OASIS3 User Guide

oasis3-mct-1.0

Edited by:
S. Valcke, CERFACS/CNRS URA No1875

CERFACS TR/CMGC/XX/XX

July 2012

Copyright Notice

How to get assistance?

Assistance can be obtained as listed below.

Phone Numbers and Electronic Mail Adresses

Name	Phone	Affiliation	e-mail
Sophie Valcke	+33-5-61-19-30-76	CERFACS	oasishelp(at)cerfacs.fr

How to get documentation ?

The documentation can be downloaded from the OASIS web site under the URL :
https://verc.enes.org/models/software-tools/oasis

Contents

Chapter 1

Acknowledgments

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

Arnaud Caubel (FECIT/Fujitsu)

Damien Declat (CERFACS)

Italo Epicoco (CMCC)

Veronika Gayler (MPI-M&D)

Josefine Ghattas (CERFACS)

Jean Latour (Fujitsu-Fecit)

Eric Maisonnave (CERFACS)

Silvia Mocavero (CMCC)

Elodie Rapaport (CERFACS)

Hubert Ritzdorf (CCRLE-NEC)

Sami Saarinen (ECMWF)

Eric Sevault (Météo-France)

Laurent Terray (CERFACS)

Olivier Thual (CERFACS)

Sophie Valcke (CERFACS)

Reiner Vogelsang (SGI Germany)

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)

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)
Marie-Alice Foujols (IPSL)
Gilles Garric (CERFACS)
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)
Olivier Marti (IPSL/LSCE)
Claude Mercier (IDRIS)
Pascale Noyret (EDF)
Andrea Piacentini (CERFACS)
Marc Pontaud (Météo-France)
Adam Ralph (ICHEC)
René Redler (MPI-M)
Tim Stockdale (ECMWF)
Rowan Sutton (UGAMP)
Véronique Taverne (CERFACS)
Jean-Christophe Thil (UKMO)
Nils Wedi (ECMWF)

Chapter 2

Introduction

In 1991, CERFACS decided to tackle coupled climate modelling and to develop a software interface to couple existing numerical General Circulation Models of the ocean and of the atmosphere. Today, the OASIS3 coupler, which is the result of more than 20 years of evolution is used by about 30 modelling groups in Europe, Australia, Asia and North America, on the different computing platforms used by the climate modelling community. The list of coupled models realized with OASIS3 and previous versions and the platforms onto which they were run on in the few past years can be found in Appendix C.

OASIS3 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 presently reinforced with additinal resources coming from IS-ENES project funded by the EU (FP7 - GA no 228203).

OASIS3 is a portable set of Fortran 77, Fortran 90 and C routines. Portability and flexibility are OASIS3 key design concepts. At run-time, OASIS3 acts as a coupling layer, which main function is to interpolate the coupling fields exchanged between the component models, and as a communication library linking component models, via the OASIS3 PRISM Model Interface Library (PSMILe). OASIS3 supports coupling of general two dimensional fields. Unstructured grids are also supported using a one dimension degeneration of the two dimensional structures. To communicate directly with another model, or to perform I/O actions, a component model needs to include few specific PSMILe calls. OASIS3 PSMILe supports parallel communication between parallel component models as well as parallel interpolation via Message Passing Interface (MPI) and file I/O using netcdf.

This version has been significantly refactored. There is no longer a separate OASIS coupler component. All coupling is now direct and all transforms are executed in parallel on a set of component processes. This version of OASIS3 leverages the Argonne National Laboratory's Model Coupling Toolkit (MCT) in the coupling layer. In spite of the significant changes in underlying implementation, usage has largely remained unchanged. The prism fortran interfaces used in component models are unchanged. The use statement has been updated and now requires a single "use mod_prism" statement instead of the various use statements required in prior OASIS3 versions. The *namcouple* spec is also largely unchanged relative to OASIS3, although several options are either not used or not supported. There are a few new keywords in *namcouple* including `MAPPING` which allows a user to specify a mapping file generated externally. Some features like vector mapping and second order mapping have been delayed in implementation while other new features like parallel mapping have been added. And currently, only MPI1 job launching is supported.

See appendix B for a more detailed list of changes in this version.

2.1 Step-by-step use of OASIS3

To use OASIS3 for coupling models (and/or perform I/O actions), one has to follow these steps:

1. Obtain OASIS3 source code (see chapter 3).

2. Identify the coupling or I/O fields and adapt the component models to allow their exchange with the PSMILe library based on MPI1 message passing. The PSMILe library uses NetCDF and therefore can be used to perform I/O actions from/to disk files. For more detail on how to interface a model with the PSMILe, see chapter 4.

   The tutorial coupled model gives a practical example of a coupled model; the sources are given in directories `examples/tutorial` ; more detail on the tutorial and how to compile and run it can be found in chapter 8.

3. Define all coupling and I/O parameters and the transformations required to adapt each coupling field from its source model grid to its target model grid; on this basis, prepare OASIS3 configuring file *namcouple* (See chapter 5).

   OASIS3 supports different interpolation algorithms as is described in chapter 6. Regridding files can be compute online using the SCRIP options or offline and read using the MAPPING transformation.

4. Generate required auxiliary data files (see chapter 7).
5. Compile OASIS3, the component models and start the coupled experiment. Chapter 8 describes how to compile and run OASIS3 and the TOYOASIS3 coupled model.

If you need extra help, do not hesitate to contact us (see contact details on the back of the cover page).

Chapter 3

OASIS3 sources, license and Copyright

3.1 OASIS3 sources

OASIS3 sources, related libraries, and TOYOASIS3 coupled model sources and data are available from CERFACS SVN server. To obtain more detail on how to download the sources, please contact us (see contact details on the back of the cover page).

OASIS3 directory structure is the following one:

```
- oasis3/lib/psmile      PRISM System Model Interface Library
            /scrip       SCRIPR interpolation library
            /mct         Model Coupling Toolkit Coupling Software

- oasis3/doc             OASIS3 documentation

- oasis3/util/make_dir   Utilities to compile OASIS3 (see section 8.1)

- oasis3/examples/tutorial  environment to run the tutorial toymodel
                              (see section 8.3)
```

3.2 License and Copyright

3.2.1 OASIS3 license and copyright statement

of the OASIS3 software are not responsible for any damage that may result from correct or incorrect use of this software.

3.2.2 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 4

Interfacing a model with the OASIS3-MCT library

At run-time, the OASIS3-MCT coupling layer supports coupling data between two components as well as interpolation and transformation of coupling fields. Different communication techniques have been historically developed in OASIS. With OASIS3-MCT, communication is performed by the Model Couling Toolkit library (MCT, see http://www.mcs.anl.gov/research/projects/mct/) based on message passing (the keyword $CHANNEL in the configuration file *namcouple* (see chapter 5 has to be `MPI1`).

For a practical test case using the OASIS3-MCT library, see the sources in `examples/tutorial` and more details in chapter 8.

To communicate with another component model or to perform I/O actions, a component model needs to be interfaced with the OASIS3-MCT library, which sources can be found in `oasis3-mct/lib/psmile` directory. The OASIS3-MCT library supports:

- parallel communication between parallel component models,
- an ability to couple a component on a subset of it's processeses only,
- automatic sending and receiving actions at appropriate times following user's choice indicated in the *namcouple*,
- time integration or accumulation of the coupling fields,
- some transformations such as mapping (interpolation) between grids,
- I/O actions from/to files.

To adapt a component model to OASIS3-MCT, specific calls of the following classes have to be implemented in the code:

1. Initialisation (section 4.2)
2. Grid data file definition (section 4.3)
3. Partition definition (section 4.4)
4. I/O-coupling field declaration (section 4.5)
5. End of definition phase (section 4.6)
6. I/O-coupling field sending and receiving (section 4.7)
7. Termination (section 4.8)
8. Optional auxiliary routines (section 4.9)

Finally, in section 4.10, different coupling algorithms are illustrated and details on how to reproduce them with OASIS3-MCT are provided. More information on the `LAG` and `SEQ` indices are also given in that section.

4.1 Use

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

- `USE mod_prism`

  ** OR **
- `USE mod_oasis`

Both use statements are valid and use of just one or the other is recommended in a particular component model. A single use statement now provides all the methods that required multiple use statements in previous OASIS3 versions. The methods, datatypes, and capabilities are identical for both the `mod_prism` or `mod_oasis` interfaces. The only difference is 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 now recommended 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 is defined and a single description of the interface arguments is provided.

4.2 Initialisation

4.2.1 Coupling initialisation

- `CALL oasis_init_comp (compid, model_name, ierror)`
- `CALL prism_init_comp_proto (compid, model_name, ierror)`
  - `compid [INTEGER; OUT]`: component model ID
  - `model_name [CHARACTER*6; IN]`: component model name (as in *namcouple*)
  - `ierror [INTEGER; OUT]`: returned error code.

  This routine must called by all component processes to initialise the coupling.[1]

4.2.2 Communicator for internal parallelisation

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

  If needed, this routine may be called by the component processes to get the value of a local communicator to be used by the component for its internal parallelisation.

  This may be needed as all component models started in a pseudo-MPMD mode with MPI1 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 based on the name given to `oasis_init_comp/prism_init_comp_proto` routine; its value is returned as the first argument of the routine, `local_comm`.

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

4.2.3 Coupling through a subset of the component model processes

If only a subset of the component processes participate in the coupling (e.g. a component is setup to run on 80 processes but 16 of those processes are associated with a distinct task, like I/O), a communicator gathering only these processes must be defined, with either `oasis/prism_create_couplcomm` or `oasis/prism_set_couplcomm`.

If such communicator does not exist yet in the code, the component processes should use, to create it:

- `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 must be called by all component processes with `icpl=1` for processes participating in the coupling and with `icpl=MPI_UNDEFINED` for the others. Argument `local_comm` is the MPI communicator associated with all processes of the component. The new coupling communicator is returned in `coupl_comm` argument and the internal coupling communicator is also set to that value.

If this communicator already exist in the code, the component should use, to provide it to OASIS3-MCT:

- `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 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.

These routines should be called after the `oasis_init_comp/prism_init_comp_proto` call but before the grid, partition, or coupling field declaration. All OASIS3-MCT interface routine, besides the grid definition (see section 4.3) and the "puts" and " gets" per se (see section 4.7), are collective and must be called by all processes[2]. In particular, the coupling field partition must be described across the coupling processes only even though the `oasis_def_partition/prism_def_partition_proto` must be called on all processes (but with `ig_paral(:)=0` for the processes not involved in the coupling.)

Here is a coding sample of how to use these routines:

```
CALL oasis_init_comp (comp_id, comp_name, ierror )
CALL oasis_get_localcomm ( localComm, ierror )

!--- create communicator gathering coupling processes (every other)
CALL MPI_Comm_Rank ( localComm, mype, ierror )
couplingpe = .false.
if (mod(mype,2) == 0) couplingpe = .true.
icpl = MPI_UNDEFINED
if (couplingpe) icpl = 1
```

[2] In fact, the `oasis_def_var/prism_def_var_proto` must be called by at least the root process of the coupler communicator but can be called from all processes.

```
CALL MPI_COMM_Split(localComm,icpl,1,couplComm,ierror)
!
!--- provide this communicator to OASIS3-MCT
CALL oasis_set_couplcomm(couplComm, ierror)

! The call to MPI_COMM_Split and oasis_set_couplcomm could be replaced by
! CALL oasis_create_couplcomm(icpl,localComm,couplComm,ierror)

CALL oasis_def_partition ( ... )
CALL oasis_def_var ( ... )
CALL oasis_enddef ( ... )

!--- do loop
   ! ...
   if (couplingpe) CALL oasis_put( ... )
   ! ...
   if (couplingpe) CALL oasis_get( ... )
   ! ...
!--- enddo

CALL oasis_terminate ( ... )
```

4.3 Grid data file definition

The grid data files *grids.nc, masks.nc* and *areas.nc* can be created by the user before the run or can be written directly at run time by the master process of each component model. Fields can be added to the grid files by the model but fields on the grid file are NEVER overwritten.

This section describes the PSMILe routines to be called by the master process of each component model to write, at run time, grid information to the grid data files. These routines have to be called just after `prism_init_comp`.

As an example, see the tutorial test case use these routines to write the grid data files.

- `CALL prism_start_grids_writing (flag)`
- `CALL oasis_start_grids_writing (flag)`
  - `flag [INTEGER; OUT]`: UNUSED
- `CALL prism_write_grid (cgrid, nx, ny, lon, lat)`
- `CALL oasis_write_grid (cgrid, nx, ny, lon, lat)`
  - `cgrid [CHARACTER*4; IN]`: grid name prefix (see 5.3)
  - `nx [INTEGER; IN]` : first grid dimension (x)
  - `ny [INTEGER; IN]` : second grid dimension (y)
  - `lon [REAL, DIMENSION(nx,ny); IN)` : array of longitudes (degrees East)
  - `lat [REAL, DIMENSION(nx,ny); IN)` : array of latitudes (degrees North)

  Writing of the model 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 OASIS (which is essential to have a correct conservative remapping `SCRIPR/CONSERV`, see section 6.3).

- `CALL prism_write_corner (cgrid, nx, ny, nc, clon, clat)`
- `CALL oasis_write_corner (cgrid, nx, ny, nc, clon, clat)`
  - `cgrid [CHARACTER*4; IN]`: grid name prefix
  - `nx [INTEGER; IN]` : first grid dimension (x)
  - `ny [INTEGER; IN]` : second grid dimension (y)
  - `nc [INTEGER; IN]` : number of corners per grid cell (always 4 in the version)
  - `lon [REAL, DIMENSION (nx,ny,nc);IN]` : array of corner longitudes (in degrees_East)
  - `lat [REAL, DIMENSION (nx,ny,nc);IN]` : array of corner latitudes (in degrees_North)

  Writing of 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 some transformations (see section 7.2). If called, prism_write_corners needs to be called after prism_write_grids.
- `CALL prism_write_angle (cgrid, nx, ny, angle)`
- `CALL oasis_write_angle (cgrid, nx, ny, angle)`
  - `cgrid [CHARACTER*4; IN]`: grid name prefix
  - `nx [INTEGER; IN]` : first grid dimension (x)
  - `ny [INTEGER; IN]` : second grid dimension (y)
  - `angle [REAL, DIMENSION (nx,ny);IN]` : array of angles

  Writing of the grid angles; needed only if coupling fields are vector fields defined on a grid which has a local coordinate system not oriented in the zonal and meridional directions. The angle is defined as the angle between the vector first component and the zonal direction. See SCRIPR/CONSERV in section 6.2.

  If called, prism_write_angle needs to be called after prism_write_grids.
- `CALL prism_write_mask (cgrid, nx, ny, mask)`
- `CALL oasis_write_mask (cgrid, nx, ny, mask)`
  - `cgrid [CHARACTER*4; IN]`: grid name prefix
  - `nx [INTEGER; IN]` : first grid dimension (x)
  - `ny [INTEGER; IN]` : second grid dimension (y)
  - `mask [INTEGER, DIMENSION(nx,ny) ;IN]` : mask array (0 - not masked, 1 - masked)

  Writing of the model grid mask.
- `CALL prism_write_area (cgrid, nx, ny, area)`
- `CALL oasis_write_area (cgrid, nx, ny, area)`
  - `cgrid [CHARACTER*4; IN]`: grid name prefix
  - `nx [INTEGER; IN]` : first grid dimension (x)
  - `ny [INTEGER; IN]` : second grid dimension (y)
  - `area [REAL, DIMENSION(nx,ny); IN]` : array of grid cell areas

  Writing of the model grid cell areas. Writing of areas is optional as area information is needed only for some transformations (see section 7.2).
- `CALL prism_terminate_grids_writing()`
- `CALL oasis_terminate_grids_writing()`

  Termination of grids writing. This call is required.

4.4 Partition definition

When a component of the coupled system is a parallel code, each coupling field is usually scattered among the different processes. With the PSMILe library, each process exchanging coupling data has to define its local partition in the global index space.

- `CALL prism_def_partition_proto (il_part_id, ig_paral, ierror)`
- `CALL oasis_def_partition (il_part_id, ig_paral, ierror)`
  - `il_part_id [INTEGER; OUT]`: partition ID
  - `ig_paral [INTEGER, DIMENSION(:), IN]`: vector of integers describing the local partition in the global index space
  - `ierror [INTEGER; OUT]`: returned error code.

The vector of integers describing the process local partition, `ig_paral`, has a different expression depending on the type of the partition. In OASIS3, 4 types of partition are supported: Serial (no partition), Apple, Box, and Orange.

4.4.1 Serial (no partition)

This is the choice for a monoprocess model. In this case, we have `ig_paral(1:3)`:

- `ig_paral(1)` = 0 (indicates a Serial "partition")
- `ig_paral(2)` = 0
- `ig_paral(3)` = the total grid size.

4.4.2 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 4.1 illustrates an Apple partition over 3 processes.

4.4.3 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[3]
- `ig_paral(5)` = the global extent in x.

Figure 4.2 illustrates a Box partition over 3 processes.

[3]The maximum value of the local extent in y is presently 338; it can be increased by modifying the value of `Clim_MaxSegments` in `oasis3/lib/clim/src/mod_clim.F90` and in `oasis3/lib/psmile/src/mod_prism_proto.F90` and by recompiling OASIS3 and the PSMILe library.

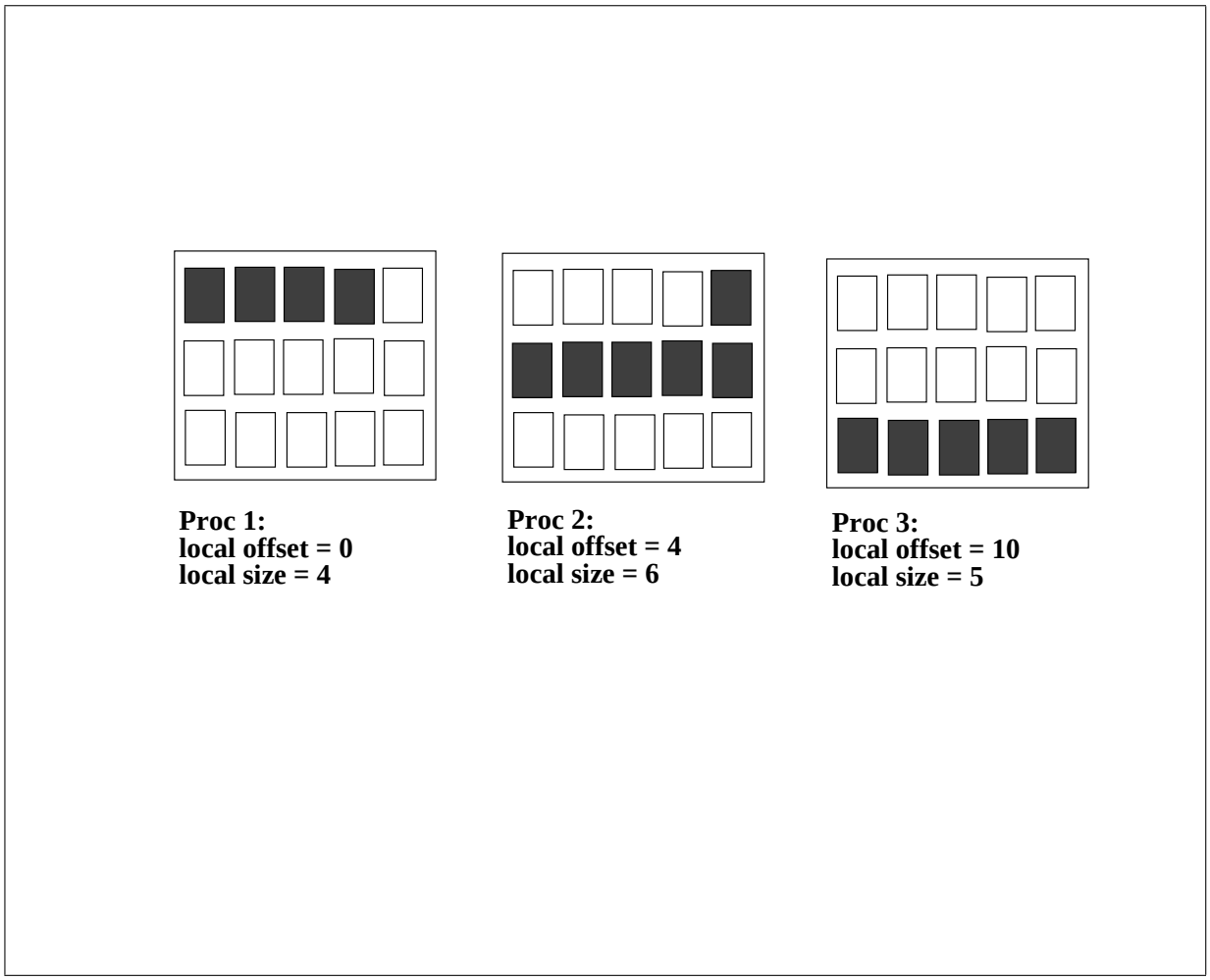


Figure 4.1: Apple partition. It is assumed here that the index start at 0 in the upper left corner.

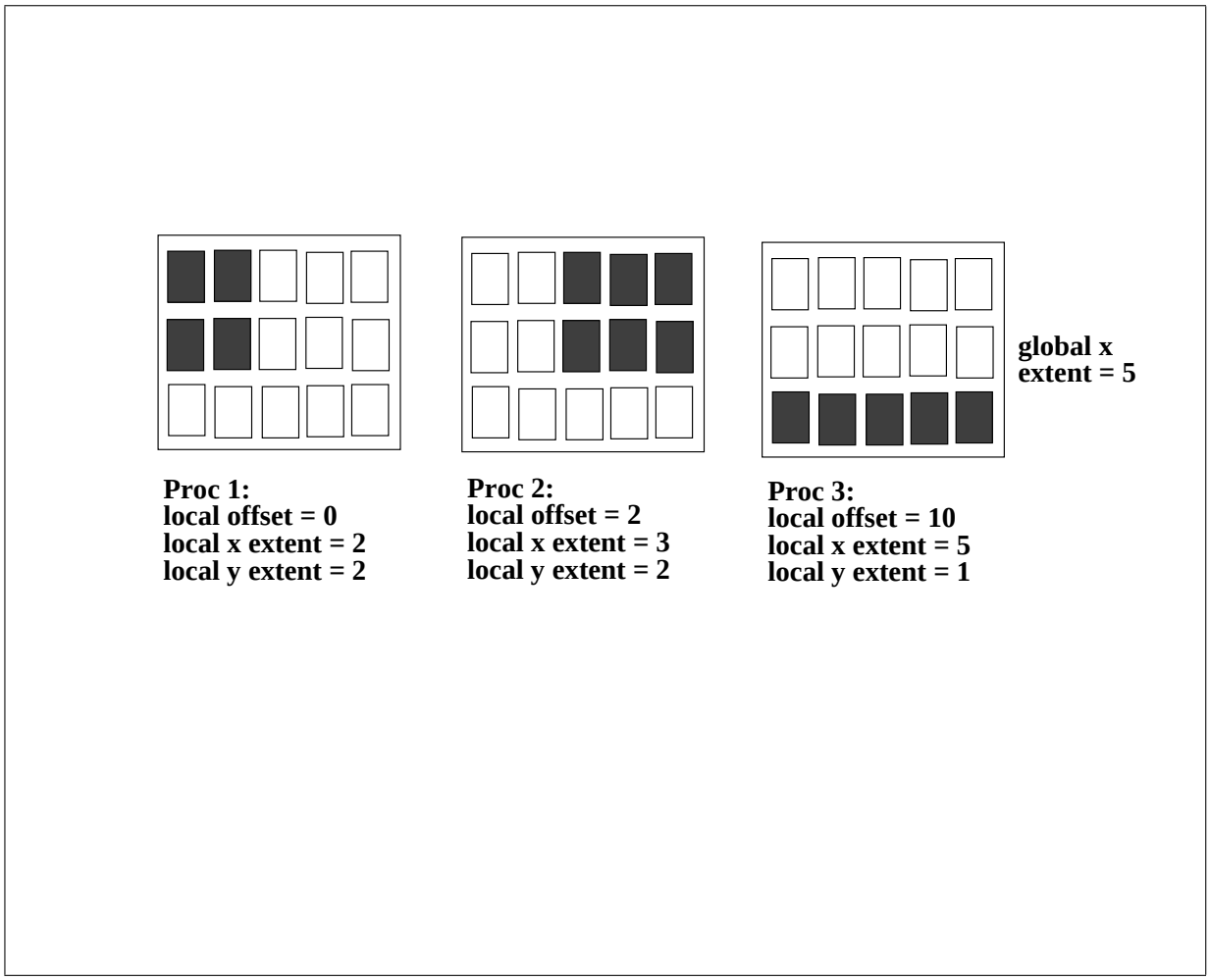
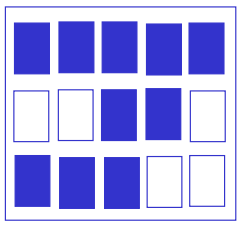


Figure 4.2: Box partition. It is assumed here that the index start at 0 in the upper left corner.



Proc 1:
nbr of segments = 3

1st segment offset = 0
1st segment size = 5
2nd segment offset = 7
2nd segment size = 2
3rd segment offset = 10
3rd segment size = 3

Figure 4.3: Orange partition for one process. It is assumed here that the index start at 0 in the upper left corner.

4.4.4 Orange partition

Each partition is an ensemble of segments of 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`[4].

- `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 4.3 illustrates an Orange partition with 3 segments for one process. The other process partitions are not illustrated.

[4]As for the Box partition, the maximum number of segments is presently 338; it can be increased by modifying the value of `Clim_MaxSegments`

4.5 I/O-coupling field declaration

Each process exchanging coupling data declares each field it will send or receive during the simulation.

- `CALL prism_def_var_proto(var_id, name, il_part_id, var_nodims, kinout, var_actual_shape, var_type, ierror)`
- `CALL oasis_def_var (var_id, name, il_part_id, var_nodims, kinout, var_actual_shape var_type, ierror)`
  - `var_id [INTEGER; OUT]`: coupling field ID
  - `name [CHARACTER*8; IN]`: field symbolic name (as in the *namcouple*)
  - `il_part_id [INTEGER; IN]`: partition ID (returned by `prism_def_partition_proto`)
  - `var_nodims [INTEGER, DIMENSION(2); IN]`: var_nodims(1) is the rank of field array (1 or 2); var_nodims(2) is the number of bundles (always 1 for OASIS3).
  - `kinout [INTEGER; IN]`: `PRISM_In` for fields received by the model, or `PRISM_Out` for fields sent by the model
  - `var_actual_shape [INTEGER, DIMENSION(2*var_nodims(1)); IN]`: vector of integers giving the minimum and maximum index for each dimension of the coupling field array; for OASIS3, the minimum index has to be 1 and the maximum index has to be the extent of the dimension.
  - `var_type [INTEGER; IN]`: type of coupling field array; put `PRISM_Real` 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 this interface.
  - `ierror [INTEGER; OUT]`: returned error code.

4.6 End of definition phase

Each process exchanging coupling data closes the definition phase.

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

4.7 Sending and receiving actions

4.7.1 Sending a coupling field

In the model time stepping loop, each process sends its part of the I/O or coupling field.

- `CALL prism_put_proto(var_id, date, field_array, info)`
- `CALL oasis_put (var_id, date, field_array, info)`
  - `var_id [INTEGER; IN]`: field ID (from corresponding prism_def_var_proto)
  - `date [INTEGER; IN]`: number of seconds in the run at the time of the call
  - `field_array [REAL, IN]`: I/O or coupling field array
  - `info [INTEGER; OUT]`: returned info code, current unused.

This routine may be called by the model at each timestep. The sending is actually performed only if the time obtained by adding the field lag (see 4.10) to the argument `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 5). A field will not be sent at all if its coupling or I/O period indicated in the *namcouple* is greater than the total run time.

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 6), it is automatically performed and the resulting field is finally sent at the coupling or I/O frequency. For non-instant transformations, partially transformed fields will be written to the restart file at the end of the run for use on the next model startup.

For a coupling field with a positive lag (see 4.10), the OASIS3 restart file (see section 7.3) is automatically written by the last `prism_put_proto` call of the run, if its argument `date` + the field lag corresponds to a coupling or I/O period.

4.7.2 Receiving a coupling field

In the model time stepping loop, each process receives its part of the I/O-coupling field.

- `CALL prism_get_proto(var_id, date, field_array, ierror)`
- `CALL oasis_get (var_id, date, field_array, ierror)`
  - `var_id [INTEGER; IN]`: field ID (from corresponding prism_def_var_proto)
  - `date [INTEGER; IN]`: number of seconds in the run at the time of the call
  - `field_array [REAL, OUT]`: I/O or coupling field array
  - `info [INTEGER; OUT]`: returned info code, current unused

This routine may be called by the model 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. A field will not be received at all if its coupling or I/O period indicated in the *namcouple* is greater than the total run time.

4.8 Termination

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

  All processes of the component model must terminate the coupling by calling `prism_terminate_proto`[5] (normal termination). OASIS3 will terminate after all processes called prism_terminate_proto.
- `CALL prism_abort_proto(compid, routine_name, abort_message)`
- `CALL oasis_abort (compid, routine_name, abort_message)`
  - `compid [INTEGER; IN]`: component model ID (from prism_init_comp_proto)
  - `routine_name[CHARACTER*; IN]`: name of calling routine
  - `abort_message[CHARACTER*; IN]`: message to be written out.

  If a process needs to abort voluntarily, it should do so by calling `prism_abort_proto`. This will ensure a proper termination of all processes in the coupled model communicator. This routine writes the name of the calling model, the name of the calling routine, and the message to the job standard output (stdout). This routine cannot be called before `prism_init_comp_proto`.

4.9 Auxiliary routines

Not all auxiliary routines available in oasis3_3 are available.

- `CALL prism_get_debug(debug_value)`

[5] If the process called `MPI_Init` (before calling `prism_init_comp_proto`), it must also call `MPI_Finalize` explicitly, but only after calling `prism_terminate_proto`.

- `CALL oasis_get_debug(debug_value)`
  - `debug_value [INTEGER; OUT]`: debug value

This routine may be called at any time to retrieve the current debug level internal to oasis. This is useful if changing debug levels and the user wants to return the debug value later to its prior value, or if a user wants to key off the oasis debug level for model debug diagnostics.

- `CALL prism_set_debug(debug_value)`
- `CALL oasis_set_debug(debug_value)`
  - `debug_value [INTEGER; IN]`: debug value

This routine may be called at any time to change the debug level in oasis. The debug level is initially set for all processors and all models by the NLOGPRT namcouple input. This method allows users to vary the debug level by model, task, or at different points in the model integration.

- `CALL prism_get_intercomm(new_comm, cdnam, kinfo)`
- `CALL oasis_get_intercomm(new_comm, cdnam, kinfo)`
  - `new_comm [INTEGER; OUT]`: mpi intercomm communicator
  - `cdnam [CHARACTER*; IN]`: other model name
  - `kinfo [INTEGER; OUT; OPTIONAL]`: returned error code

This routine sets up an MPI intercomm communicator between the root processors of two components, the local component and the component associated with cdnam. This must be called by both components at the same time otherwise a deadlock will occur. In addition, this call is collective across the tasks of the two components but other components are not involved.

- `CALL prism_get_intracomm(new_comm, cdnam, kinfo)`
- `CALL oasis_get_intracomm(new_comm, cdnam, kinfo)`
  - `new_comm [INTEGER; OUT]`: mpi intracomm communicator
  - `cdnam [CHARACTER*; IN]`: other model name
  - `kinfo [INTEGER; OUT; OPTIONAL]`: returned error code

This routine sets up an MPI intracomm communicator between the root processors of two components, the local component and the component associated with cdnam. This must be called by both components at the same time otherwise a deadlock will occur. In addition, this call is collective across the tasks of the two components but other components are not involved.

4.10 Coupling algorithms - SEQ and LAG concepts

Using the PSMILe library, the user has full flexibility to reproduce different coupling algorithms. In the component codes, the sending and receiving routines, respectively `oasis_put/prism_put_proto` and `oasis_get/prism_get_proto`, can be called at each model 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" (see section 4.7.1). This `date` argument is automatically analysed by the PSMILe and depending on the coupling period and the lag (LAG) chosen by the user for each coupling field in the configuration file *namcouple*, different coupling algorithms can be reproduced without modifying anything in the component model codes themselves.

With OASIS3-MCT, the SEQ index is no longer needed in the namcouple input to specify the sequencing order of different coupling fields. 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.

The lag concept and indices are explained in more detail below and some examples are provided.

4.10.1 The lag concept

The lag (LAG) value tells the coupler to modify the time at which that data is sent (put) by the amount of the lag. The lag must be expressed in "number of seconds" 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 restart-ability and dead-locking. When a component model calls a `oasis_put/prism_put_proto`, the value of the lag is automatically added to the value of the `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/prism_get_proto` for which the `date` argument is the same coupling time. The lag only shifts the time data is sent. It cannot be used to shift the time data is received yet.

When the lag is positive, a restart file must be available to initiate the coupling and in those cases, the restart file is then updated at the end of the run. A positive lag acts like a send occurred before the model started. In fact, for a field with positive lag, the source component model automatically reads the field in the restart file during the coupling initialization phase (below the `oasis_enddef/prism_enddef_proto`) and send the data to match the `oasis_get/prism_get_proto` performed at time=0 in the target component model. The final coupling data on the source side will then be automatically written to the restart file for use in the next run.

When there is a lag, the first and last instance of the source field in the debug netCDF file (EXPOUT fields, see section 5.3) always correspond respectively to the field read from and written to the restart file.

1. LAG concept first example

   A first coupling algorithm, exploiting the LAG concept, is illustrated on figure 4.4.

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

   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 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/prism_put_proto` 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 at times 12 and 24 below the `oasis_get/prism_get_proto` called by model B. For F_2, the `oasis_put/prism_put_proto` 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 below the `oasis_get/prism_get_proto` called by model A.

   At the beginning of the run, as their LAG index is greater than 0, the first `oasis_get/prism_get_proto` 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/prism_put_proto` as the `date+lag` equals the total run time. The analogue is true for F_2. These values will automatically be read in at the beginning of the next run below the respective `oasis_get/prism_get_proto`.

2. LAG concept second example

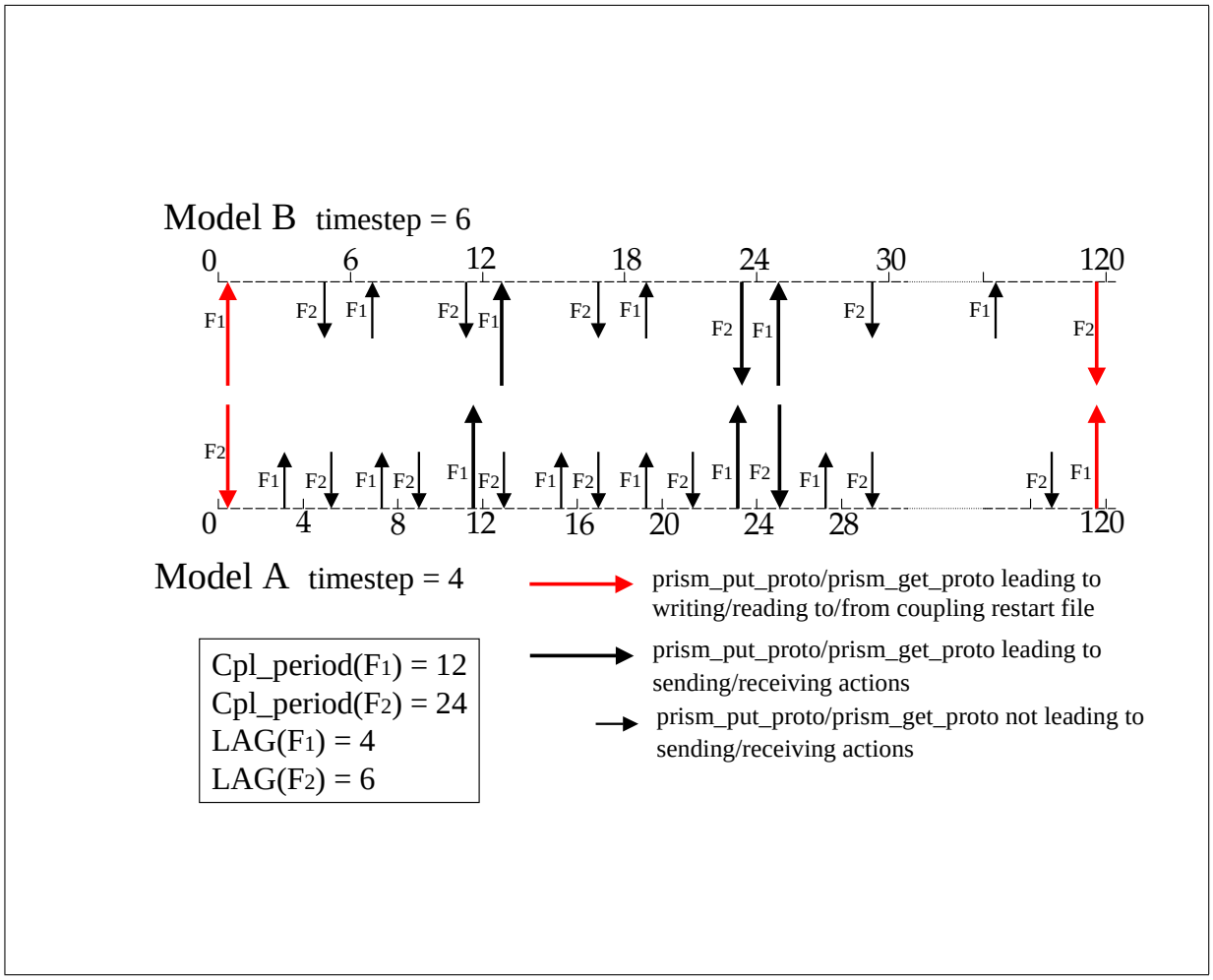


Figure 4.4: LAG concept first example

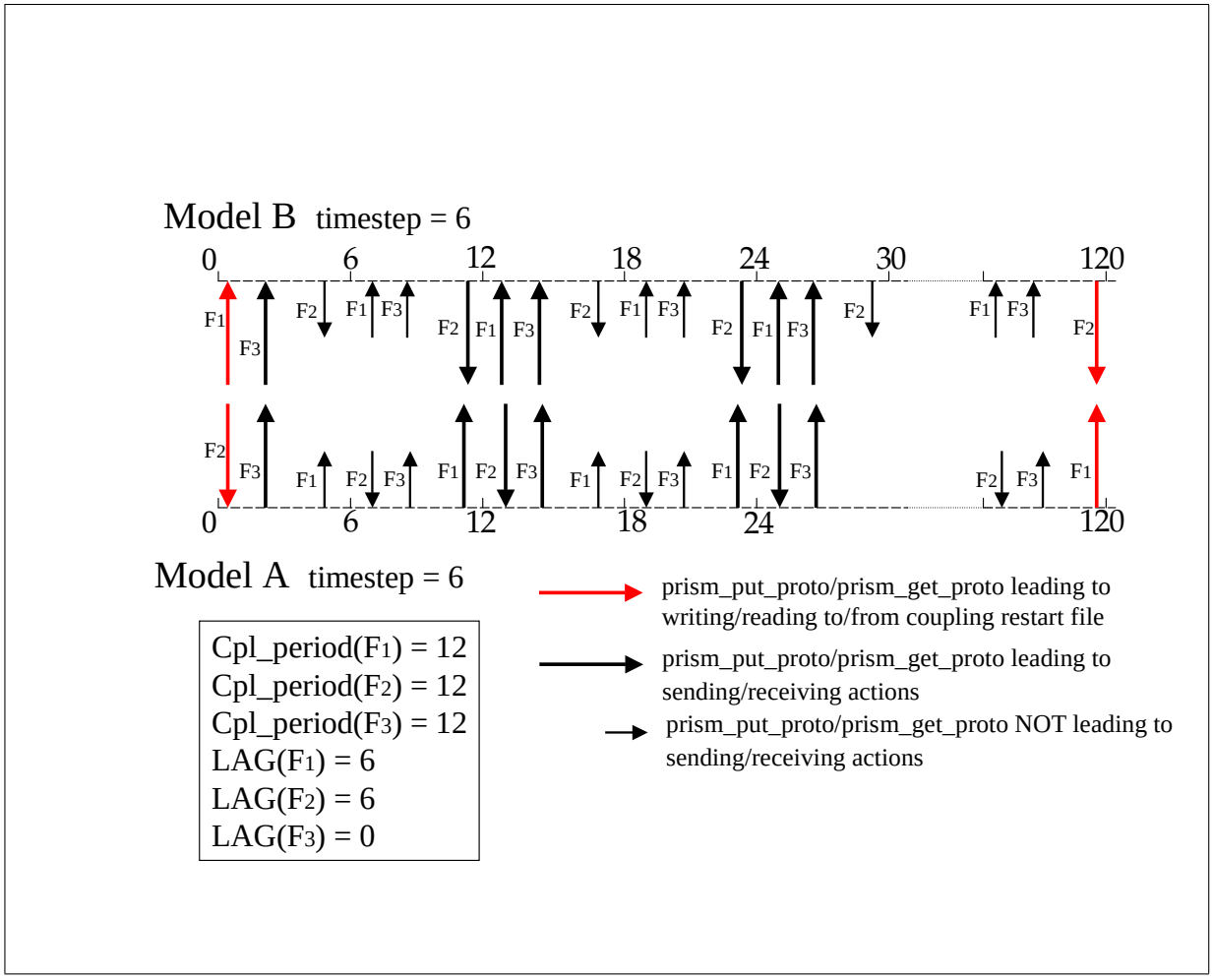


Figure 4.5: LAG concept second example

A second coupling algorithm exploiting the LAG concept is illustrated on figure 4.5. 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. 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 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/prism_put_proto` 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 below the `oasis_get/prism_get_proto` 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/prism_get_proto` 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/prism_put_proto` performed at the end of the run automatically write them to their coupling restart file. For F_3, no coupling restart file is needed nor used as at each coupling period the coupling field produced by model A can be directly "consumed" by model B.

We see here how the introduction of appropriate LAG indices results in "get" in the target model,

coupling fields produced by the source model the timestep before; this is, in some coupling configurations, essential to avoid deadlock situations.

4.10.2 The sequence concept

The order of coupling operations in the system is determined solely by the order of calls to send (put) and receive (get) data in the models in conjunction with the setting of the lag in the namcouple. Data that is received (get) is always blocking while data that is sent (put) is non-blocking with respect to the model making that call. It is possible to deadlock the system if the relative orders of puts and gets in different models 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 models compared to the SEQ specified in the namcouple. If there are any inconsistencies, the model 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.

Take the following two examples. In both 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 and each model sends (puts) the data first and then receives the data second at each timestep.

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

The put from model1 for fld1 is received by the get in model2 and the put from model2 for fld2 is received by the get in model1. 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 SEQ is set to 1 for fld1 and 2 for fld2 in this case, then the model will abort because at runtime, the coupling 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 this dependency is known, then there is a benefit in using SEQ=1 for fld1 and SEQ=2 for fld2. At runtime, if the sequencing of both model1 and model2 do not match the above diagram, the model will abort gracefully. For instance, if model2 has the dependency shown above but model1 does not have consistent ordering of the put and get as required by model2, then if SEQ is unused, the model will deadlock and hang. If SEQ is set properly, the coupling layer will detect that the required sequence has not been followed and will exit gracefully with an error message.

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

Chapter 5

The OASIS3 configuration file *namcouple*

The OASIS3 configuration file *namcouple* contains, below pre-defined keywords, all user's 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 not allowed.**

The first part of *namcouple* is devoted to configuration of general parameters such as the number of models involved in the simulation, the number of fields, the communication technique, etc. The second part gathers specific information on each coupling or I/O field, e.g. their coupling period, the list of transformations or interpolations to be performed by OASIS3 and associated configuring lines (described in more details in chapter 6), etc.

Several *namcouple* inputs have been deprecated (become unsupported or unused) in the current version, but for backwards compatability, they are still allowed. These inputs will be noted in the following text using the notation "UNUSED" and not fully described.

In the next sections, a simple *namcouple* example is given and all configuring parameters are described. The additional lines containing the different parameters required for each transformation are described in section 6. An example of a realistic *namcouple* can be found in `oasis3/examples/toyoasis3/input/` directory.

5.1 An example of a simple *namcouple*

The following simple *namcouple* configures a run in which an ocean, an atmosphere and an atmospheric chemistry models are coupled. The ocean provides only the SOSSTSST field to the atmosphere, which in return provides the field CONSFTOT to the ocean. One field (COSENHFL) is exchanged directly from the atmosphere to the atmospheric chemistry, and one field (SOALBEDO) is read from a file by the ocean.

```
#############################################################
# First section
#
 $SEQMODE
    1
#
 $CHANNEL
```

```
    MPI2    NOBSEND
    1    1   arg1
    3    1   arg2
    3    1   arg3
#
 $NFIELDS
    4    9
#
 $JOBNAME
    JOB
#
 $NBMODEL
    3  ocemod   atmmod  chemod  55  70   99
#
 $RUNTIME
    432000
#
 $INIDATE
    00010101
#
 $MODINFO
    NOT
#
 $NLOGPRT
   2
#
 $CALTYPE
   1
#
#############################################################################
# Second section
#
 $STRINGS
#
# Field 1
#
 SOSSTSST SISUTESU 1 86400  5  sstoc.nc  EXPORTED
 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
  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  MAPPING CHECKOUT
#
  ACCUMUL
  INT=1
  map_atmo_toce_conserv.nc src opt
  INT=1
#
# Field 3
#
 COSENHFL  SOSENHFL  37  86400   1  flda3.nc  IGNOUT
 atmo   atmo LAG=+7200
 LOCTRANS
 AVERAGE
#
# Field 4
#
 SOALBEDO SOALBEDO  17  86400  0  SOALBEDO.nc  INPUT
#
#####################################################################
```

5.2 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 in the second column. The first ten keywords are described hereafter:

- `$SEQMODE`: UNUSED. On the line below this keyword is an integer that plays no role in the implementation in the current version. 4.10).
- `$CHANNEL`: UNUSED. On the line below this keyword should be the string MPI1 but this plays no role in the implementation.
- `$NFIELDS`: On the line below this keyword is the total number of fields exchanged and described in the second part of the *namcouple*.
- `$JOBNAME`: On the line below this keyword is a `CHARACTER*3` or `CHARACTER*4` variable giving an acronym for the given simulation.
- `$NBMODEL`: On the line below this keyword is the number of models running in the given experiment followed by `CHARACTER*6` variables giving their names, which must correspond to the name announced by each model when calling `prism_init_comp_proto` (second argument, see section 4.2).

  Then the user may indicate the maximum Fortran unit number used by the models. In the example, Fortran units above 55, 70, and 99 are free for respectively the ocean, atmosphere, and atmospheric chemistry models. If no maximum unit numbers are indicated, OASIS3 PSMILe will suppose that units above 1024 are free. The unit number information is currently not used.
- `$RUNTIME`: On the line below this keyword is the total simulated time of the run, expressed in seconds.
- `$INIDATE`: On the line below this keyword is the initial date of the run. The format is `YYYYMMDD`. This date is important only for printing information in OASIS3 log file *cplout*.
- `$MODINFO`: UNUSED. On the line below this keyword should be the string NOT.
- `$NLOGPRT`: The line below this keyword refers to the amount of information that will be written to the OASIS log files for each model and processor. The value of `$NLOGPRT` may be:

  - 0 : one file debug.root.xx is open by the master proces of each model and one file debug_notroot.xx is open for all the other processes of each model to write error information.
  - 1 : one file debug.root.xx is open by the master proces of each model to write information equivalent to level 10 (see below); one file debug_notroot.xx is open for all the other processes of each model to write error information.
  - 2 : one file debug.xx.xxxxxx is open by each process of each model to write normal production diagnostics
  - 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

  This value can be changed at runtime with the `prism_set_debug` method.
- `$CALTYPE`: UNUSED. The line below this keyword contains an integer that plays not role in the implementation.

5.3 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 :

- `AUXILARY`: UNUSED
- `EXPORTED`: exchanged between component models and transformed by OASIS3 .
- `EXPOUT`: exchanged, transformed and also written to two debug NetCDF files, one before the sending action in the source model below the `prism_put_proto` call, and one after the receiving action in the target model below the `prism_get_proto` call (should be used when debugging the coupled model only). The name of the debug NetCDF file (one per field) is automatically defined by the PSMILe based on the field and component model names.
- `IGNORED`: this setting is equivalent to and converted to EXPORTED
- `IGNOUT`: this setting is equivalent to and converted to EXPOUT
- `INPUT`: simply read in from the input file by the target model PSMILe below the `prism_get_proto` call at appropriate times corresponding to the input period indicated by the user in the *namcouple*. See section 7.5 for the format of the input file.
- `OUTPUT`: simply written out to an output debug NetCDF file by the source model PSMILe below the `prism_put_proto` call, after local transformations `LOCTRANS` and `BLASOLD`, at appropriate times corresponding to the output period indicated by the user in the *namcouple*.

5.3.1 Second section of *namcouple* for `EXPORTED`, `AUXILARY` and `EXPOUT` fields

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

```
SOSSTSST SISUTESU 1 86400  5  sstoc.nc  sstat.nc EXPORTED
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 model (CHARACTER*8). It has to match the argument name of the corresponding field declaration in the source model; see prism_def_var_proto in section 4.5.
  - SISUTESU : symbolic name for the field in the target model (CHARACTER*8). It has to match the argument name of the corresponding field declaration in the target model; see prism_def_var_proto in section 4.5.
  - 1 : thie feature is not currently supported, but it is meant to be the index in auxiliary file cf_name_table.txt used by OASIS3 and PSMILe to identify corresponding CF standard name and units (see 7.1).
  - 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 (CHARACTER*8); mandatory even if no coupling restart file is effectively used. (for more detail, see section 7.3);
  - sstat.nc : UNUSED but still required for parsing
  - EXPORTED : field status.
- Field second line:
  - 182 : number of points for the source grid first dimension (optional if a netCDF coupling restart file is used).
  - 149 : number of points for the source grid second dimension (optional if a netCDF coupling restart file is used).
  - 128 : number of points for the target grid first dimension (optional if a netCDF coupling restart file is used).
  - 64 : number of points for the target grid second dimension (optional if a netCDF coupling restart file is used).
  - toce : prefix of the source grid name in grid data files (see section 7.2) (CHARACTER*4)
  - atmo : prefix of the target grid name in grid data files (CHARACTER*4)
  - LAG=+14400: optional lag index for the field expressed in seconds
  - SEQ=+1: optional sequence index for the field (see section 4.10)
- 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. There is then one or more additional configuring lines describing some parameters for each transformation. These additional lines are described in more details in the chapter 6.

5.3.2 Second section of *namcouple* for OUTPUT fields

The first 2 lines for fields with status OUTPUT are as follows:

```
COSHFTOT  COSHFTOT   7   86400  0  fldhftot.nc OUTPUT
atmo   atmo
```

where the different entries are as for EXPOUT fields, except that the source symbolic name must be repeated twice on the field first line, the restart file name is needed only for LOCTRANS transformations, there is no output file name on the first line and no LAG or SEQ index at the end of the second line. The

name of the output file is automatically determined by the PSMILe based on the field and component model names.

The third line is `LOCTRANS` if this transformation is chosen for the field. Note that `LOCTRANS` is the only transformation supported for `OUTPUT` fields.

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

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

```
SOALBEDO SOALBEDO  17  86400  0  SOALBEDO.nc  INPUT
```

- `SOALBEDO`: symbolic name for the field in the target model (`CHARACTER*8` repeated twice)
- 17: index in auxiliary file cf_name_table.txt (see above for EXPORTED fields)
- 86400: input period in seconds
- 0: number of transformations (always 0 for `INPUT` fields)
- `SOALBEDO.nc`: `CHARACTER*32` giving the input file name (for more detail on its format, see section 7.5)
- `INPUT`: field status.

Chapter 6

The transformations and interpolations in OASIS3

Different transformations and 2D interpolations are available in OASIS3 to adapt the coupling fields from a source model grid to a target model grid. They are divided into five general classes that have precedence one over the other in the following order: time transformation, pre-processing, interpolation, "cooking", and post-processing. This order of precedence is conceptually logical, but is also constrained by the OASIS3 software internal structure.

In the following paragraphs, a description of each transformation with its corresponding configuring lines is given. Features that are now deprecated (non functional) compared to prior versions will be noted with the string UNUSED but not described.

6.1 The time transformations

- **LOCTRANS**:

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

  Non INSTANT time transformations are now supported more generally with use of a restart file. The restart file allows the partial time transformation to be saved at the end of the run for exact restart at the start of the next run. For that reason, restart filenames are now required for all *namcouple* transforms that use LOCTRANS with non INSTANT values. In this mode, oasis will exit gracefully with an error message if a restart filename is not provided. This is the reason an optional restart file is now provided on the OUTPUT *namcouple* input line.

  ```
  # 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 prism_put_proto calls, not weighted by the time interval between these calls);
  - 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 prism_put_proto calls, not weighted by the time interval between these calls);

  - 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

6.2 The pre-processing transformations

The following transformations are available in the pre-processing part of OASIS3, controlled by `preproc.f`.

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

  CHECKIN calculates the global minimum, the maximum and the sum of the the source field values and prints them to the OASIS debug file (for the master process of the source component model only). This operation does not transform the field.

  The generic input line is as follows, even if $NINT has no impact in OASIS3-MCT:

```
# CHECKIN operation
    $INT = $NINT
```

- **CORRECT**: UNUSED

6.3 The interpolation

The following transformations are available in OASIS3.

- **BLASOLD**:

  BLASOLD allows the source fields 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. This transform 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 current field, and $NBFIELDS is UNUSED and should be set to 0 or 1. To add a scalar, an additional input line is required:

```
# nbfields lines
     CONSTANT  $AVALUE
```

  where CONSTANT is required and $AVALUE is the constant that is added to the field.

- **MAPPING**:

  The MAPPING keyword is used to specify an input file to be read and used for mapping (ie. regridding or interpolation); the MAPPING file must follow the SCRIPR format. This is an alternative method to SCRIPR for setting the mapping file.

  In the current implementation, each pair of source and target points in the MAPPING file can be linked by only one weight, i.e. remappings such as SCRIPR/BICUBIC involving at each ssource

grid point the value of the field, of the gradients and of the cross-gradient, or second-order conservative remapping are not supported.

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

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

- `$MAPNAME` is the name of the mapping file to read. This is a netcdf file consistent with the OASIS/SCRIPR map file format.
- `$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 on the destination grid to the destination model and processors; this is the default. With `dst`, the mapping is done on the destination processors after the data is sent from the source model on the source grid.
- `$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 without leveraging a partial sum computation. 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.

Note that if `SCRIPR` (see below) is used to calculate the remapping file, `MAPPING` can still be listed in the `namcouple` to specify a name for the remapping file generated by `SCRIPR` different from the default and/or to specify a `$MAPLOC` or `$MAPSTRATEGY` option.

- **SCRIPR**:

`SCRIPR` gathers the interpolation techniques offered by Los Alamos National Laboratory SCRIP 1.4 library[1](1). `SCRIPR` routines are in `oasis3/lib/scrip`. See the SCRIP 1.4 documentation in `oasis3/doc/SCRIPusers.pdf` for more details on the interpolation algorithms. In the current implementation, only first order mapping is supported through either scrip weights generation or reading from a file generated off-line. As a result, second order `CONSERV` or `BICUBIC` are not currently supported. Second order mapping methods will be added in future versions.

The following types of interpolations are available:

- `DISTWGT` performs a distance weighted nearest-neighbour interpolation (N neighbours). All types of grids are supported.
  - Masked target grid points: the zero value is associated to masked target grid points.
  - Non-masked target grid points having some of the N source nearest neighbours masked: a nearest neighbour algorithm using the remaining non masked source nearest neighbours is applied.
  - Non-masked target grid points having all of the N source nearest neighbours masked: by default, the nearest non-masked source neighbour is used.

  The configuring line is:

```
# SCRIPR (for DISWGT)
    $CMETH $CGRS $CFTYP $REST $NBIN $NV $ASSCMP $PROJCART
```

  - `$CMETH = DISTWGT`.
  - `$CGRS` is the source grid type (`LR`, `D` or `U`)- see annexe A.
  - `$CFTYP` is the field type: `SCALAR` if the field is a scalar one, or `VECTOR_I` or `VECTOR_J` whether the field represents respectively the first or the second component of a vector field

[1] See the copyright statement in appendix 3.2.2.

(see paragraph **Support of vector fields** below). The option VECTOR, which in fact leads to a scalar treatment of the field (as in the previous versions), is still accepted.

  * $REST is the search restriction type: LATLON or LATITUDE (see SCRIP 1.4 documentation SCRIPusers.pdf). Note that for D or U grid, the restriction may influence sligthly the result near the borders of the restriction bins. (XXX to be checked)
  * $NBIN the number of restriction bins (see SCRIP 1.4 documentation SCRIPusers.pdf).
  * $NV is the number of neighbours used.
  * $ASSCMP: optional, for VECTOR_I or VECTOR_J vector fields only; the source symbolic name of the associated vector component.
  * $PROJCART: optional, for vector fields only; should be PROJCART if the user wants the vector components to be projected in a Cartesian coordinate system before interpolation (see paragraph **Support of vector fields** below).

– GAUSWGT performs a N nearest-neighbour interpolation weighted by their distance and a gaussian function. All grid types are supported.

  * Masked target grid points: the zero value is associated to masked target grid points.
  * Non-masked target grid points having some of the N source nearest neighbours masked: a nearest neighbour algorithm using the remaining non masked source nearest neighbours is applied.
  * Non-masked target grid points having their N nearest neighbours all masked: the zero value will be associated to these target points.

  The configuring line is:

```
# SCRIPR (for GAUSWGT)
    $CMETH  $CGRS  $CFTYP  $REST  $NBIN  $NV $VAR $ASSCMP $PROJCART
```

  all entries are as for DISTWGT, except that:

  * $CMETH = GAUSWGT
  * $VAR, which must be given as a REAL value (e.g 2.0 and not 2), 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 the average distance between two source grid points (calculated automatically by OASIS3).

– BILINEAR performs an interpolation based on a local bilinear approximation (see details in chapter 4 of SCRIP 1.4 documentation SCRIPusers.pdf)

  For BILINEAR and BICUBIC, Logically-Rectangular (LR) and Reduced (D) source grid types are supported.

  * 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 bilinear or bicubic interpolation masked: a N nearest neighbour algorithm using the remaining non masked source points is applied.
  * Non-masked target grid points having their N nearest neighbours all masked: the zero value will be associated to these target points.

  The configuring line is:

```
# SCRIPR  (for BILINEAR or BICUBIC)
    $CMETH  $CGRS  $CFTYP  $REST  $NBIN $ASSCMP $PROJCART
```

  * $CMETH = BILINEAR or BICUBIC
  * $CGRS is the source grid type (LR or D)

  * $CFTYP, $NBIN, $ASSCMP $PROJCART are as for DISTWGT.
  * $REST is as for DISTWGT, except that only LATITUDE is possible for a Reduced (D) source grid.

- BICUBIC is currently unsupported as a mapping option becuase it is higher order. It performs an interpolation based on a local bicubic approximation (see details in chapter 5 of SCRIP 1.4 documentation SCRIPusers.pdf) See BILINEAR for configure line
- 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. Note that 2nd order conservative mapping files can be generated but not used currently.

  The configuring line is:

```
# SCRIPR (for CONSERV)
    $CMETH  $CGRS  $CFTYP  $REST  $NBIN  $NORM  $ORDER $ASSCMP $PROJCART
```

  * $CMETH = CONSERV
  * $CGRS is the source grid type: LR, D and U are supported for first-order remapping if the grid corners are given by the user in the grid data file grids.nc ; only LR is supported if the grid corners are not available in grids.nc and therefore have to be calculated automatically by OASIS3. For second-order remapping, only LR is supported because the gradient of the coupling field used in the transformation has to be calculated automatically by OASIS3.
  * $CFTYP, $REST, $NBIN, $ASSCMP,and $PROJCART are as for DISTWGT.
  * $NORM is the NORMalization option:
    · FRACAREA: 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.
    · DESTAREA: 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.
    · FRACNNEI: as FRACAREA, except that at least the source nearest unmasked neighbour is used for unmasked target cells that intersect only masked source cells. Note that a zero value will be assigned to a target cell that does not intersect any source cells (masked or unmasked), even with FRACNNEI option.
  * $ORDER: FIRST or SECOND for first or second order remapping respectively (see SCRIP 1.4 documentation). Note that CONSERV/SECOND is not positive definite and has not been fully validated yet.

Precautions related to the use of the SCRIPR/CONSERV remapping in particular

- 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 4 corners of each cell is known), the library assumes that the borders are linear in latitude and longitude between two corners. In general, this assumption becomes less valid closer to the pole and for latitudes above the north_thresh or below the south_thresh values specified in oasis3/lib/scrip/remap_conserv.F, the library evaluates the intersection between two border segments using a Lambert equivalent azimuthal projection. Problems have been observed in some cases for the grid cell located around this north_thresh or south_thresh latitude.

- Another limitation of the SCRIP 1st order conservative remapping algorithm is that is also supposes, 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.
- For a proper consevative remapping, the corners of a cell have to coincide with the corners of its neighbour cell.
- If two cells of a grid overlay, at least the one with the greater numerical index must be masked (they also can be both masked) for a proper conservative 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*, OASIS 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` version.
- A target grid cell intersecting no source cell (either masked or non masked) at all i.e. falling in a "hole" of the source grid will in all cases get a zero value.
- If a target grid cell intersects only masked source cells, it will still get a zero value unless:

  - the `FRACNNEI` normalisation option is used, in which case it will get the nearest non masked neighbour value, or

  - the routines `oasis3/lib/scrip/src/scriprmp.f` or `vector.F90` - for vector interpolation - are compiled with `ll_weightot=.true.` in which case, the value 1.0E+20 will be assigned to these target grid cell intersecting only masked source cells (for easier identification).

Precautions related to the use of the SCRIPR remappings in general

- For using `SCRIPR` interpolations, linking with the NetCDF library is mandatory and the grid data files (see section 7.2) must be NetCDF files (binary files are not supported).
- 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. The file is created in the working directory and is called `rmp_`*srcg*`_to_`*tgtg*`_`*INTTYPE*`_`*NORMAOPT*`.nc`, where *srcg* and *tgtg* are the acronyms of respetively the source and the target grids, *INTTYPE* is the interpolation type (i.e. `DISTWGT`, `GAUSWGT`, `BILINEA`, `BICUBIC`, or `CONSERV`) and *NORMAOPT* is the normalization option (i.e. `DESTAREA`, `FRACAREA` or `FRACNNEI` for `CONSERV` only). The problem comes from the fact that the weights and addresses will also differ whether or not the `MASK` and `EXTRAP` transformations are first activated during the pre-processing phase (see section 6.2) and this option is not stored in the remapping file name. Therefore, the remapping file used will be the one created for the first field having the same source grid, target grid, and interpolation type (and the same normalization option for `CONSERV`), even if the `MASK` and `EXTRAP` transformations are used or not for that field. (This inconsistency is however usually not a problem as the `MASK` and `EXTRAP` transformations are usually used for all fields having the same source grid, target grid, and interpolation type, or not at all.)

Support of vector fields with the SCRIPR remappings

Vector mapping is NOT supported in this version of OASIS. Vector fields are treated as if each field were an independent scalar field. Vector mapping capabilities will be added in future versions.

- **INTERP**: UNUSED
- **MOZAIC**: UNUSED
- **FILLING**: UNUSED

6.4 The "cooking" stage

The following transformations are available in the "cooking" part of OASIS3.

- **CONSERV**:

  CONSERV ensures a global modification of the coupling field. This analysis requires the areas of the source and target grid meshes to be transfered to the coupler with the oasis_write_area/prism_write_area call (see section 4.3). In the *namcouple*, CONSERVrequires one input line with one argument and one optional argument:

  ```
  # CONSERV operation
       $CMETH  $CONSOPT
  ```

  where:

  - $CMETH is the method desired with the following choices
    - 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; this option ensures global conservation of the field
    - with $CMETH = GLBPOS, the same operation is performed except that the residual is distributed proportionally to the value of the original field; this option ensures the global conservation of the field and does not change the sign of the field
    - with $CMETH = BASBAL, the operation is analogous to GLOBAL except that the non masked 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 surface of the target grid
    - with $CMETH = BASPOS, 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; therefore, 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.
  - $CONSOPT is an optional argument specifying the algorithm. $CONSOPT can be bfb or opt. The bfb option enforces a bit-for-bit transformation regardless of the grid decomposition or process count. The opt option carries out the conservation using an optimal algorithm using less memory and a faster approach. Option bfb is the default setting.

  Note that for this operation to be correct, overlapping grid cells on the source grid or on the target grid must be masked.

- **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. The prior feature that supported linear combinations of the current coupling field with any other fields after the interpolation has been deprecated.

  This analysis requires the same input line as BLASOLD.

- **MASKP**: UNUSED

6.5 The post-processing

The following analyses are available in the post-processing part of OASIS3.

- **REVERSE**: UNUSED

- **CHECKOUT**:

  CHECKOUT calculates the global minimum, the maximum and the sum of the the target field values and prints them to the OASIS debug file (for the master process of the target component model only). This operation does not transform the field. The generic input line is as for CHECKIN (see above).
- **GLORED**: UNUSED

Chapter 7

OASIS3 auxiliary data files

OASIS3 needs auxiliary data files describing coupling and I/O field names and units, defining the grids of the models being coupled, containing the field coupling restart values or input data values, as well as a number of other auxiliary data files used in specific transformations.

7.1 Field names and units

The text file `cf_name_table.txt`, has been available to users in the past. Use of this file is currently not supported but will be in future versions.

7.2 Grid data files

The grids of the models being coupled must be given by the user, or directly by the model through PSMILe specific calls that write grid data. Note that if the grid data files exist in the working directory, fields in those files will not be overwritten by the PSMILe specific calls (see section 4.3). These files are netCDF.

The arrays containing the grid information are dimensioned `(nx, ny)`, where `nx` and `ny` are the grid first and second dimension. Unstructured grids are supported by setting the ny dimension to 1 and then nx is the total number of grid points.

1. *grids.nc*: contains the model grid longitudes, latitudes, and local angles (if any) in single or double precision `REAL` arrays (depending on OASIS3 compilation options). 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 5.3), and of a suffix (4 characters); this suffix is ".lon" or ".lat" for respectively the grid point longitudes or latitudes.

   If the `SCRIPR/CONSERV` remapping is used, longitudes and latitudes for the source and target grid **corners** must also be available in the *grids.nc* file as arrays dimensioned `(nx,ny,4)` or `(nbr_pts,1,4)` where `4` is the number of corners (in the counterclockwize sense). 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 model through PSMILe specific calls (see section 4.3).

   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 OASIS.

   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.

If vector fields are defined on a grid which has a local coordinate system not oriented in the usual zonal and meridional directions, the local angle of the grid coordinate system must be given in *grids.nc* file in an array which name must be composed of the grid prefix and the suffix ".ang". The angle is defined as the angle between the first component and the zonal direction (which is also the angle between the second component and the meridional direction). If one of the `SCRIPR` interpolations is requested for a vector field, OASIS3 automatically performs the rotation from the local coordinate system to the geographic spherical coordinate system for a source grid, or vice-versa for a target grid.

2. *masks.nc*: contains the masks for all component model grids in `INTEGER` arrays (0 -not masked i.e. active- or 1 -masked i.e. not active- for each grid point). The array names must be composed of the grid prefix and the suffix ".msk". This file, *masks* or *masks.nc*, is mandatory.
3. *areas.nc*: this file contains mesh surfaces for the component model grids in single or double precision `REAL` arrays (depending on OASIS3 compilation options). 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 all the same (in `INTERP/GAUSSIAN`, it is assumed that the units are m^2 but they are used for statistics calculations only.) This file *areas* or *areas.nc* is mandatory for `CHECKIN`, `CHECKOUT` or `CONSERV`, and used for statistic calculations in `INTERP/GAUSSIAN`; it is not required otherwise.

7.3 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 section 4.10). When needed, these files are also automatically updated by the last `prism_put_proto` call of the run (see section 4.7.1) . **Warning**: the date is not written or read to/from the restart file; therefore, the user has to make sure that the appropriate restart file is present in the working directory.

Note that all restart files have to be present in the working directory at the beginning of the run even if one model is delayed with respect to the others.

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 5.3). 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.

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, except for fields given on unstructured for which the arrays are dimensioned `(nt,1)`, where `nt` is the total number of grid points. 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.

NetCDF formats are supported; for NetCDF file the suffix .nc is not mandatory. In the NetCDF restart files, the field arrays must have the source symbolic name indicated in the *namcouple*. (see section 5.3).

7.4 Mapping files

The mapping files to be read using the MAPPING option are consistent with the files generated in OASIS3. The files are netcdf and the key fields are num_links (the number of weights for each grid pair), src_grid_size (the 2d size of the source grid), dst_grid_size (the 2d size of the destination grid), num_wgts

(the total number of weights), remap_matrix (the weights), dst_address (the global destination index for the weight), src_address (the global source index for the weight).

7.5 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 PSMILe below the `prism_get_proto` 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 5.3.3). 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 field symbolic name in the *namcouple* and dimensioned `(nx,ny,time)` or `(nbr_pts,1,time)`. The time variable has to be an array `time(time)` expressed in "seconds since beginning of run". The "time" dimension has to be the unlimited dimension.

7.6 Transformation auxiliary data files

Some transformations need auxiliary data files. In particular, interpolation requires that mapping files be generated either offline and set through the `MAPPING` *namcouple* keyword or be generated via scrip in OASIS using the `SCRIPR` *namcouple* keyword.

7.6.1 Auxiliary data files for `SCRIPR`

The NetCDF files containing the weights and addresses for the `SCRIPR` remappings (see section 6.3) are automatically generated at runtime by OASIS3. Their structure is described in detail in section 2.2.3 of the SCRIP documentation available in `oasis3/doc/SCRIPusers.pdf`. In particular, they are netCDF files with the following one fields

- src_grid_size is a scalar integer indicating the total number of global grid cells for the source grid. This field in a netCDF dimension.
- dst_grid_size is a scalar integer indicating the total number of global grid cells for the destination (or target) grid. This field in a netCDF dimension.
- num_links is a scalar integer indicating the total number of associated grid pairs in the file. This is typically a large number. This field is a netCDF dimension.
- num_wgts is a scalar integer indicating the number of weights per associated grid pair. For first order mapping, this is 1. This field in 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. This field is a netCDF variable.

The mapping implementation does the following using a parallel approach

```
! this is pseudocode for first order mapping

! src_arr is an input, dst_array is an output
! src_address, dst_address, remap_matrix are read from mapping file
! while remap_matrix supports num_wgts > 1, for first order
!   num_wgts = 1 and the sparse matrix multiply below is hardcoded
```

```
!   for first order mapping only.

real :: src_arr(:), dst_arr(:), remap_matrix(:,:)
integer :: src_address(:),dst_address(:)

allocate(src_arr(src_grid_size))
allocate(dst_arr(dst_grid_size))

allocate(src_address(num_links))
allocate(dst_address(num_links))
allocate(remap_matrix(num_wgts,num_links))

read(src_address)
read(dst_address)
read(remap_matrix)

src_arr(:) = from_input(:)
dst_arr(:) = 0.0
do n = 1,num_links
   sindex = src_address(n)
   dindex = dst_address(n)
   dst_arr(dindex) = dst_arr(dindex) + src_array(sindex)*remap_matrix(1,n)
enddo
```

Chapter 8

Compiling and running OASIS3 and TOYOASIS3

8.1 Compiling OASIS3 and debugging

8.1.1 Compilation with TopMakefileOasis3

Compiling OASIS3 can be done in directory `oasis3/util/make_dir` with Makefile `TopMakefileOasis3` which must be completed with a header file `make.`*your_platform* specific to the compiling platform used and specified in `oasis3/util/make_dir/make.inc`. One of the header files distributed with the release can by used as a template. The root of the OASIS3 tree can be anywhere and must be set in the variable `COUPLE` in the `make.`*your_platform* file.

The following commands are available:

- `make -f TopMakefileOasis3`
  compiles OASIS3 libraries *mct* and *scrip*
- `make realclean -f TopMakefileOasis3`:
  removes OASIS3 and PSMILe library compiled sources and librairies.

Log and error messages from compilation are saved in the files COMP.log and COMP.err in make_dir.

During compilation, a new compiling directory, defined by variable `ARCHDIR` is created. After successful compilation, resulting executables are found in the compiling directory in `/bin`, libraries in `/lib` and object and module files in `/build`.

The different pre-compiling flags used for OASIS3 and its associated PSMILe library are described in section 8.1.2.

8.1.2 CPP keys

The following CPP keys are coded in OASIS3 and associated PSMILe library and can be specified in `CPPDEF` in `make.`*your_platform* file.

- To ensure, in `SCRIPR/CONSERV` remapping (see section 6.3), that if two cells of the source grid overlay, at least the one with the greater numerical index is masked (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 are to be respected.
  - `TREAT_OVERLAY`

- To reproduce default behaviour of SCRIPR/DISTWGT before version oasis3_3, i.e. the zero value is associated to the target points having all of the N source nearest neighbours masked (see section 6.3 for details).
  - NOT_NNEIGHBOUR

8.1.3 Debugging

If you experience problems while running your coupled model with OASIS3, you can obtain more information on what is happening with:

- Increasing $NLOGPRT in your *namcouple* (see section 5.2)

8.2 Running OASIS3 in coupled mode with TOYOASIS3

In order to test the OASIS3 coupler in a light coupled configuration, CERFACS has written 3 "toy" component models, mimicking an atmosphere model (atmoa3), an ocean model (oceoa3), and a chemistry model (cheoa3), which sources can be found in oasis3/examples/toyoasis3/src. These "toy" component models are 'empty' in the sense that they do not model any real physics or dynamics. The coupled combination of these 3 "toy" component models through OASIS3 coupling software is refered to as the TOYOASIS3 coupled model; the TOYOASIS3 coupling is realistic as the coupling algorithm linking the toy component models, the size and the grid of the 2D coupling fields, and the operations performed by OASIS3 on the coupling fields are realistic.

The current version of OASIS3 and its TOYOASIS3 example coupled model was successfully compiled and run on:

- Linux neolith1 2.6.18-194.8.1.el5, with ifort/icc and openmpi - 1.4 and scalimpi 3.13.8, thanks to C. Basu from NSC (Sweden)
- cluster BULLX based on Intel/westmere, with Intel 11.1.073 compiler and "bullxmpi 1.0.2" MPI library (compilation only)
- CRAY XT platforms, with Cray Fortran Compiler crayftn and MPI library xt-mpt, thanks to C. Henriet from Cray
- (XXX to be completed)

Previous versions were compiled and run on many other platforms.

Compiling OASIS3 was described in section 8.1. In the following section, the TOYOASIS3 example coupled model is first described in more detail (see section 8.2.1), then instructions on how to compile and run TOYOASIS3 are given in section 8.2.2.

8.2.1 TOYOASIS3 description

The oceoa3 model

The oceoa3 model has a 2D logically-rectangular, streched and rotated grid of 182x149 points, which corresponds to a real ocean model grid (with two poles of convergence over America and Asia). Oceoa3 timestep is 14400 seconds; it performs therefore 36 timesteps per 6-day run.

OASIS3 PSMILe routines are detailed in section 4. At the beginning of a run, oceoa3 performs appropriate PSMILe calls to initialize the coupling, define its grids, and declare its I/O or coupling fields. As oceoa3 is not parallel, it calls the PSMILe prism_def_partition routine to define only one Serial partition containing the 182X149 grid points.

Then, oceoa3 starts its timestep loop. At the beginning of its timestep, oceoa3 calls the PSMILe prism_get routine 6 times to request the fields named Field3, Field4, Field6 to Field9 on table 8.1. At the end of its

timestep, oceoa3 calls PSMILe prism_put routine to send fields named Field1 and Field2 on table 8.1. The fields will be effectively received or sent only at the coupling frequency defined by the user (see section 5.3).

Finally, at the end of the run, oceoa3 performs the PSMILe finalization call.

The atmoa3 model

The atmoa3 model has a realistic atmospheric Gaussian reduced grid with 6232 points. Its timestep is 3600 seconds; it therefore performs 144 timesteps per 6-day run.

As oceoa3, atmoa3 performs, at the beginning of a run, appropriate PSMILe calls to initialize the coupling, define its grids, and declare its I/O or coupling variables. Then atmoa3 retrieves a local communicator for its internal parallelization with a call to PSMILe prism_get_localcomm routine, useful if the `MPI1` communication technique is chosen by the user (see section 4.2), and defines its local partition calling the PSMILe prism_def_partition routine.

Then, atmoa3 starts its timestep loop. At the beginning of its timestep, atmoa3 calls the PSMILe prism_get routine 3 times to request the fields named Field1, Field2 and Field11 on table 8.1. At the end of its timestep, atmoa3 calls PSMILe prism_put routine to send fields named Field4 to Field10 on table 8.1. The fields will be effectively received or sent only at the coupling frequency defined in the *namcouple* (see section 5.3) of the coupled model that one can find in `oasis3/examples/toyoasis3/input`.

Finally, at the end of the run, atmoa3 performs the PSMILe finalization call.

The cheoa3 model

Cheoa3 is integrated on the same atmospheric model grid than atmoa3. Its timestep is 7200 seconds; it therefore performs 72 timesteps per 6-day run.

As the other toymodels, cheoa3 performs, at the beginning of a run, appropriate PSMILe calls to initialize the coupling, define its grids, and declare its I/O or coupling variables; it also retrieves a local communicator if needed. As cheoa3 has the same grid than atmoa3, a direct exchange of coupling fields can occur between those two models, without going through OASIS3 interpolation process. To insure this, the coupling field must have a field status 'IGNORED' or 'IGNOUT' in the OASIS3 configuration file *namcouple* (see section 5.3) and the two models must have also the same parallel decomposition. Cheoa3 decomposition is hardcoded the same way than atmoa3, and if the user modifies the atmoa3 decomposition, he has to modify the cheoa3 decomposition the same way by changing cheoa3 values for `il_nbcplproc` and `cdec` (see below).

At the beginning of its timestep, cheoa3 calls the PSMILe prism_get routine to request Field10 (see table 8.1). At the end of its timestep, cheoa3 calls PSMILe prism_put routine to send Field11.

Finally, at the end of the run, cheoa3 performs the PSMILe finalisation call.

TOYOASIS3 coupling algorithm

The coupling algorithm between the TOYOASIS3 component models oceoa3, atmoa3, and cheoa3 is described here.

Table 8.1 lists the coupling fields exchanged between those 3 model components, giving the symbolic name used in each component and indicating whether the model produces the field (src) or receives it (tgt).

Figure 8.1 illustrates the coupling algorithm between the 3 TOYOASIS3 toy models for $Field_1$, $Field_3$, $Field_4$, $Field_{10}$, and $Field_{11}$.

	oceoa3	atmoa3	cheoa3	restart
Field1	SOSSTSST (src)	SISUTESU (tgt)		fldo1.nc
Field2	SOICECOV (src)	SIICECOV (tgt)		fldo2.nc
Field3	SOALBEDO (tgt)			SOALBEDO.nc
Field4	SONSHLDO (tgt)	CONSFTOT (src)		flda1.nc
Field5		COSHFTOT (src)		
Field6	SOWAFLDO (tgt)	COWATFLU (src)		flda3.nc
Field7	SORUNOFF (tgt)	CORUNOFF (src)		flda4.nc
Field8	SOZOTAUX (tgt)	COZOTAUX (src)		flda5.nc
Field9	SOMETAUY (tgt)	COMETAUY (src)		flda6.nc
Field10		COSENHFL (src)	SOSENHFL (tgt)	flda7.nc
Field11		COTHSHSU (tgt)	SOTHSHSU (src)	flda8.nc

Table 8.1: Coupling and I/O fields of the TOYOASIS3 coupled model. The symbolic name used in each toy model is given and it is indicated whether the model produces the field (src) or receives it (tgt).

$Field_1$ is sent from oceoa3 component to atmoa3 component at the coupling frequency dtF_1 defined by the user in the configuring file *namcouple*. As interpolation is needed between oceoa3 and atmoa3 grids, this exchange must go through OASIS3 interpolation process. In the *namcouple*, $Field_1$ field status must therefore be $EXPORTED$ and the interpolation must be defined. If the user wants the field to be also automatically written to files before being sent (below the prism_put), and after being received (below the prism_get), he can choose the field status $EXPOUT$. In oceoa3 and atmoa3 codes, the prism_put and prism_get routines are respectively called every timestep with an argument corresponding to the time at the beginning of the timestep. The lag of $Field_1$, defined as 4 hours (14400 seconds) in the *namcouple*, is automatically added to the prism_put time argument; the prism_put called at the oceoa3 timestep preceeding the coupling period therefore matches the prism_get called in atmoa3 at the coupling period.

At the beginning of the run (i.e. at time = 0), the oceoa3 prism_put for $Field_1$ is not activated (as a positive lag is defined for $Field_1$) and OASIS3 automatically read $Field_1$ in its coupling restart file, fldo1.nc, and sends it to atmoa3 component after interpolation.

The exchange of $Field_2$ from oceoa3 to atmoa3 and $Field_4$, $Field_6$, $Field_7$, $Field_8$ and $Field_9$ from atmoa3 to oceoa3 follow exactly the same logic as for $Field_1$.

$Field_3$ as a status `INPUT` in the *namcouple*. $Field_3$ will therefore not be exchanged between two models but will be read from a file automatically below the target model oceoa3 prism_get calls, at the user-defined frequency in the input file also specified in the *namcouple*, `SOALBEDO.nc`.

$Field_5$ as a status of `OUTPUT` in the *namcouple*. It will therefore be only automatically written to a file at the user-defined frequency, below the source model atmoa3 prism_put calls. The name of the file will be automatically composed of the field symbolic name (here `COSHFTOT`) and of the begin and end dates of the run.

$Field_{10}$ and $Field_{11}$ are exchanged respectively from atmoa3 to cheoa3 and from cheoa3 to atmoa3. The fields status chosen by the user for those fields in the *namcouple* should therefore be `IGNORED` (or `IGNOUT` if the user wants the fields also automatically be written to files below the prism_put and after the prism_get). At the beginning of the run (i.e. at time = 0), the oceoa3 prism_get called to receive those fields will automatically read the fields in their corresponding coupling restart files flda7.nc and flda8.nc.

8.2.2 Compiling and Running TOYOASIS3

The TOYOASIS3 compiling and running environment is available in `oasis3/examples/toyoasis3`.

Subdirectory `/data` contains auxiliary grid data files (see 7.2) and coupling restart files (see 7.3). Subdirectory `/input` contains the file `cf_name_table.txt` (see 7.1) the configuration file (see chapter 5) used in the classic (not parallel) mode, *namcouple*, and the configuration files used in the IPSL parallel

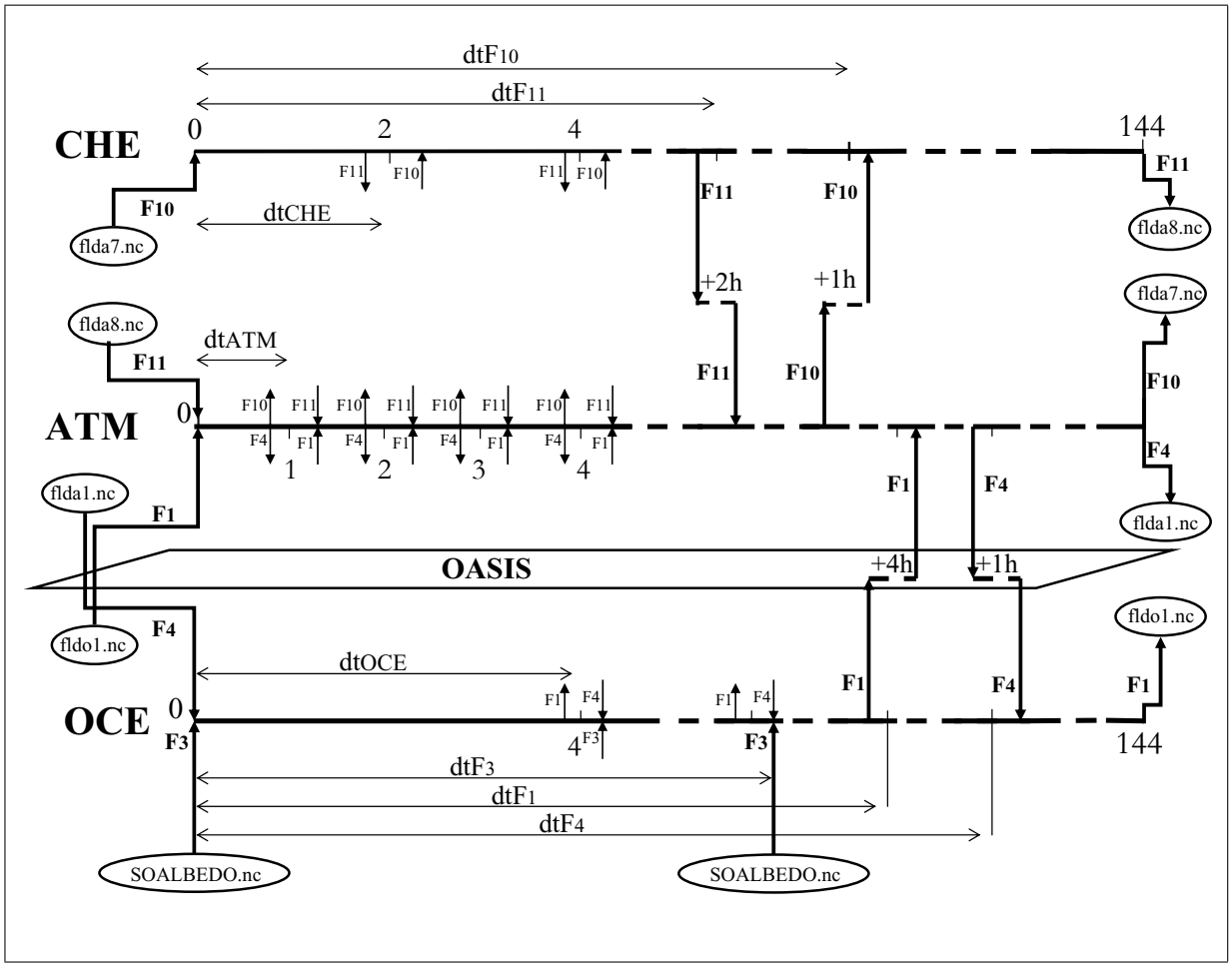


Figure 8.1: Exchange algorithm between the 3 TOYOASIS3 component models for fields $Field_1$, $Field_3$, $Field_4$, $Field_{10}$, and $Field_{11}$.

mode, *namcouple_0* and *namcouple_1*.

To run TOYOASIS3, first compile OASIS3 and the 3 TOYOASIS3 component models (see section 8.1). Go in directory `oasis3/examples/toyoasis3/src` and type `make`. This will automatically compile OASIS main executable and PSMILe library, if not done before hand, and the three component models `atmoa3.x`, `cheoa3.x` and `oceoa3.x`, using the header file specified in `oasis3/util/make_dir/make.inc`.

The next step is to adapt the "User's section" of the running script `run_toyoasis3` in subdirectory `/script` and to launch it. The script `run_toyoasis3` supports Linux PC, NEC SX, IBM Power4, CRAYX1, CRAYXD1 and CRAYXT platforms (see `arch` variable). If your platform is not supported, the script will have to be adapted.

8.3 Known problems when compiling or running OASIS3 on specific platforms

- Notes on porting to BlueGeneL (XXX to be detailed):
- Notes on porting to BlueGeneP (XXX to be detailed):

Appendix A

The grid types for the transformations

As described in section 6, the different transformations in OASIS3 support different types of grids. The characteristics of these grids are detailed here.

1. Grids supported for the SCRIPR interpolations
   - '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 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 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, 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 versions

Here is a list of changes between the different official OASIS3 versions.

B.1 Changes between `oasis3_3` and `oasis3-mct`

Much of the underlying implementation was refactored to leverage the Model Coupling Toolkit (MCT). All communication is now parallel and direct between models. The central oasis coupler running on it's own processors no longer plays a role. MCT is used to carry out interpolation in parallel on either the source or destination processors before or after communication. This location of the interpolation is set by the user as is the parallelization strategy. Several *namcouple* features have been deprecated in this version, but this version supports a *namcouple* format that mostly backwards compatable with OASIS3. The ability to read mapping files is now provided with the *namcouple* transformation MAPPING. Several other changes are highlighted below.

- General Features:
  - MPI2 job launching is NOT supported. Only MPI1 is allowed.
  - All coupling is now direct. Gets are blocking, puts are non-blocking. Get and put calls between models must match adequately to avoid deadlocks.
  - Like oasis3, the first coupling occurs at time=0 and the final coupling occurs at (ncouplings-1)*dt.
  - All input and output files must be netcdf, binary not supported
  - Ability to send and receive more than 1 field with a coupling operation. Use colon delimited field names in *namcouple* input.
  - A model output field (put) can be associated with more than one destination model (get). In that case, the source model only needs to put the data once and that data will arrive at multiple destinations as specified by *namcouple* input file. Different coupling frequencies and transforms are allowed for different coupling interactions of the same field. In addition, a single coupling field can be sent to the same destination model using multiple transforms as long as the destination field name is unique. For example
    - TSURF from model 1 to TSURF in model 2
    - TSURF from model 1 to TSURFAVG in model2
    - TSURF from model 1 to TSURF in model 3

    TSURF only needs to be put from model 1 "once" per timestep and the data will arrive at the destination at the coupling frequency and with transforms specified in the *namcouple* file for each coupling interaction. The inverse of this coupling feature is not allowed, a single destination (get) field CANNOT be associated with multiple source (put) fields.

  - Only first order mapping methods are supported. Second order conserv and bicubic mapping is not supported.
  - Vector mapping is currently not supported.
  - Grid corners will NOT be compute automatically for LR grids if they are needed but not provided.

- Coupling Interfaces:
  - use mod_prism OR use mod_oasis instead of many use statements
    - mod_prism retains prior interface names
    - mod_oasis has updated interface names, changing "prism" to "oasis" and dropping "_proto" in the interface name
  - NO MPI2 support yet
  - prism_put_inquire not supported yet
  - prism_put_restart_proto not supported yet
  - prism_get_freq not supported yet

- *namcouple* Changes:
  - *namcouple* should be mostly backwards compatable, but several things are NOT used or supported and are ignored
    - INIDATE is not used yet
    - CALTYPE is not used
    - AUXILARY is not used
    - ONCE time transformation not supported
    - REDGLO not supported
    - INVERT not supported
    - MASK not supported
    - EXTRAP not supported
    - CORRECT not supported
    - INTERP not supported
    - MOZAIC not supported
    - FILLING not supported
    - SUBGRID not supported
    - MASKP not supported
    - REVERSE not supported
    - GLORED not supported
  - BLASOLD and BLASNEW only supports multiplication and addition terms to the fields, not field combinations.
  - IGNORED and INGOUT no longer needed, redundant with EXPORTED and EXPOUT use of IGNORED and INGOUT converts to EXPORTED and EXPOUT respectively
  - NLOGPRT sets debugging output levels for the coupling software and will lead to output to all log files.
  - Non-INSTANT LOCTRANS options will use a restart file
  - Fields of type OUTPUT now support a restart file argument to be compatable with LOCTRANS restart files
  - SEQ setting has a slightly different meaning. SEQ is no longer needed to ensure correct coupling sequencing. Sequencing is dependent only on the order of get and put calls in models.

SEQ is never required. Use of SEQ allows the coupling layer to detect potential deadlocks before they happen and to exit gracefully. Use of SEQ is recommended when it makes sense to do so.

- There is a new MAPPING transform that allows use of previously generated mapping file for interpolation to be read in.
- Support to couple multiple fields via a single communication. This is supported through colon delmited field lists in the *namcouple* input, for example ATMTAUX:ATMTAUY:ATMHFLUX:ATMPREC TAUX:TAUY:HEATFLUX:FWFLUX in a single *namcouple* definition. All fields will use the *namcouple* settings for that definition. The prism_get and prism_put interfaces still only support sending (put) or receiving (get) one field at a time. Inside oasis, the fields are stored and and a single mapping and send or recieve is executed for all fields. This is useful in cases where multiple fields have the same coupling transforms or to reduce communication costs by aggregating multiple fields into a single communication.
- SCRIPR map generation supports a subset of options including BILINEAR, first order CONSERV, DISTWGT, and GAUSSWGT. No second order mapping is supported.

B.2 Changes between `oasis3_3` and `oasis3_prism_2_5`

The changes between version `oasis3_3` and version `oasis3_prism_2_5` delivered in September 2006 are the following:

- Bug corrections:
  - In `oasis3/lib/scrip/src/remap_bilinear.f, remap_bicubic.f, remap_bilinear_reduced.f, remap_bicubic_reduced.F90`: (r2084) we observed a wrong behaviour of routines remap_bilinear.f and remap_bicubic.f on the NEC SX9 when compiled with NEC SX compiler revision 400. We got round this problem by adding an explicit instruction to prevent the vectorisation of one loop.

    As remap_bilinear_reduced.f and remap_bicubic_reduced.F90 have a very similar loop, we introduced the same instruction in these routines, although nothing specific was observed with them.
  - `oasis3/lib/scrip/src/remap_bicubic_reduced.F90`: (r1883) Two bugfixes: 1) Memory fault when a target point was falling on the before-last latitude circle or in the before-last latitude band of the source reduced grid; 2) Wrong neighbours for target points south of the before-last latitude circle (i.e in the last latitude band or Southern).
  - `oasis3/lib/psmile/src/mod_psmile_io.F90`: (r2380) correction to ensure that when `INVERT` is used, the corner latitudes and longitudes are also inverted (and not only the center latitudes and longitudes as before).
  - `oasis3/lib/scrip/src/scriprmp.F`: (r1547) the calculation of the average for the line of points at the pole was bugged and did not have any effect. It is now debugged but commented.
  - In `oasis3/lib/scrip/src/vector.F90`:

    - correction of wrong sequences in declarations, at least for Intel & NAG compilers (thanks to L. Kornblueh from MPI); bug fix announced to the mailing list diff-oasis@cerfacs.fr on 31/10/2006.

    - (r1698 - 2008-08-20) bugfix to make sure that OASIS does not automatically calculates corners of target grid as this calculation is correct only for LR grids and target grid type is not known (thanks to S. Calmanti from Météo-France)

    - Modifications so that last 4 arguments of call to grid_init are always arrays even when corners are not defined (error detected with Intel Fortran V10.1.012 by Mike Rezny, SGI, Australia)

- In oasis3/lib/clim.GSIP/src/CLIM_Init_Oasis.F, correction of a wrongly positioned #endif (thanks to L. Kornblueh from MPI); bug fix announced to the mailing list diff-oasis@cerfacs.fr on 31/10/2006.
- In oasis3/lib/psmile/src/prism_enddef_proto.F, the call to MPI_Errhandler_set was moved after the test on the value of mpi_err returned by MPI_Buffer_Detach (thanks to I. Bethke from NERSC); bug fix announced to the mailing list diff-oasis@cerfacs.fr on 14/12/2006.
- Routine oasis3/lib/psmile/src/prism_terminate_proto.F90 was modified to ensure proper deallocation of all allocated arrays (thanks to Adam Ralph from ICHEC)
- In oasis3/lib/scrip/remap_conserv.F, small bugfix having no impact on the results, it just avoids misleading messages of type "Error sum wts map1:grid2_add ..." to be printed in the cplout log file.
- In oasis3/src/getfld.F, givfld.F, driver.f and closerst.F, correction of a bug observed by A.Caubel from CEA for coupling fields having a sequence index greater than 1. For first iteration, closing of netcdf restart files is done in driver.f by calling new routine closerst.F. Closing is therefore removed from getfld.F. A minor correction (useless opening and closing of first netcdf restart file) was also added to givfld.F. Bug fix announced to the mailing list diff-oasis@cerfacs.fr on 09/02/2006.
- In oasis3/src/inipar.F, bugfix for SEQMODE greater than 9 (thanks to T. Silva, Oregon State U.) and to avoid array overbound.
- In oasis3/util/make_dir/TopMakefileOasis3: typo error: libmpp_io.a instead of libmppio.a (thanks to J.M. Epitalon from CERFACS)
- In oasis3/lib/psmile/src/prism_init_comp_proto.F: initialisation of iprcou (bug fix thanks to J.M. Epitalon).
- In oasis3/src/filling.f: rewind of file in the "AN" case (thanks to S. Calmanti from Météo-France)
- In oasis3/lib/psmile/src/mod_prism_put_proto.F90: bug fix to avoid array overbounds (bug identified T. van Noije from KNMI).
- In oasis3/lib/psmile/src/mod_psmile_io.F90: thanks to A. Caubel from CEA, modification so that the grid point longitudes and latitudes do not have to appear before the corner longitudes and latitudes in the grids.nc file (revision 13/01/2009).
- Bugfix in oasis3/examples/testNONE/PROG/calc_errorfield.f90 to use the absolute value of the error in the non masked point mean error calculation.
- Modifications in oasis3/lib/clim/src/CLIM_Init_Oasis.F and oasis3/lib/psmile/src/prism_init_comp_proto.F to allow communication log file (*.prt* files) for OASIS and/or component models to run on up to 9999 processes.

- Other major modifications
  - New directory structure. See section 3 for details.
  - Modification of many routines to allow using more than one OASIS3 executable in a coupled model resulting in pseudo-parallelisation of OASIS3 on a field-per-field basis. See section **??** for more detail.
  - New directory structure and update of compiling environement to work with the new directory structure. In particular, the location of directory created for compilation (see ARCHDIR in the Makefile headers make.xxx in oasis3/util/make_dir) can be arbitrarily chosen by the user.
  - Modification of routine oasis3/lib/scrip/src/remap_distwgt.F so that SCRIPR/DISTWGT that has by default the same behaviour than SCRIPR/BILINEAR, /BICUBIC and /CONSERV (with FRACNNEI option) i.e. the non-masked nearest neighbour is used

for target grid points having their N nearest neighbour all masked. To reproduce the previous default behaviour, one has to compile with CPP key NOT_NNEIGHBOUR. See section 6.3 for details.

- Inclusion of an additional number below the $NFIELDS keyword in the *namcouple* (after the total number of fields exchanged, on the same line). This number, corresponding to the maximum number of prism_def_var_proto called by ANY component model in the coupled system, is needed only if it is greater than twice the number of fields listed in the *namcouple*; this may be the case if OASIS3 is used in pseudo-parallel mode or if fields declared with prism_def_var_proto call (and corresponding to sending - prism_put_proto call- or receiving - prism_get_proto call - actions in the component models) do not appear in the *namcouple* (in this case, the sending and receiving calls simply return without any action performed).
- added options GLBPOS, BASPOS, BASBAL for the cooking stage transformation CONSERV. For details, see section 6.4.
- New CPP key TREAT_OVERLAY : to ensure that if two cells of the source grid overlay, at least the one with the greater numerical index is masked (they also can be both masked). 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 ensures that these rules are respected. This is mandatory for SCRIPR/CONSERV remapping, see section 6.3.
- Optimisation of SCRIPR (XXX to be detailed) interpolation weigths-and-address files, thanks to CMCC. The weights and addresses are now read once per run and stored.
- mode stats consommation Eric M (XXX to be detailed)
- mod_prism_get_comm (XXX to be detailed)
- Release of a new toy coupled model TOYOASIS3. This new toy model is described in 8.2. The toy model sources are available in oasis3/src/mod/oceoa3/, atmoa3, cheoa3. It running environment is available in oasis3/src/mod/oasis3/examples/toyoasis3. The grids of the TOYOASIS3 component models and the interpolation performed have been updated compared to the previous toy coupled model TOYCLIM (which is no longer distributed with the official release).
- In oasis3/src/mod/oasis3/src/extrap.F: optimisation (thanks to T. Schoenemeyer from NEC) and idoitold changed from 1000000 to 10000000 to support higher resolution grids (thanks to E.Maisonnave from CERFACS). XXX and CMCC
- In oasis3/lib/psmile/src/mod_prism_grids_writing.F90:

  - routine prism_write_angle was added to to allow a component model to write the angle of its grid (see section 4.3 for more detail).

  - changed logical netcdf for l_netcdf (to run on NEC SX9, thanks to E. Maisonnave, CERFACS)
- In may 2007, we moved from CVS to SVN for source management.

- Other modifications
  - The names of the log files for the communication information **.prt** are now always ending with 4 digits indicating the rank of the component process (e.g. model1.prt0002 or model1.prt9999) or the rank of the oasis process (e.g. Oasis.prt0001 or Oasis.prt0010). These modifications impacted routines oasis3/lib/clim/src/CLIM_Init_Oasis.F and oasis3/lib/psmile/src/prism_init_comp_proto.F.
  - The following routines were modified for the NAG compiler: oasis3/src/getfld.F, inipar.F, interp.F and oasis3/lib/scrip/src/remap_write.F

- Some routines in oasis3/lib/mpp_io/src/ were modified so that all debug and log messages are now written to stdout unit and to include modifications done in the OASIS4 version for CRAY pointers and for bundles.
- Added CPP key __SILENT CPP key to reduce log outputs to .prt files during the psmile library exchanges (thanks to S. Lorenz from MPI)
- In oasis3/src/chkfld.f, modified misleading comment about masked points written to *cplout* log file (thanks to T. Craig from BOM).
- In oasis3/lib/psmile/src/mpp_io/src/mpp_io_mod_oa.F90 modified "lowercase" function so to avoid using "transfer" function (thanks to Mike Rezny for SGI Melbourne) which causes problem with pgf90 5.2.4, 6.1.3 or 7.0 in 64 bit mode, and with gfortran 4.2.1 and 4.2.5 SUSE Linux.

B.3 Changes between oasis3_prism_2_5 and oasis3_prism_2_4

The changes between version oasis3_prism_2_5 and version oasis3_prism_2_4 delivered in December 2004 are listed here after. Please note that those modifications should not bring any difference in the interpolation results, except for SCRIPR/DISTWGT (see below).

- Bug corrections:
  - In prism/src/lib/scrip/src/scriprmp.F: initialisation of dst_array(:); bug fix announced to the mailing list diff-oasis@cerfacs.fr on 02/02/2006.
  - In prism/src/lib/psmile/src/prism_enddef_proto.F and prism/src/lib/clim/src/CLIM_Start_MPI.F: the call to MPI_barrier (that created a deadlock when not all processes of a component model were exchanging data with the coupler) was changed for a call to MPI_wait on the previous MPI_Isend; bug fix announced to the mailing list diff-oasis@cerfacs.fr on 02/23/2006.
  - For SCRIPR/DISTWGT, in prism/src/lib/scrip/src/remap_distwgt.f: line 190 was repeated without epsilon modification; bug fix announced to the mailing list diff-oasis@cerfacs.fr on 03/21/2006.
  - In prism/src/lib/psmile/src/mod_prism_put_proto.F90, for prism_put_proto_r28 and prism_put_proto_r24, the reshape of the 2d field was moved after the test checking if the field is defined in the *namcouple* (thanks to Arnaud Caubel from LSCE).
- Modification in SCRIP interpolations
  - For SCRIPR interpolations (see section 6.3), the value 1.0E+20 is assigned to target grid points for which no value has been calculated if prism/src/lib/scrip/src/scriprmp.f or vector.F90 (for vector interpolation) are compiled with ll_weightot = .true..
  - For SCRIPR/GAUSWGT: if routine prism/src/lib/scrip/src/remap_gauswgt.f is compiled with ll_nnei=.true., the non-masked nearest neighbour is used for target point if all original neighbours are masked (see section 6.3).
  - For SCRIPR/BICUBIC (routine prism/src/lib/scrip/src/remap_bicubic.f), the convergence criteria was modified so to ensure convergence even in single precision.
  - For SCRIPR/CONSERV (routine prism/src/lib/scrip/src/remap_conserv.f), a test was added for non-convex cell so that integration does not stall.
  - The routine prism/src/lib/scrip/src/corners.F was modified so to abort if it is called for the target grid, as the automatic calculation of corners works only for Logically-Rectangular (LR) grids and as the target grid type is unknown. If needed, the reverse remapping, in which the current target grid become the source grid, can be done .
- Other important modifications

  - A new PSMILe routine `prism/src/lib/psmile/src/prism_get_freq.F` was added; this routine can be used to retrieve the coupling period of field (see section **??**).
  - The routines of the `mpp_io` library in `prism/src/lib/mpp_io` changed name and were merged with the OASIS4 `mpp_io` library.
  - Routine `prism/src/mod/oasis3/src/extrap.F` was modified to ensure that the extrapolation works even if the `MASK` value is very big (thanks to J.M. Epitalon).
  - In the *namcouple*, there is no need anymore to define a lag greater than 0 (e.g. LAG=+1) for fields in mode NONE.
  - Diverse modifications were included for successful compilation with NAGW compiler: non portable use of "kind", etc. (thanks to Luis Kornblueh from MPI).
  - In `prism/src/lib/psmile/mod_prism_get_proto.F90`, a potential deadlock was removed (the master process was sending a message to itself)(thanks to Luis Kornblueh from MPI).
  - Routine `prism/src/lib/scrip/src/scriprmp_vector.F90` was completely rewritten for more clarity.
  - Obsolete transformations INVERT and REVERSE were removed from the toy coupled model TOYCLIM (in file `prism/util/running/toyclim/input/namcouple`. This change does not affect the statistics printed in the `cplout` but changes the orientation of some fields in the NetCDF ouput files (see the results in `prism/data/toyclim/outdata`).

- Other minor modifications:

  - In `prism/src/lib/psmile/src/prism_enddef_proto.F`, allocation is done only for rg_field_trans or dg_field_trans depending on precision for REAL (but not for both, to save memory).
  - In few routines in `prism/src/lib/clim` and in `prism/src/mod/oasis3`, parentheses were added to make sure that && has priority over || in CPP instructions (thanks to A. Caubel from LSCE).
  - Routines `scrip/src/corners.f`, `netcdf.f`, and `scriprmp.f` were renamed `corners.F`, `netcdf.F`, `scriprmp.F` and the line "INCLUDE 'netcdf.inc' " was changed for "#include <netcdf.inc> "

B.4 Changes between `oasis3_prism_2_4` and `oasis3_prism_2_3`

The changes between versions tagged `oasis3_prism_2_4` and `oasis3_prism_2_3` delivered in July 2004 are the following:

- Update of compiling and running environments with version `prism_2-4` of PRISM Standard Compiling Environment (SCE) and PRISM Standard Running Environment (SRE), which among other improvements include the environments to compile and run on the CRAY X1 (see the directories with `<node>=baltic1`), thanks to Charles Henriet from CRAY France, and on a Linux station from Recherche en Prévision Numérique (Environnement Canada, Dorval, Canada) (see the directories with `<node>=armc28`).
- `prism/src/mod/oasis3/src/iniiof.F`: the opening of the coupling restart files is done only if the corresponding field has a lag greater than 0; note that this implies that all fields in mode NONE must now have a lag greater than 0 (e.g. LAG=+1) (thanks to Veronika Gayler from M&D).
- `prism/src/lib/psmile/src/prism_def_var_proto.F`: contrary to what was previously described in the documentation, `PRISM_Double` is not supported as 7^{th} argument to describe the field type; `PRISM_Real` must be given for single or double precision real arrays.

- prism/src/mod/oasis3/src/inipar.F90: For upward compatibility of SCRIPR interpolation, "VECTOR" is still accepted in the *namcouple* as the field type and leads to the same behaviour as before (i.e. each vector component is treated as an independent scalar field). To have a real vector treatment, one has to indicate "VECTOR_I" or "VECTOR_J" (see section 6.3).
- Bug corrections in:
  - prism/src/lib/scrip/src/scriprmp_vector.F90: In some cases, some local variables were not deallocated and variable dimid was declared twice.
  - prism/src/lib/psmile/src/mod_psmile_io.F90: correct allocation of array hosting the longitudes (thanks to Reiner Vogelsang from SGI Germany).
  - prism/src/lib/psmile/src/write_file.F90: to remove a deadlock on some architecture (thanks to Luis Kornblueh from MPI).
  - prism/src/lib/psmile/src/prism_enddef_proto.F: the error handler is now explicitely set to MPI_ERRORS_RETURN before the call to MPI_Buffer_Detach to avoid abort on some architecture when the component model is not previously attached to any buffer (thanks to Luis Kornblueh from MPI).
  - prism/src/lib/scrip/src/remap_conserv.f (thanks to Veronika Gayler from M&D).
  - prism/src/mod/oasis3/src/inicmc.F
  - prism/src/lib/scrip/src/remap_distwgt.f

B.5 Changes between oasis3_prism_2_3 and oasis3_prism_2_2

The changes between versions tagged oasis3_prism_2_3 delivered in July 2004 and oasis3_prism_2_2 delivered in June 2004 are the following:

- Bug correction of the previous bug fix regarding ordering of grid and data information contained in I/O files when INVERT or REVERSE transformations are used: the re-ordering now occurs only for source field if INVERT is used, and only for target field if REVERSE is used.
- LGPL license: OASIS3 is now officially released under a Lesser GNU General Public License (LGPL) as published by the Free Software Foundation (see prism/src/mod/oasis3/COPYRIGHT and prism/src/mod/oasis3/src/couple.f)
- Upgrade of compiling and running environments: The compiling and running environments have been upgraded to the PRISM Standard Compiling and Running Environment version dated August 5th 2004, that should be very close to "prism_2-3".
- Treament of vector fields: The interpolation algorithms using the SCRIP library now support vector fields, including automatic rotation from local to geographic coordinate system, projection in Cartesian coordinate system and interpolation of 3 Cartesian components, and support of vector components given on different grids. New routines have been added in prism/src/lib/scrip/src: scriprmp_vector.F90 and rotations.F90. For more detail, see SCRIPR in section 6.3.
- All include of mpif.h are now written '#include <mpif.h>'.
- The output format of CHECKIN and CHECKOUT results is now E22.7

B.6 Changes between oasis3_prism_2_2 and oasis3_prism_2_1

The changes between versions tagged oasis3_prism_2_2 delivered in June 2004 and oasis3_prism_2_1 delivered to PRISM in April 2004 are the following:

- Bug corrections
  - INTERP/GAUSSIAN and SCRIPR/GAUSWGT transformations work for 'U' grids.

  - The grid and data information contained in I/O files output by the PSMILe library have now a coherent ordering even if `INVERT` or `REVERSE` transformations are used.

- OASIS3 and the TOYCLIM coupled model are ported to IBM Power4 and Linux Opteron, which are now included in the Standard Compiling and Running Environments (SCE and SRE).
- SIPC technique communication is re-validated.
- `Clim_MaxSegments = 338` in `prism/src/lib/clim/src/mod_clim.F90` and in `prism/src/lib/ps` 338 is presently the largest value needed by a PRISM model.
- `MPI_BSend`: below the call to `prism_enddef_proto`, the PSMILe tests whether or not the model has already attached to an MPI buffer. If it is the case, the PSMILe detaches from the buffer, adds the size of the pre-attached buffer to the size needed for the coupling exchanges, and reattaches to an MPI buffer. The model own call to `MPI_Buffer_Attach` must therefore be done before the call to `prism_enddef_proto`. Furthermore, the model is not allowed to call `MPI_BSend` after the call to `prism_terminate_proto`, as the PSMILe definitively detaches from the MPI buffer in this routine. See the example in the toyatm model in `prism/src/mod/toyatm/src`.

B.7 Changes between `oasis3_prism_2_1` and `oasis3_prism_1_2`

The changes between versions tagged `oasis3_prism_1_2` delivered in September 2003 and `oasis3_prism_2_1` delivered to PRISM in April 2004 are the following:

- Bug corrections
  - Thanks to Eric Maisonnave, a bug was found and corrected in /prism/src/lib/scrip/src/scriprmp.f: "sou_mask" and "tgt_mask" were not properly initialised if weights and addresses were not calculated but read from file.
  - Some deallocation were missing in prism_terminate_proto.F ("ig_def_part", "ig_length_part", "cg_ignout_field").
  - Thanks to Arnaud Caubel, a bug was found and corrected in /prism/src/lib/psmile/src/write_file.F90. In case of parallel communication between a model and OASIS3 main process, the binary coupling restart files were not written properly (NetCDF coupling restart files are OK).
- Routines renamed

  The routines `preproc.f, extrap.f, iniiof.f` in `prism/src/mod/oasis3/src` were renamed to `preproc.F, extrap.F, iniiof.F`, as a CPP key 'key_openmp' was added. Please note that this key, allowing openMP parallelisation, is not fully tested yet.
- Modifications in the *namcouple*
  - The third entry on the field first line now corresponds to an index in the new auxiliary file *cf_name_table.txt* (see sections 5.3 and 7.1).
  - For `IGNORED, IGNOUT` and `OUTPUT` fields, the source and target grid locator prefixes must now be given on the field second line (see section **??**)
- A new auxiliary file *cf_name_table.txt*

  For each field, the CF standard name used in the OASIS3 log file, *cplout*, is now defined in an additional auxiliary file *cf_name_table.txt* not in `inipar.F` anymore. This auxiliary file must be copied to the working directory at the beginning of the run. The user may edit and modify this file at her own risk. In *cf_name_table.txt*, an index is given for each field standard name and associated units. The appropriate index has to be indicated for each field in the *namcouple* (third entry on the field first line, see section 5.3).

  This standard name and the associated units are also used to define the field attributes "long_name" and "units" in the NetCDF output files written by the PSMILe for fields with status `EXPOUT, IGNOUT` and `OUTPUT`.

For more details on this auxiliary file, see section 7.1.

- Many timesteps for mode NONE

  In mode NONE, OASIS3 can now interpolate at once all time occurrences of a field contained in an input NetCDF file. The time variable in the input file is recognized by its attribute "units". The acceptable units for time are listed in the udunits.dat file (3). This follows the CF convention.

  The keyword $RUNTIME in the *namcouple* has to be the number of time occurrences of the field to interpolate from the input file. The "coupling" period of the field (4th entry on the field first line) must be always "1". Note that if $RUNTIME is smaller than the total number of time ocurrences in the input file, the first $RUNTIME occurrences will be interpolated.

  For more details, see section **??**.

- Model grid data file writing

  The grid data files *grids.nc, masks.nc* and *areas.nc* can now be written directly at run time by the component models, if they call the new routines prism_start_grids_writing, prism_write_grid, prism_write_corner prism_write_mask, prism_write_area, prism_terminate_grids_writing.

  The writing of those grid files by the models is driven by the coupler. It first checks whether the binary file *grids* or the netCDF file *grids.nc* exists (in that case, it is assumed that *areas* or *areas.nc* and *masks* or *masks.nc* files exist too) or if writing is needed. If *grids* or *grids.nc* exists, it must contain all grid information from all models; if it does not exist, each model must write its grid informations in the grid data files.

  See section 4.3 for more details.

- Output of CF compliant files

  The NetCDF output files written by the PSMILe for fields with status EXPOUT, IGNOUT and OUTPUT are now fully CF compliant.

  In the NetCDF file, the field attributes "long_name" and "units" are the ones corresponding to the field index in *cf_name_table.txt* (see above and section 7.1). The field index must be given by the user as the third entry on the field first line in the *namcouple*.

  Also, the latitudes and the longitudes of the fields are now automatically read from the grid auxiliary data file *grids.nc* and written to the output files. If the latitudes and the longitudes of the mesh corners are present in *grids.nc*, they are also written to the ouput files as associated "bounds" variable. This works whether the *grids.nc* is given initially by the user or written at run time by the component models (see above). However, this does not work if the user gives the grid definition in a binary file *grids*.

- Removal of pre-compiling key "key_BSend"

  The pre_compiling key "key_BSend" has been removed. The default has changed: by default, the buffered MPI_BSend is used, unless NOBSEND is specified in the *namcouple* after MPI1 or MPI2, in which case the standard blocking send MPI_Send is used to send the coupling fields.

Appendix C

The coupled models realized with OASIS

Here is a list of (some of) the coupled models realized with OASIS within the past 5 years or so in Europe and in other institutions in the world: XXX to be updated XXX

Lab	Cnt	Vrs	Atm	Oce	Comp
Environment Canada	Canada	3.0	MEC	GOM	IBM Power4
IRI	USA	2.4	ECHAM4	MOM3	SGI Origin IBM Power3
JPL(NASA)	USA	2.4	QTCM	Trident	SGI
JAMSTEC	Japan	2.4	ECHAM4	OPA 8.2	ES SX5
U. of Tasmania	Aus-tral.	3.0	Data atm. model	MOM4	SGI O3400 Compaq
BMRC	Aus-tral.	3.0 2.4	BAM4 BAM3 T47L34	MOM4 ACOM2 180X194X25	
CAS-IIT	India	3.0	MM5	POM	
IAP-CAS	China		AGCM	LSM	

Table C.1: List of couplings realized with OASIS within the past 5 years in institutions outside Europe . The columns list the institution, the country, the OASIS version used, the atmospheric model, the ocean model, and the computing platform used for the coupled model run.

Lab	Cnt	Vrs	Atm	Oce	Comp
IPSL	Fr	3.0	LMDz 96x71x19	ORCA2 182x149x31	SX6
			+ ORCH/INCA	+ LIM	
		2.4	LMDz 96x71x19	ORCA2 182x149x31	VPP5000
		2.4	LMDz 72x45x19	ORCA4 92x76x31	VPP5000
		2.4	LMDZ 120X90X1	OPA ATL3 1/3 deg	
		2.4	LMDZ 120X90X1	OPA ATL1 1 deg	
Lodyc-ISAO	Fr,It	2.3	ECHAM4 T30/T42 L14	ORCA2 182x149x31	SX4,SX5
Météo-Fr	Fr	3.0	ARPEGE 4	ORCA2	VPP5000
		2.4	ARPEGE medias	OPA med 1/8e	VPP5000
		2.2	ARPEGE 3	OPA 8.1 + Gelato	VPP5000
Mercator	Fr	3.0	interp. mode	PAM (OPA)	
CERFACS	Fr	3.0	ARPEGE 4	OPA9/NEMO	VPP5000
					CRAY XD1
					PC Linux
		2.4	ARPEGE 3	ORCA2-LIM	VPP5000
		2.2	ARPEGE 3	OPA 8.1	VPP700
ECMWF	UK	2.2	IFS T63/T255	E-HOPE 2deg/1deg	IBM Power 4
		2.2	IFS Cy23r4 T159L40	E-HOPE 256L29	VPP700
		2.2	IFS Cy23r4 T95L40	E-HOPE 256L29	VPP700
MPI	Ger-	3.0	ECHAM5	MPI-OM	IBM Power4
	ma-	2.4	ECHAM5 T42/L19	C-HOPE T42+L20	NEC-SX
	ny	2.4	PUMAT42/L19	C-HOPE 2deg GIN	NEC-SX
		2.4	EMAD	E-HOPE T42+L20	CRAY C-90
		2.4	ECHAM5 T42/L19	E-HOPE T42+L20	NEC-SX
IFM-GEOMAR	D	3.0	ECHAM5	NEMO	
CGAM	UK	3.0	HadAM3 2.5x3.75 L20	ORCA2 182x149x31	NEC SX6
		2.4	HadAM3 2.5x3.75 L20	ORCA 182x149x31	T3E
SMHI	Sw	3.0	ECHAM-RCA(reg.)		SGI O3800
		2.3	RCA-HIRLAM (reg.)	RCO-OCCAM (reg.)	
INGV	It	3.0	ECHAM5	MPIOM	NEC SX6
KNMI	Nl	3.0	ECHAM5	MPIOM	SGI IRIX64
DMI	Dk	3.0	ECHAM (glob.)		NEC SX6
U.Bergen	Nw	3.0	MM5	ROMS	
NERSC	Nw		ARPEGE	MICOM	

Table C.2: List of couplings realized with OASIS within the past 5 years in Europe. The columns list the institution, the country, the OASIS version used, the atmospheric model, the ocean model, and the computing platform used for the coupled model run.

Bibliography

[1] http://gcmd.nasa.gov/records/LANL-SCRIP.html

[2] http://www.gfdl.noaa.gov/~ vb/mpp_io.html

[3] http://www.unidata.ucar.edu/packages/udunits/udunits.dat

[4] S. Valcke, 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.

[5] S. Valcke, 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.

[6] S. Valcke, 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.

[7] L. Terray, 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.

[8] C. Cassou, 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.

[9] L. Terray, 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

[10] E. Guilyardi, 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.

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

[12] M. Pontaud, 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.

[13] L. Terray, 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.

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

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

[16] L. Terray, 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.

[17] J. Larson, R. Jacob, and E.Ong The Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models, *International Journal of High Performance Computing Applications*, 19(3):277-292, 2005.