_C-multi_intracomm`: illustrates the set-up of an intracommunicator between a Fortran, a C and a python components using OASIS3-MCT; a bcast is then realised to share some data. In this example, an additional component is also launched at start but does not participate in the coupling and hence uses the `coupled` third argument of `oasis_initi_comp`.

- `10-grid`: a single Box-parallel component defines and writes two grids `pyoa` and `mono`, the first one with distributed calls from all the processes, the second one from the master process only.
- `11-test-interpolation`: one exchange of a coupling bundle field defined on real grids involving a first-order concervative regridding between an Apple-parallel sender and a serial receiver. In the Fortran and C examples, the grids are fixed, while in the python example, the user chooses the source and target grids interactively, among the ones available in the files available in the `common_data` directory. This example produces graphical output of the received fields if the following packages are installed
  - pip3 install matplotlib
  - pip3 install scipy
  - pip3 install cartopy
  - pip3 uninstall shapely
  - pip3 install shapely –no-binary shapely
- `12-grid-functions`: Graphical version of `10-grid` (i.e. the `pyoa` grid layout is displayed if the same graphical packages than for `11-test-interpolation` are installed).

The different examples can be launched with the `Makefile` from directory `/pyoasis` using targets `examples`, `examples_f` or `examples_c` to run respectively python, Fortran and C examples.

6.4 Debugging

6.4.1 Debug files

If you experience problems while running your coupled model with OASIS3-MCT, you can obtain more information on what is happening by increasing the `$NLOGPRT` value in your *namcouple*, see section 3.2 for details.

6.4.2 Time statistics files

The variable TIMER_Debug, defined in the *namcouple* (second number on the line below $NLOGPRT keyword), is used to obtain time statistics over all the processors for each routine.

Different output are written (in files named `*.timers_xxxx`) depending on TIMER_Debug value :

- TIMER_Debug=0 : nothing is calculated, nothing is written.
- TIMER_Debug=1 : the times are calculated and written in a single file by the process 0 as well as the min and the max times over all the processes.
- TIMER_Debug=2 : the times are calculated and each process writes its own file ; process 0 also writes the min and the max times over all the processes in its file.
- TIMER_Debug=3 : the times are calculated and each process writes its own file ; process 0 also writes in its file the min and the max times over all processes and also writes in its file all the results for each process.

The time given for each timer is calculated by the difference between calls to `oasis_timer_start()` and `oasis_timer_stop()` and is the accumulated time over the entire run. Here is an overview of the meaning of the different timers as implemented by default. [1]

- 'total' : total time of the simulation, implemented in `mod_oasis_method` (i.e. between the end of `oasis_init_comp` and the `mpi_finalize` in routine `oasis_terminate`).

[1] Many other measures can be obtained by defining the logical `local_timers_on` as `.true.` in different routines or by implementing other timers. Of course, OASIS3_MCT and the model code then have to be recompiled.

- 'init_thru_enddef' : time between the end of `oasis_init_comp` and the end of `oasis_enddef`, implemented in `mod_oasis_method`.
- 'part_definition' : time spent in routine `oasis_def_partition`.
- 'oasis_enddef' : time spent in routine `oasis_enddef`; this routine performs basically all the important steps to initialize the coupling exchanges, e.g. the internal management of the partition and variable definition, the definition of the patterns of communication between the source and target processes, the reading of the remapping weight-and-address file and the initialisation of the sparse matrix vector multiplication.
- 'grcv_00x' : time spent in the reception of field x in `mct_recv` (including communication and possible waiting time linked to unbalance of components).
- 'wout_00x' : time spent in the I/O for field x in routine `oasis_advance_run`.
- 'gcpy_00x' : time spent in routine `oasis_advance_run` in copying the field x just received in a local array.
- 'pcpy_00x' : time spent in routine `oasis_advance_run` in copying the local field x in the array to send (i.e. with local transformation besides division for averaging).
- 'pavg_00x' : time spent in routine `oasis_advance_run` to calculate the average of field x (if done).
- 'pmap_00x'/'gmap_00x' : time spent in routine `oasis_advance_run` for the matrix vector multiplication for field x on the source/target processes.
- 'psnd_00x' : time spent in routine `oasis_advance_run` for sending field x (i.e. including call to `mct_waitsend` and `mct_isend`).
- 'wtrn_00x' : time spent in routine `oasis_advance_run` to write fields associated with non-instant loctrans operations to restart files (see section 5.2 for details).
- 'wrst_00x' : time spent in routine `oasis_advance_run` to write fields to restart files (see section 5.2 for details).

6.5 Load balancing analysis of coupled model components

An efficient use of the allocated computing resources in a coupled system requires the harmonisation of the component execution speed. This operation, called load balancing, is often neglected, either because of the apparent resource abundance or practical difficulties. To facilitate this work, a load balancing analysis functionality is included in OASIS3-MCT and can be activated by setting to 1 the third number under `$NLOGPRT` in the *namcouple* configuration file (see section 3.2). Some details on this functionality are provided here and more information can be found in the `balancing_documentation.pdf` file in `oasis3-mct/util/load_balancing` directory.

When activated, the load balancing analysis functionality outputs the full timeline of all OASIS3-MCT related events, for any of the allocated resources. This timeline is saved in one NetCDF file per coupled component, `timeline_XXX_component.nc` where `XXX` is the component name. It provides the comprehensive sequence of all operations related to the coupling (field send and receive through MPI, field output on disk, field interpolation and mapping, field reading on disk, restart writing, initialisation and termination phase of the OASIS3-MCT setup) so that any simulation slow down in link with the use of the OASIS3-MCT library can be identified.

The analysis of the coupling field exchanges, amongst all coupling events, allows to not only identify the waste of resources by components which are recurrently waiting for their coupling fields but it also reveals other bottlenecks such as disk access or model internal load imbalance. The full picture of these events makes possible an optimal load balancing, even for the most complex configurations.

In addition to the detailed timeline saved in the NetCDF file, more general computing information (simulation time, speed, waiting time, etc.) is also provided in a text file `load_balancing_info.txt` for

the coupled model and for each component. In simple cases, this global information can help to allocate resources in a balanced way.

Appendix A

The grid types for the transformations

As described in section 4 for the different SCRIP remappings, OASIS3-MCT support different types of grids. The characteristics of these grids are detailed here:

- `'LR' grid`: The longitudes and the latitudes of 2D Logically-Rectangular (LR) grid points can be described by two arrays `longitude(i,j)` and `latitude(i,j)`, where i and j are respectively the first and second index dimensions. Streched or/and rotated grids are LR grids. Note that previous OASIS3 A, B, G, L, Y, or Z grids are all particular cases of LR grids.
- `'U' grid`: Unstructured (U) grids do have any particular structure. The longitudes and the latitudes of 2D Unstructured grid points must be described by two arrays `longitude(nbr_pts,1)` and `latitude(nbr_pts,1)`, where nbr_pts is the total grid size.
- `'D' grid`: The Gaussian Reduced (D) grid is composed of a certain number of latitude circles, each one being divided into a varying number of longitudinal segments. In OASIS3-MCT, the grid data (longitudes, latitudes, etc.) must be described by arrays dimensioned `(nbr_pts,1)`, where `nbr_pts` is the total number of grid points. There is no overlap of the grid, and no grid point at the equator nor at the poles. There are grid points on the Greenwich meridian.

Appendix B

Changes between the different versions of OASIS3-MCT

The evolution between the different versions of OASIS3-MCT can be followed in real-time by registering on the Redmine project management site at https://inle.cerfacs.fr/ (see "Register" at the right top of the page). On this site, registered users can browse the sources and consult tickets describing bug fixes and developments. To follow day to day evolution of the OASIS3-MCT sources, it is also possible to have your e-mail added to the list of addresses to which the log files of the SVN checkins are automatically sent; contact oasishelp@cerfacs.fr if you wish to be added to that list.

B.1 Changes between OASIS3-MCT_5.0 and OASIS3-MCT_4.0

The last version of the coupler, OASIS3-MCT_5.0, comes with the following novelties:

- Python, C and C++ bindings, see sections 2.3, 2.4 and 6.3.4
- A new load balancing analysis tool, see section 6.5
- An environment to use either SCRIP, ESMF or XIOS to generate regridding weights and to analyse the quality of the regridding, see directory `examples/regrid_environment` and section 6.3.3
- A new locally conservative remapping, to be used in particular for runoffs, see `LOCCUNIF`, `LOCCDIST` and `LOCCGAUS` in section 4.3
- Extension of `BLASNEW` operation to support combination of coupling fields, see section 4.4
- Improved and additional diagnostics in `CHECKIN` and `CHECKOUT`, see sections 4.2 and 4.4
- `SCRIPR/CONSERV` option for normalisation by the true area of the grid cells, see `DESTARTR`, `FRACARTR` or `FRACNNTR` in section 4.3
- New `GSSPOS` and `BSSPOS` options for global CONSERV (as `GLBPOS` and `BASPOS` options respectively except that the multiplicative term is computed separately for positive and negative values of the field), see section 4.4
- Extension of `oasis_get_intracomm` to support multiple components, see `oasis_get_multi_intracomm` in section 2.2.9
- Communication/exchange of simple scalars, see section 2.5.2
- Update examples in `oasis3-mct/examples` directory, see section 6.3
- Update to MCT 2.11
- Update of compiling environment, see 6.1
- Migration from SVN to GIT for source management
- Migration of the OASIS3-MCT web site from DKRZ (content manager Plone) to Cerfacs (content manager Wordpress), see the new site at `https://oasis.cerfacs.fr/en/`

- For DISTWGT, DISTWGTNF, GAUSWGT, GAUSWGTNF LOCCUNIF, LOCCDIST or LOCCGAUS, inclusion of the number of neighbours used in the remapping file name
- For SCRIPR/CONSERV, specification through the *namcouple* of North thresh $NTHRESH and South thresh $STHRESH values, above which a Lambert projection is activated
- New $NCDFTYP NetCDF file format setting through the *namcouple*, see section 3.2
- Overload oasis_def_var interface to support excluding the argument id_var_shape from the argument list
- Systematic tests of NetCDF returned error code
- Update oasis_abort to also write to unit 0 (stderr)
- Bugfixes

  1. Component name argument in oasis_get_intercomm (ticket #2776)
  2. BICUBIC/BILINEAR sequence in weights generation (ticket # 2725)
  3. Array bounds check for cplfind (ticket # 2655)
  4. Fix error in maxloops/kfac computation in oasis_mpi_reducelists (ticket # 2654)
  5. Exact calculation of local distance in GAUSWGT interpolation (ticket #2500)
  6. Compatibility of load balancing analysis tool with CRAY compiler (ticket # 2473)
  7. Problem with SCRIPR/BILINEAR periodicity (ticket # 2419)
  8. Problem in m_MCTWorld.F90 in the routine initm (ticket # 1321)
  9. Argument id_var_nodim defined as IN in mod_oasis_var.F90 to compile with NEMO_4.0 and NEMO trunk (git commit 28f4fe59)
  10. In SCRIPR/CONSERV , condition on coincidence of segments is reinitialised to false for each cell (bugfix "lcoinc" ; see [Valcke and Piacentini 2019])

B.2 Changes between OASIS3-MCT_4.0 and OASIS3-MCT_3.0

Different developments were realised to improve the parallel performance of OASIS3-MCT_4.0. These developments are detailed in [Valcke et al 2018] .

- A new communication method, using the remapping weights to define the intermediate mapping decomposition, offers a significant gain at run time, especially for high-resolution cases running on a high number of tasks, thanks to reduced communication. However, as expected, the new method takes longer to initialize, partly due to the fact that the mapping weight file has to be read twice but also due to the extra cost for the initialization of the different MCT routers. That initialization cost is largely mitigated by an upgrade to MCT 2.10.beta1 which reduces the penalty to few seconds. Generally, it should be worth the extra initial cost to speed up the run time. See $NMAPDEC in section 3.2.
- The hybrid MPI+OpenMP parallelisation of the SCRIP library (previously fully sequential) leads to great improvement in the calculation of the remapping weights. The results obtained here show a reduction in the weight calculation time of 2 to 3 orders of magnitude with the new parallel SCRIP library for high-resolution grids. Details are available in [Piacentini et al 2018]. The test_interpolation environment (see section **??**) gives a practical example on how to use OASIS3-MCT_4.0 to pre-calculate (i.e. in a separate job prior to the "real" simulation) the remapping weight and address file.

  Thanks to some preliminary work, few bugs were fixed, in particular in the bounding box definition of the grid cells. This solves an important bug observed in the Pacific near the equator for the bilinear and bicubic interpolations for Cartesian grids.

However, given these modifications, one cannot expect to get exactly the same results for the interpolation weight-and-adress remapping files with this new parallel SCRIP version as compared to the previous SCRIP version in OASIS3-MCT_3.0. We checked in many different cases that the interpolation error is smaller or of the same order than before. We also observed that the parallelisation does not ensure bit reproducible results when varying the number of processes or threads.

- The new methods introduced in the global CONSERV operation reduce its calculation costs by one order of magnitude while still ensuring an appropriate level of reproducibility. This removes the bottleneck foreseen at high resolution with this important, and in few cases still unavoidable, global operation. These new methods are detailed in section 4.4 .

The other new features offered by OASIS3-MCT_4.0 are the following:

- Support for bundled coupling fields.

  Bundled fields is now supported in the `oasis_put` and `oasis_get` interfaces to allow easier coupling of multi-level or other related fields via a single namcouple coupling definition and a single call to `oasis_put` or `oasis_get`. Further details are provided in sec 2.2.7

- Automatic coupling restart writing

  An optional argument `write_restart` was added to the `oasis_put` routine. This argument is false by default but if it is explicitly set to true in the code, a coupling restart file will be written for that field only for that coupling timestep, saving the data that exists at the time of the call (see section 2.2.7).

- Exact consistency between the number of weights and fields

  Exact consistency is now required between number of weights fields in the coupling restart file and the arrays passed as arguments to the `oasis_put` routine. For example, for a 2nd order conservative remapping (CONSERV SECOND), 3 weights are needed and 3 fields must be provided as arguments: the value of the field, its gradient with respect to the longitude and its gradient with respect to the latitude. For a first order conservative remapping (CONSERV FIRST), only one weight and one field are needed. Using a weight file with 3 weights for a first order conservative remapping is no longer allowed.

- Upgrade in the namcouple configuration file

  The namcouple reading routine was cleaned up including a refactoring of the gotos and continue statements, addition of few reusable routines including an abort routine, removal of some dead code, addition of support for blank lines (which are now considered comments), removal of requirement that keywords start at character 2 on a line, removal of requirement for `$END` in the namcouple, and updates to some error messages.

- Other new functionalities with corresponding new namcouple keywords (see section 3.2)

  - `$NUNITNO`: specifies the minimum and maximum unit numbers to be used for input and output files in the coupling layer.
  - `$NMATXRD`: indicates the method used to read mapping weights, either `orig` or `ceg`. In both methods, the weights are read in chunks by the model master task. With the `orig` option, the weights are then broadcast to all other tasks and each task then saves the weights that will be applied to its grid points. With the `ceg` option, the master task reads the weights and then identifies to which other task each weight should be sent. A series of exchanges are then done with each other task involving just the weights needed by that other task. The `orig` method sends much more data but is more parallel, while the `ceg` method does most of the work on the master task but less data is communicated.
  - `$NWGTOPT`: indicates how to handle bad interpolation weights.
  - `$NNOREST` : if true, OASIS3-MCT will initialise any variable that normally requires a coupling restart file with zeros if that file does not exist.

B.3 Changes between OASIS3-MCT_3.0 and OASIS3-MCT_2.0

The main evolution of OASIS3-MCT_3.0 with respect to OASIS3-MCT_2.0 is the support of coupling exchanges between parallel components deployed in much more diverse configurations than before, for example, within one same executable between components running concurrently on separate sets of tasks or between components running sequentially on overlapping sets of tasks. All details are provided in section 2.1.

This new version also includes:

- memory and performance upgrades
- a new LUCIA tool for load balancing analysis
- new memory tracking tool (gilt)
- improved error checking and error messages
- doxygen documentation
- expanded test cases and testing automation
- testing at high resolution (¿ 1M gridpoints), high processor counts (32k pes), and with large variable counts (¿ 1k coupling fields)
- many bug fixes

B.4 Changes between OASIS3-MCT_2.0 and OASIS3-MCT_1.0

The main changes and bug fixes new in OASIS3-MCT_2.0 are the following:

- Support of `BICUBIC` interpolation, see paragraph `BICUBIC` in section 4.3. If the source grid is not a gaussian reduced grid (D), the gradient in the first dimension, the gradient in the second dimension, and the cross-gradient of the coupling field must be calculated by the model and given as arguments to `oasis_put`, as explained in section 2.2.7. If the source grid is a gaussian reduced grid (D), OASIS3-MCT_2.0 can calculate the interpolated field using only the values of the source field points.
- Support of `CONSERV/SECOND` remapping, see paragraph `CONSERV/SECOND` in section 4.3.
- Support of components exchanging data on only a subdomain of the global grid: a new optional argument, ig_size was added to oasis_def_partition, that provides the user with the ability to define the total number of grid cells on the grid (see section 2.2.3).
- The variable `TIMER_Debug` controlling the amount of time statistics written out is now an optional argument read in the *namcouple*; see the `NLOGPRT` line in 3.2 and all details about time statistics in section 6.4.2.
- Specific problems in writing out the time statistics when all the processors are not coupling were solved (see Redmine issue #497)
- The problem with restart files when one coupling field is sent to 2 target components was solved (see Redmine ticket #522)
- A memory leak in mod_oasis_getput_interface.F90 was fixed thanks to R. Hill from the MetOffice (see Redmine ticket #437)
- A bug fix was provided to ensure that the nearest neighbour option is activated when the option `FRACNNEI` is defined in the *namcouple* for the conservative interpolation .
- The behaviour of OASIS3-MCT was changed in the case a component model tries to send with `oasis_put` a field declared with a `oasis_def_var` but not defined in the configuration file *namcouple*; this will now lead to an abort. In this case, the field ID returned by the `oasis_def_var` is equal to -1 and the corresponding `oasis_put` should not be called. Conversely, all coupling fields

appearing in the *namcouple* must be defined with a call to `oasis_def_var`; this constraint is imposed to avoid that a typo in the namcouple would lead to coupling exchanges not corresponding to what the user intends to activate.

- OASIS3-MCT developments are now continuously tested and validated on different computers with a test suite under Buildbot, which is a software written in Python to automate compile and test cycles required in software project (see https://inle.cerfacs.fr/projects/oasis3-mct/wiki/Buildbot on the Redmine site).

B.5 Changes between OASIS3-MCT_1.0 and OASIS3.3

B.5.1 General architecture

- OASIS3-MCT is (only) a coupling library

  Much of the underlying implementation of OASIS3 was refactored to leverage the Model Coupling Toolkit (MCT). OASIS3-MCT is a coupling library to be linked to the component models and that carries out the coupling field transformations (e.g. remappings/interpolations) in parallel on either the source or target processes and that performs all communication in parallel directly between the component models; there is no central coupler executable anymore[1].

- `MAPPING` transformation to use a pre-defined mapping file

  With `MAPPING`, OASIS3-MCT has the ability to read a predefined set of weights and addresses (mapping file) specified in the *namcouple* to perform the interpolation/remapping. The user also has the flexibility to choose the location and the parallelization strategy of the remapping with specific `MAPPING` options (see section 4.3).

- Mono-process mapping file generation with the SCRIP library

  But as before, OASIS3-MCT_1.0 can also generate the mapping file using the SCRIP library [Jones 1999]. When this is the case, the mapping file generation is done on one process of the model components; all previous `SCRIPR` remapping schemes available in OASIS3.3 are still supported besides `BICUBIC` and `CONSERV/SECOND`. (Note: these remapping schemes, not available in OASIS3-MCT_1.0 were reactivated in OASIS3-MCT_2.0, see B.4.)

- MPI2 job launching is NOT supported.

  Only MPI1 start mode is allowed. As before with the MPI1 mode, all component models must be started by the user in the job script in a pseudo-MPMD mode; in this case, they will automatically share the same `MPI_COMM_WORLD` communicator and an internal communicator created by OASIS3-MCT needs to be used for internal parallelization (see section 2.2.2).

B.5.2 Changes in the coupling interface in the component models

- Use statement

  The different OASIS3.3 `USE` statements were gathered into one `USE mod_oasis` (or one `USE mod_prism`), therefore much simpler to use.

- Support of previous `prism_xxx` and new `oasis_xxx` interfaces

  OASIS3-MCT retains prior interface names of OASIS3.3 (e.g. `prism_put_proto`) to ensure full backward compatibility. However, new interface names such as `oasis_put` are also available and should be prefered. Both routine names are listed in chapter 2.

[1] As with OASIS3.3, the "put" calls are non-blocking but the "get" calls are blocking. As before, the user has to take care of implementing a coupling algorithm that will result in matching "put" and "get" calls to avoid deadlocks (see section 2.2.7). The lag (LAG) index works as before in OASIS3.3 (see section 2.5.3)

- Auxiliary routines not supported yet

  Auxiliary routines `prism_put_inquire`, `prism_put_restart_proto`, `prism_get_freq` are not supported yet. (Note: `prism_put_inquire` and `prism_get_freqs` were reintroduced in OASIS3-MCT_3.0 and equivalent of `prism_put_restart_proto` in OASIS3-MCT_4.0.)

- Support of components for which only a subset of processes participate in the coupling

  New routines `oasis_create_couplcomm` and `oasis_set_couplcomm` are now available to create or set a coupling communicator in the case only a subset of the component processes participate in the coupling. But even in this case, all OASIS3-MCT interface routines, besides the grid definition (see section 2.2.4) and the "put" and " get" call per se (see section 2.2.7), are collective and must be called by all processes. (Note: this has changed with OASIS3-MCT_3.0.)

- New routines `oasis_get_debug` and `oasis_set_debug`

  New routines `oasis_get_debug` and `oasis_set_debug` are now available to respectively retrieve the current OASIS3-MCT internal debug level (set by `$NLOGPRT` in the *namcouple*) or to change it (see section 2.2.9).

B.5.3 Functionality not offered anymore

- `SCRIPR/BICUBIC` and `SCRIPR/CONSERV/SECOND` remappings

  As in OASIS3.3, the SCRIP library can be used to generate the remapping/interpolation weights and addresses and write them to a mapping file. All previous `SCRIPR` remapping schemes available in OASIS3.3 are still supported in OASIS3-MCT_1.0 besides `BICUBIC` and `CONSERV/SECOND` because these remapping involve at each source grid point the value of the field but also the value of the gradients of the field (which are not known or calculated). (Note: these remapping schemes, not available in OASIS3-MCT_1.0 were reactivated in OASIS3-MCT_2.0, see B.4.)

- Vector field remapping

  Vector field remapping is not and will not be supported (see "Support of vector fields with the SCRIPR remappings" in section 4.3).

- Automatic calculation of grid mesh corners in `SCRIPR/CONSERV`

  For `SCRIPR/CONSERV` remapping, grid mesh corners will not be compute automatically if they are needed but not provided.

- Other transformations not supported

  - The following transformations are not available in OASIS3.3 and will most probably not be implemented as it should be not too difficult to implement the equivalent operations in the component models themselves: `CORRECT`, `FILLING`, `SUBGRID`, `MASKP`
  - `LOCTRANS/ONCE` is not explicitely offered as it is equivalent to defining a coupling period equal to the total runtime.
  - The following transformations are not available as they were already deprecated in OASIS3.3 : `REDGLO`, `INVERT`, `REVERSE`, `GLORED`
  - `MASK` and `EXTRAP` are not available but the corresponding linear extrapolation can be replaced by the more efficient option using the nearest non-masked source neighbour for target points having their original neighbours all masked. This is now the default option for `SCRIPR/DISTWGT`, `GAUSWGT` and `BILINEAR` interpolations. It is also included in `SCRIPR/CONSERV` if `FRACNNEI` normalization option is chosen (see section 4.3).
  - `INTERP` interpolations are not available; `SCRIPR` should be used instead.
  - `MOZAIC` is not available as `MAPPING` should be used instead.
  - `NOINTERP` does not need to be specified anymore if no interpolation is required.

  - Field combination with BLASOLD and BLASNEW; these transformations only support multiplication and addition terms to the fields (see section 4.2).

- Using the coupler in interpolator-only mode

  This is not possible anymore as OASIS3-MCT is now only a coupling library. However, it is planned, in a further release, to provide a toy coupled model that could be use to check the quality of the remapping for any specific couple of grids. (Note: this was done in OASIS3-MCT_2.0.)

- Coupling field CF standard names

  The file cf_name_table.txt is not needed or used anymore. The CF standard names of the coupling fields are not written to the debug files.

- Binary auxiliay files

  All auxiliary files, besides the *namcouple* must be NetCDF; binary files are not supported anymore.

B.5.4 New functionality offered

- Better support of components for which only a subset of processes participate in the coupling

  In OASIS3.3, components for which only a subset of processes participated in the coupling were supported in a very restricted way. In fact, this subset had to be composed of the N first processes and N had to be specified in the *namcouple*. Now, the subset of processes can be composed of any of the component processes and does not have to be pre-defined in the *namcouple*. New routines oasis_create_couplcomm and oasis_set_couplcomm are now available to create or set a coupling communicator gathering only these processes (see section 2.2.2). (Note: this was further improved in OASIS3-MCT_3.0.)

- Exact restart for LOCTRANS transformations

  If needed, LOCTRANS transformations write partially transformed fields in the coupling restart file at the end of a run to ensure an exact restart of the next run (see section 4.1). For that reason, coupling restart filenames are now required for all *namcouple* entries that use LOCTRANS (with non INSTANT values). This is the reason an optional restart file is now provided on the OUTPUT *namcouple* input line. If the coupling periods of two (or more) coupling fields are different, it is necessary to define two (or more) restart files, one for each field.

- Support to couple multiple fields via a single communication.

  This is supported through colon delimited field lists in the *namcouple*, for example

  ```
  ATMTAUX:ATMTAUY:ATMHFLUX TAUX:TAUY:HEATFLUX 1 3600 3 rstrt.nc EXPORTED
  ```

  in a single *namcouple* entry. All fields will use the *namcouple* settings for that entry. In the component model codes, these fields are still sent ("put") or received ("get") one at a time. Inside OASIS3-MCT, the fields are stored and a single mapping and send or receive instruction is executed for all fields. This is useful in cases where multiple fields have the same coupling transformations and to reduce communication costs by aggregating multiple fields into a single communication. If a LOCTRAN transformation is needed for these multiple fields, it is necessary to define a restart file for these multiple fields. The coupling fields must be sent and received in the same order as they are defined in the corresponding single entry of the namcouple (not relevant in further versions of OASIS3-MCT).

- Matching one source filed with multiple targets

  A coupling field sent by a source component model can be associated with more than one target field and model (get). In that case, the source model needs to send ("put") the field only once and the corresponding data will arrive at multiple targets as specified in the *namcouple* configuration file. Different coupling frequencies and transformations are allowed for different coupling exchanges of the same field. If coupling restart files are required (either if a LAG or if a LOCTRANS transformation is specified), it is mandatory to specify different files for the different fields.

The inverse feature is not allowed: a single target (get) field CANNOT be associated with multiple source (put) fields.

- The debug files

  The debug mode was greatly improved as compared to OASIS3.3. The level of debug information written out to the OASIS3-MCT debug files for each model process is defined by the $NLOGPRT value in the *namcouple*. All details are provided in section 3.2.

B.5.5 Changes in the configuration file *namcouple*

- The *namcouple* configuration file of OASIS3-MCT is fully backward compatible with OASIS3.3. However, several *namcouple* keywords have been deprecated even if they are still allowed. These keywords are noted "UNUSED" in sections 3.2 and 3.3 and are not fully described. Information below these keywords will not be read and will not be used: $SEQMODE , $CHANNEL, $JOBNAME, $INIDATE, $MODINFO, $CALTYPE.
- Also the following inputs should not appear in the *namcouple* anymore as the related functionality are not supported anymore in OASIS3-MCT (see above): field status AUXILARY, time transformation ONCE, REDGLO, INVERT, MASK, EXTRAP, CORRECT, INTERP, MOZAIC, FILLING, SUBGRID, MASKP, REVERSE, GLORED.
- To get 2D fields in the debug output NetCDF files, the 2D dimensions of the grids must be provided in the *namcouple* (except if the field has the status OUTPUT); otherwise, the fields in the debug output files will be 1D.

B.5.6 Other differences

- IGNORED and IGNOUT fields are converted to EXPORTED and EXPOUT respectively.
- The file `areas.nc` is not needed anymore to calculate some statistics with options CHECKIN and/or CHECKOUT.
- SEQ index is no longer needed to ensure correct coupling sequencing within the coupler. Use of SEQ allows the coupling layer to detect potential deadlocks before they happen and to exit gracefully (see section 2.5.4).
- The I/O library mpp_io is no longer used to write the restart and output files.

Bibliography

[Cassou et al 1998] Cassou, C., P. Noyret, E. Sevault, O. Thual, L. Terray, D. Beaucourt, and M. Imbard: Distributed Ocean-Atmosphere Modelling and Sensitivity to the Coupling Flux Precision: the CATHODe Project. *Monthly Weather Review*, 126, No 4: 1035-1053, 1998.

[Chavas et al 2013] Chavas, J., E. Audit, L. Coquart, and S. Valcke: Conservative Regridding When Grid Cell Edges Are Unknown - Case of SCRIP *Cerfacs Technical Report, TR-CMGC-13-6*, *Technical Report N^o 0001, Maison de la Simulation, 91400 Saclay, France*, CECI, Université de Toulouse, CNRS, Toulouse, France

[Coquart et al 2018] Coquart, L., E. Maisonnave, E. and S. Valcke: Using Open MP in OASIS3-MCT for the N-nearest-neighbor remapping *Technical Report, WN/CMGC/18/19*, CECI, Université de Toulouse, CNRS, Toulouse, France

[Craig et al 2017] Craig, A., S. Valcke and L. Coquart : Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3.0 *Geosci. Model Dev.*, 10: 3297–3308 https://doi.org/10.5194/gmd-10-3297-2017

[Craig et al 2018] Craig, A., S. Valcke: OASIS3-MCT4.0 Timing Study with MCT 2.10.beta1 *Technical Report, WN/CMGC/18/38*, CECI, Université de Toulouse, CNRS, Toulouse, France

[Craig 2019] Craig, A. : GSSPOS and BSSPOS options for the global conservation in OASIS3-MCT *Technical Report, WN/CMGC/19/128*, CECI, Université de Toulouse, CNRS, Toulouse, France https://oasis.cerfacs.fr/wp-content/uploads/sites/114/2021/08/GLOBC_TR_Craig_oasis_map_conserv_092019.pdf

[Guilyardi et al 1995] Guilyardi, E., G. Madec, L. Terray, M. Déqué, M. Pontaud, M. Imbard, D. Stephenson, M.-A. Filiberti, D. Cariolle, P. Delecluse, and O. Thual. Simulation couplée océan-atmosphère de la variabilité du climat. *C.R. Acad. Sci. Paris*, t. 320, série IIa:683–690, 1995.

[He and Ding 2001] He, Y. and C. H. Q Ding. Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Numer- ical Applications. The Journal of Supercomputing, 18, 259 https://doi.org/10.1023/A:1008153532043, 2001.

[Jacob et al 2005] Jacob, R., J. Larson, and E. Ong: MxN Communication and Parallel Interpolation in CCSM3 Using the Model Coupling Toolkit. *Int. J. High Perf. Comp. App.*, 19(3), 293-307 2005

[Jones 1999] Jones, P.: Conservative remapping: First- and second-order conservative remapping. Mon Weather Rev, 127, 2204-2210, 1999.

[Jonville and Valcke 2019] Jonville, G. and S. Valcke: Analysis of SCRIP conservative remapping in OASIS3-MCT – Part B *Technical Report, TR/CMGC/19/155*, CERFACS, Toulouse, France, 2020. https://oasis.cerfacs.fr/wp-content/uploads/sites/114/2021/08/GLOBC_TR_Jonville-SCRIP_CONSER_TRNORM_partB_2019.pdf

[Larson et al 2005] Larson, J., R. Jacob, and E. Ong: The Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models. *Int. J. High Perf. Comp. App.*, 19(3), 277-292, 2005

[Maisonnave 2020] Maisonnave, E. Locally conservative OASIS interpolation using target grid nearest neighbours *Technical Report, TR/CMGC/20/12*, CERFACS, Toulouse, France,

2020. https://oasis.cerfacs.fr/wp-content/uploads/sites/114/2021/08/GLOBC_TR_Maisonnave-locally_conserv_interpolation_2020.pdf

[Maisonnave et al 2020] Maisonnave, E., L. Coquart, and A. Piacentini: A better diagnostic of the load imbalance in OASIS based coupled systems *Technical Report, TR/CMGC/20/176*, CERFACS, Toulouse, France, 2020. https://oasis.cerfacs.fr/wp-content/uploads/sites/114/2021/08/GLOBC_TR_Maisonnave-load-balancing_2020.pdf

[Mirin and Worley 2012] Mirin, A. A., and P. H. Worley : Improving the Per- formance Scalability of the Community Atmosphere Model. *Int. J. High Perf. Comp. App.*, 26, 17–30 https://doi.org/10.1177/1094342011412630, 2012.

[Noyret et al 1994] Noyret, P., E. Sevault, L. Terray and O. Thual. Ocean-atmosphere coupling. *Proceedings of the Fall Cray User Group (CUG) meeting*, 1994.

[Piacentini et al 2018] Piacentini, A., E. Maisonnave, G. Jonville, L. Coquart and S. Valcke: A parallel SCRIP interpolation library for OASIS, *Technical Report, WN/CMGC/18/34*, CERFACS, Toulouse, France, 2018. https://oasis.cerfacs.fr/wp-content/uploads/sites/114/2021/08/GLOBC_WN_Piacentini_Parallel_SCRIP_cmgc_18_34_2018.pdf

[Piacentini and Maisonnave 2020] Piacentini, A. and E. Maisonnave: Interactive visualisation of OASIS coupled models load imbalance *Technical Report, TR/CMGC/20/177*, CERFACS, Toulouse, France, 2020. https://oasis.cerfacs.fr/wp-content/uploads/sites/114/2021/08/GLOBC_TR_Piacentini_Interactive_visualisation_of_OASIS_2020.pdf

[Pontaud et al 1995] Pontaud, M., L. Terray, E. Guilyardi, E. Sevault, D. B. Stephenson, and O. Thual. Coupled ocean-atmosphere modelling - computing and scientific aspects. In *2nd UNAM-CRAY supercomputing conference, Numerical simulations in the environmental and earth sciences* Mexico-city, Mexico, 1995.

[Sevault et al 1995] Sevault, E., P. Noyret, and L. Terray. Clim 1.2 user guide and reference manual. *Technical Report TR/CGMC/95-47*, CERFACS, 1995.

[Terray and Thual 1995b] Terray, L. and O. Thual. Oasis: le couplage océan-atmosphère. *La Météorologie*, 10:50–61, 1995.

[Terray and Thual 1993] Terray, L. and O. Thual. Coupled ocean-atmosphere simulations. In *High Performance Computing in the Geosciences, proceedings of the Les Houches Workshop* F.X. Le Dimet Ed., Kluwer Academic Publishers B.V, 1993.

[Terray et al 1995] Terray, L., E. Sevault, E. Guilyardi and O. Thual OASIS 2.0 Ocean Atmosphere Sea Ice Soil User's Guide and Reference Manual *Technical Report TR/CGMC/95-46*, CERFACS, 1995.

[Terray et al 1995b] Terray, L. O. Thual, S. Belamari, M. Déqué, P. Dandin, C. Lévy, and P. Delecluse. Climatology and interannual variability simulated by the arpege-opa model. *Climate Dynamics*, 11:487–505, 1995

[Terray et al 1999] Terray, L., S. Valcke and A. Piacentini: OASIS 2.3 Ocean Atmosphere Sea Ice Soil, User's Guide and Reference Manual, *Technical Report TR/CMGC/99-37*, CERFACS, Toulouse, France, 1999.

[Valcke et al 2021] Valcke, S., Piacentini, A. and Jonville, G. Benchmarking of regridding libraries used in Earth System Modelling: SCRIP, YAC, ESMF and XIOS *Technical Report, WN/CMGC/21/145*, CERFACS, Toulouse, France, 2021. https://oasis.cerfacs.fr/wp-content/uploads/sites/114/2021/11/GLOBC-TR_Valcke_Report_regridding_analysis_final_2021.pdf

[Valcke and Piacentini 2019] Valcke, S. and A. Piacentini Analysis of SCRIP conservative remapping in OASIS3-MCT *Technical Report, WN/CMGC/19/129*, CERFACS, Toulouse, France, 2019. https://oasis.cerfacs.fr/wp-content/uploads/sites/114/2021/08/GLOBC_TR_Valcke-SCRIP_CONSERV_TRNORM_partA_2019.pdf

[Valcke et al 2018] Valcke, S., L. Coquart, A. Craig, G. Jonville, E. Maisonnave, A. Piacentini Multithreaded or thread safe OASIS version including performance optimizations to adapt to many-

core architectures, IS-ENES2 deliverable D2.3 *Technical Report, WN/CMGC/18/74*, CERFACS, Toulouse, France, 2018.

[Valcke et al 2017] Valcke, S., G. Jonville, R. Ford, M. Hobson, A. Porter and G. Riley Report on benchmark suite for evaluation of coupling strategies *Technical Report, TR/CMGC/17/87*, CERFACS, Toulouse, France, 2018. http://cerfacs.fr/wp-content/uploads/2017/05/GLOBC-TR-IS-ENES2_D10.3_MAI2017.pdf

[Valcke et al 2015] Valcke, S., T. Craig, L. Coquart: OASIS3-MCT User Guide, OASIS3-MCT 3.0, *Technical Report, WN/CMGC/15/38*, CERFACS, Toulouse, France, 2010.

[Valcke 2013] Valcke, S.: The OASIS3 coupler: a European climate modelling community software *Geosci. Model Dev.*, 6:373–388

[Valcke et al 2013] Valcke, S., T. Craig, L. Coquart: OASIS3-MCT User Guide, OASIS3-MCT 2.0, *Technical Report, WN/CMGC/13/17*, CERFACS, Toulouse, France, 2013.

[Valcke et al 2012] Valcke, S., T. Craig, L. Coquart: OASIS3-MCT User Guide, OASIS3-MCT 1.0, *Technical Report, WN/CMGC/12/49*, CERFACS, Toulouse, France, 2012.

[Valcke et al 2011] Valcke, S., M. Hanke, L. Coquart: OASIS4_1 User Guide *Technical Report TR/CMGC/11/50*, CERFACS, Toulouse, France, 2011.

[Valcke 2006b] Valcke, S.: OASIS3 User Guide (prism_2-5) *Technical Report TR/CMGC/06/73*, CERFACS, Toulouse, France, 2006.

[Valcke 2006a] Valcke, S.: OASIS4 User Guide (OASIS4_0_2), *Technical Report TR/CMGC/06/74*, CERFACS, Toulouse, France, 2006.

[Valcke et al 2004] Valcke, S., A. Caubel, R. Vogelsang, and D. Declat: OASIS3 User's Guide (oasis3_prism_2-4), *PRISM Report No 2, 5th Ed.*, CERFACS, Toulouse, France, 2004.

[Valcke et al 2003] Valcke, S., A. Caubel, D. Declat and L. Terray: OASIS3 Ocean Atmosphere Sea Ice Soil User's Guide, *Technical Report TR/CMGC/03-69*, CERFACS, Toulouse, France, 2003.

[Valcke et al 2000] Valcke, S., L. Terray and A. Piacentini: OASIS 2.4 Ocean Atmosphere Sea Ice Soil, User's Guide and Reference Manual, *Technical Report TR/CMGC/00-10*, CERFACS, Toulouse, France, 2000.

[Voldoire 2020] Voldoire, A. River to ocean models interpolation *Research Report*, CNRM, Université de Toulouse, Météo-France, CNRS, Toulouse, France, 2020 https://hal-meteofrance.archives-ouvertes.fr/meteo-02986574/file/interpolation_runoffs_en.pdf