OASIS3 User Guide

oasis3_3

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

CERFACS TR/CMGC/XX/XX

May 2010

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 15 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), into which the parallel OASIS4 version of the coupler is also currently being developed.

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 separate executable, which main function is to interpolate the coupling fields exchanged between the component models, and as a communication library linked to the component models, the OASIS3 PRISM Model Interface Library (PSMILe). OASIS3 supports 2D coupling fields in the longitude and latitude dimensions. To communicate with OASIS3, or directly with another model, or to perform I/O actions, a component model needs to include few specific PSMILe calls. OASIS3 PSMILe supports in particular parallel communication between a parallel component model and OASIS3 interpolation executable based on Message Passing Interface (MPI) and file I/O using the mpp_io library from GFDL. New with this version, the OASIS3 interpolation executable can be run on many processes, each process interpolating a subset of the coupling fields, resulting in a parallelisation of OASIS3 on a field-per-field basis. For each coupling exchange, OASIS3 performs the transformations and regridding needed to express the source field on the grid of the target model. The current OASIS3_3 version and its toy coupled model TOYOASIS3 were compiled and run on NEC SX6, IBM Power4, CRAY XD1, and Linux PC (XXX list to be updated).

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 sources (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 or MPI2 message passing[1]. The PSMILe library is interfaced with the `mpp_io` library from GFDL (2) 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 TOYOASIS3 coupled model gives a practical example of a coupled model; the sources are given

[1] The SIPC, PIPE and GMEM communication techniques available in previous versions are not maintained anymore.

in directories `/oasis3/examples/toyoasis3/src` ; more detail on TOYOASIS3 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. We strongly recommend that one tests off-line the quality of the chosen transformations and regriddings in the "testNONE" environment (see section 8.4.2).

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/src              OASIS3 main code

- oasis3/lib/anaisg       GAUSSIAN interpolation library
             /anaism      SURFMESH interpolation library
             /clim        CLIM/MPI1-MPI2 communication library
             /fscint      INTERP interpolation library
             /mpp_io      I/O library
             /NAG_dummies Dummy library for NAG compiler
             /psmile      PRISM System Model Interface Library
             /scrip       SCRIPR interpolation library

- oasis3/doc              OASIS3 documentation

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

- oasis3/examples/toyoasis3  environment to run the TOYOASIS3 toymodel
                             (see section 8.3)
                  /testinterp environment to test few predefined OASIS3
                             interpolations (see section 8.4.1)
                  /testNONE  environment to evaluate the quality of
                             one's interpolation (see section 8.4.2)
                  /toysimple environment to test a ping pong exchange
                             between two coupled models
                  /tutorial  training to learn how to use OASIS3 step
                             by step by interfacing the two toy models
                             with the Psmile library.
```

3.2 License and Copyright

3.2.1 OASIS3 license and copyright statement

3.2.2 The SCRIP 1.4 license copyright statement

The SCRIP 1.4 copyright statement reads as follows:

Chapter 4

Interfacing a model with the PSMILe library

At run-time, OASIS3 acts as a separate executable which drives the coupled run, interpolates and transforms the coupling fields. To communicate with OASIS3 or directly between the component models, different communication techniques have been historically developed. The technique used for one particular run is defined by the user with the keyword $CHANNEL in the configuration file *namcouple* (see chapter 5). In OASIS3, the CLIM communication technique based on MPI1 or MPI2 message passing and the associated model interface library PSMILe, should be used as the SIPC, PIPE and GMEM communication techniques from previous versions are not maintained anymore. For a practical toy model using the PSMILe library, see the sources in `/oasis3/examples/toyoasis3/src` and more details in chapter 8.

To communicate with OASIS3 or directly with another component model using the CLIM-MPI1/2 communication technique, or to perform I/O actions, a component model needs to be interfaced with the PRISM System Model Interface library, PSMILe, which sources can be found in `oasis3/lib/psmile` directory. PSMILe supports:

- parallel communication between a parallel component model and OASIS3 executable,
- direct communication between two parallel component models when no transformations and no repartitioning are required,
- automatic sending and receiving actions at appropriate times following user's choice indicated in the *namcouple*,
- time integration or accumulation of the coupling fields,
- I/O actions from/to files.

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

1. Initialisation (section 4.1)
2. Grid data file definition (section 4.2)
3. Partition definition (section 4.3)
4. I/O-coupling field declaration (section 4.4)
5. End of definition phase (section 4.5)
6. I/O-coupling field sending and receiving (section 4.6)
7. Termination (section 4.7)

Finally, in section 4.8, different coupling algorithms are illustrated, and explanations are given on how to reproduce them with PSMILe by defining the appropriate indices of lag and sequence for each coupling field.

4.1 Initialisation

All processes of the component model initialise the coupling and, if required, retrieve a local communicator for the component model internal parallelisation.

- `USE mod_prism_proto`

  Module to be used by the component models.

- `CALL prism_init_comp_proto (compid, model_name, ierror)`

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

  Routine called by all component model processes, which initialises the coupling.[1]

- `CALL prism_get_localcomm_proto (local_comm, ierror )`

  - `local_comm [INTEGER; OUT]`: value of local communicator
  - `ierror [INTEGER; OUT]`: returned error code.

  If needed, routine called by all model processes to get the value of a local communicator to be used by the model for its internal parallelisation (CLIM-MPI1 communication technique only).

  With CLIM-MPI1, all component models started in a pseudo-MPMD mode share automatically the same MPI_COMM_WORLD communicator. Another communicator has to be used for the internal parallelisation of each model. OASIS3 creates this model local communicator based on the name of the calling model; its value is returned as the first argument of prism_get_localcomm_proto routine.

  With CLIM-MPI2, OASIS3 executable spawns the component model executables at the beginning of the run; the components keep their internal parallelisation context unchanged with respect to their standalone mode. In this case, calling the prism_get_localcomm_proto routine is useless but if called, the communicator MPI_COMM_WORLD will be returned as local communicator.

4.2 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 (except when OASIS3 is used in "IPSL" parallel mode, see section 8.2.1).

If written by the component models, the writing of those grid files is driven by OASIS3 main process. It first checks whether the binary file *grids* or the netCDF file *grids.nc* exists (if it is the case, it assumes 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; the file will not be completed or overwritten even if the following routines are explicitely called. If *grids* or *grids.nc* does not exist, each model must write its grid, mask and area definition in the grid data files with the following routines.

The coupler sends the information on whether or not writing is needed to the models following an OASIS3 internal order (below prism_start_grids_writing called by the component master process). If the grid data files already exist in the working directory, nothing happens when the component model calls the `prism_write_grid`, `prism_write_corner`, `prism_write_ang`, `prism_write_mask`, `prism_write_area` routines; the grid information is NOT overwritten in the grid files. If writing is needed, the first model creates the files, writes the data arrays when calling the appropriate routines, and then sends a termination flag to the coupler (below

[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`.

prism_terminate_grids_writing call). The coupler will then send the starting flag to the next model; this ensures that only one model accesses the files at a time.

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

As an example, see the TOYOASIS3 coupled model components that use these routines to write the grid data files (effective if gridswr=1 in the running script run_toyoasis3, see section 8.3).

- USE mod_prism_grids_writing

  Module to be used by the component model to call grid writing routines.

- CALL prism_start_grids_writing (flag)

  - flag [INTEGER; OUT]: returns 1 or 0 if grids writing is needed or not needed

  Initialisation of grids writing.

- CALL prism_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 6.4).

- CALL prism_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)

  - 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 6.3.

If called, prism_write_angle needs to be called after prism_write_grids.

- `CALL prism_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)`
  - `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()`

  Termination of grids writing. A flag stating that all needed grid information was written will be sent to OASIS3 main process.

4.3 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 can send directly its partition to OASIS3 interpolation executable, or directly to the other component model if no transformation and no repartition are required. To do so, each process exchanging coupling data has to define its local partition in the global index space.

- `USE mod_prism_def_partition_proto`

  Module to be used by the component model to call `prism_def_partition_proto`.
- `CALL prism_def_partition_proto (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.

4.3.1 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.

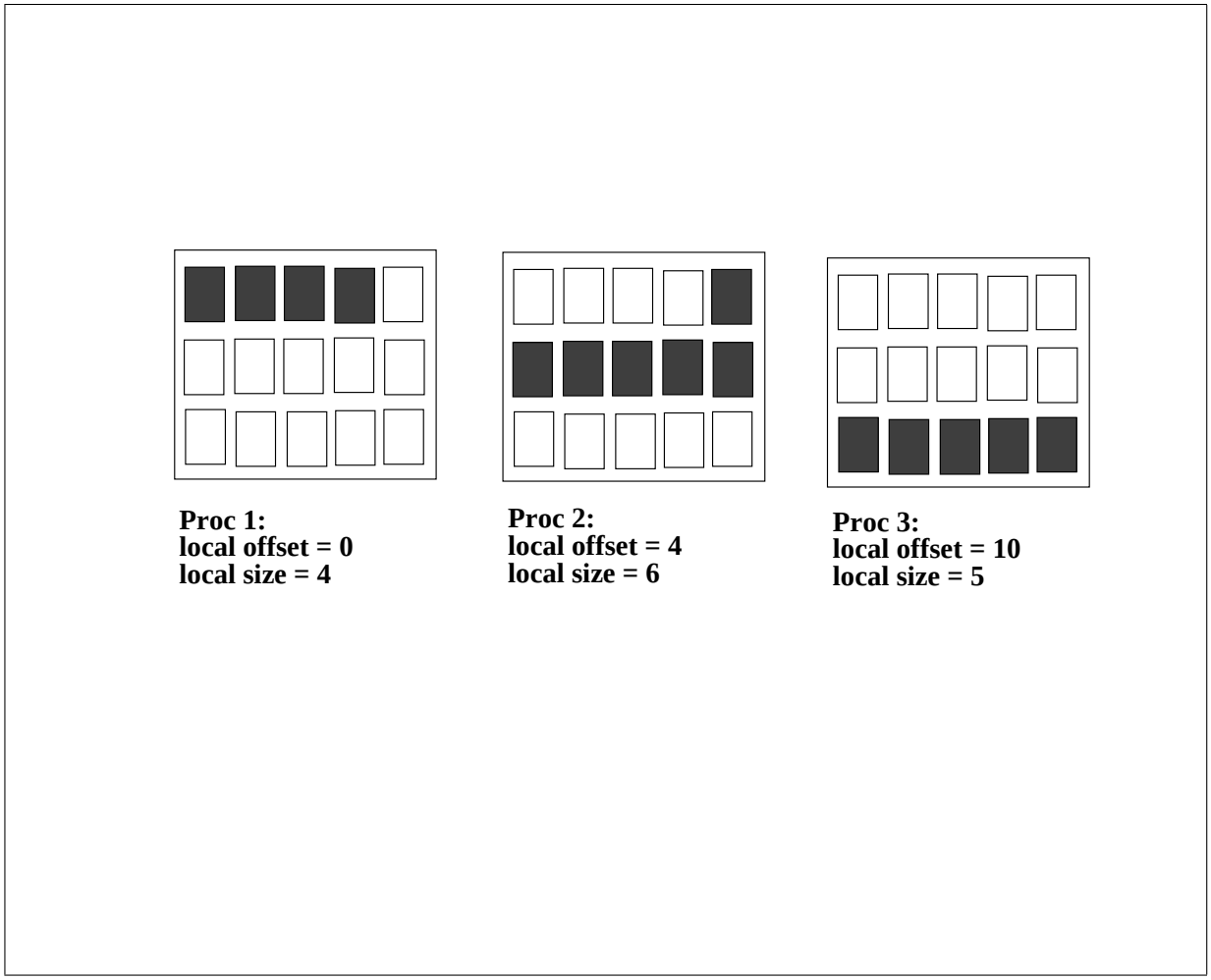

Figure 4.1: Apple partition. It is assumed here that the index start at 0 in the upper left corner.

4.3.2 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.

4.3.3 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[2]

[2]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

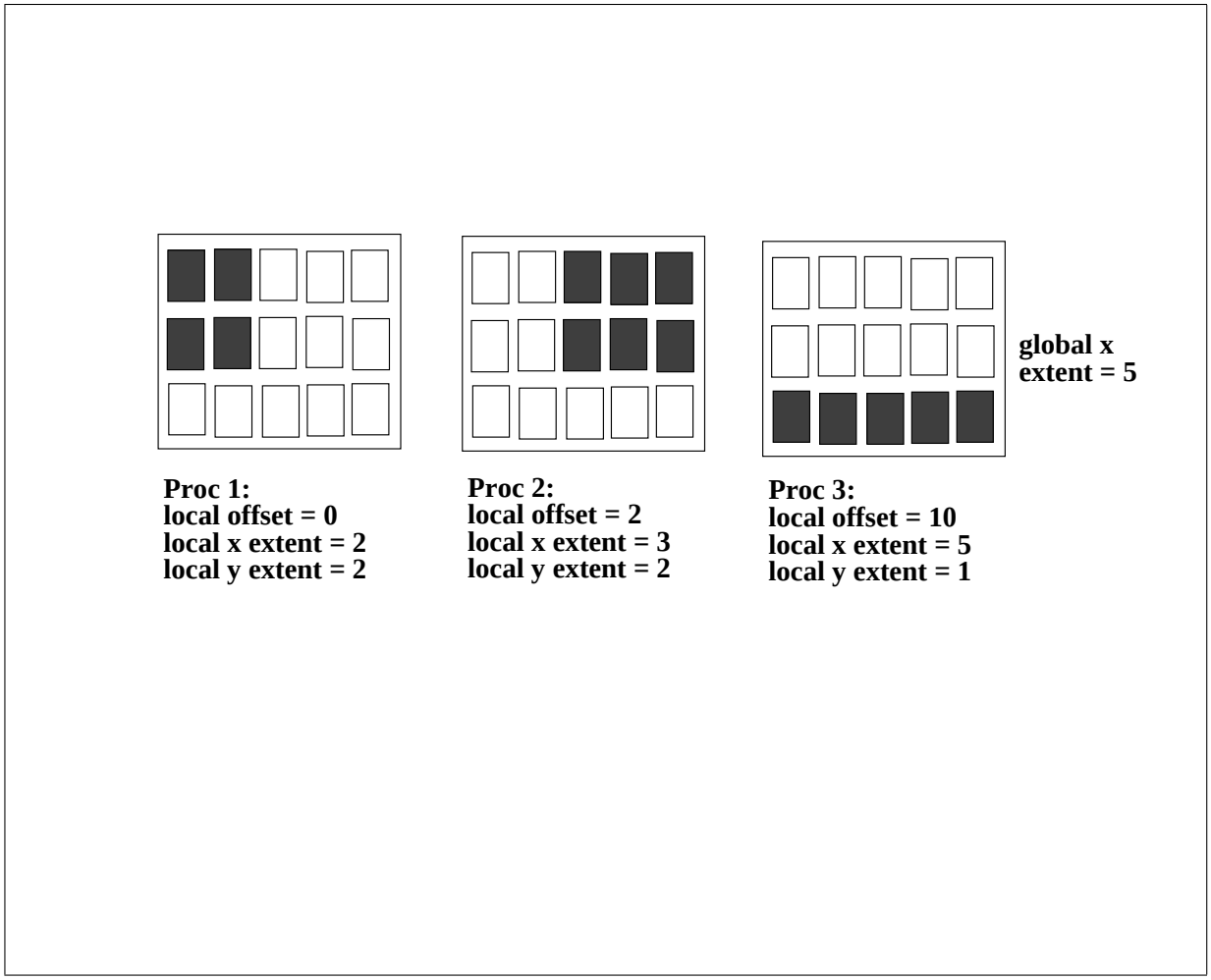

Figure 4.2: Box partition. It is assumed here that the index start at 0 in the upper left corner.

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

Figure 4.2 illustrates a Box partition over 3 processes.

4.3.4 Orange partition

WARNING: *I/O do not work for Orange partition; therefore, fields having an Orange partition cannot have* `EXPOUT`, `IGNOUT`, `INPUT`,`OUTPUT` *field status (see section 5.3).*

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`[3].

- `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

`oasis3/lib/psmile/src/mod_prism_proto.F90` and by recompiling OASIS3 and the PSMILe library.

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

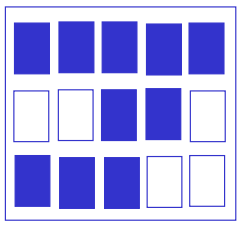

Figure 4.3: Orange partition for one process. It is assumed here that the index start at 0 in the upper left corner.

- ...
- `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.

4.4 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)`
  - `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[4]. Note that no automatic conversion is implemented; therefore, all coupling fields exchanged through OASIS3 main process must be of same type
- `ierror [INTEGER; OUT]`: returned error code.

4.5 End of definition phase

Each process exchanging coupling data closes the definition phase.

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

4.6 Sending and receiving actions

4.6.1 Sending a coupling field

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

- `USE mod_prism_put_proto`

  Module to be used by the component model to call `prism_put_proto`.
- `CALL prism_put_proto(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 i.e.
    - PRISM_Sent(=4) if the field was sent to another model (directly or via OASIS3 main process)
    - PRISM_LocTrans (=5) if the field was only used in a time transformation (not sent, not output)
    - PRISM_ToRest (=6) if the field was written to a restart file only
    - PRISM_Output (=7) if the field was written to an output file only
    - PRISM_SentOut (=8) if the field was both written to an output file and sent to another model (directly or via OASIS3 main process)
    - PRISM_ToRestOut (=9) if the field was written both to a restart file and to an output file.
    - PRISM_Ok (=0) otherwise and no error occurred.

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 4.8) 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.

[4]PRISM standard is to exchange coupling fields declared `REAL(kind=SELECTED_REAL_KIND(12,307))`. By default, all real variables are declared as such in OASIS3. To exchange single precision coupling fields, OASIS3 has to be compiled with the CPP key `use_realtype_single`, and the coupling fields must be declared `REAL(kind=SELECTED_REAL_KIND(6,37))` in the component models (see also chapter 8).

For a coupling field with a positive lag (see 4.8), 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. To force the writing of the field in its coupling restart file, one can use `prism_put_restart_proto` (see below).

This routine can use the buffered MPI_BSend (by default) or the standard send MPI_Send (if `NOBSEND` is specified in the namcouple -see `$CHANNEL` section 5.2) to send the coupling fields.

4.6.2 Receiving a coupling field

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

- `USE mod_prism_get_proto`
  Module to be used by the component model to call `prism_get_proto`.
- `CALL prism_get_proto(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, OUT]`: I/O or coupling field array
  - `info [INTEGER; OUT]`: returned info code
    * PRISM_Recvd(=3) if the field was received from another model (directly or via OASIS3 main process)
    * PRISM_FromRest (=10) if the field was read from a restart file only (directly or via OASIS3 main process)
    * PRISM_Input (=11) if the field was read from an input file only
    * PRISM_RecvOut (=12) if the field was both received from another model (directly or via OASIS3 main process) and written to an output file
    * PRISM_FromRestOut (=13) if the field was both read from a restart file (directly or via OASIS3 main process) and written to an output file
    * PRISM_Ok (=0) otherwise and no error occurred.

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.

4.6.3 Auxiliary routines

- `CALL prism_put_inquire(var_id, date, 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
  - `info [INTEGER; OUT]`: returned info code.

This routine may be called at any time to inquire what would happen to the corresponding field (i.e. with same `var_id` and at same `date`) below the corresponding `prism_put_proto`. The possible value of the returned info code are as for `prism_put_proto`:

- PRISM_Sent(=4) if the field would be sent to another model (directly or via OASIS3 main process)
- PRISM_LocTrans (=5) if the field would be only used in a time transformation (not sent, not output)
- PRISM_ToRest (=6) if the field would be written to a restart file only

- PRISM_Output (=7) if the field would be written to an output file only
- PRISM_SentOut (=8) if the field would be both written to an output file and sent to another model (directly or via OASIS3 main process)
- PRISM_ToRestOut (=9) if the field would be written both to a restart file and to an output file.
- PRISM_Ok (=0) otherwise and no error occurred.

This is useful when the calculation of the corresponding `field_array` is CPU consuming and should be avoided if the field is not effectively used below the `prism_put_proto`.

- `CALL prism_put_restart_proto(var_id, date, 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
  - `info [INTEGER; OUT]`: returned error code (should be PRISM_ToRest=6 if the restart writing was successful)

This routine forces the writing of the field with corresponding `var_id` in its coupling restart file (see section 7.3). If a time operation is specified for this field, the value of the field as calculated below the last `prism_put_proto` is written. If no time operation is specified, the value of the field transferred to the last `prism_put_proto` is written.

- `CALL prism_get_freq (var_id, period, ierror)`
  - `var_id [INTEGER; IN]`: field ID (from corresponding prism_def_var_proto)
  - `period [INTEGER; OUT]`: period of coupling (in number of seconds)
  - `ierror [INTEGER; OUT]`: returned error code

This routine can be used to retrieve the coupling period of field with corresponding `var_id`, as defined in the *namcouple* (see also section 5.3.1).

- `CALL prism_abort_proto(compid, routine_name, abort_message)`
  - `compid [INTEGER; IN]`: component model ID (from prism_init_comp_proto)
  - `routine_name; IN]`: name of calling routine
  - `abort_message; 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`.

4.7 Termination

- `CALL prism_terminate_proto(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. With MPI2, the run may be considered finished when OASIS3 terminates; to avoid problem, place the call to prism_terminate_proto at the very end in the component model code.

[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`.

4.8 Coupling algorithms - SEQ and LAG concepts

Using PSMILe library, the user has full flexibility to reproduce different coupling algorithms. In the component codes, the sending and receiving routines, respectively `prism_put_proto` and `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". This `date` argument is automatically analysed by the PSMILe and depending on the coupling period, the lag and sequencing indices (LAG and SEQ), 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. The lag and sequence concepts and indices are explained in more details here below. These mechanisms are valid for fields exchanged through OASIS3 main process and for fields exchanged directly between the component models.

4.8.1 The lag concept

If no lag index or if a lag index equal to 0 is given by the user in the *namcouple* for a particular coupling field, the sending or receiving actions will actually be performed, below the `prism_put_proto` called in the source model or below the `prism_get_proto` called in the target model respectively, each time the `date` argument on both sides matches an integer number of coupling periods.

To match a `prism_put_proto` called by the source model at a particular `date` with a `prism_get_proto` called by the target model at a different `date`, the user has to define in the *namcouple* an appropriate lag index, LAG, for the coupling field(see section 5). The value of the LAG index must be expressed in "number of seconds"; its value is automatically added to the `prism_put_proto date` value and the sending action is effectively performed when the sum of the `date` and the lag matches an integer number of coupling periods. This sending action is automatically matched, on the target side, with the receiving action performed when the `prism_get_proto date` argument equals the same integer number of coupling periods.

Note that when there is a lag, the first instance of the source field is missing in the debug file (EXPOUT or IGNOUT fields, see section 5.3) because the first source field is not sent by the source model with a `prism_put_proto` but directly read by OASIS3 from a coupling 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 `prism_put_proto` or `prism_get_proto` called in the component model that do not lead to any sending or receiving action; long black arrows correspond to `prism_put_proto` or `prism_get_proto` called in the component models that do effectively lead to a sending or receiving action; long red arrows correspond to `prism_put_proto` or `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 (either directly or through OASIS3 main process).

   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 sending action by model A/B was used at a coupling timestep to match the model B/A receiving action, a deadlock would occur as both models would be initially waiting on a receiving 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 receiving 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 `prism_put_proto` performed at time 8 and 20 by model A will then lead to sending actions (as 8 + 4 = 12 and 20 + 4 = 24 which are coupling periods) that match the receiving actions performed at times 12 and 24 below the `prism_get_proto` called by model B. For F_2, the

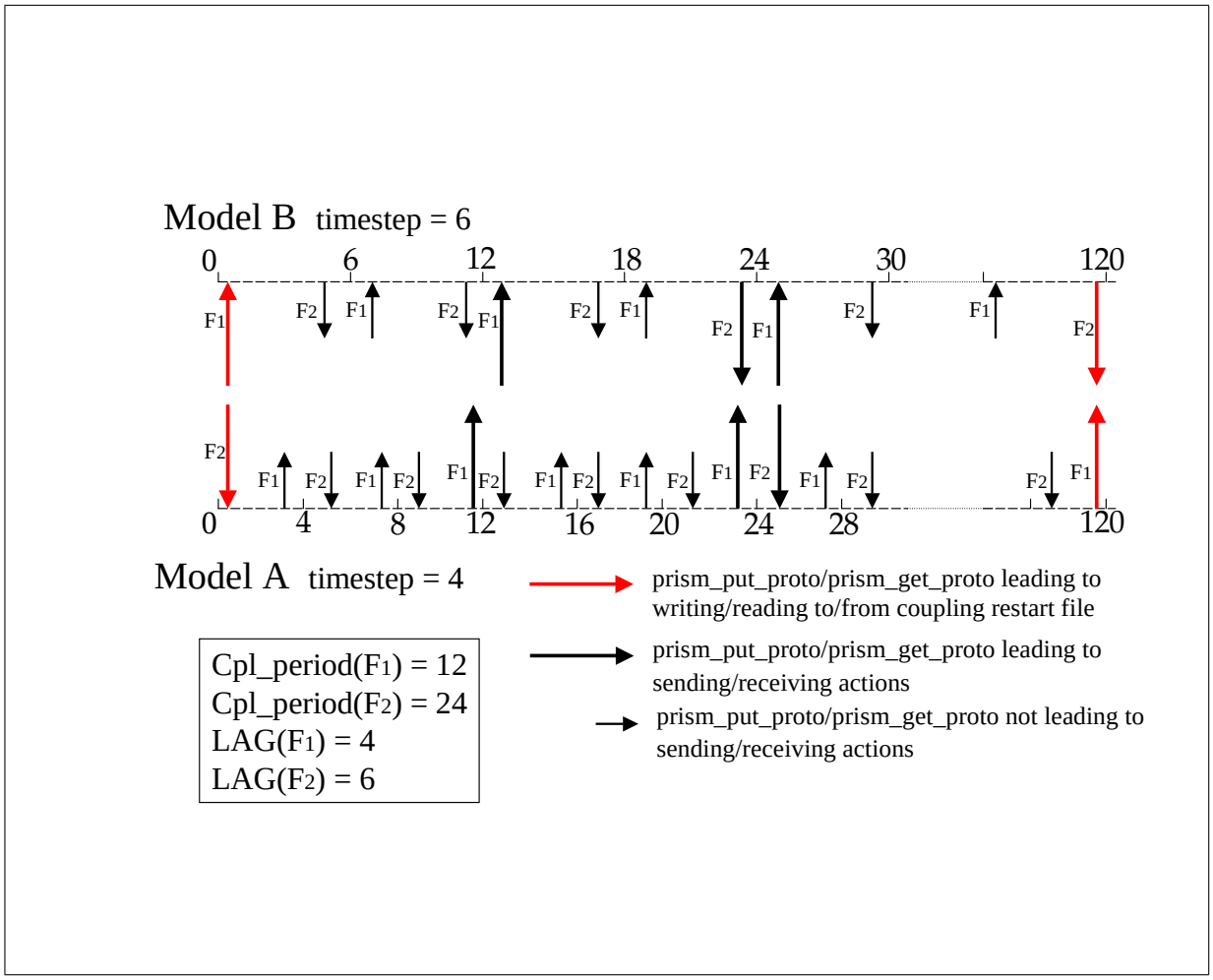

Figure 4.4: LAG concept first example

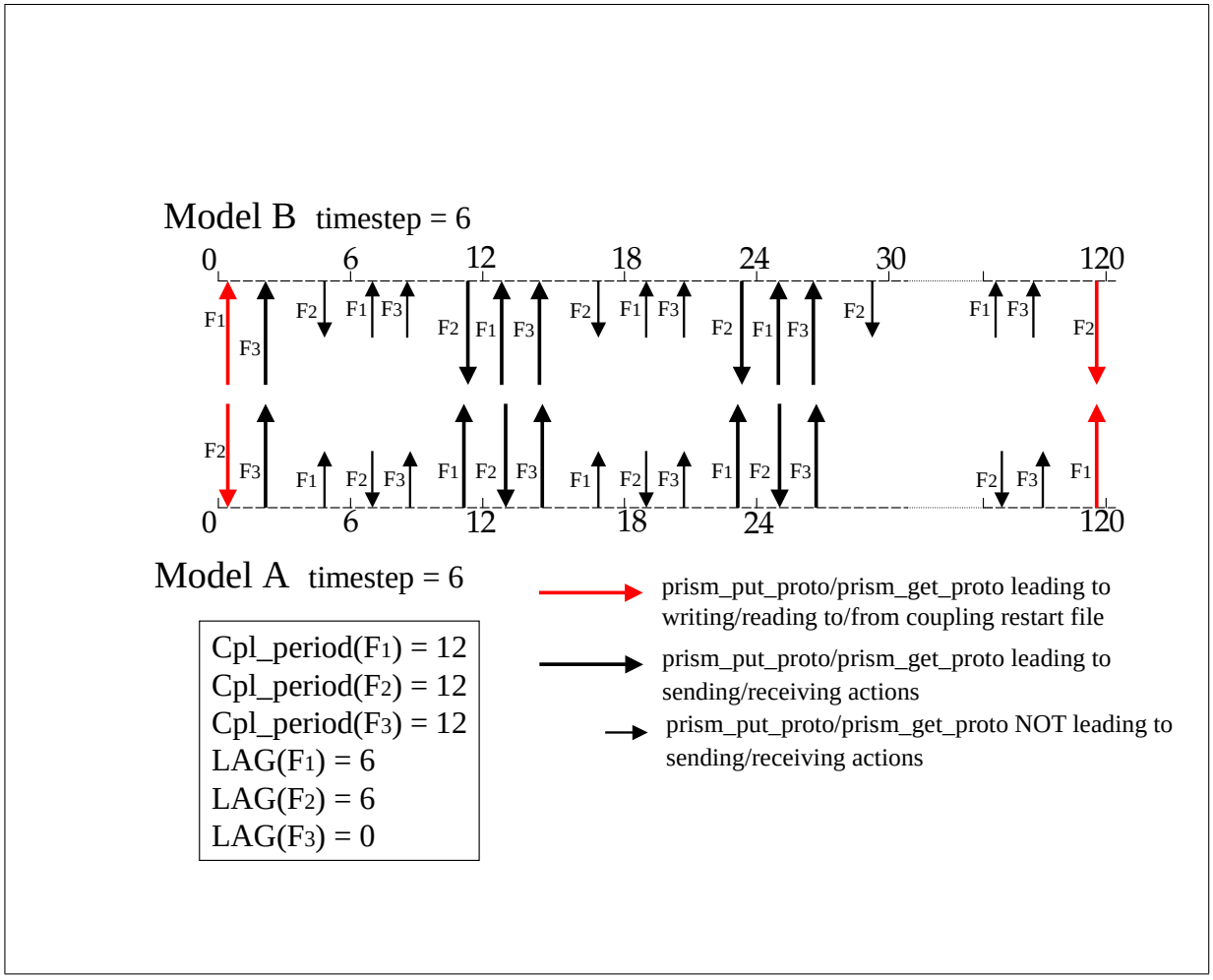

Figure 4.5: LAG concept second example

`prism_put_proto` performed at time 18 by model B then leads to a sending action (as 18 + 6 = 24 which is a coupling period) that matches the receiving action performed at time 24 below the `prism_get_proto` called by model A.

At the beginning of the run, as their LAG index is greater than 0, the first `prism_get_proto` of F_1 and F_2 will automatically be fulfilled by OASIS3 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 `prism_put_proto` as the `date` + F_1 lag equals a coupling 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 `prism_get_proto`.

2. 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 sending action by model A/B was used at a coupling timestep to match the model B/A receiving action, a deadlock would occur as both models would be waiting on a receiving 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 receiving actions, which means that a lag of 6 must be defined for both F_1 and F_2. For both coupling fields, the `prism_put_proto` performed at times 6 and 18 by the source model then lead to sending actions (as 6 + 6 = 12 and 18

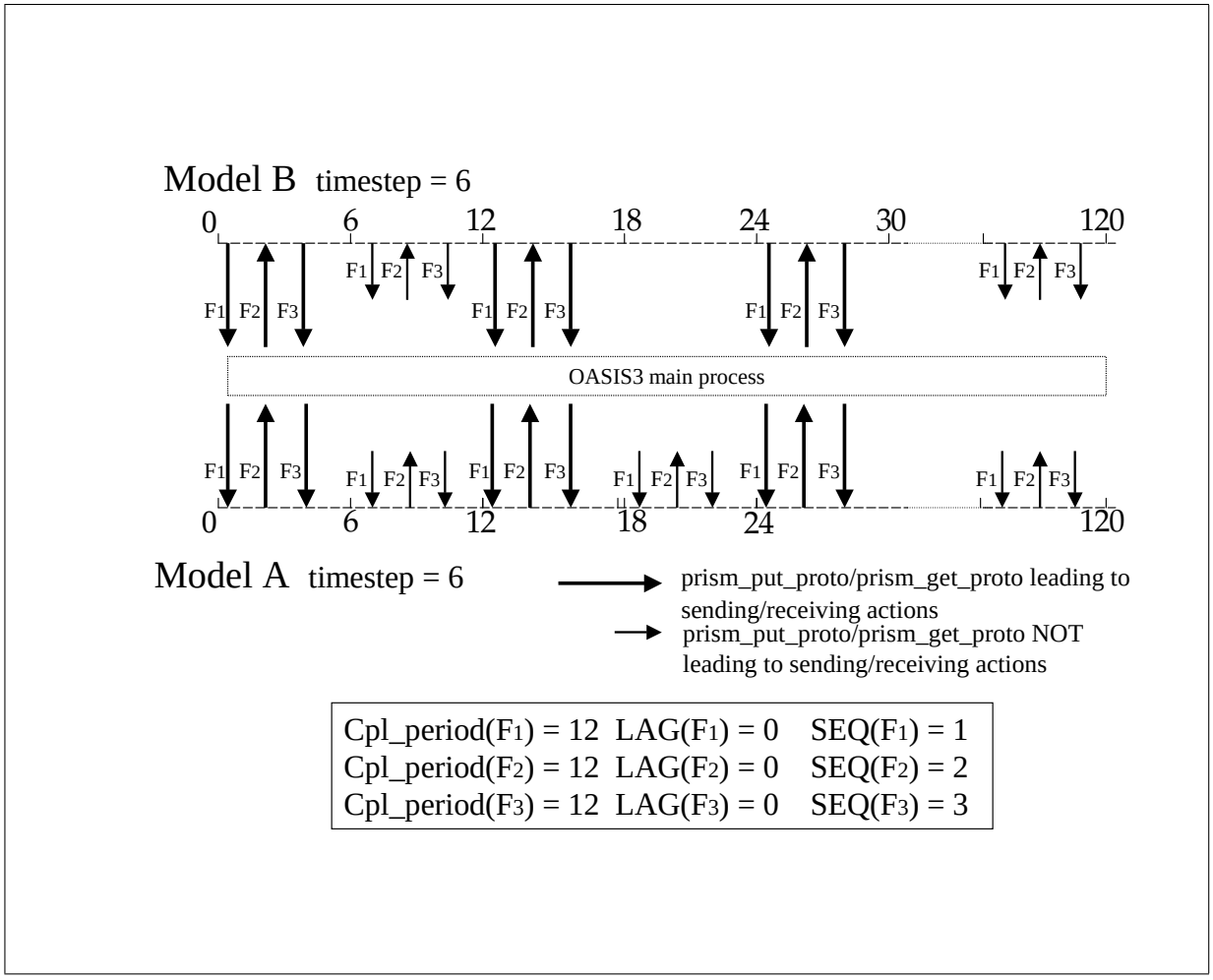

Figure 4.6: The SEQ concept

+ 6 = 24 which are coupling periods) that match the receiving action performed at time 12 and 24 below the `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 `prism_get_proto` performed at the beginning of the run for F_1 and F_2, automatically read them from their coupling restart files, and the last `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 receiving 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.

4.8.2 The sequence concept

To exchange the coupling fields going through OASIS3 main process (i.e. with status EXPORTED, AUXILARY, or EXPOUT, see section 5), in a given order at each coupling timestep, a sequence index SEQ must be defined for each of them. This is not required for I/O fields or for coupling fields exchanged directly between the component models, i.e. with status IGNOUT, INPUT or OUTPUT. SEQ gives the position of the coupling field in the sequence.

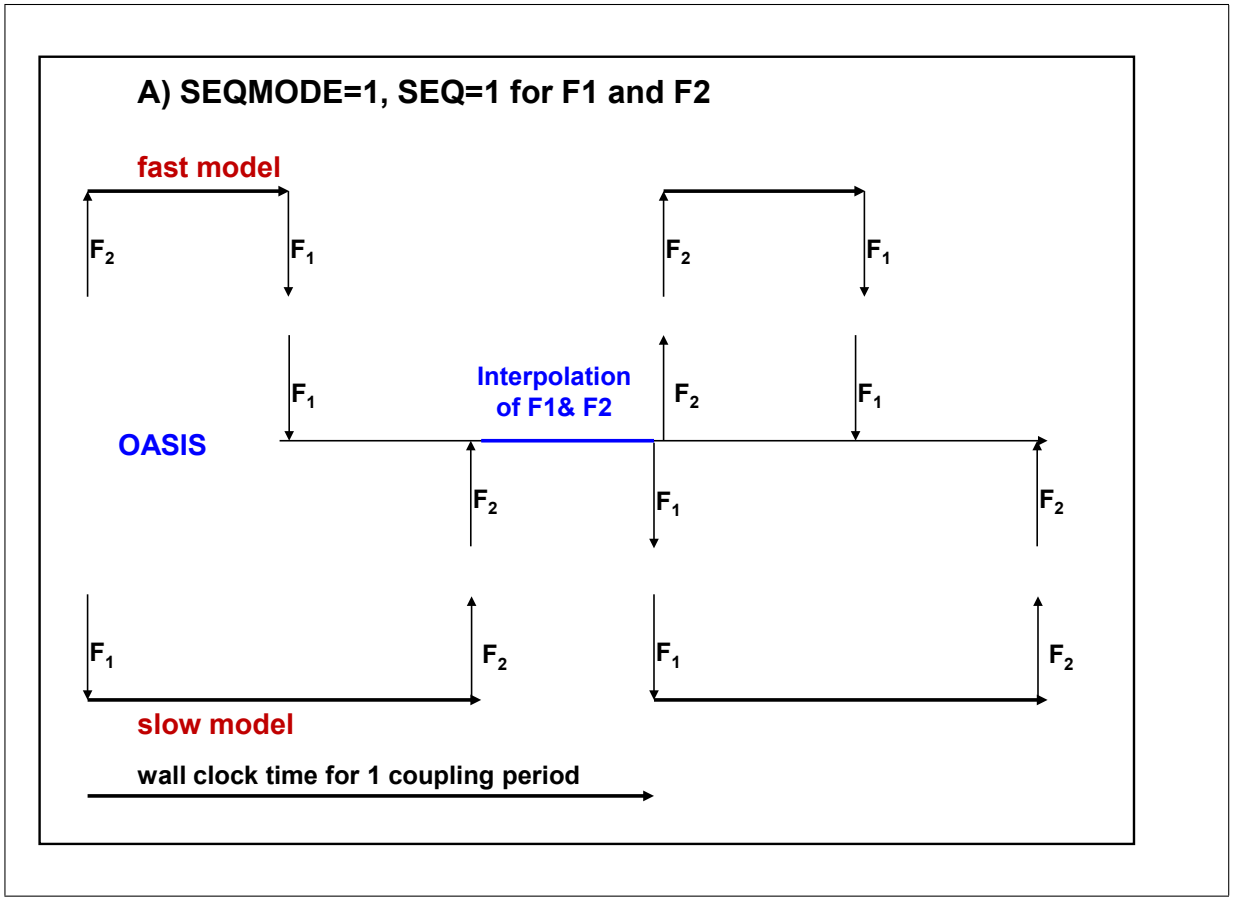

Figure 4.7: Optimisation of a coupled run using the SEQ index : case A

A coupling algorithm, showing the SEQ concept, is illustrated on figure 4.6. All coupling field produced by the source model at the coupling timestep can be "consumed" by the target model at the same timestep without causing any deadlock situation; therefore, LAG = 0 for all coupling fields. However, at each coupling timestep, a particular order of exchange must be respected; F_1 must be received by model A before it can send F_2, which in turn must be received by model B before it can send F_3. Therefore, SEQ = 1, 2, 3 must be defined respectively for F_1, F_2 and F_3. As all fields can be consumed at the time they are produced (LAG=0 for all fields), there no reading/writing from/to coupling restart files.

An appropriate use of the SEQ index can optimise a coupled run when one of the component model is much slower than the other. This is illustrated on figure 4.7 and figure 4.8.

In figure 4.7 , a coupled run with no use of the SEQ index is illustrated. When the fast model reaches a coupling period, it sends its output coupling fields to OASIS that receives them and then waits until the slow model has also reaches the coupling period. Once OASIS3 has received the fields from both the fast and the slow components, it transforms them and send them respectively to the slow and the fast components. Only then the fast and the slow components are able to go on. The total elapse time for a coupling period is the sum of the slow model running time and the OASIS3 working time.

In the figure 4.8, an index SEQ = 1 is assigned to the coupling fields going from the fast to the slow component and an index SEQ = 2 is assigned to the coupling fields going from the slow to the fast component. When the fast model reaches a coupling period, it sends its output coupling fields to OASIS3 that receives them, treats them and sends them to the slow model. As soon as the slow model also reaches the coupling period, it sends its output coupling fields to OASIS, receives its input coupling fields right after, and is then able to go on without any delay. Concurrently, OASIS3 treats the slow model output component fields and sends them to the fast model that is then able to go on. Here the total elapse time for a coupling period is very close to the slow model running time only. One can see that using the SEQ index in this way results in "hiding" OASIS3 working time behind the slow model running time. Note

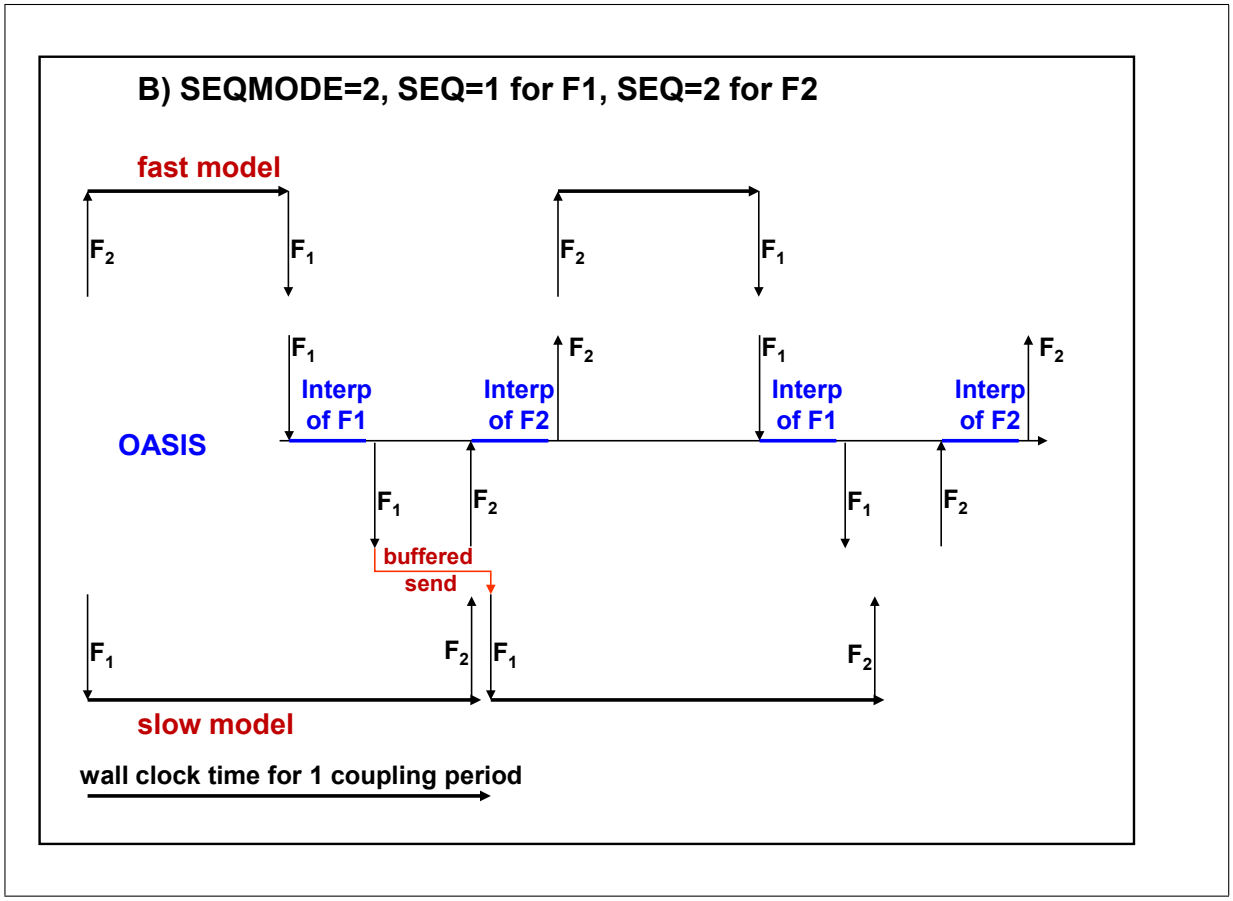

Figure 4.8: Optimisation of a coupled run using the SEQ index : case B

that in this case, the default buffered send must be used (i.e the `NOBSEND` option cannot be specified in the *namcouple*, see section 5.2).

4.8.3 A mix of lag and sequence: the sequential coupled model

One can run the same component models simultaneously or sequentially by defining the appropriate LAG and SEQ indices. In the example illustrated on figure 4.9, the models perform their `prism_put_proto` and `prism_get_proto` calls exactly as in the first lag example above (see 1. of 4.8.1): 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 both 12. By defining a LAG index of -8 for F_1, the models will now run sequentially.

As the LAG for F_2 is positive (6), a reading of F_2 in its coupling restart file is automatically performed by OASIS3 to fulfill the initial `prism_get_proto`. As the LAG for F_1 is negative (-8), no reading from file is performed initially and model B waits; at time 8, a sending action is effectively performed below model A F_1 `prism_put_proto` (as 8 + LAG (-8) = 0 which is the first coupling timestep) and matches the initial model B F_1 `prism_get_proto`. Below the last model A F_1 `prism_put_proto` of the run at time 116, a sending action is effectively performed, as $116 + LAG(-8) = 108$ is a coupling period (as the LAG is negative, the field is not written to its coupling restart file). Below the last model B F_2 `prism_put_proto` of the run at time 114, a writing of F_2 to its restart file is performed, as $114 + LAG(6) = 120$ is a coupling period and as the LAG is positive.

If the coupling fields are transformed through OASIS3 executable, it is important to indicate a sequence index. In fact, at each OASIS3 coupling timestep, F_1 must be necessarily treated after F_2. Therefore, $SEQ(F_1) = 2$ and $SEQ(F_2) = 1$.

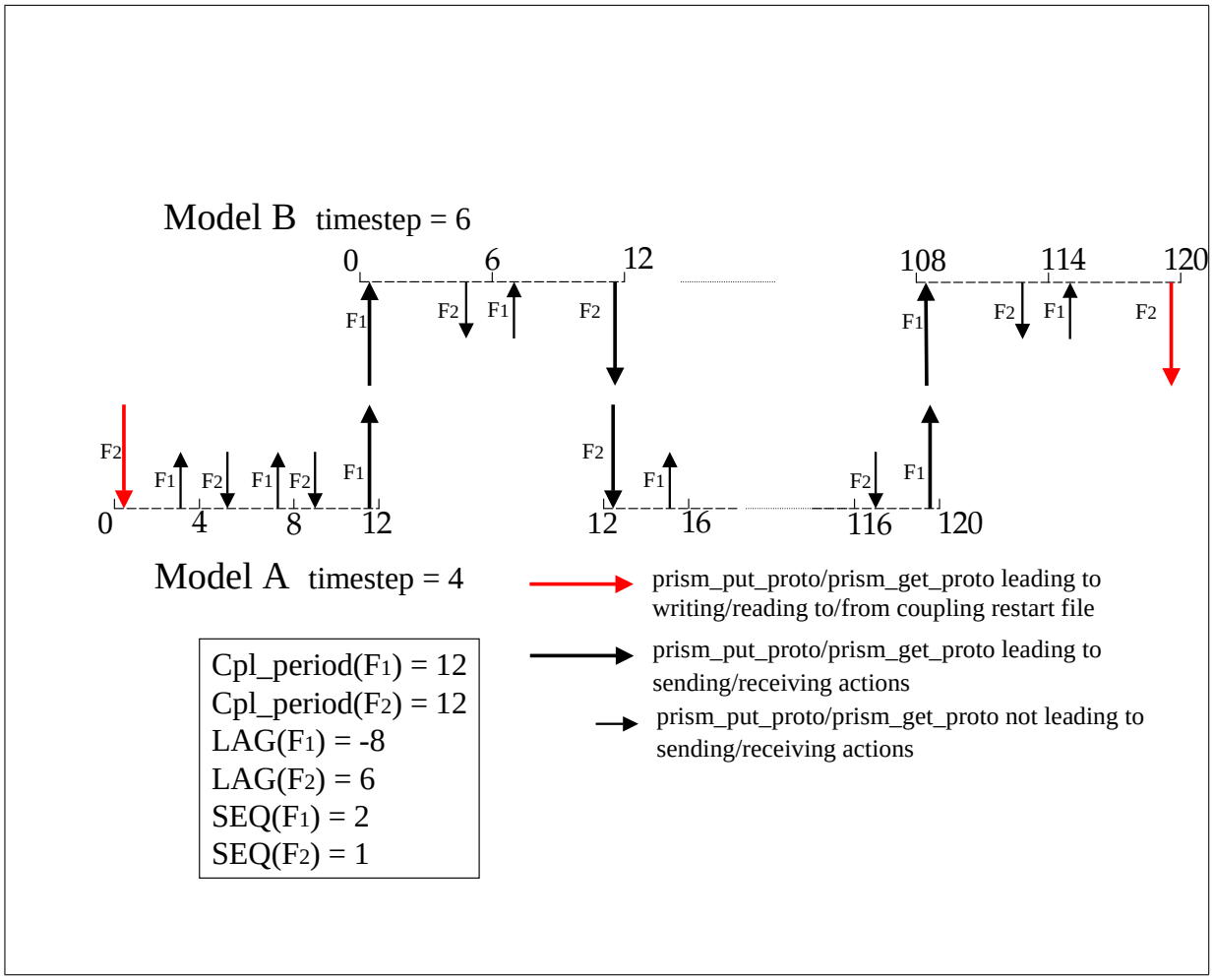

Figure 4.9: Mix of LAG and SEQ concepts

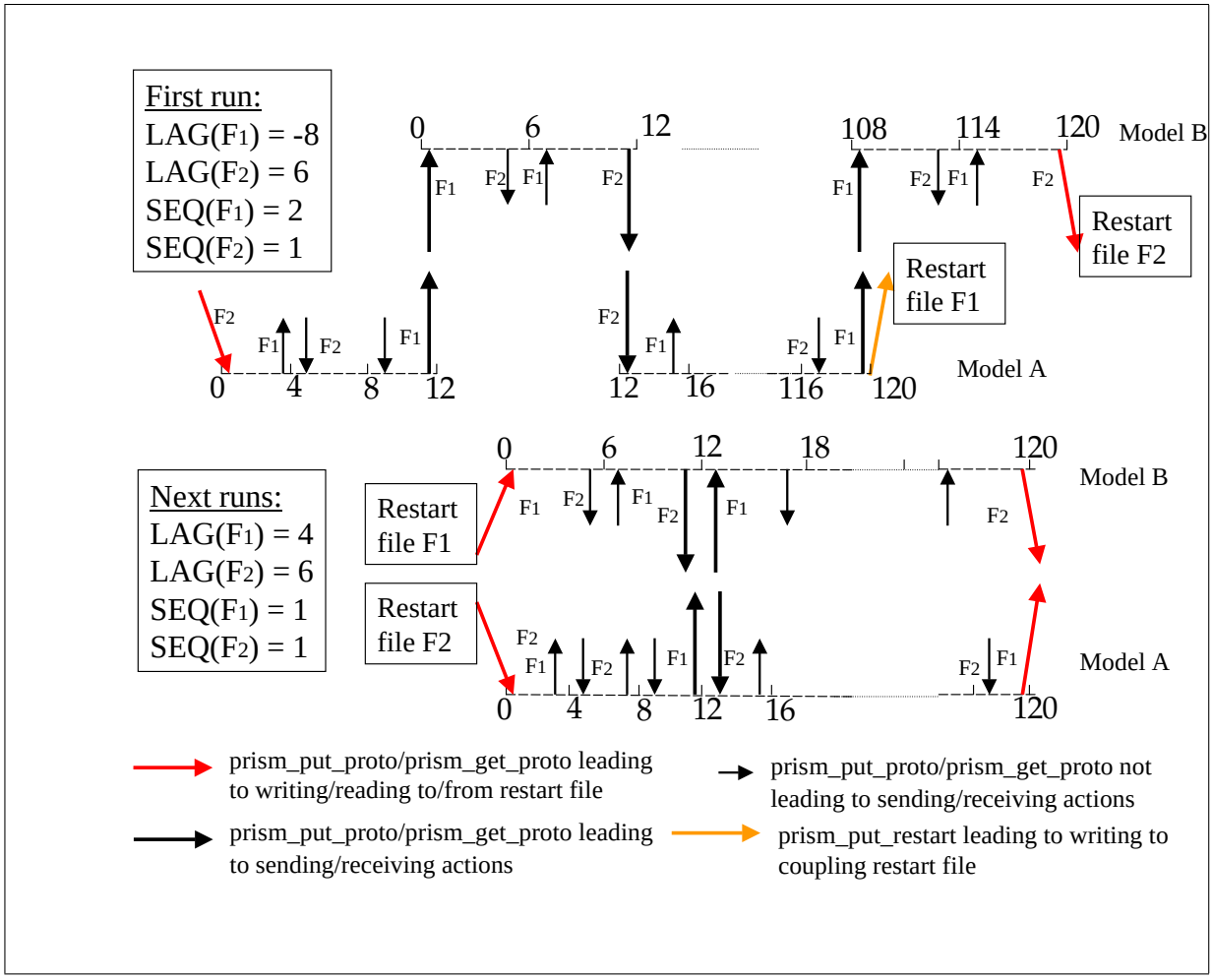

Figure 4.10: An example using prism_put_restart_proto

4.8.4 Mixing sequential and parallel runs using `prism_put_restart_proto`

In the example illustrated on figure 4.10, the models run sequentially for the first run only and then run simultaneously. For the first run, the LAG and SEQ indices must be defined as in section 4.8.3. After the first run, the situation is similar to the one of section 4.8.1, and positive LAG must be defined for F_1 and F_2. As their lag is positive, their corresponding first `prism_get_proto` will automatically lead to reading F_1 and F_2 from coupling restart files. In this case, model A has to write F_1 to its restart file explicitly by calling `prism_put_restart_proto` (illustrated on the figure by an orange arrow) at the end of the first run; in fact, F_1 lag being then negative, such writing is not automatically done below the last `prism_put_proto` of the first run.

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.

If the OASIS3 executable runs on many processes with the 'IPSL' type of parallelisation (resulting in a parallelisation of OASIS3 on a field-per-field basis), one *namcouple* file per process must be provided by the user. In this case, OASIS3 must be compiled with the CPP key `use_oasis_para` and the *namcouple* files must be named `namcouple_x` where x is the number of the corresponding OASIS3 process. For more details on using OASIS3 in this parallel mode, see section 8.2.1.

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.

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 MOZAIC  BLASNEW CHECKOUT
#
  AVERAGE
  INT=1
  at31topa   91    2   48
  CONSTANT     273.15
  INT=1
#
```

```
# Field 2
#
 CONSFTOT SOHEFLDO 6 86400  4   flxat.nc  EXPORTED
 atmo   toce  LAG=+14400  SEQ=+2
 P 0 P 2
 LOCTRANS  CHECKIN  SCRIPR CHECKOUT
#
  ACCUMUL
  INT=1
  CONSERV  LR  SCALAR  LATLON   10  FRACAREA   FIRST
  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:

- `$SEQMODE`: On the line below this keyword is the maximum value of the coupling field SEQ index specified in the *namcouple* (or 1 if no SEQ indices are specified for the coupling fields, see below and also section 4.8).
- `$CHANNEL`: On the line below this keyword is the communication technique chosen. Choices are `MPI1` or `MPI2` for the CLIM communication technique and related PSMILe library. To run OASIS3 as an interpolator only, put `NONE` (see also section 6.1). The communication techniques available in previous OASIS version, i.e. `SIPC`, `PIPE`, or `GMEM` are not officially supported anymore.

  To use the `CLIM/MPI2` communication technique, the lines below `$CHANNEL` are, e.g. for 3 models:

  ```
  $CHANNEL
  MPI2 NOBSEND
  1    1  arg1
  3    1  arg2
  3    1  arg3
  ```

  where `MPI2` is the message passing used in CLIM and PSMILe, and `NOBSEND` indicates that standard basic send `MPI_Send` should be used in place of the default buffered `MPI_BSend` to send the coupling fields. The standard basic send `MPI_Send` can be used only if the coupling fields are sent and received in the same order by the different component models, i.e. the order into which they are listed in the *namcouple*, or on platforms for which `MPI_Send` is implemented with a sufficiently large mailbox. The less efficient default buffered send `MPI_BSend` should be used on platforms

for which MPI_Send is not implemented with a mailbox[1] if the coupling fields are not sent and received in the order into which they are listed in the *namcouple*[2].

The following lines (one line per model listed on the $NBMODEL line) indicate for each model the total number of processes, the number of processes implied in the coupling, and possibly launching arguments[3]. Here the first model runs on one process which is of course implied in the coupling and the argument passed to the model is "arg1"; the second and third models run on 3 processes but only one process is implied in the coupling (i.e. exchanging information with OASIS3 main process), and the argument passed to the models are respectively "arg2" and "arg3".

To use the CLIM/MPI1 communication technique, the $CHANNEL lines are as for MPI2 except that MPI2 is replaced by MPI1 and there is no launching arguments. With MPI1, models have to be started by the user in a pseudo-MPMD mode in the order they are introduced in the *namcouple*. The way to do this depends on the computing platform. With MPI1, OASIS3 main process and the component models automatically share the same MPI_COMM_WORLD communicator; in this communicator OASIS3 main process is assumed to have rank 0 and the other component models are assumed to have ranks in the order of which they are introduced in namcouple. If this is not the case, a deadlock may occur.

- $NFIELDS: On the line below this keyword is the total number of fields exchanged and described in the second part of the *namcouple*.

  In some cases, an additional number has to be specified on the same line after the total number of fields exchanged. This number, corresponding to the maximum number of prism_def_var_proto called by ANY component model in the coupled system, is mandatory if it is greater than twice the number of fields listed in the *namcouple*; this may be the case if OASIS3 is used in IPSL parallel mode (see section 8.2.1) or if fields declared with prism_def_var_proto call in the component code are not activated in the *namcouple* (in this case, the corresponding sending and receiving calls in the component code simply return without any action performed).
- $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 4.1). In MPI2 mode, these names also have to correspond to the model executable names.

  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. If $CHANNEL is NONE, $NBMODEL has to be 0 and there should be no model name and no unit number.
- $RUNTIME: On the line below this keyword is the total simulated time of the run, expressed in seconds. If $CHANNEL is NONE, $RUNTIME has to be the number of time occurrences of the field to interpolate from the restart file.
- $INIDATE: On the line below this keyword is the initial date of the run. The format is YYYYMMDD. This date is important only for the FILLING transformation and for printing information in OASIS3

[1]Note that it was observed that the MPI_Bsend does not work correctly on NEC SX8 with MPI libraries versions 7.2.0 et 7.2.1; this problem was solved with MPI version 7.2.4.

[2]Note that 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 atmoa3.F90 model in oasis3/examples/toyoa3/atmoa3/src.

[3]The cases that have been fully tested are: 1- when all processes participate in the coupling; 2- when only the master process participate in the coupling. The case where a subset of more than one process participate in the coupling has not been tested.

log file *cplout*.

- $MODINFO: If coupling restart files are binary files (see section 7.3), the line below this keyword indicates if a header is encapsulated or not: it can be YES or NOT.
- $NLOGPRT: The line below this keyword refers to the amount of information that will be written to the OASIS3 log file *cplout* during the run. With 0, there is practically no output written to the cplout; with 1, only some general information on the run, the header of the main routines, and the names of the fields when treated appear in the *cplout*. Finally, with 2, the full output is generated.
- $CALTYPE: This keyword gives the type of calendar used. The calendar type is important only if FILLING analysis is used for a coupling field in the run and for printing information in OASIS3 log file *cplout*. Below this keyword, a number (0, 1 or n) must be indicated by the user:
  - 0 : a 365 day calendar (no leap year)
  - 1 : a 365 or 366 (leap years) day calendar A year is a leap year if it can be divided by 4; however if it can be divided by 4 and 100, it is not a leap year; furthermore, if it can be divided by 4, 100 and 400, it is a leap year.
  - n : $n \geq 1$ day month calendar.

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: sent by the source component, received and used by OASIS3 for the transformation of other fields, but not sent to any target component.
- EXPORTED: exchanged between component models and transformed by OASIS3 .
- EXPOUT: exchanged, transformed and also written to two output 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)
- IGNORED: exchanged directly between the component models without being transformed by OASIS3 (the grid and the partitioning of the source and target models have to be identical)
- IGNOUT: exchanged directly between the component models without being transformed by OASIS3 and written to two output 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 (the grid and the partitioning of the source and target models have to be identical)
- 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 7.4 for the format of the input file.
- OUTPUT: simply written out to an output file by the source model PSMILe below the prism_put_proto call at appropriate times corresponding to the output period indicated by the user in the *namcouple*. The name of the output file (one per field) is automatically built based on the field name and initial date of the run ($INIDATE).

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

The first 3 lines for fields with status EXPORTED, AUXILARY 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 4.4.
  - 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 4.4.
  - 1 : 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. If $CHANNEL is NONE (see section 6.1), put "1".
  - 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); it may be a binary or netCDF file (for more detail, see section 7.3); mandatory even if no coupling restart file is effectively used.
  - sstat.nc : name of the field output file[4], may be indicated for NONE (and PIPE) communication techniques only. It may be a binary of netCDF file (see section 7.3).
  - 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 4.8)
- 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.

[4] In interplator-only mode (i.e. $CHANNEL = NONE), there must be one input file and one output file per field.

5.3.2 Second section of *namcouple* for IGNORED and IGNOUT fields

The first 2 lines for fields with status IGNORED or IGNOUT are as follows:

```
COSENHFL SOSENHFL 37 86400 1 flda3.nc IGNOUT
atmo   toce LAG=+7200
```

where the different entries are as for EXPORTED fields, except that there is no output file name on the first line and no SEQ index at the end of the second line.

For IGNORED fields, the name used in the coupling restart file (if any) must be the target symbolic name.

The third line is LOCTRANS if this transformation is chosen for the field. Note that LOCTRANS is the only transformation supported for IGNORED, IGNOUT and OUTPUT fields (as it is performed directly in the PSMILe below the prism_put_proto call). If LOCTRANS is chosen, a fourth line giving the name of the time transformation is required. For more detail on LOCTRANS, see section 6.2.

5.3.3 Second section of *namcouple* for OUTPUT fields

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

```
COSHFTOT  COSHFTOT   7   86400  0  OUTPUT
atmo   atmo
```

where the different entries are as for IGNORED fields, except that the source symbolic name must be repeated twice on the field first line, that there is no coupling restart file name, no LAG index and no SEQ index. The name of the output file is automatically determined by the PSMILe.

The third line is LOCTRANS if this transformation is chosen for the field. Note that LOCTRANS is the only transformation supported for OUTPUT fields (see section 5.3.2).

5.3.4 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 7.4)
- 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 (with `CLIM/MPI1-MPI2` communication technique only), 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, it is first described how to use OASIS3 in an interpolator-only mode. Then a description of each transformation with its corresponding configuring lines is given.

6.1 Using OASIS3 in the interpolator-only mode

OASIS3 can be used in an interpolator-only mode, in which case it transforms fields without running any model. It is recommended to use first OASIS3 in this mode to test different transformations and interpolations without having to run the whole coupled system. In the interpolator-only mode, all transformations, except the time transformations, are available.

To run OASIS3 in an interpolator-only mode, the user has to prepare the *namcouple* as indicated in sections 5.2 and 5.3. In particular, `NONE` has to be chosen below the keyword `$CHANNEL`; "0" (without any model name and Fortran unit number) must be given below the keyword `$NBMODEL`; `$RUNTIME` has to be the number of time occurrences of the field to interpolate from the NetCDF input file[1]; finally, the "coupling" period of the field (4th entry on the field first line) must be always "1". Note that if `$RUNTIME` is smaller than the total number of time ocurrences in the input file, the first `$RUNTIME` occurrences will be interpolated.

The name of the input file which contains the fields to interpolate is given by the 6th entry on the field first line (see 5.3). After their transformation, OASIS3 writes them to their output file which name is the 7th entry on the first line. Note that there must be one input file and one output file per field.

The time variable in the input file, if any, is recognized by the its attribute "units". The acceptable units for time are listed in the udunits.dat file (3). This follows the CF convention.

To compile OASIS3 in interpolator-only mode, see section 8.1.1. Practical examples on how to use OASIS3 in a interpolator-only mode are given in `oasis3/examples/testinterp` (see also section 8.4.1) and `oasis3/examples/testNONE` (see also section 8.4.2)

The configuring parameters that have to be defined in the *namcouple* for each transformation in the interpolator-only mode or in the coupling mode are described here after.

[1] For binary input file, only one time occurence may be interpolated

6.2 The time transformations

Transformation LOCTRANS[2] requires one configuring line on which a time transformation, automatically performed below the call to PSMILe prism_put_proto, should be indicated:

- 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: only one prism_put_proto or prism_get_proto will be performed; this is equivalent to giving the length of the run as coupling or I/O period.

6.3 The pre-processing transformations

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

- **REDGLO**

  This transformation is deprecated in the current OASIS3 version as interpolations for Gaussian Reduced grid now exist; this transformation should not be used anymore.

  REDGLO (routine redglo.f) performs the interpolation from a Reduced grid to a Gaussian one. The interpolation is linear and performed latitude circle per latitude circle. When present, REDGLO must be the first pre-processing transformation performed. The configuring line is as follows:

  ```
  # REDGLO operation
      $NNBRLAT  $CDMSK
  ```

  where xxx is half the number of latitude circles of the Gaussian grid. For example, for a T42 with 64 latitude circles, $NNBRLAT is "NO32". In the current version, it can be either NO16, NO24, NO32, NO48, NO80, NO160. $CDMSK is a flag indicating if non-masked values have to be extended to masked areas before interpolation ($CDMSK = SEALAND) using the Reduced grid mask (see section 7.2) or if the opposite has to be performed ($CDMSK = LANDSEA). If $CDMSK = NOEXTRAP, then no extrapolation is performed.

- **INVERT**:

  This transformation is deprecated in the current OASIS version and should be used anymore. The fields and corresponding grids can be given in any direction as long as they are coherent.

  INVERT (routine invert.f) reorders a field so that it goes from south to north and from west to east (the first point will be the southern and western most one; then it goes parallel by parallel going from south to north). Note that INVERT does not transform the associated grid or mask. INVERT should be used only for fields associated to A, B, G, L, Z, or Y grids (see annexe A) but produced by the source model from North to South and/or from East to West. INVERT does not work for Reduced ('D') or unstructured ('U') grids (see annexe A).

  The generic input line is as follows:

[2] Note that LOCTRANS cannot be chosen with the deprecated SIPC, PIPE, or GMEM communication techniques.

```
# INVERT operation
    $CORLAT  $CORLON
```

$CORLAT = NORSUD or SUDNOR and $CORLON = ESTWST or WSTEST describes the orientation of the source field in longitude and latitude, respectively.

- **MASK**:

  MASK (routine masq.f) is used before the analysis EXTRAP. A given REAL value VALMASK is assigned to all masked points following the source grid mask (see section 7.2), so they can be detected by EXTRAP.

  The generic input line is as follows:

```
# MASK operation
    $VALMASK
```

  Make sure that $VALMASK conforms to a REAL value, e.g. “99999999.” not “99999999” . Problems may arise if the value chosen approaches the maximum value that your computing platform can represent; choose a value well outside the range of your field values but not too large.

- **EXTRAP**:

  EXTRAP (routine extrap.f) performs the extrapolation of a field over its masked points. The analysis MASK must be used just before, so that EXTRAP can identify masked points. Note that EXTRAP does not work for Reduced (’D’) or unstructured (’U’) grids (see appendix A).

  Two methods of extrapolation are available. With NINENN, a N-nearest-neighbour method is used. The procedure is iterative and the set of remaining masked points evolves at each iteration. The configuring line is:

```
# EXTRAP operation for $CMETH = NINENN
    $CMETH  $NV  $NIO  $NID
```

  $CMETH = NINENN; $NV is the minimum number of neighbours required to perform the extrapolation (with a maximum of 4)[3]; $NIO is the flag that indicates if the weight-address-and-iteration-number dataset will be calculated and written by OASIS3 ($NIO= 1), or only read ($NIO= 0) in file *nweights* (see section 7.5); $NID is the identificator for the weight-address-iteration-number dataset in all the different EXTRAP/NINENN datasets in the present coupling.[4]

  With $CMETH = WEIGHT, an N-weighted-neighbour extrapolation is performed. In that case, the user has to build the grid-mapping file, giving for each target grid point the weights and addresses of the source grid points used in the extrapolation; the structure of this file has to follow the OASIS3 generic structure for transformation auxiliary data files (see section 7.5).

  The configuring line is:

```
# EXTRAP operation for $CMETH = WEIGHT
    $CMETH  $NV  $CFILE  $NUMLU  $NID
```

  $CMETH = WEIGHT; $NV is the maximum number of neighbours required by the extrapolation operation; $CFILE and $NUMLU are the grid-mapping file name and associated logical unit; $NID is the identificator for the relevant grid-mapping dataset in all different EXTRAP/WEIGHT transformations in the present coupling.

- **CHECKIN**:

  CHECKIN (routine chkfld.f) calculates the mean and extremum values of the source field and prints them to the coupler log file *cplout* (this operation does not transform the field).

  The generic input line is as follows:

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

[3] For some grids, the extrapolation may not converge if $NV is too large.

[4] An EXTRAP/NINENN analysis is automatically performed within GLORED analysis but the corresponding datasets have to be distinct; this is automatically checked by OASIS3 at the beginning of the run.

$NINT = 1 or 0, depending on whether or not the source field integral is also calculated and printed.

- **CORRECT**:

  CORRECT (routine correct.f) reads external fields from binary files and uses them to modify the coupling field. This transformation can be used, for example, to perform flux correction on the field.

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

  ```
  # CORRECT operation
      $XMULT   $NBFIELDS
  ```

  $XMULT is the multiplicative coefficient of the current field, and $NBFIELDS the number of additional fields to be combined with the current field. For each additional field, an additional configuring line is required:

  ```
  # nbfields lines
      $CLOC   $AMULT   $CFILE   $NUMLU
  ```

  $CLOC and $AMULT, $CFILE and $NUMLU are respectively the symbolic name, the multiplicative coefficient, the file name and the associated logical unit on which the additional field is going to be read. The structure of the file has to follow the structure of OASIS3 binary coupling restart files (see section 7.3).

6.4 The interpolation

The following transformations, controlled by interp.f, are available in OASIS3.

- **BLASOLD**:

  BLASOLD (routine blasold.f) performs a linear combination of the current coupling field with other coupling fields or with a constant before the interpolation *per se*.

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

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

  $XMULT is the multiplicative coefficient of the current field, and $NBFIELDS the number of additional fields to be combined with the current field. For each additional field, an additional input line is required:

  ```
  # nbfields lines
      $CNAME   $AMULT
  ```

  where $CNAME and $AMULT are the symbolic name and the multiplicative coefficient for the additional field. To add a constant value to the original field, put $XMULT = 1, $NBFIELDS = 1, $CNAME = CONSTANT, $AMULT = value to add.

- **SCRIPR**:

  SCRIPR gathers the interpolation techniques offered by Los Alamos National Laboratory SCRIP 1.4 library[5](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.

  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.

[5]See the copyright statement in appendix 3.2.2.

  * 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. Warning: this default behaviour has changed since release `oasis3_prism_2_5`[6].

The configuring line is:

```
# SCRIPR/DISWGT
 $CMETH $CGRS $CFTYP $REST $NBIN $NV $ASSCMP $PROJCART
```

  * `$CMETH = DISTWGT`.
  * `$CGRS` is the source grid type (`LR`, `D` or `U`)- see annexe A.
  * `$CFTYP` is the field type: `SCALAR` if the field is a scalar one, or `VECTOR_I` or `VECTOR_J` whether the field represents respectively the first or the second component of a vector field (see paragraph **Support of vector fields** below). The option `VECTOR`, which is fact leads to a scalar treatment of the field (as in the previous versions), is still accepted.
  * `$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 value 1.0E+20 will however be assigned to these non-masked target grid points if routines `scriprmp.f` or `vector.F90` (for vector interpolation) in `oasis3/lib/scrip/src/` are compiled with `ll_weightot=.true.`. Furthermore, if `oasis3/lib/scrip/src/remap_gauswgt.f` is compiled with `ll_nnei=.true.`, the non-masked nearest neighbour value will be given to these target grid points.

The configuring line is:

```
 # SCRIPR/GAUSWGT
$CMETH  $CGRS  $CFTYP  $REST  $NBIN  $NV $VAR $ASSCMP $PROJCART
```

  * `$CMETH = GAUSWGT`
  * `$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

[6]To reproduce the previous default behaviour, one has to compile with CPP key `NOT_NNEIGHBOUR`. In this case, the zero value is associated to the target points having all of the N source nearest neighbours masked. However, the value 1.0E+20 will be assigned to these non-masked target grid points if routines `scriprmp.f` or `vector.F90` (for vector interpolation) in `oasis3/lib/scrip/src/` are compiled with `ll_weightot=.true.`.

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)
- BICUBIC performs an interpolation based on a local bicubic approximation (see details in chapter 5 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 value 1.0E+20 will however be assigned to these non-masked target grid points if routines scriprmp.f or vector.F90 (for vector interpolation) in oasis3/lib/scrip/src/ are compiled with ll_weightot=.true.. Furthermore, if oasis3/lib/scrip/src/remap_bicubic.f or remap_bilinear.f is compiled with ll_nnei=.true., the non-masked nearest neighbour value will be given to these target grid points.

  The configuring line is:

```
# SCRIPR/BILINEAR or SCRIPR/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.

- 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.

  The configuring line is:

```
 # SCRIPR/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 normal-

ization 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 6.3) 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

SCRIPR supports 2D vector interpolation. Note however that this functionality has been validated only in a reduced number of test cases. The two vector components have to be identified by assigning VECTOR_I or VECTOR_J to $CFTYP and have to be associated by giving, for each component field, the source symbolic name of the associated vector component to $ASSCMP (see above). The grids of the two vector components can be different but have to have the same number of points, the same overlap, the same mask; the same interpolation must be used for the two components. A proper example of vector interpolation is given in the interpolator-only mode example testinterp (see section 8.4.1). The details of the vector treatment, performed by the routines scriprmp_vector.F90 and rotations.F90 in oasis3/lib/scrip/src are the following:

- If the angles of the source grid local coordinate system are defined in the *grids.nc* data file (see section 7.2), an automatic rotation from the local to the geographic spherical coordinate system is performed.
- If the two source vector components are not defined on the same source grid, one component is automatically interpolated on the grid of the other component. Note that if the components are not given in a Cartesian coordinate system, this operation is not exact as the coordinate system is not fixed spatially (for grids not covering the North or South poles, the error is however small at the scale of the grid mesh).
- If the user puts the PROJCART keyword at the end of the SCRIPR configuring line (see above), projection of the two vector components in a Cartesian coordinate system, interpolation of the resulting 3 Cartesian components, and projection back in the spherical coordinate system are performed. In debug mode (compilation with __DEBUG pre-compiling key), the resulting vertical component in the spherical coordinate system after interpolation is written to a file projection.nc; as the source vector is horizontal, this component should be very close to 0.
- If the user did not put the PROJCART keyword at the end of the SCRIPR configuring line, the two spherical coordinate system components are interpolated. Note that this operation is not exact as the coordinate system is not fixed spatially (in most cases, the error is however small at the scale of the grid mesh).
- If the angles of the target grid local coordinate system are defined in the *grids.nc* data file (see section 7.2), an automatic rotation from the geographic spherical to the local coordinate system is performed.
- The first and second components of the interpolated vector field are then present in the target fields associated respectively to the first and second source vector component. The target grids for the two vector components can be different.

- **INTERP**:

INTERP gathers different techniques of interpolation controlled by routine fiasco.f. The following interpolations are available:

- BILINEAR performs a bilinear interpolation using 4 neighbours.
- BICUBIC performs a bicubic interpolation.
- NNEIBOR performs a nearest-neighbour interpolation.

  These three interpolations are performed by routines in /oasis3/lib/fscint and support only A, B, G, L, Y, or Z grids (see appendix A). All sources grid points, masked or not, are used in the calculation. To avoid the 'contamination' by masked source grid points, transformations MASK and EXTRAP should be used. Values are calculated for all target grid points, masked or not.

  The configuring line is as follows:

  ```
  # BILINEAR or BICUBIC or NNEIBOR interpolation
     $CMETH $CGRS $CFTYP
  ```

  * $CMETH = BILINEAR, BICUBIC or NNEIBOR
  * $CGRS is the source grid type (A, B, G, L, Y, or Z, see appendix A)
  * $CFTYP the field type (SCALAR or VECTOR). VECTOR has an effect for target grid points located near the pole: the sign of a source value located on the other side of the pole will be reversed.
- SURFMESH (routines in /oasis3/lib/anaism) is a first-order conservative remapping from a fine to a coarse grid (the source grid must be finer over the whole domain) and supports only Lat-Lon grids (see appendix A). For a target grid cell, all the underlying not masked source grid cells are found and the target grid field value is the sum of the source grid field values weighted by the overlapped surfaces. No value is assigned to masked cells. Note that it is not recommended to use this interpolation anymore, as the more general SCRIPR/CONSERV remapping is now available. The configuring line is as follows:

  ```
  # SURFMESH remapping
      $CMETH $CGRS $CFTYP $NID $NV $NIO
  ```

  * $CMETH = SURFMESH
  * $CGRS and $CFTYP are as for BILINEAR
  * $NID is the identificator for the weight-address dataset in all the different INTERP/SURFMESH datasets in the present coupling. This dataset will be calculated by OASIS3 if $NIO= 1, or only read if $NIO= 0.
  * $NV is the maximum number of source grid meshes used in the remapping.
- GAUSSIAN (routines in /oasis3/lib/anaisg) is a gaussian weighted nearest-neighbour interpolation technique. The user can choose the variance of the function and the number of neighbours considered. The $NV non masked nearest source grid points are automatically used and no value are calculated for masked target grid points.

  The configuring line is:

  ```
  # GAUSSIAN interpolation
  $CMETH $CGRS $CFTYP $NID $NV $VAR  $NIO
  ```

  * $CMETH = GAUSSIAN
  * $CGRS is the source grid type; all grids are supported[7]
  * $CFTYP is the field type SCALAR or VECTOR.

[7] Note that for "U" grids, the average distance between two source grid point $\overline{d}^2$, used in the calculation of the Gaussian weighting function, is not exact but this has only a limited impact as this value is in all cases multiplied by the $VAR value defined by the user (see SCRIPR/GAUSWGT above).

  - * $NID is the identificator for the weight-address dataset in all the different INTERP/GAUSSIAN datasets in the present coupling. This weight-address dataset will be calculated by OASIS3 if $NIO= 1, or only read if $NIO= 0.
  - * $NV is the number of neighbours used in the interpolation.
  - * $VAR is as for SCRIPR/GAUSWGT (see above).

- **MOZAIC**:

  MOZAIC performs the mapping of a field from a source to a target grid. The grid-mapping dataset, i.e. the weights and addresses of the source grid points used to calculate the value of each target grid point are defined by the user in a file (see section 7.5). The configuring line is:

```
# MOZAIC operation
     $CFILE  $NUMLU  $NID  $NV
```

  - $CFILE and $NUMLU are the grid-mapping file name and associated logical unit on which the grid-mapping dataset is going to be read),
  - $NID the identificator for this grid-mapping dataset in all MOZAIC grid-mapping datasets in the present coupling
  - $NV is the maximum number of target grid points use in the mapping.

- **NOINTERP**:

  NOINTERP is the analysis that has to be chosen when no other transformation from the interpolation class is chosen. There is no configuring line.

- **FILLING**:

  FILLING (routine oasis3/src/filling.f) performs the blending of a regional data set with a climatological global one for a Sea Surface Temperature (SST) or a Sea Ice Extent field. This occurs when coupling a non-global ocean model with a global atmospheric model. FILLING can only handle fields on Logically Rectangular grid (LR, but also A, B, G, L, Y, and Z grids, see section A.

  The global data set has to be a set of SST given in Celsius degrees (for the filling of a Sea Ice Extent field, the presence or absence of ice is deduced from the value of the SST). The frequency of the global set can be interannual monthly, climatological monthly or yearly.

  The blending can be smooth or abrupt. If the blending is abrupt, only model values are used within the model domain, and only the global data set values are used outside. If the blending is smooth, a linear interpolation is performed between the two fields within the model domain over narrow bands along the boundaries. The linear interpolation can also be performed giving a different weight to the regional or and global fields.

  The smoothing is defined by parameters in oasis3/src/mod_smooth.F90. The lower smoothing band in the global model second dimension is defined by *nsltb* (outermost point) and *nslte* (innermost point); the upper smoothing band in the global model second dimension is defined by *nnltb* (outermost point) and *nnlte* (innermost point). The parameter *qalfa* controls the weights given to the regional and to the global fields in the linear interpolation. *qalfa* has to be $1/(nslte - nsltb)$ or $1/(nnltb - nnlte)$. For the outermost points (*nsltb* or *nnltb*) in the smoothing band, the weight given to the regional and global fields will respectively be 0 and 1; for the innermost points (*nslte* or *nnlte*) in the smoothing band, the weight given to the regional and global fields will respectively be 1 and 0; within the smoothing band, the weights will be a linear interpolation of the outermost and innermost weights.

  The smoothing band in the global model first dimension will be a band of *nliss* points following the coastline. To calculate this band, OASIS3 needs *nwlgmx*, the greater first dimension index of the lower coastline and *nelgmx*, the smaller first dimension index on the upper coastline. The parameter *qbeta* controls the weights given to the regional and to the global fields in the linear interpolation.

qbeta has to be $1/(nliss - 1)$. The weights given to the regional and global fields in the global model first dimension smoothing bands will be calculated as for the second dimension.

The user must provide the climatological data file with a specific format described in 7.5. When one uses FILLING for SST with smooth blending, thermodynamics consistency requires to modify the heat fluxes over the blending regions. The correction term is proportional to the difference between the blended SST and the original SST interpolated on the atmospheric grid and can be written out on a file to be read later, for analysis CORRECT for example. In that case, the symbolic name of the flux correction term read through the input file *namcouple* must correspond in FILLING and CORRECT analyses.

In case the regional ocean model includes a coastal part or islands, a sea-land mask mismatch may occur and a coastal point correction can be performed if the field has been previously interpolated with INTER/SURFMESH. In fact, the mismatch could result in the atmosphere undesirably "seeing" climatological SST's directly adjacent to ocean model SST's. Where this situation arises, the coastal correction consists in identifying the suitable ocean model grid points that can be used to extrapolate the field, excluding climatological grid points.

This analysis requires one configuring line with 3, 4 or 6 arguments.

1. If FILLING performs the blending of a regional data set with a global one for the Sea Ice Extent, the 3-argument input line is:

   ```
   # Sea Ice Extent FILLING operation
        $CFILE  $NUMLU  $CMETH
   ```

   the file name for the global data set, $NUMLU the associated logical unit. $CMETH, the FILLING technique, is a CHARACTER*8 variable: the first 3 characters are either SMO, smooth filling, or RAW, no smoothing ; the next three characters must be SIE for a Sea Ice Extent filling operation; the last two define the time characteristics of the global data file, respectively MO, SE and AN for interannual monthly, climatological monthly and yearly. Note that in all cases, the global data file has to be a Sea Surface Temperature field in Celsius degrees.

2. If FILLING performs the blending of a regional data set with a global one for the Sea Surface Temperature without any smoothing, the 4-argument input line is:

   ```
   #Sea Surface Temperature FILLING operation without smoothing
        $CFILE  $NUMLU  $CMETH  $NFCOAST
   ```

   $CFILE, $NUMLU are as for the SIE filling. In this case however, $CMETH(1:3) = RAW, $CMETH(4:6) = SST, and the last two characters define the time characteristics of the global data file, as for the SIE filling. $NFCOAST is the flag for the calculation of the coastal correction (0 no, 1 yes).

3. If FILLING performs the blending of a regional data set with a global one for the Sea Surface Temperature with smoothing, the 6-argument input line is:

   ```
   #Sea Surface Temperature FILLING operation with smoothing
        $CFILE  $NUMLU  $CMETH  $NFCOAST  $CNAME  $NUNIT
   ```

   where $CFILE, $NUMLU and $NFCOAST are as for the SST filling without smoothing. In this case, $CMETH(1:3) = SMO, $CMETH(4:6) = SST, and the last two characters define the time characteristics of the global data file, as for the SIE filling. $CNAME is the symbolic name for the correction term that is calculated by OASIS3 and $NUNIT the logical unit on which it is going to be written.

6.5 The "cooking" stage

The following transformations are available in the "cooking" part of OASIS3, controlled by cookart.f.

- **CONSERV**:

  CONSERV (routine oasis3/src/conserv.f) ensures a global modification of the coupling field. This 'cooking" stage operation should not be mixed with SCRIPR/CONSERV conservative remapping (see section 6.4).

  This analysis requires one input line with one argument:

```
# CONSERV operation
     $CMETH
```

  where $CMETH is the method required:

  - 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.

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

- **SUBGRID**:

  SUBGRID can be used to interpolate a field from a coarse grid to a finer target grid (the target grid must be finer over the whole domain). Two types of subgrid interpolation can be performed, depending on the type of the field.

  For solar type of flux field ($SUBTYPE = SOLAR), the operation performed is:

$$\Phi_i = \frac{1-\alpha_i}{1-\alpha} F$$

  where Φ_i (F) is the flux on the fine (coarse) grid, α_i (α) an auxiliary field on the fine (coarse) grid (e.g. the albedo). The whole operation is interpolated from the coarse grid with a grid-mapping type of interpolation; the dataset of weights and addresses has to be given by the user.

  For non-solar type of field ($SUBTYPE = NONSOLAR), a first-order Taylor expansion of the field on the fine grid relatively to a state variable is performed (for instance, an expansion of the total heat flux relatively to the SST):

$$\Phi_i = F + \frac{\partial F}{\partial T}(T_i - T)$$

  where Φ_i (F) is the heat flux on the fine (coarse) grid, T_i (T) an auxiliary field on the fine (coarse) grid (e.g. the SST) and $\frac{\partial F}{\partial T}$ the derivative of the flux versus the auxiliary field on the coarse grid. This operation is interpolated from the coarse grid with a grid-mapping type of interpolation; the dataset of weights and addresses has to be given by the user.

  This analysis requires one input line with 7 or 8 arguments depending on the type of subgrid interpolation.

  1. If the the SUBGRID operation is performed on a solar flux, the 7-argument input line is:

```
# SUBGRID operation with $SUBTYPE=SOLAR
  $CFILE  $NUMLU  $NID  $NV  $SUBTYPE  $CCOARSE $CFINE
```

`$CFILE` and `$NUMLU` are the subgrid-mapping file name and associated logical unit (see section 7.5 for the structure of this file); `$NID` the identificator for this subgrid-mapping dataset within the file build by OASIS based on all the different `SUBGRID` analyses in the present coupling; `$NV` is the maximum number of target grid points use in the subgrid-mapping; `$SUBTYPE = SOLAR` is the type of subgrid interpolation; `$CCOARSE` is the auxiliary field name on the coarse grid (corresponding to α) and `$CFINE` is the auxiliary field name on fine grid (corresponding to α_i). These two fields needs to be exchanged between their original model and OASIS3 main process, at least as `AUXILARY` fields. This analysis is performed from the coarse grid with a grid-mapping type of interpolation based on the `$CFILE` file.

2. If the the SUBGRID operation is performed on a nonsolar flux, the 8-argument input line is:

```
# SUBGRID operation with $SUBTYPE=NONSOLAR
  $CFILE $NUMLU $NID $NV $SUBTYPE $CCOARSE $CFINE $CDQDT
```

`$NV` are as for a solar subgrid interpolation; `$SUBTYPE = NONSOLAR`; `$CCOARSE` is the auxiliary field name on the coarse grid (corresponding to T) and `$CFINE` is the auxiliary field name on fine grid (corresponding to T_i); the additional argument `$CDQDT` is the coupling ratio on the coarse grid (corresponding to $\frac{\partial F}{\partial T}$) These three fields need to be exchanged between their original model and OASIS3 main process as `AUXILARY` fields. This operation is performed from the coarse grid with a grid-mapping type of interpolation based on the `$CFILE` file.

- **BLASNEW**:

  `BLASNEW` (routine `oasis3/src/blasnew.f`) performs a linear combination of the current coupling field with any other fields after the interpolation. These can be other coupling fields or constant fields.

  This analysis requires the same input line as `BLASOLD`.

- **MASKP**:

  A new analysis `MASKP` can be used to mask the fields after interpolation. `MASKP` has the same generic input line as `MASK`.

6.6 The post-processing

The following analyses are available in the post-processing part of OASIS3, controlled by `oasis3/src/postpro.f`.

- **REVERSE**:

  This transformation is obsolete in the current OASIS version.

  `REVERSE` (routine `oasis3/src/reverse.f`) reorders a field.

  This analysis requires the same input line as `INVERT`, with `$CORLON` and `$CORLAT` being now the resulting orientation. `REVERSE` does not work for U and D grids (see appendix A). Note that `INVERT` does not transform the associated grid or mask.

- **CHECKOUT**:

  `CHECKOUT` (routine `oasis3/src/chkfld.f`) calculates the mean and extremum values of an output field and prints them to the coupler output *cplout* (this operation does not transform the field).

  The generic input line is as for `CHECKIN` (see above).

- **GLORED**

  This transformation is obsolete in the current OASIS version as coupling fields can be directly interpolated to a target Reduced grid, if needed; this transformation should not be used anymore.

GLORED performs a linear interpolation of field from a full Gaussian grid to a Reduced grid. When present, GLORED must be the last analysis performed.

Before doing the interpolation, non-masked values are automatically extrapolated to masked points with EXTRAP/NINENN method (see above); to do so, the masked grid points are first replaced with a predefined value. The required global grid mask must be present in data file masks or masks.nc (see section 7.2).

The generic input line is as follows:

```
# GLORED operation
     $NNBRLAT  $NV   $NIO   $NID
```

is as for REDGLO (see REDGLO description above). The next 3 parameters refer to the EXTRAP/NINENN extrapolation (see EXTRAP/NINENN description above). The value assigned to all land points before interpolation is given by amskred in oasis3/src/blkdata.f; as for the $VALMASK in MASK analysis, it has to be chosen well outside the range of your field values but not too large to avoid any representation problem.

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`, that can be found in directory `oasis3/examples/toyoasis3/input` directory, contains a list of CF standard names and associated units identified with an index. The appropriate index has to be given by the user for each coupling or I/O field as the third entry on the field first line (see 5.3). This information will be used by OASIS3 for its log messages to *cplout* file and by the PSMILe to produce CF compliant NetCDF files.

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 in grid data files. Note that if the grid data files exist in the working directory, they are not overwritten by the PSMILe specific calls (see section 4.2). These files can be all binary or all NetCDF. In `oasis3/examples/toyoasis3/data`, NetCDF examples can be found.

The arrays containing the grid information are dimensioned `(nx, ny)`, where `nx` and `ny` are the grid first and second dimension, except for Unstructured (U) and Reduced (D) grid, for which the arrays are dimensioned `(nbr_pts,1)`, where `nbr_pts` is the total number of grid points.

1. *grids* or *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 (see `oasis3/src/mod_label.F90`.)

   For `SCRIPR` interpolations, the grid data files must be NetCDF files. 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 4.2). For source grids of Logically Rectangular LR type only, the grid corners will however be automatically calculated

and stored by OASIS if they are not initially available in *grids.nc*[1].

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). For example, the angles of the `torc` grid are given in array `torc.ang` in the grids.nc file in `oasis3/examples/testinterp/input`. 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.

File *grids* or *grids.nc* must be present with at least the grid point longitudes and latitudes for all component model.

2. *masks* or *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* or *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.
4. *maskr* or *maskr.nc*: *(this file is obsolete with the current OASIS3 version and should not be used anymore)* this file contains Reduced (D) grid mask in `INTEGER` arrays dimensioned `array(nbr_pts)` where `nbr_pts` is the total number of the Reduced grid points (0 -not masked- or 1 -masked- for each grid point). This file is required only for grids to which the `REDGLO` or `GLORED` transformation is applied. *As mentionned above, these transformations should not be used anymore as interpolations are now available for Reduced grids directly.* If used, the mask array name must be "MSKRDxxx" where "xxx" is half the number of latitude circles of the reduced grid (032 for a T42 for example).

If the binary format is used, *grids*, *masks*, *areas*, and *maskr* must have the following structure. The array name is first written to the file to locate a data set corresponding to a given grid. The data set is then written sequentially after its name. Let us call "brick" the name and its associated data set. The order in which the bricks are written doesn't matter. All the bricks are written in the grid data file in the following way:

```
...
WRITE(LU) array_name
WRITE(LU) auxildata
...
```

[1]Tip: to automatically calculate the corners of a Logically Rectangular LR target grid, use the corresponding reverse remapping in which the current target grid becomes the source grid.

- LU is the associated unit,
- array_name is the name of the array (CHARACTER*8),
- auxildata is the REAL or INTEGER array dimensioned (nx, ny) or (nbr_pts,1) containing the grid data.

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 4.8). When needed, these files are also automatically updated by the last prism_put_proto call of the run (see section 4.6.1) . To force the writing of the field in its coupling restart file, one can use the routine prism_put_restart_proto (see section 4.6.3). **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. (Note that in the NONE techniques, output files with the same format are also created for writing the resulting field after transformation.)

In the coupling restart files, the fields must be provided on the source grid in single or double precision REAL arrays (depending on PSMILe and OASIS3 compilation options) 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 ('U') and Reduced ('D) grid, for which the arrays are dimensioned (nbr_pts,1), where nbr_pts 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.

Both binary and NetCDF formats are supported; for NetCDF file the suffix .nc is not mandatory. If the coupling restart file for the first field is in NetCDF format, OASIS3 will assume that all coupling restart files (and output files for NONE communication techniques) are NetCDF[2].

In the NetCDF restart files, the field arrays must have the source symbolic name indicated in the *namcouple*, except for IGNORED fields for which the target symbolic name must be used. (see section 5.3).

In binary restart file, each field is written in the following way:

```
...
WRITE(LU) array_name
WRITE(LU) restartdata
...
```

- LU is the associated unit,
- array_name is the source symbolic name of the field (CHARACTER*8),
- restartdata is the restart field REAL array dimensioned (nx, ny) or (nbr_pts,1)[3]

Note that if using OASIS in the IPSL parallel mode (see section 8.2.1), the different OASIS3 executables cannot share the same coupling restart file. The recommendation here is to use one separate coupling restart file per coupling field.

[2]Note that even if the grid auxiliary data files are in NetCDF format, the restart coupling files may be in binary format, or vice-versa.

[3]If REDGLO is the first transformation applied on a Reduced grid field, the Reduced field must be given is an array restartdata(nx*ny) where nx and ny are the global Gaussian grid dimensions and the Reduced field is completed by trailing zeros. *Note that this transformation is obsolete in the current OASIS3 version and should not be used anymore.*

7.4 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 5.3.4). There must be one input file per INPUT field, containing a time sequence of the field in a single or double precision REAL array (depending on PSMILe compilation options), named with the field symbolic name in the *namcouple* and dimensioned (nx,ny,time) or (nbr_pts,1,time). The time variable as to be an array time(time) expressed in "seconds since beginning of run". The "time" dimension has to be the unlimited dimension. For a practical example, see the file SOALBEDO.nc in oasis3/examples/toyoasis3/data.

7.5 Transformation auxiliary data files

Many transformation need auxiliary data files, such as the grid-mapping files used for an interpolation. Some of them are created automatically by OASIS3, others have to be generated by the user before starting the coupled run.

7.5.1 Auxiliary data files for EXTRAP/NINENN, EXTRAP/WEIGHT, INTERP/SURFMESH, INTERP/GAUSSIAN, MOZAIC, and SUBGRID

The auxiliary data files containing the weights and addresses used in these transformations have a similar structure; their names are given in Table 7.1.

File name	Description
nweights	weights, addresses and iteration number for EXTRAP/NINENN interpolation
any name	weights and addresses for EXTRAP/WEIGHT extrapolation
mweights	weights and addresses for INTERP/SURFMESH interpolation
gweights	weights and addresses for INTERP/GAUSSIAN interpolation
any name	weights and addresses for MOZAIC interpolation
any name	weights and addresses for SUBGRID interpolation

Table 7.1: Analysis auxiliary data files

The files *nweights*, *mweights* and *gweights* can be created by OASIS3 if their corresponding $NIO = 1 (see EXTRAP/NINENN, INTERP/SURFMESH, INTERP/GAUSSIAN in sections 6.3 and 6.4).

The name of the (sub)grid-mapping files for MOZAIC, EXTRAP/WEIGHT and SUBGRID analyses can be chosen by the user and have to be indicated in the *namcouple* (see respectively sections 6.3 and 6.4 and 6.5). These files have to be generated by the user before starting the coupled run.

The structure of these files is as follows:

```
...
CHARACTER*8 cladress,clweight
INTEGER iadress(jpnb,jpo)
REAL weight(jpnb,jpo)
OPEN(unit=90, file='at31topa', form='unformatted')
WRITE(clweight,'(''WEIGHTS'',I1)') knumb
WRITE(cladress,'(''ADRESSE'',I1)') knumb
WRITE (90) clweight
```

```
      WRITE (90) weight
      WRITE (90) cladress
      WRITE (90) iadress
```

where

- jpnb is the maximum number of neighbors used in the transformation ($NVOISIN in the *namcouple*)
- jpo is the total dimension of the target grid
- at31topa is the name of the grid-mapping data file ($CFILE in *namcouple*)
- knumb is the identificator of the data set ($NID in *namcouple*)
- cladress is the locator of the address dataset
- clweight is the locator of the weight dataset
- iadress (i,j) is the address on the source grid of the i^e neighbor used for the mapping of the j^e target grid point. The address is the index of a grid point within the total number of grid points.
- weight(i,j) is the weight affected to the i^e neighbor used for the transformation of the j^e target grid point

For file *nweights*, there is an additional brick composed of a CHARACTER*8 variable (formed by the characters INCREME and by the data set identificator) and of an INTEGER array(N) which is the iteration number within EXTRAP/NINENN at which the extrapolation of the n^e grid point is effectively performed.

7.5.2 Auxiliary data files for FILLING

For the FILLING analysis, the global data set used can be either interannual monthly, climatological monthly or yearly (see 6.4). The name of the global data file can be chosen by the user and has to be indicated in the *namcouple* have to be given to OASIS through the input file *namcouple*. In case of monthly data, the file must be written in the following way:

```
      ...
      REAL field_january_year_01(jpi, jpj)
      ...
      WRITE(NLU_fil) field_january_year_01
      WRITE(NLU_fil) field_february_year_01
      WRITE(NLU_fil) field_march_year_01
      etc...
      WRITE(NLU_fil) field_december_year_01
C
C if climatology, one stops here
C
      WRITE(NLU_fil) field_january_year_02
      etc...
```

where

- field_... is the global dataset
- jpi and jpj are the dimensions of the grid on which FILLING is performed
- NLU_fil is the logical unit associated to the global data file and is defined in the input file *namcouple*

Note that the first month needs not to be January. This is the only file within OASIS in which the fields are not read using a locator.

7.5.3 Auxiliary data files for SCRIPR

The NetCDF files containing the weights and addresses for the SCRIPR remappings (see section 6.4) 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.

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 anywere and must be set in the variable `COUPLE` in the `make.`*your_platform* file. The choice of MPI1, MPI2 or NONE (interpolator-only mode, see section 6.1) is done by prescribing the value of `CHAN` and by activating the CPP key `-Duse_comm_$(CHAN)` in the `make.`*your_platform* header file.

The following commands are available:

- `make -f TopMakefileOasis3`
  compiles OASIS3 libraries *clim*, *anaisg*, *anaism*, *fscint*, *scrip* and main sources and creates OASIS3 main executable `oasis3.$CHAN.x` (where `$CHAN` is `MPI1`, `MPI2` or `NONE`) ;
- `make -f TopMakefileOasis3 oasis3_psmile`
  creates OASIS3 main executable as above, and compiles *mpp_io* and *psmile* sources to create the PSMILe library `libpsmile.$CHAN.a` (where `$CHAN` is `MPI1`, `MPI2`) that needs to be linked to the component executables;
- `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.

- Mandatory to indicate which communication technique will be used (see sections 4.1 and 5.2):

  - use_comm_MPI2 (by default): CLIM/MPI2
  - use_comm_MPI1 : CLIM/MPI1
  - use_comm_NONE : no communication technique for OASIS (interpolator-only mode NONE)

  The previous SIPC, PIPE and GMEM communication techniques are not available anymore.
- Mandatory when linking OASIS3 and PSMILe with a netCDF library (which is highly recommended[1])
  - use_netCDF
- Mandatory for compiling the *mpp_io* and *psmile* libraries:
  - use_libMPI
- Mandatory for compiling the *mpp_io* library if LAM implementation of MPI is used:
  - use_LAM_MPI
- To compile OASIS3 in IPSL or CMCC pseudo-parallel mode (see section 8.2 for details and restrictions).
  - use_oasis_para or use_oasis_cmcc_para
- To ensure, in SCRIPR/CONSERV remapping (see section 6.4), 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 6.4 for details).
  - NOT_NNEIGHBOUR
- To indicate the precision for REAL variables:
  - use_realtype_double (by default): to exchange double precision coupling fields declared as REAL(kind=SELECTED_REAL_KIND(12,307))
  - use_realtype_single: to exchange single precision coupling fields declared as REAL(kind=SELECTED_REAL_KIND(6,37))

  Note that if use_realtype_single is activated the compiling option promoting reals should be removed from F90FLAGS_1.
- For more information in *cplout* and in log files **.prt** during the psmile library exchanges (in particular, a message is printed when entering and leaving each main routine):
  - __VERBOSE
- For more debugging information to the log files **.prt** from the *mpp_io* library:
  - DEBUG
- The CPP key __DEBUG to activate :
  - deadlock detection in *clim* and *psmile* librairies at reception of a coupling field: a (non-standard) sleep function is called for one second in a loop testing if the field has been received; the code aborts after icountmax seconds if not (the length of the loop can be ajusted with the value of icountmax in CLIM_Import.F and mod_prism_get_proto.F90).
  - more debugging information in log files **.prt** during the *psmile* library I/Os;
  - in SCRIPR vector transformation, for writing the resulting vertical component in the spherical coordinate system after interpolation to a file projection.nc (see section 6.4).

[1] Linking with netCDF is mandatory when using SCRIPR transformations (see section 6.4).

- To get some statistics on the wall clock time spent in the coupling
  - `balance`
- To compile the PSMILe communication library without the I/O functionality (see section 5.3), i.e to compile only empty routines in `oasis3/lib/mpp_io`:
  - `key_noIO`
- For compiling without linking the SCRIP interpolation library:
  - `key_noSCRIP`
- To compile on NEC SX platforms (optimisation in `oasis3/src/extrap.F` and proper value for `ip_i8_p` in `oasis3/lib/psmile/src/mod_kinds_model.F90` and `oasis3/lib/mpp_io/src/mod_kinds_mpp.F90`:
  - `SX`
- Other platform dependent CPP keys are used in `oasis3/lib/mpp_io/include/os.h`, `oasis3/lib/psmile/include/psmile_os.h` and `oasis3/src/mod_kinds_oasis.F90`; they should be automatically activated on the corresponding platforms.

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:

- Putting `$NLOGPRT = 2` in your namcouple (see section 5.2)
- Compiling with CPP key `__VERBOSE` which will result in more information printed in *cplout* and in log files **.prt** during the psmile library exchanges (in particular, a message is printed when entering and leaving each main routine)
- Compiling with CPP key `DEBUG` which will result in more information printed in the log files **.prt** from the *mpp_io* library

8.2 Running OASIS3 in parallel mode

Two modes of parallelisation, both working on a field-per-field basis, were developed concurrently for OASIS3, one by IPSL finalized by CERFACS, and one by CMCC. These modes allows different OASIS3 processes to treat different subsets of coupling fields. These parallelisation modes can be applied with any number of OASIS3 processes; they are in fact only limited by the number of coupling fields exchanged within the coupled model.

With both modes, OASIS3 log file (i.e. *cplout_x*) and auxiliary files for transformations `EXTRAP/NINENN`, `INTERP/SURFMESH` and `INTERP/GAUSSIAN` (i.e. *mweights_x*, *nweights_x*, *gweights_x*, *anaisout_x*, see 7.5.1) , and for `SCRIPR` (see 7.5.3) are suffixed with the number of the corresponding OASIS3 process performing the transformation. This is done automatically by the OASIS3 process when the file is created by OASIS3 or has to do so by the user if he provides the file before hand.

Note that if the CPP keys `use_oasis_para` or `use_oasis_cmcc_para` are not activate, these new parallelisation modes have no impact on the use of OASIS3. Here are the details on the two modes.

8.2.1 IPSL parallelisation

To run the OASIS3 executable on more than one process in IPSL parallelisation mode, OASIS3 must be compiled with the CPP key `use_oasis_para` (see 8.1.2).

With IPSL parallelisation mode, each OASIS3 process is totally independent of the others. Therefore, the user has to provide one separate configuration file per process and the different files must be named *namcouple_x* where x is the number of the corresponding OASIS3 process. Each OASIS3 process will

receive, treat, and send the coupling fields described in its configuration file *namcouple_x*. Note that if OASIS3 was compiled with the CPP key `use_oasis_para`, the configuration file suffix is mandatory, even if OASIS3 runs in fact with only one process - in this case, the suffix is _0.

Although this parallelisation mode presents the advantage of removing the bottleneck that can appear when only one OASIS process receives and sends all the coupling fields, there are few constraints associated:

- As stated above, the configuration file `namcouple` has to be splitted into different `namcouple_x`, one per process. In one `namcouple_x` the total number of coupling fields treated by OASIS3 process `x` must be given on the line below the `$NFIELDS` keyword and only these coupling fields must be detailed.

  If the maximum number of prism_def_var_proto called by ANY component model in the coupled system is greater than twice the number of fields listed in the *namcouple_x* (which is often the case when OASIS3 is used in parallel mode), **this maximum number of prism_def_var_proto has to be specified on the same line** (below the `$NFIELDS` keyword) after the total number of fields exchanged (see also section 5.2).
- This mode is available only with the `MPI1` CLIM communication technique (see 5.2).
- In this mode, the grid data files have to be created by the user before the run (see also 4.2).
- Binary restart files do not work with this mode; NetCDF files have to be used
- The different OASIS3 executables cannot share the same coupling restart file (see section 7.3). Therefore, the same coupling restart file cannot be indicated in different *namcouple_x* configuration files. The recommendation here is to use one separate coupling restart file per coupling field.

8.2.2 CMCC parallelisation

To run the OASIS3 executable on more than one process in CMCC parallelisation mode, OASIS3 must be compiled with the CPP key `use_oasis_cmcc_para` (see 8.1.2). (XXX to be completed)

8.3 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 8.3.1), then instructions on how to compile and run TOYOASIS3 are given in section 8.3.2.

8.3.1 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 4.1), 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).

	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).

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

$Field_1$ 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.

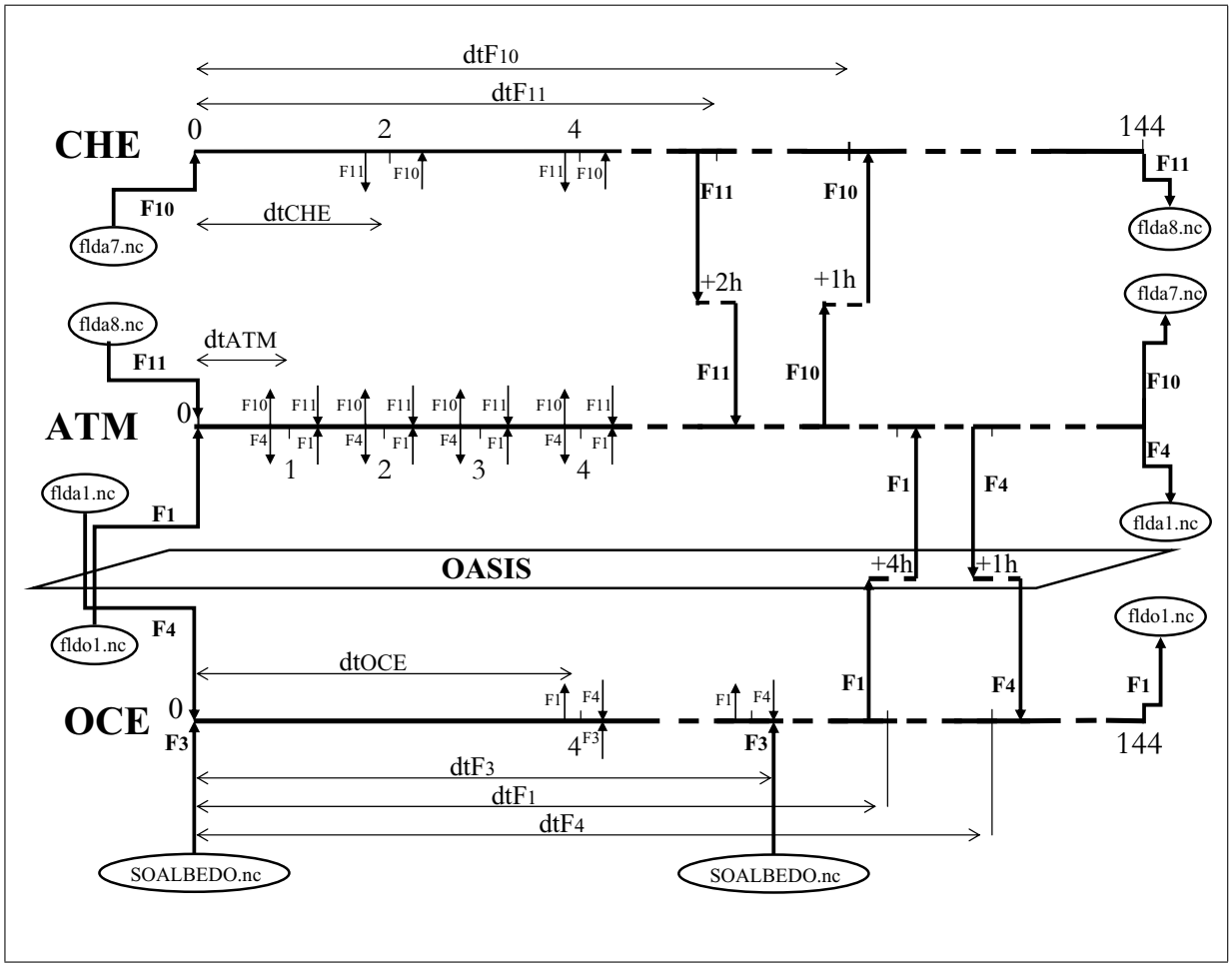

Figure 8.1: Exchange algorithm between the 3 TOYOASIS3 component models for fields $Field_1$, $Field_3$, $Field_4$, $Field_{10}$, and $Field_{11}$.

8.3.2 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 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.

Different modes can be tested.

- To run in the classic (not parallel) mode, compile without the CPP key use_oasis_para or use_oasis_cmcc_para, and set ipsl_comp_para=0 and cmcc_comp_para=0 in the running script. One can now choose between running atmoa3 and cheoa3 toy models on one process only (nproc_atmche=1 and ncpl_atmche=1) or on 3 processes each (nproc_atmche=3 and ncpl_atmche=3). The results for the first and second cases are respectively available in oasis3/examples/toyoasis3/outdata/results_mono_111 and oasis3/examples/toyoasis3/outdata/results_mono_313 (XXX this last directory is not updated yet).

  Atmoa3 can run on 1 or 3 processes, depending on the value of the variable nproc_atmche in script run_toyoasis3. When running on 3 processes, atmoa3 can either exchange coupling data through its 3 processes (ncpl_atmche = 3 in run_toyoasis3 and il_nbcplproc = 3 in atmoa3.F90) or only through the master process (ncpl_atmche = 1 in run_toyoasis3 and il_nbcplproc = 1 in atmoa3.F90). When atmoa3 runs or exchanges data with only one process, it defines one Serial partition containing the 96X48 grid points. If it runs and exchanges coupling data with 3 processes, its decomposition depends on the cdec parameter hard coded in routine decomp_def.F90. When cdec = APPLE, each of the 3 atmoa3 processes calls the PSMILe prism_def_partition routine to define 1 segment of an APPLE decomposition (1536 grid points per segment). If cdec = BOX, each process will define 1 'box' of a BOX decomposition and will treat a box of 32X48 points. If the user hardcodes cdec = ORANGE, each process will define a partition of two segments of 768 points distant of 1536 points.

- Setting gridswr=1 in the script run_toyoasis3 will make the toy models to create the grid data files *grids.nc, masks.nc* and *areas.nc* and to write their grids, corners, areas and masks in these files using the prism_write_grid, prism_write_corner, prism_write_mask and prism_write_area routines (see 4.2). This does not change the results.

- To run in IPSL parallelisation mode, set ipsl_comp_para=1. In this case, make sure that the CPP key use_oasis_para was used for compilation. The script will launch OASIS3 on two processes treating each one a subset of the coupling fields; the first OASIS3 process will treat the coupling fields listed in *namcouple_0* while the second will treat the coupling fields listed in *namcouple_1*. For more details in the IPSL parallelisation mode, see section 8.2.1. The results one should get with ipsl_comp_para=1 (in the case the atmoa3 and cheoa3 toy models run on one process only) are available in oasis3/examples/toyoasis3/outdata/results_para. (XXX This last directory is not updated yet XXX).

- To run in CMCC parallelisation mode, set cmcc_comp_para=1. In this case, make sure that the CPP key use_oasis_cmcc_para was used for compilation. For more details in the CMCC parallelisation mode, see section 8.2.2. The results one should get with cmcc_comp_para=1 are available in ... (XXX to be completed)

8.4 Running OASIS3 in interpolator-only mode

OASIS3 can be used in an interpolator-only mode, in which case it transforms fields without running any model (see section 6.1). Two test-cases are provided with OASIS3 to illustrate its uses in this mode, the "testinterp" test-case (see section 8.4.1) and the "testNONE" test-case (see section 8.4.2).

8.4.1 The "testinterp" test-case

The "testinterp" test-case can be run to test the interpolation of different source fields corresponding to analytical functions and to evaluate the error between the interpolated fields and the same analytical functions calculated on the different target grids.

All files needed to run this test-case can be found in `oasis3/examples/testinterp/input` and `oasis3/examples/testinterp/restart`.

To run "testinterp", OASIS3 first has to be compiled (see section 8.1.1) in interpolator-only mode NONE, i.e. by putting `CHAN = NONE` in the TopMakefileOasis3 header file. Then the programs that will calculate the interpolation error, i.e. `gen_error.f90` and `gen_error_vector.f90` (for vector fields) in directory `oasis3/examples/testinterp/error` have to be compiled (see script `sc_comp_error`).

Then, one has to adapt and execute the running script `oasis3/examples/testinterp/sc_run_testinterp`. With `TIME=ONE`, the configuration file `oasis3/examples/testinterp/input/namcouple_ONE`, the input files `flda1.nc, flda2.nc, flda3.nc, fldb1.nc, fldo1.nc` and `fldz1.nc` from `oasis3/examples/testinterp/restart` and the input files `aaIin.nc` and `caJin.nc` from `oasis3/examples/testinterp/restart/vector` are used. This example also shows one vector interpolation (field components `a_at42_I` and `c_at42_J`). The test-case automatically writes the error fields in `error_*.nc` files and error statistics in `log_*` files.

To run the example into which OASIS3 interpolates many time occurrences from one input file, put `TIME=MANY` in `sc_run_testinterp`. The configuration file `oasis3/examples/testinterp/input/namcouple_MANY` and the input file `fldin.nc` in `oasis3/examples/testinterp/restart` are then used.

The results obtained after running the testinterp test-case should match the ones in `oasis3/examples/testinterp/outdata`. (XXX this directory is not updated yet).

8.4.2 The "testNONE" test-case

All files to run the "testNONE" test-case can be found in `oasis3/examples/testNONE`. This test-case provides a flexible environnement to test the interpolation specified in the `INPUT/namcouple` configuration file from a source grid to a target grid, both grids being defined in regular OASIS3 grid data files *grids.nc*, *masks.nc*, *areas.nc* (see section 7.2).

To run "testNONE", OASIS3 first has to be compiled (see section 8.1.1) in interpolator-only mode NONE, i.e. by putting `CHAN = NONE` in the TopMakefileOasis3 header file. The user then has to adapt the "User specifications" part of the running script `sc_run_NONE`. In particular, he has to specify:

- `DIRWORK` : the directory where to run the test-case
- `BINDIR` : the directory containing OASIS3 executable
- `SRCROOT` : the source of the `oasis3` directory
- `ARCH` : the platform onto which the test-case will run
- `DIR_GRD` : the directory for the grid data files
- `DIR_INP` : the directory for the `namcouple` and `cf_name_table.txt` files
- `SRC_GRID` and `TGT_GRID` : the source and target grid prefixes in the `namcouple` and in the grid data files (see section 5.3.1),
- `FLD_NBR` : the number of the analytic function chosen,
- `MASKERROR` : whether or not the error on the target grid will be calculated on all points (`MASKERROR=NOT`) or only on non masked points (`MASKERROR=YES`).

When launched, the running script `sc_run_NONE`:

- creates a working directory

- compiles and runs the program `PROG/create_inputfield.f90` that creates an input field using the chosen analytical function on the specified source grid in file `fldin.nc`
- copy all required input and data file to the working directory
- run OASIS3 that interpolates the analytical field from `fldin.nc` with the interpolation specified in the `namcouple`
- compile and run the program `PROG/create_errorfield.f90` that calculates the error between the resulting interpolated field and the field defined by the chosen analytical function on the specified target grid, and writes it to the file `error.nc`

8.5 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 INTERP interpolations (see section 6.4)

   - 'A' grid: this is a regular Lat-Lon grid covering either the whole globe or an hemisphere, going from South to North and from West to East. There is no grid point at the pole and at the equator, and the first latitude has an offset of 0.5 grid interval. The first longitude is 0^o (the Greenwhich meridian) and is not repeated at the end of the grid ($CPER = P and $NPER= 0). The latitudinal grid length is 180/NJ for a global grid, 90/NJ otherwise. The longitudinal grid length is 360/NI.
   - 'B' grid: this is a regular Lat-Lon grid covering either an hemisphere or the whole globe, going from South to North and from West to East. There is a grid point at the pole and at the equator (if the grid is hemispheric or global with NJ odd). The first longitude is 0^o (the Greenwhich meridian), and is repeated at the end of the grid ($CPER = P and $NPER= 1). The latitudinal grid length is 180/(NJ-1) for a global grid, 90/(NJ-1) otherwise. The longitudinal grid length is 360/(NI-1).
   - 'G' grid: this is a irregular Lat-Lon Gaussian grid covering either an hemisphere or the whole globe, going from South to North and from West to East. This grid is used in spectral models. It is very much alike the A grid, except that the latitudes are not equidistant. There is no grid point at the pole and at the equator. The first longitude is 0^o (the Greenwhich meridian) and is not repeated at the end of the grid ($CPER = P and $NPER= 0). The longitudinal grid length is 360/NI.
   - 'L' grid: this type covers regular Lat-Lon grids in general, going from South to North and from West to East.. The grid can be described by the latitude and the longitude of the southwest corner of the grid, and by the latitudinal and longitudinal grid mesh sizes in degrees.
   - 'Z' grid: this is a Lat-Lon grid with non-constant latitudinal and longitudinal grid mesh sizes, going from South to North and from West to East. The deformation of the mesh can be described with the help of 1-dimensional positional records in each direction. This grid is periodical ($CPER = P) with $NPER overlapping grid points.
   - 'Y' grid: this grid is like 'Z' grid except that it is regional ($CPER = R and $NPER = 0).

2. 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_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 8.2 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 6.4 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 6.5.
- 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 6.4.
- 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 8.3. 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 4.2 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.2 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 6.4), 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 6.4).
  - 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 4.6.3).
  - 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.3 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 6.4).
- 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.4 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 6.4.
- All include of mpif.h are now written '#include <mpif.h>'.
- The output format of CHECKIN and CHECKOUT results is now E22.7

B.5 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.6 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 5.3.2)
- 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 6.1.

- 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 4.2 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.