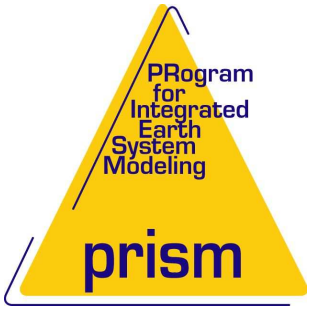
PRISM
An Infrastructure Project for Climate Research in Europe



OASIS3 User Guide

prism_2-5

Edited by:
S. Valcke, CERFACS

PRISM–Support Initiative
Report No 3

September 2006

Copyright Notice

How to get assistance?

Assistence can be obtained as listed below.
PRISM documentations can be downloaded from the WWW PRISM web site under the URL: <http://prism.enes.org>

Phone Numbers and Electronic Mail Adresses

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

Contents

Chapter 1

Acknowledgements

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):

Arnaud Caubel (FECIT/Fujitsu)

Damien Declat (CERFACS)

Veronika Gayler (MPI-M&D)

Josefine Ghattas (CERFACS)

Jean Latour (Fujitsu-Fecit)

Eric Maisonnave (CERFACS)

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):

Dominique Astruc (IMFT)

Sophie Belamari (Météo-France)

Dominique Bielli (Météo-France)

Gilles Bourhis (IDRIS)

Pascale Braconnot (IPSL/LSCE)

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)
René Redler (NEC-CCRLE)
Tim Stockdale (ECMWF)
Rowan Sutton (UGAMP)
Véronique Taverne (CERFACS)
Jean-Christophe Thil (UKMO)
Nils Wedi (ECMWF)

Chapter 2

Introduction

OASIS3 is the direct evolution of the OASIS coupler developed since more than 10 years at CERFACS (Toulouse, France). OASIS3 is a portable set of Fortran 77, Fortran 90 and C routines. At run-time, OASIS3 acts as a separate mono process executable, which main function is to interpolate the coupling fields exchanged between the component models, and as a library linked to the component models, the OASIS3 PRISM Model Interface Library (OASIS3 PSMILe). OASIS3 supports 2D coupling fields only. To communicate with OASIS3, 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 main process based on Message Passing Interface (MPI) and file I/O using the mpp_io library from GFDL. Portability and flexibility are OASIS3 key design concepts. OASIS3 has been extensively used in the PRISM demonstration runs and is currently used by approximately 15 climate modelling groups in Europe, USA, Canada, Australia, India and Brasil. The current OASIS3 version and its toy coupled model TOYCLIM were compiled and run on NEC SX6, IBM Power4 and Linux PC cluster, and previous OASIS3 versions were run on many other platforms.

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 TOYCLIM coupled model gives a practical example of a coupled model; the sources are given in directories `/prism/src/mod/` to `yat`, `/toyoce`, `/toyche` ; more detail on TOYCLIM 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; prepare OASIS3 configuring file *namcouple* (See chapter 5). OASIS3 supports different interpolation algorithms as is described in chapter 6.
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 TOYCLIM coupled model.

The appendix D lists (some of) the coupled models realized with OASIS within the past 5 years or so.

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

[1]The SIPC, PIPE and GMEM communication techniques available in previous versions should still work but are not maintained anymore and were not tested.

Chapter 3

The OASIS3 sources

The sources and data of OASIS3, all related libraries, and TOYCLIM coupled model are available from CERFACS CVS server `alter` and from CERFACS anonymous ftp.

The CVS repository is `/home/oasis/PRI SMC VS`. To obtain the CVS login and password as well as the most recent OASIS3 tag, please contact us (see contact details on the back of the cover page).

Note that OASIS3 is temporarily released without the corresponding PRISM Standard Compile Environment and Running Environment (SCE/SRE); they will be included when the migration from CVS to Subversion will be realized at CERFACS.

OASIS3 directory structure follows the PRISM standard one:

```
- prism/data/tes   ti nt er p      data  and  results   for  OASIS3   in  mode  NONE
             /toyclim              data  and  results   for  TOYCLIM   coupled   model

- prism/src/lib/   an ai sg        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

- prism/src/mod/   oa si s3 /sr c  OASIS3   main   code
                         /doc     OASIS3   documentation
                         /util    Utilities    to  compile   OASIS3

                  /toyatm          TOYCLIM   component    model  1
                  /toyoce          TOYCLIM   component    model  2
                  /toyche          TOYCLIM   component    model  3

- prism/util/run   ni ng /t est in te rp    environment    to  test  OASIS3   interpolation
                       /testNONE           (interpolator-o    nl y mode  NONE)
                       /toyclim            environment    to  run  the  TOYCLIM   toymodel
```

Chapter 4

Interfacing a model with the PSMILe library

At run-time, OASIS3 acts as a separate mono process 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 in the configuration file *namcouple* (see section 5). In OASIS3, the CLIM communication technique based on MPI1 or MPI2 message passing and the associated model interface library PSMILe, should be used[1]. For a practical toy model using the PSMILe library, see the sources in `/prism/src/mod /toyatm, /toyche, /toyoce` 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 `prism/src/lib/psmile` directory. PSMILe supports:

- parallel communication between a parallel component model and OASIS3 main process,
- 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.

[1]The SIPC, PIPE and GMEM communication techniques available in previous versions should still work but are not maintained anymore and were not tested.

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.[2]

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

  For CLIM-MPI1 communication technique: routine called by all model processes to get the value of a local communicator to be used by the model for its internal parallelisation.

  In fact, 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 following a "coloring scheme"; its value is returned as the first argument of prism_get_localcomm_proto routine.

  With CLIM-MPI2, the communicator MPI_COMM_WORLD will be returned as local communicator.

  Besides that, the differences between using PSMILe with MPI1 or MPI2 message passing are
  - The $CHANNEL in the namcouple; see section 5.2.
  - The way the models are started. With MPI2, only OASIS3 needs to be started at the command line; it will then spawn the component models at the beginning of the run. With MPI1, models have to be started by the user in a pseudo-MPMD mode; the way to do this depends on the computing platform. For more details, see section **??**.

4.2 Grid data file definition

The grid data files *grids.nc, masks.nc* and *areas.nc* must be created by the user before the run or can be written directly at run time by the master process of each component model.

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 (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 definition in the grid data files.

The coupler sends the information on whether or not writing is needed to the models following an OASIS internal order (below prism_start_grids_writing). If no writing is needed, nothing happens when calling the following prism_write_xxxx routines. If writing is needed, the first model creates the files, writes the data arrays (with `prism_write_grid` , `prism_write_corner` , `prism_write_mask` , `prism_write_area` calls), and then sends a termination flag to the coupler (below

[2]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 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 that may 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 .

The TOYCLIM coupled model uses those routines to write its grid data files if gridswr=1 in the running script run_toyclim (see section **??**).

- 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 1/0 if grids writing is needed/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] : grid dimension in x-direction
  - ny [INTEGER; IN] : grid dimension in y-direction
  - 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] : grid dimension in x-direction
  - ny [INTEGER; IN] : grid dimension in y-direction
  - nc [INTEGER; IN] : number of corners per grid cell (4)
  - 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_mask (cgrid, nx, ny, mask)

  - cgrid [CHARACTER*4; IN] : grid name prefix
  - nx [INTEGER; IN] : grid dimension in x-direction
  - ny [INTEGER; IN] : grid dimension in y-direction
  - 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] : grid dimension in x-direction
  - ny [INTEGER; IN] : grid dimension in y-direction
  - 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 sends directly its partition to OASIS3 main process, or directly to the other component model if no transformation nor repartition is required. To do so, each process implied in the coupling 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.

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) :

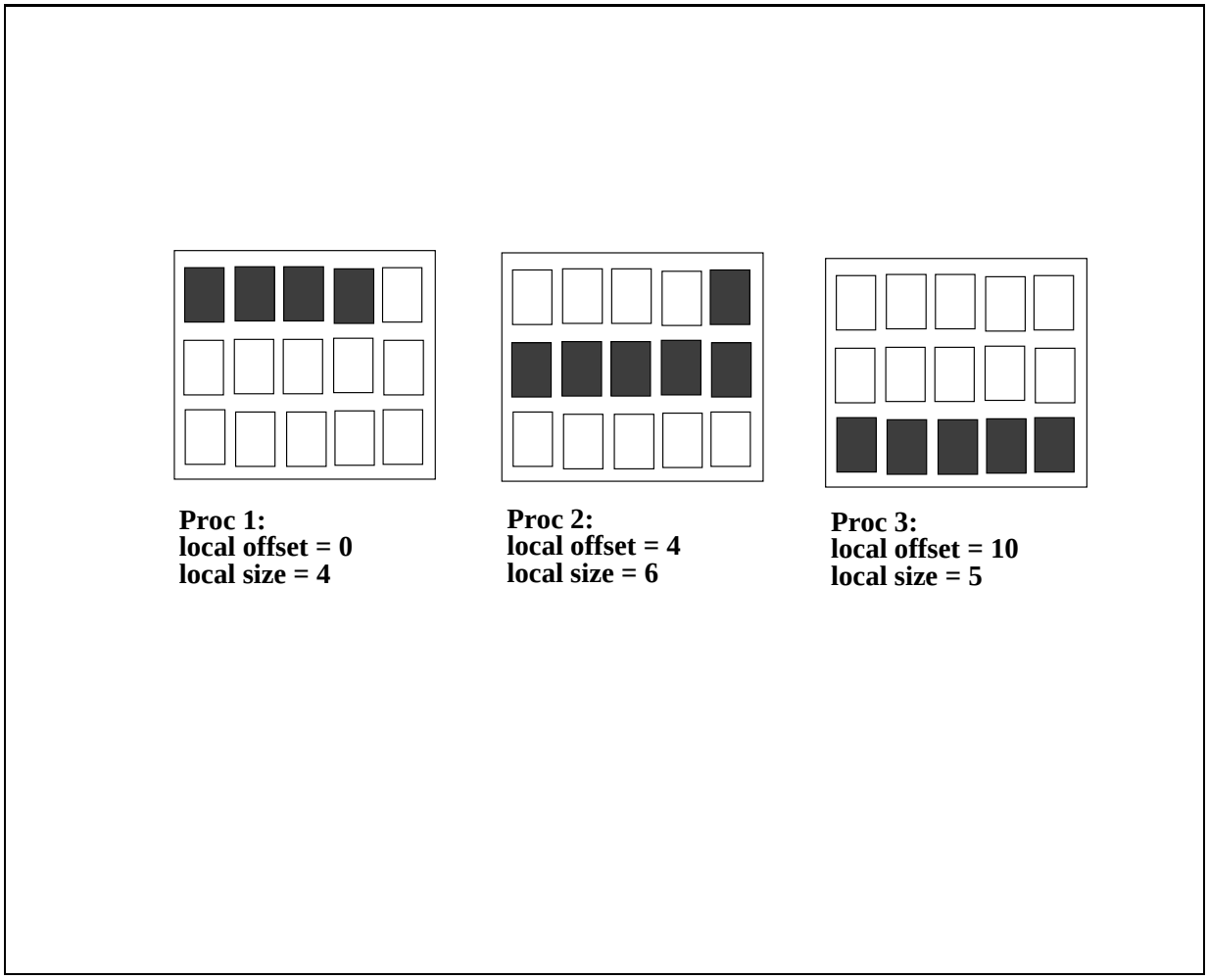


Figure 4.1: Apple partition

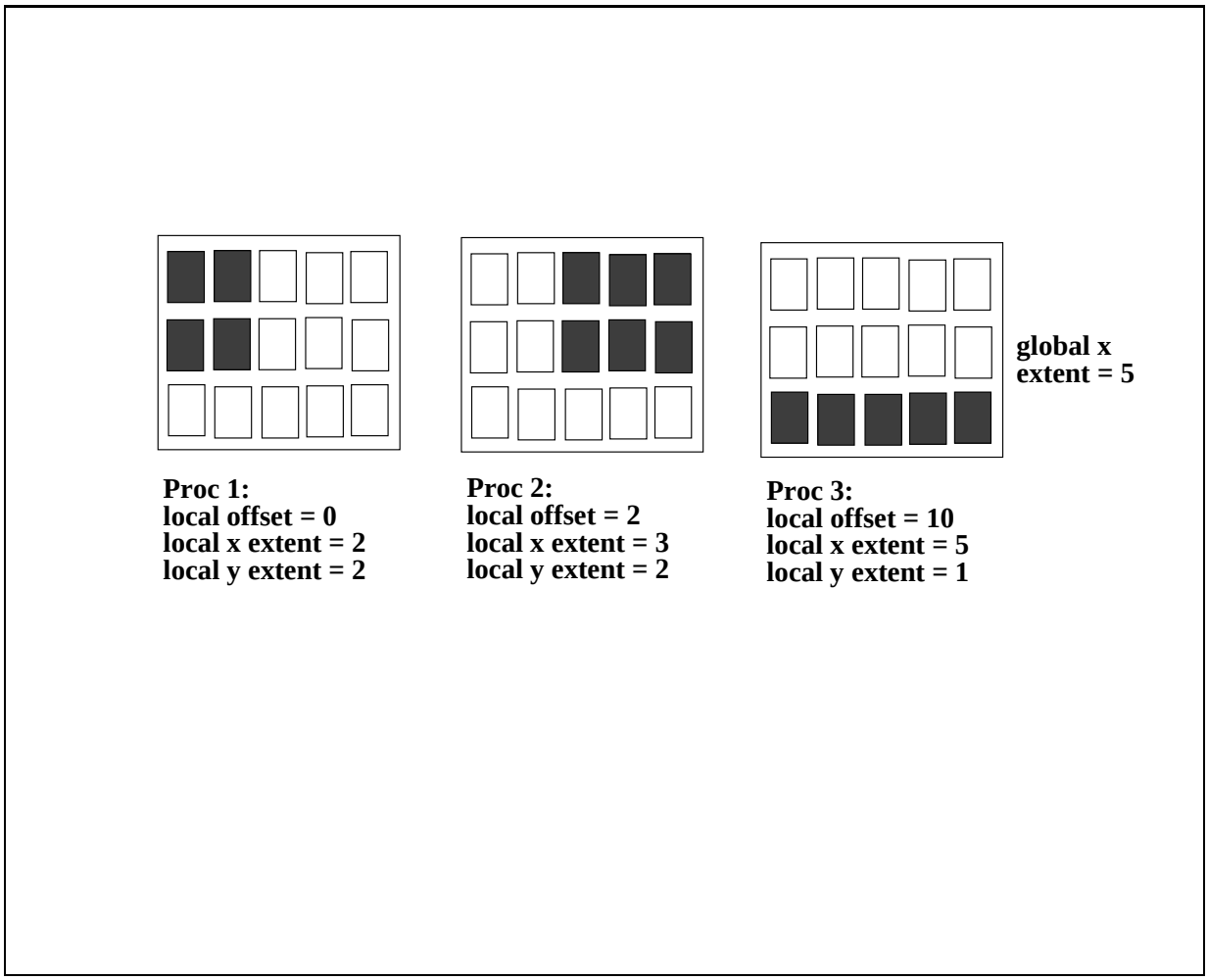


Figure 4.2: Box partition

- `ig_paral(1)` = 2 (indicates a Box partition)
- `ig_paral(2)` = the upper left corner global offset
- `ig_paral(3)` = the local extent in X
- `ig_paral(4)` = the local extent in Y[3]
- `ig_paral(5)` = the global extent in X.

Figure 4.2 illustrates a Box partition over 3 processes.

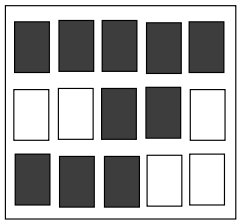
4.3.4 Orange partition

Each partition is an ensemble of segments of the global domain. Each segment is described by its global offset and its local extent. In this case, we have `ig_paral(1:N)` where `N = 2 + 2*number of segments` [4].

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

[3] The maximum value of the local extent in Y is presently 338; it can be increased by modifying the value of `Clim_MaxSegments` in `prism/src/lib/clim/src/mod_clim.F90` and in `prism/src/lib/psmile/src/mod_prism_proto.F90` and by recompiling Oasis3 and the PSMILe library.

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



Proc 1:
nbr of segments = 3

1st segment offset = 0
1st segment size = 4
2nd segment offset = 7
2nd segment size = 2
3rd segment offset = 10
3rd segment size = 3

Figure 4.3: Orange partition for one process

- `ig_paral(4)` = the first segment local extent
- `ig_paral(5)` = the second segment global offset
- `ig_paral(6)` = the second segment local extent
- ...
- `ig_paral(N-1)` = the last segment global offset
- `ig_paral(N)` = the last segment local extent

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

4.4 I/O-coupling field declaration

Each process implied in the coupling 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[5]. No automatic conversion is implemented; therefore, all coupling fields exchanged through OASIS3 main process must be of same type[6].
- ierror [INTEGER; OUT] : returned error code.

4.5 End of definition phase

Each process implied in the coupling 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 implied in the coupling 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 beginning of the timestep
  - 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.

[5]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 pre-compiling key use_realtype_single, the coupling fields must be declared REAL(kind=SELECTED_REAL_KIND(6,37)) in the component models (see also chapter 8).

[6]Coupling fields exchanged directly between two component models can have a type different from the ones exchanged through OASIS3 main process, as long as they are single or double precision real arrays in both models.

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.

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 blocking 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 implied in the coupling 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, ierror)
  - var_id [INTEGER; IN] : field ID (from corresponding prism_def_var_proto)
  - date [INTEGER; IN] : number of seconds in the run at the beginning of the timestep
  - 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 beginning of the timestep
  - 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 beginning of the timestep
  - 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).

4.7 Termination

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

  All processes must terminate the coupling by calling prism_terminate_proto [7] (normal termination). Oasis will terminate after all processes called prism_terminate_proto. With MPI2, the run may be considered finished when Oasis terminates; to avoid problem, place the call to prism_terminate_proto at the very end in the component model code.
- 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

[7] 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 .

– abort_message; IN] : message to be written out.

If a process needs to abort (abnormal termination), it must 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).

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 arguments on both sides match 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.

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.

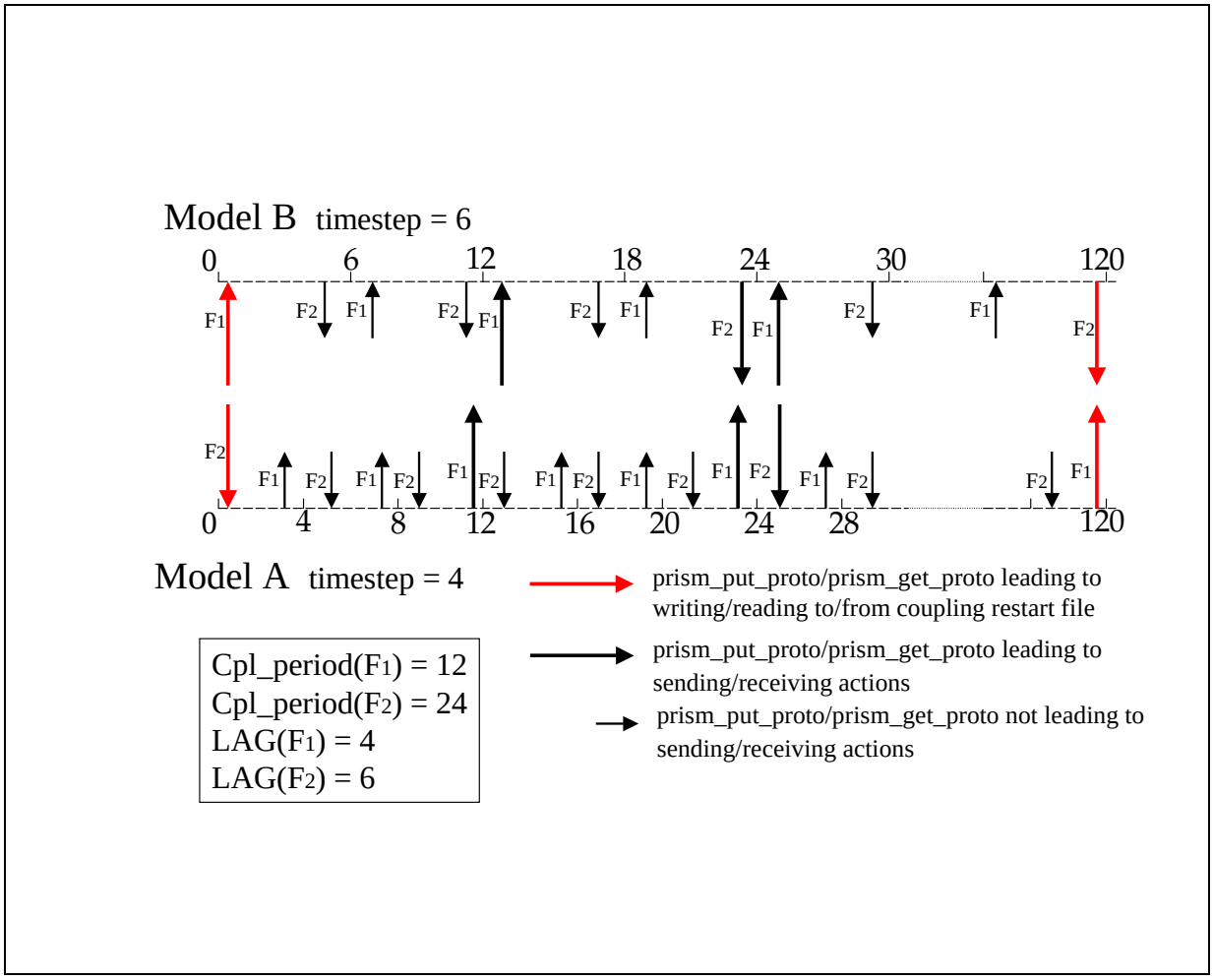


Figure 4.4: LAG concept first example

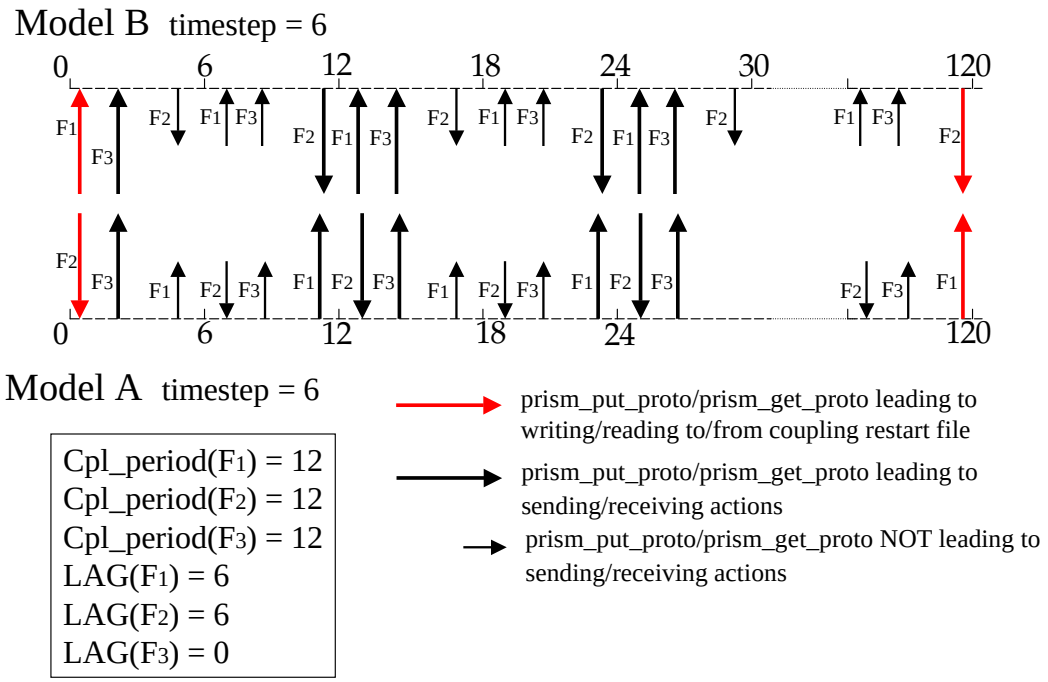


Figure 4.5: LAG concept second example

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 `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` will automatically lead to reading F_1 and F_2 from their coupling restart files. The user therefore have to create those coupling restart files for 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` if 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 + 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, below the prism_get_proto in the target model, coupling fields produced, below the prism_put_proto 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.

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.

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 4.7, the models perform their prism_put_proto and prism_get_proto calls exactly as in the first lag example above: 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 below 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

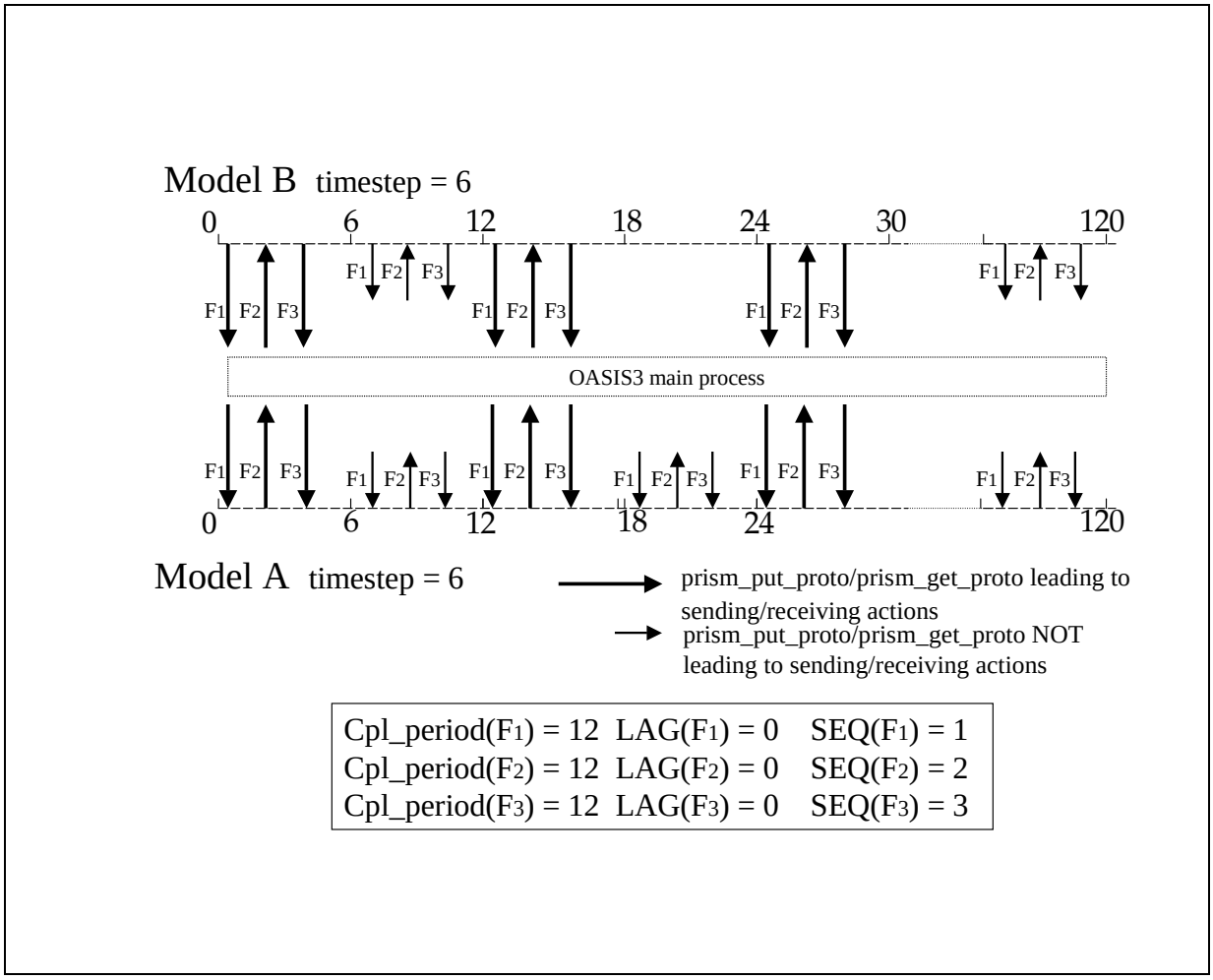


Figure 4.6: The SEQ concept

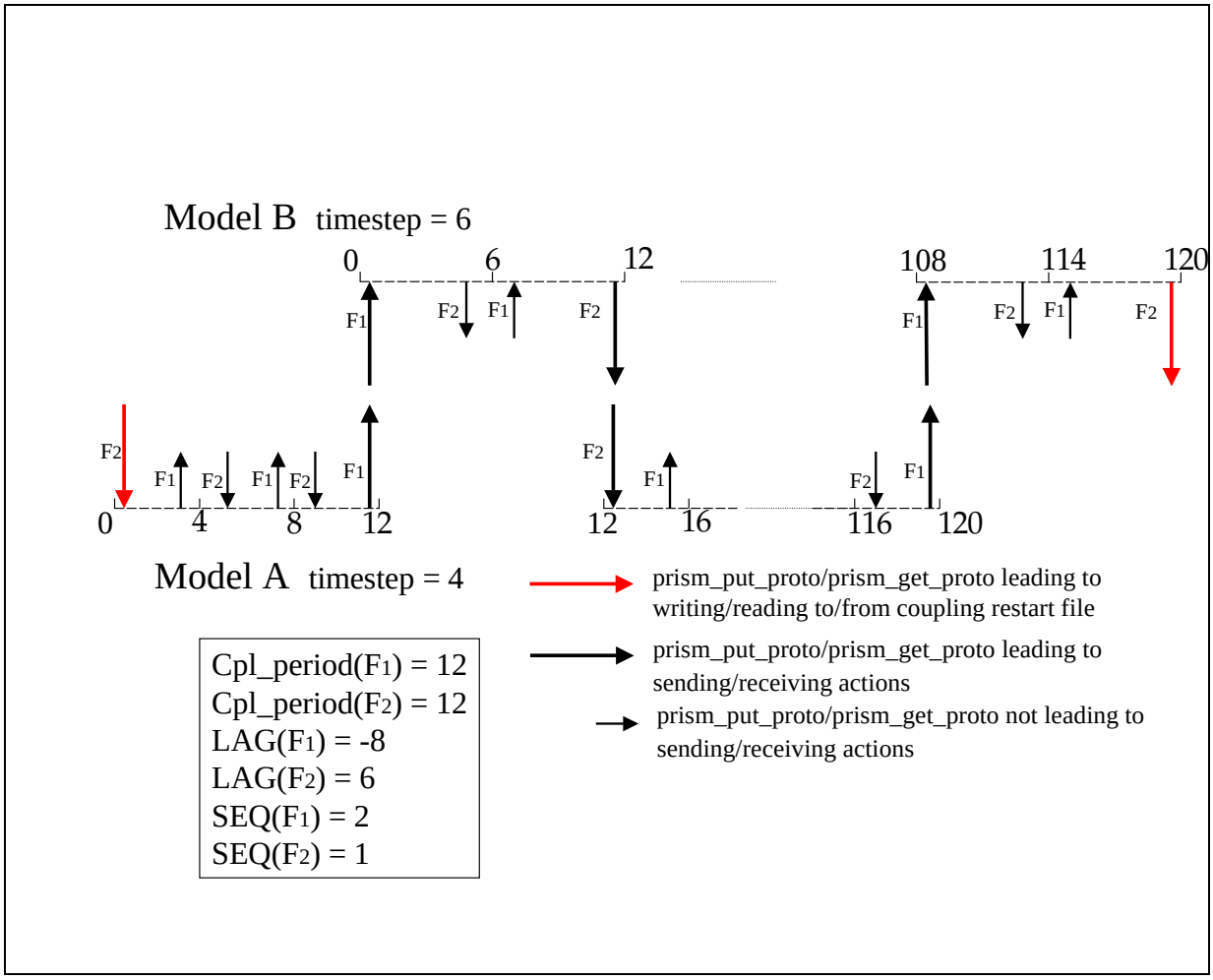


Figure 4.7: Mix of LAF and SEQ concepts

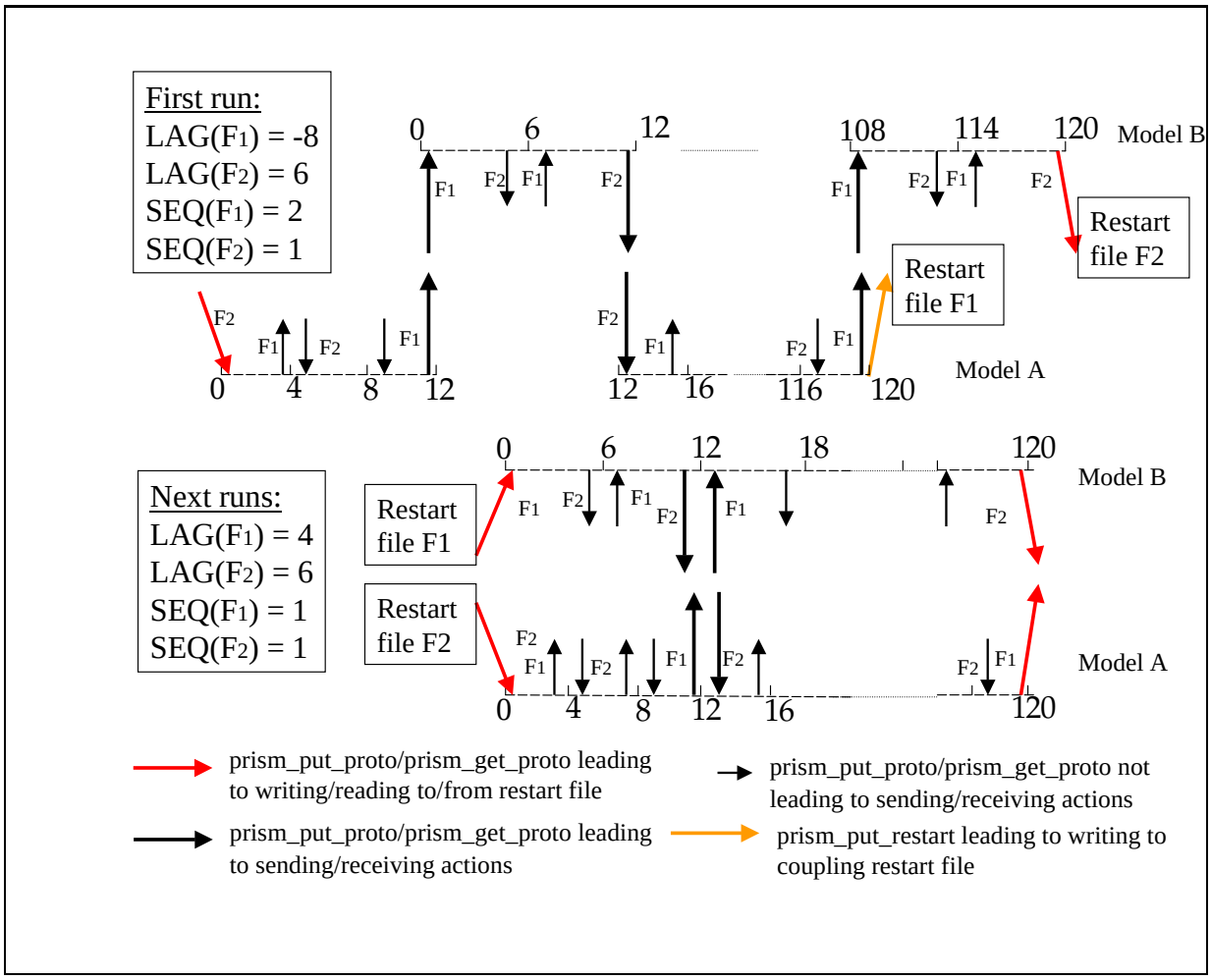


Figure 4.8: An example using prism_put_restart_proto

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 main process, it is important to indicate a sequence index. In fact, at each OASIS3 main process coupling timestep, F_1 is necessarily treated after F_2. Therefore, $SEQ(F_1) = 2$ and $SEQ(F_2) = 1$.

4.8.4 Mixing sequential and parallel runs using `prism_put_restart_proto`

In the example illustrated on figure 4.8, 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. 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 their associated configuring lines (described in more details in section 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 directory `/prism/util/running/toyclim/input/namcouple`.

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
#
 $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    7    flxat.nc     EXPORTED
 atmo     toce    LAG=+14400     SEQ=+1
 P  0  P  2
 LOCTRANS     CHECKIN     SCRIPR    CHECKOUT
```

```
#
  ACCUMUL
  INT=1
  CONSERV     LR    SCALAR     LATLON      10    FRACAREA       FIRST
  INT=1
#
# Field
#
 COSENHFL     SOSENHFL     37   86400      1   flda3.nc     IGNOUT
 atmo     atmo   LAG=+7200      SEQ=+1
 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 number of fields that have to be, at one particular coupling timestep, necessarily exchanged sequentially in a given order. For $SEQMODE ≥ 1, the position of each coupling field in the sequence has to be given by its SEQ index (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, using MPI1 or MPI2 message passing. 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 should still work but are not officially supported anymore and were not tested.

  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 blocking send MPI_Send should be used in place of the buffered MPI_BSend to send the coupling fields.[1]

[1] Use the standard blocking send MPI_Send if the coupling fields are necessarily sent and received in the same order, or on platforms for which MPI_Send is implemented with a mailbox (e.g. VPPs; in this case, make sure that the size of the mailbox is sufficient). Use the less efficient buffered send MPI_BSend on platforms for which MPI_Send is not implemented with a mailbox if the coupling fields are not sent and received in the same order. 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

If NOBSEND is not specified, the buffered send MPI_BSend will be used.

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. 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 are as for MPI2 except that MPI2 is replaced by MPI1 and there is no launching arguments[2].

- $NFIELDS : On the line below this keyword is the total number of fields exchanged and described in the second part of the *namcouple*.
- $JOBNAME : On the line below this keyword is a CHARACTER *3 or CHARACTER *4 variable giving an acronym for the given simulation.
- $NBMODEL : On the line below this keyword is the number of models running in the given experiment followed by CHARACTER *6 variables giving their names. 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 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 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 new keyword gives the type of calendar used. For now, 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.

from the MPI buffer in this routine. See the example in the toyatm model in prism/src/mod/toyatm/src

[2]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.

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 status may be the following (AUXILARY and EXPORTED are supported by all communication techniques, while the others are supported only by the PSMILe i.e. the CLIM/MPI1 or CLIM/MPI2 communication technique):

- AUXILARY : sent by the source model, received and used by OASIS3 main process for the transformation of other fields.
- EXPORTED : exchanged between component models and transformed by OASIS3 main process.
- 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.
- IGNORED : exchanged directly between the component models without being transformed by OASIS3 main process. The grid and partitioning of the source and target models have to be identical.
- IGNOUT : exchanged directly between the component models without being transformed by OASIS3 main process 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 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, put "1".)
  - 5 : number of transformations to be performed on this field.

  - sstoc.nc : name of the coupling restart file for the field (CHARACTER*8); it may be a binary of netCDF file (for more detail, see section 7.3).
  - sstat.nc : name of the field output file, 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 (CLIM/MPI1 or CLIM/MPI2 communication technique only, see section 4.8). Note that in mode NONE a LAG has to be defined so that the input file is opened initially.
  - SEQ=+1 : optional sequence index for the field (CLIM/MPI1 or CLIM/MPI2 communication technique only, 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 section 6.

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

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

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

entries are as for EXPORTED fields, except that there is no output file name on the first line.

For OUTPUT fields, there is no target model and therefore no target symbolic name; the source symbolic name must be repeated twice on the field first line. Also, there is no coupling restart file name (flda3.nc here), no LAG index and no SEQ index.

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 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 and PSMILe 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 all fields have to be present in the same restart file.

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 prism/util/running/toymodel/testinterp (see also section 8.3.1) and prism/util/running/toymodel/testNONE (see also section 8.3.2)

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

LOCTRANS can be chosen as first transformation if CLIM/MPI1-MPI2 communication and the PSMILe interface are used. LOCTRANS 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 transferred;
- AVERAGE : the field averaged over the previous coupling period is transferred;
- 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 obsolete in the current OASIS 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 obsolete in the current OASIS version and should be used anymore.

  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:

  ```
  # 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
```

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)[2]; $NIO is the flag that indicates if the weight-address-and-iteration-number dataset will be calculated and written by OASIS3 ($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.[3]

  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*.

  The generic input line is as follows:

```
# CHECKIN operation
    $NINT
```

  $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.

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

[3] 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.

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 interpolations, 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[4](1). SCRIPR routines are in prism/src/lib/ scrip . See the SCRIP 1.4 documentation in prism/src/mod/ oasis3/doc/SCRIPuser.pdf for more details on the interpolation algorithms. Linking with NetCDF library is mandatory when using SCRIPR interpolations.

  The following types of interpolations are available:

  – DISTWGT performs a distance weighted nearest-neighbour interpolation (N neighbours). All grid types are supported.

    Masked target grid points: no values are calculated for masked target grid points.

    Non-masked target grid points: if the N nearest neighbours of a non-masked target grid point are masked, no value is calculated for that target point (note that transformations MASK and EXTRAP can be used to avoid this situation); the value 1.0E+20 is assigned to that non-masked target grid point if prism/src/lib/scrip/src/scrip_mp.f or vector.F90 (for vector interpolation) are compiled with ll_weightot=.true. .

    The configuring line is:

[4] See the copyright statement in appendix C.2.

```
# 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). Note that VECTOR , which is fact leads to a scalar treatment of the field (as in the previous versions), is still supported.
* $REST is the search restriction type: LATLON or LATITUDE (see SCRIP 1.4 documentation). Note that for D or U grid, the restriction may influence sligthly the result near the borders of the restriction bins.
* $NBIN the number of restriction bins (see SCRIP 1.4 documentation).
* $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: no values are calculated for masked target grid points.

Non-masked target grid points: if the N nearest neighbours of a non-masked target grid point are masked, no value is calculated for that target point, except that if prism/src/lib/scrip/src/remap_gauswgt.f is compiled with ll_nnei=.true. , in which case the non-masked nearest neighbour is used. As for DISTWGT , the value 1.0E+20 is assigned to non-masked target grid points for which no value is calculated if prism/src/lib/scrip/src/scriprmp.f or vector.F90 (for vector interpolation) are compiled with ll_weightot=.true. .

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 inversely proportional to $exp(-1/2 \cdot d^2/\sigma^2)$ where d is the distance between the source and target grid points, and $\sigma^2 = \$VAR \cdot \overline{d}^2$ where $\overline{d}^2$ is the average distance between two source grid points (calculated automatically by OASIS3).

– BILINEAR performs bilinear interpolation.

– BICUBIC performs a bicubic interpolation.

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

Masked target grid points: no values are calculated for masked target grid points.

Non-masked target grid points: if the some of the source grid points normally used in the bilinear or bicubic interpolation are masked, another algorithm is applied; at least, the nearest non-masked source neighbour is used.

The configuring line is:

```
 # SCRIPR/BILINEA   R 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 target cell. Note that the SCRIP library supposes that the borders of the cells are linear in the longitude-latitude space between the corners defined by the users or calculated automatically by OASIS3 (see $CGRS). In this space the border of a cell has to be coincident with the border of its neighbour cell to give proper results.

  The target grid mask is never considered in CONSERV , except with normalisation option FRACNNEI (see below). To have a value calculated, a target grid cell must intersect at least one source cell. However, the NORMlisation option (that takes into account the source grid mask, see below) may result in a null value calculated for those target grid cells. In that case (i.e. at least one intersecting source cell, but a null value finally calculated because of the normalisation option), the value 1.0E+20 is assigned to those target grid points if prism/src/lib/scrip/src/scripprm.f or vector.F90 (for vector interpolation) are compiled with ll_weightot=.true ..

  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 1st-order remapping if the grid corners are given by the user in the grid data file which is, in this case, necessarily a netCDF file (grids.nc, see section 7.2); only LR is supported if the grid corners are not available in the grid data file 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 . Note that for CONSERV the restriction does not influence the result.
  - $NORM is the NORMlization option:
    - FRACAREA : The sum of the non-masked source cell intersected areas is used to NORMlise 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 NORMlise 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 no value will be calculated for a target cell not intersecting any source cells (masked or unmasked), even with FRACNNEI option.
  - $ORDER : FIRST or SECOND [5] for first or second order remapping respectively (see SCRIP 1.4 documentation).

Support of vector fields

[5] CONSERV/SECOND has not been tested in detail.

SCRIPR supports 2D vector interpolation. Please note however that this functionality is relatively new and has been tested and validated only in a reduced number of test cases. The two vector components have to be identified by replacing $CFTYP by VECTOR_I or VECTOR_J and have to be associated by replacing $ASSCMP, for each component field, by the source symbolic name of the associated vector component in (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 (see details in prism/util/running/testinterp/README_testinterp). The details of the vector treatment, performed by the routines scriprmp_vector.F90 and rotations.F90 in prism/src/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.
- If the user put 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.
- 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.

Known problems with SCRIPR

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*_*XXXXXXX*_*NORMAOPT*.nc, where *srcg* and *tgtg* are the acronyms of respetively the source and the target grids, *XXXXXXX* is the interpolation type (i.e. DISTWGT, GAUSWGT, BILINEA, BICUBIC, or CONSERV) and *NORMAOPT* is the normalization option (i.e. DESTAREA or FRACAREA 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.

- **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 /prism/src/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 /prism/src/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 /prism/src/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 masked source grid points are not 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 (LR, D or U) and $CFTYP is as for the DISTWGT
  * $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 /prism/src/mod/ 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 /prism/src/mod/ 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 analyses are available in the "cooking" part of OASIS3, controlled by cookart.f .

- **CONSERV**:

  CONSERV (routine /prism/src/mod /oasis3/src/conserv.f) performs global flux conservation. The flux is integrated on both source and target grids, without considering values of masked points, and the residual (target - source) is calculated. Then all flux values on the target grid are uniformly modified, according to their corresponding surface. This analysis requires one input line with one argument:

```
# CONSERV  operation
    $CMETH
```

version, only global flux conservation can be performed. Therefore $CMETH must be GLOBAL .

- **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=NONSO  LAR
       $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 /prism/src/mod/ 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 /prism/src/mod/oasis3/src/ postpro.f.

- **REVERSE**:

  This transformation is obsolete in the current OASIS version.

  REVERSE (routine /prism/src/mod/ 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 /prism/src/mod /oasis3 /src /chkfld .f) calculates the mean and extremum values of an output field and prints them to the coupler output *cplout*.

  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 /prism/src/mod/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 `prism/util/running/toyclim/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 (see section 4.2) in grid data files. These files can be all binary or all NetCDF. In `/prism/data/toyclim/input_toyclim_standard_standard_prism_2-2.tar.gz`, 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 `/prism/src/mod/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); furthermore, for Logically Rectangular LR source grids only, the grid corners will be automatically calculated if they

are not available in *grids.nc* (if needed, the corresponding reverse remapping can be done in which the current target grid become the source grid).

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. Note also that 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). In the grid file in `/prism/data/toy_clim/input_toyclim_standard_standard_prism_2-2.tar.gz`, the angles of the `torc` grid are given in array `torc.ang`. 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- or 1 -masked- 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/GAUSSIA_N`, 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/GAUSSIA_N`; it is not required otherwise.
4. *maskr* or *maskr.nc*: 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
...
```

- `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 (see section 4.8). If 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.

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 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[1].

In the NetCDF restart files, the field arrays must have the source symbolic name indicated in the *namcouple* (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) [2]

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 5.3.3). There must be one input file per INPUT field, containing a time sequence of the field in a single or double precision REAL array (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"

[1] 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.

[2] 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.

dimension has to be the unlimited dimension. For a practical example, see the file SOALBEDO.nc in /prism/data/toy_clim/input_toyclim_standard_standard_prism_2-2.tar.gz .

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_y    ear_01(jpi,  jpj)
       ...
       WRITE(NLU_fil)     field_january_ye    ar_01
       WRITE(NLU_fil)     field_february_y    ear_01
       WRITE(NLU_fil)     field_march_year    _01
       etc...
       WRITE(NLU_fil)     field_december_y    ear_01
C
C if  climatology,    one  stops  here
C
       WRITE(NLU_fil)     field_january_ye    ar_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 a 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 prism/src/mod/oasis3/doc/SCRIPusers.pdf.

Chapter 8

Compiling and running OASIS3

8.1 Compiling OASIS3 and TOYCLIM

Compiling OASIS3 and TOYCLIM (see section 8.2.1) can be done using the top makefile TopMakefileOasis3 and platform dependent header files as described in section 8.1.1. Note OASIS3 is temporarily released without the corresponding PRISM Standard Compile Environment and Running Environment (SCE/SRE); they will be included when the migration from CVS to Subversion will be realized in CERFACS.

During compilation, a new directory branch is created /prism/*arch*, where *arch* is the name of the compiling platform architecture (e.g. *Linux*). After successful compilation, resulting executables are found in /prism/*arch*/bin, libraries in /prism/*arch*/lib and object and module files in /prism/*arch*/build. The different pre-compiling flags used for OASIS3 and its associated PSMILe library are described in section 8.1.2.

8.1.1 Compilation with TopMakefileOasis3

Compiling OASIS3 and TOYCLIM using the top makefile TopMakefileOasis3 can be done in directory prism/src/mod/oasis3/util/make_dir. TopMakefileOasis3 must be completed with a header file make.*your_platform* specific to the compiling platform used and specified in prism/src/mod/oasis3/util/make_dir/make.inc. One of the files make.pgi_cerfacs, make.sx_frontend or make.aix can by used as a template. The root of the prism tree can be anywere and must be set in the variable PRISMHOME in the make.*your_platform* file. The choice of MPI1, MPI2 or NONE (interpolator-only mode, see section 6.1) is also done in the make.*your_platform* file (see $CHAN therein).

The following commands are available:

- make -f TopMakefileOasis3
  compiles OASIS3 libraries *clim*, *anaisg*, *anaism*, *fscint*, *scrip* and creates OASIS3 main executable oasis3.$CHAN.x (where $CHAN is MPI1, MPI2 or NONE);
- make -f TopMakefileOasis3 toyclim
  compiles OASIS3 libraries as above, compiles *mpp_io* and *psmile* librairies, and creates OASIS3 and TOYCLIM executables oasis3.MPI[1/2].x, toyatm.MPI[1/2].x, toyoce.MPI[1/2].x and toylan.MPI[1/2].x ;
- make clean -f TopMakefileOasis3:
  cleans OASIS3 and TOYCLIM compiled files, but not the libraries ;
- make realclean -f TopMakefileOasis3:
  cleans OASIS3 and TOYCLIM compiled files including libraries.

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

8.1.2 CPP keys

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

- To 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 SIPC, PIPE and GMEM communication techniques available in previous versions should still work but are not maintained anymore and were not tested.
- 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 .
- When linking OASIS3 and PSMILe with a netCDF library which is highly recommended[1] (the opposite case should work but was not fully tested):
  - 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
- For more information in log files **.prt** during the psmile library exchanges:
  - __VERBOSE
- For more debugging information to the standard output from the *mpp_io* library:
  - DEBUG
- The CPP key __DEBUG can also be activated for:
  - deadlock detection in *clim* and *psmile* librairies;
  - 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).
- To compile the PSMILe communication library without the I/O functionality (see section 5.3), i.e to compile only empty routines in prism/src/lib/mpp_io :
  - key_noIO
- For compiling without linking the SCRIP library:
  - key_noSCRIP
- Other platform dependent CPP keys, that should be automatically activated on the corresponding platforms, are defined and used in prism/src/lib/mpp_io/include

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

8.2 Running OASIS3 in coupled mode with TOYCLIM

In order to test the OASIS3 coupler in a light coupled configuration, CERFACS has written 3 "toy" component models, mimicking an atmosphere model (toyatm), an ocean model (toyoce), and a chemistry model (toyche). 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 TOYCLIM coupled model; the TOYCLIM 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 TOYCLIM example coupled model was successfully compiled and run on NEC SX6, IBM Power4, and Linux PC DELL, and previous versions were compiled and run on many other platforms.

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

8.2.1 TOYCLIM description

The toyoce model

The toyoce model, which sources can be found in `prism/src/mod/toyoce/src`, has a 2D logically-rectangular, streched and rotated grid of 182x152 points, which corresponds to a real ocean model grid (the pole of convergence is shifted over Asia). Toyoce timestep is 14400 seconds; it performs therefore 36 timesteps per 6-day run.

OASIS3 PRISM System Model Interface (PSMILe) routines are detailed in section 4. At the beginning of a run, toyoce performs appropriate PSMILe calls to initialize the coupling, define its grids, and declare its I/O or coupling fields. As toyoce is not parallel, it calls the PSMILe prism_def_partition routine to define only one Serial partition containing the 182X152 grid points.

Then, toyoce starts its timestep loop. At the beginning of its timestep, toyoce calls the PSMILe prism_get routine 7 times to request the fields named Field3 to Field9 on table 8.1. At the end of its timestep, toyoce 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). As toyoce contains no real physics or dynamics, it defines a simple feed back between Field1 and Field3 and between Field2 and Field7 such as:

$$Field1 = Field3 + 1$$

$$Field2 = Field7 + 1$$

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

The toyatm model

The toyatm model, which sources can be found in `prism/src/mod/toyatm/src`, has a realistic atmospheric T31 Gaussian grid (96x48 points). Its timestep is 3600 seconds; it therefore performs 144 timesteps per 6-day run.

As toyoce, toyatm 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 toyatm 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). Toyatm can run on 1 or 3 processes, depending on the variable `il_nbcplproc` (hardcoded to 3 by default). If the user modifies this variable and hardcodes `il_nbcplproc = 1`, toyatm runs on only one process and defines only one Serial

partition containing the 96X48 grid points. If `il_nbcplproc = 3`, toyatm runs on 3 processes and its decomposition depends on the `cdec` parameter, hardcoded to `APPLE`. In this case, each of the 3 toyatm processes calls the PSMILe prism_def_partition routine to define 1 segment of an APPLE decomposition (1536 grid points per segment). If the user changes the hardcoded value of `cdec` to `BOX`, each process will define 1 'box' of a BOX decomposition; the first two processes treat a box of 64X24 points, while the third process treats a box of 128X12 points. If the user hardcodes `cdec='ORANGE'`, each process will define a partition of two segments of 768 points distant of 1536 points.

Then, toyatm starts its timestep loop. At the beginning of its timestep, toyatm 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, toyatm 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 by the user (see section 5.3). As toyatm contains no real physics or dynamics, it defines a simple feed back between the different fields as:

$$Field5 = Field1 + 1$$

$$Field7 = Field2 + 1$$

$$Field10 = Field11 + 1$$

$$Field4 = Field1 + 1$$

$$Field8 = Field2 + 1$$

$$Field6 = Field1 + 1$$

$$Field9 = Field2 + 1$$

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

The toyche model

Toyche, which sources can be found in `prism/src/mod/toyche/src`, is integrated on the same atmospheric model grid than toyatm. Its timestep is 7200 seconds; it therefore performs 72 timesteps per 6-day run.

As the other toymodels, toyche 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 toyche has the same grid than toyatm, 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. Toyche decomposition is hardcoded the same way than toyatm, and if the user modifies the toyatm decomposition, he has to modify the toyche decomposition the same way by changing toyche values for `il_nbcplproc` and `cdec` (see above for toyatm).

At the beginning of its timestep, toyche calls the PSMILe prism_get routine to request Field10 (see table 8.1). At the end of its timestep, toyche calls PSMILe prism_put routine to send Field11. As toyche contains no real physics or dynamics, it defines a simple feed back between Field11 and Field10 such as:

$$Field11 = Field10 + 1$$

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

TOYCLIM coupling algorithm

The coupling algorithm between the TOYCLIM component models toyoce, toyatm, and toyche 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).

	toyoce	toyatm	toyche	restart	function
Field1	SOSSTSST (src)	SISUTESU (tgt)		fldo1.nc	F_1
Field2	SOICECOV (src)	SIICECOV (tgt)		fldo1.nc	F_1
Field3	SOALBEDO (tgt)			SOALBEDO.nc	
Field4	SONSHLDO (tgt)	CONSFTOT (src)		flda1.nc	F_2
Field5	*SOSHFLDO*	COSHFTOT (src)			F_2
Field6	SOWAFLDO (tgt)	COWATFLU (src)		flda1.nc	F_2
Field7	SORUNOFF (tgt)	CORUNOFF (src)		flda1.nc	F_2
Field8	SOZOTAUX (tgt)	COZOTAUX (src)		flda2.nc	F_3
Field9	SOMETAUY (tgt)	COMETAUY (src)		flda2.nc	F_3
Field10		COSENHFL (src)	SOSENHFL (tgt)	flda3.nc	F_1
Field11		COTHSHSU (tgt)	SOTHSHSU (src)	flda4.nc	F_1

Table 8.1: Coupling and I/O fields of the TOYCLIM 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). The function used to create the field in the initial restart file is also given.

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

$Field_1$ is sent from toyoce component to toyatm component at the coupling frequency dtF_1 defined by the user in the configuring file *namcouple*. As interpolation is needed between toyoce and toyatm 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 toyoce and toyatm 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 toyoce timestep preceeding the coupling period therefore matches the prism_get called in toyatm at the coupling period.

At the beginning of the run (i.e. at time = 0), the toyoce 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 toyatm component after interpolation. The different functions used to create the fields in the initial restart file are also indicated in table 8.1. They are defined as follows:

$$F_1 = 2 + cos[\pi * acos(cos(\theta)cos(\phi))]$$

$$F_2 = 2 + cos^2(\theta)cos(2\phi)$$

$$F_3 = 2 + sin^{16}(2\phi)cos(16\phi)$$

F_1 represents a global dipole reaching its maximum and minimum values at the equator, respectively at the date-line and at the Greenwich meridian. F_2 represents two dipoles reaching respectively their maximum and minimum values at the equator, respectively at the date-line and at 90^o W, and at the Greenwich meridian and at 90^o E; it is similar to a spherical harmonic with *l* = 2 and *m* = 2, where *l* is the sherical

harmonic order and *m* is the azimuthal wave number. F_3 represents a series of dipoles centered at 45^o N and 45^o S; it is similar to a spherical harmonic with *l* = 32 and *m* = 16 and is useful for testing interpolation of fields with relatively high spatial frequency and rapidly changing gradients.

The exchange of $Field_2$ from toyoce to toyatm and $Field_4$, $Field_6$, $Field_7$, $Field_8$ and $Field_9$ from toyatm to toyoce 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 toyoce 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 toyatm 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. The prism_get calls in the toyoce model will not be activated at all.

$Field_{10}$ and $Field_{11}$ are exchanged respectively from toyatm to toyche and from toyche to toyatm following $Field_1$ logic described here above, except that the exchanges take place directly between the component models without going to OASIS3 interpolation process, as toyatm and toyche have the same grid and the same parallel partition. 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 toyoce prism_get called to receive those fields will automatically read the fields in their corresponding coupling restart files flda3.nc and flda4.nc.

8.2.2 Running TOYCLIM using the script `run_toyclim`

Data for running TOYCLIM are contained in tar file `input_toyclim_standard_standard_prism_2-2.tar.gz` or `input_toyclim_standard_standard_prism_2-2_single.tar.gz` in directory `prism/data/toyclim`. Input files and script are located in directory `prism/util/running/toyclim` .

To run TOYCLLIM, one has to compile OASIS3 and the 3 TOYCLIM component models (see section 8.1), to adapt the "User's section" of the running script `prism/util/running/toyclim/script/run_toyclim` for his/her platform, and to launch it. The script `run_toyclim` was tested on Linux PC, NEC SX-6, and IBM Power4.

To run the single precision case, OASIS3 and the 3 component models have to be compiled in single precision (see section 8.1.2), the configuration file `prism/util/running/toyclim/input/namcouple_single` has to be renamed `namcouple` (in the same directory, so to be used automatically), and `comp_precision=single` has to be specified in `run_toyclim` . The configuration file `namcouple_single` has to be used because the original `namcouple` specifies some `MOZAIC` transformations using binary auxiliary files that were not converted; the results obtained with `namcouple_single` , into which the `MOZAIC` transformations are replaced by `SCRIPR/DISTWGT` interpolations, will be slightly different.

8.3 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 8.3.1) and the "testNONE" test-case (see section 8.3.2).

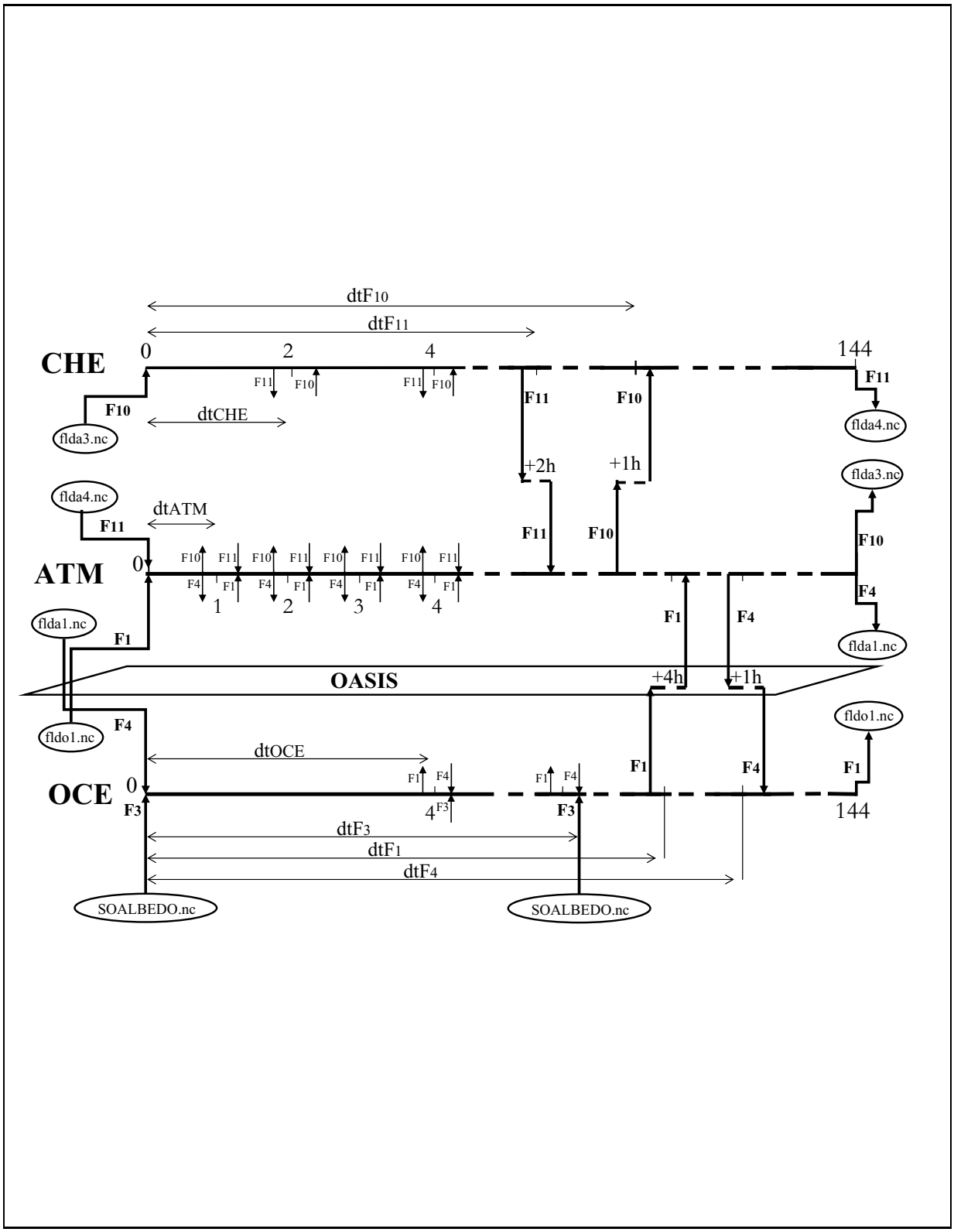


Figure 8.1: Exchange algorithm between the 3 TOYCLIM component models for fields $Field_1$, $Field_3$, $Field_4$, $Field_{10}$, and $Field_{11}$.

8.3.1 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 `prism/data/testinterp/input` and `prism/data/testinterp/restart`.

To run "testinterp", OASIS3 first has to be compiled in interpolator-only mode NONE (see section 8.1). Then the programs that will calculate the interpolation error, i.e. `gen_error.f90` and `gen_error_vector.f90` (for vector fields) in directory `prism/util/running/testinterp/error` have to be compiled (see script `sc_comp_error`).

Then, one has to adapt and execute the running script `prism/util/running/testinterp/script/sc_run_testinterp`. With `TIME=ONE`, the configuration file `prism/data/testinterp/input/namcouple_ONE`, the input files `flda1.nc`, `flda2.nc`, `flda3.nc`, `fldb1.nc`, `fldo1.nc` and `fldz1.nc` from `prism/data/testinterp/restart` and the input files `aaIin.nc` and `caJin.nc` from `prism/data/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 `prism/data/testinterp/input/namcouple_MANY` and the input file `fldin.nc` in `prism/data/testinterp/restart` is then used.

The results obtained after running the testinterp test-case should match the ones in `prism/data/testinterp/outdata`.

8.3.2 The "testNONE" test-case

All files to run the "testNONE" test-case can be found in `prism/util/running/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", the user has to adapt the "User specifications" part of the running script `sc_run_NONE`. In particular, he has to specify the directory for the grid data files, the directory for the `namcouple` and `cf_name_table.txt` files, the source and target grid prefixes in the `namcouple` and in the grid data files (see section 5.3.1), an analytic function, and 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`

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_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/scripmp.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/scripmp.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/scrip_rmp_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/toy_climio/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/toy_climio/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/cornerrs.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.2 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/scripr/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/scripr/src/remap_conserv.f (thanks to Veronika Gayler from M&D).
  - prism/src/mod/oasis3/src/inicmc.F
  - prism/src/lib/scripr/src/remap_distwgt.f

B.3 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/scripr/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.4 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/psm
  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.5 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

Copyright statements

C.1 OASIS3 copyright statement

C.2 The SCRIP 1.4 copyright statement

The SCRIP 1.4 copyright statement reads as follows:

Appendix D

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:

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 D.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 D.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.

[14] L. Terray, E. Sevault, E. Guilyardi and O. Thual OASIS 2.0 Ocean Atmosphere Seai 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.