


```

                                dda_src_epio_lat,      &
                                dda_src_epio_lon,      &
                                dda_src_epio_z,        &
                                id_src_mask,          &
                                ida_src_epio_mask,     &
                                id_err                )
                                id_err                )
INTEGER, INTENT (In)           :: id_src_epio_size
REAL, DIMENSION(id_src_epio_size), INTENT(In)      :: rda_src_epio_lat
REAL, DIMENSION(id_src_epio_size), INTENT(In)      :: rda_src_epio_lon
REAL, DIMENSION(id_src_epio_size), INTENT(In)      :: rda_src_epio_z
INTEGER, INTENT (In)           :: id_src_mask
INTEGER, DIMENSION(id_src_epio_size), INTENT (In) :: ida_src_epio_mask
INTEGER, INTENT (Out)          :: id_epio_id
INTEGER, INTENT (Out)          :: id_err
END SUBROUTINE PSMILe_Trset_src_epio3d_real

```

These two routines transfer the source epio information and arrays to the transformer. The routines give as output the epio id that will be used in the following routines.

The source epio can be sent as real or double.

One source epio and one target epio can have different types.

The intent in id_src_mask indicates if the mask is transferred (=1) or not (=0) to the transformer.

B. PSMILe Trs set tgt epio3d *

```

SUBROUTINE PSMILe_Trset_tgt_epio3d_dble(id_epio_id,      &
                                id_tgt_epio_size,      &
                                dda_tgt_epio_lat,      &
                                dda_tgt_epio_lon,      &
                                dda_tgt_epio_z,        &
                                id_tgt_mask,          &
                                ida_tgt_epio_mask,     &
                                id_err                )
INTEGER, INTENT (In)           :: id_epio_id
INTEGER, INTENT (In)           :: id_tgt_epio_size
DOUBLE PRECISION, DIMENSION(id_tgt_epio_size), INTENT(In):: dda_tgt_epio_lat
DOUBLE PRECISION, DIMENSION(id_tgt_epio_size), INTENT(In):: dda_tgt_epio_lon
DOUBLE PRECISION, DIMENSION(id_tgt_epio_size), INTENT(In):: dda_tgt_epio_z
INTEGER, INTENT (In)           :: id_tgt_mask
INTEGER, DIMENSION(id_tgt_epio_size), INTENT (In) :: ida_tgt_epio_mask
INTEGER, INTENT (Out)          :: id_err
END SUBROUTINE PSMILe_Trset_tgt_epio3d_dble

```

```

SUBROUTINE PSMILe_Trset_tgt_epio3d_real(id_epio_id,      &
                                id_tgt_epio_size,      &
                                rda_tgt_epio_lat,      &
                                rda_tgt_epio_lon,      &
                                rda_tgt_epio_z,        &
                                id_tgt_mask,          &
                                ida_tgt_epio_mask,     &
                                id_err                )
INTEGER, INTENT (In)           :: id_epio_id
INTEGER, INTENT (In)           :: id_tgt_epio_size
REAL, DIMENSION(id_tgt_epio_size), INTENT(In)      :: rda_tgt_epio_lat
REAL, DIMENSION(id_tgt_epio_size), INTENT(In)      :: rda_tgt_epio_lon

```



```

        id_epio_field_size,      &
        ida_field,              &
        id_err)
    INTEGER, INTENT (In)       :: id_transient_out_id
    INTEGER, INTENT (In)       :: id_epio_id
    INTEGER, INTENT (In)       :: id_epio_field_size
    INTEGER, DIMENSION(id_epio_field_size), INTENT (In) :: ida_field
    INTEGER, INTENT (Out)      :: id_err
END SUBROUTINE PSMILe_Trns_put_int

SUBROUTINE PSMILe_Trns_put_real(id_transient_out_id,      &
        id_epio_id,              &
        id_epio_field_size,      &
        rda_field,              &
        id_err)
    INTEGER, INTENT (In)       :: id_transient_out_id
    INTEGER, INTENT (In)       :: id_epio_id
    INTEGER, INTENT (In)       :: id_epio_field_size
    REAL, DIMENSION(id_epio_field_size), INTENT (In) :: rda_field
    INTEGER, INTENT (Out)      :: id_err
END SUBROUTINE PSMILe_Trns_put_real

SUBROUTINE PSMILe_Trns_put_dble(id_transient_out_id, &
        id_epio_id,              &
        id_epio_field_size, &
        dda_field,              &
        id_err)
    INTEGER, INTENT (In)       :: id_transient_out_id
    INTEGER, INTENT (In)       :: id_epio_id
    INTEGER, INTENT (In)       :: id_epio_field_size
    DOUBLE PRECISION, DIMENSION(id_epio_field_size), INTENT (In) :: dda_field
    INTEGER, INTENT (Out)      :: id_err
END SUBROUTINE PSMILe_Trns_put_dble

SUBROUTINE PSMILe_Trns_get_int(id_transient_in_id,      &
        id_epio_id,              &
        id_epio_field_size,      &
        ida_field,              &
        id_err)
    INTEGER, INTENT (In)       :: id_transient_in_id
    INTEGER, INTENT (In)       :: id_epio_id
    INTEGER, INTENT (In)       :: id_epio_field_size
    INTEGER, DIMENSION(id_epio_field_size), INTENT (In) :: ida_field
    INTEGER, INTENT (Out)      :: id_err
END SUBROUTINE PSMILe_Trns_get_int

SUBROUTINE PSMILe_Trns_get_real(id_transient_in_id,      &
        id_epio_id,              &
        id_epio_field_size,      &
        rda_field,              &
        id_err)
    INTEGER, INTENT (In)       :: id_transient_in_id
    INTEGER, INTENT (In)       :: id_epio_id
    INTEGER, INTENT (In)       :: id_epio_field_size
    REAL, DIMENSION(id_epio_field_size), INTENT (In) :: rda_field
    INTEGER, INTENT (Out)      :: id_err
END SUBROUTINE PSMILe_Trns_get_real

```

```

SUBROUTINE PSMILe_Trns_get_dble(id_transient_in_id,      &
                               id_epio_id,             &
                               id_epio_field_size,     &
                               dda_field,             &
                               id_err)
  INTEGER, INTENT (In)          :: id_transient_in_id
  INTEGER, INTENT (In)          :: id_epio_id
  INTEGER, INTENT (In)          :: id_epio_field_size
  DOUBLE PRECISION, DIMENSION(id_epio_field_size), INTENT (In) :: dda_field
  INTEGER, INTENT (Out)         :: id_err
END SUBROUTINE PSMILe_Trns_get_dble

```

The psmile_trs_put and psmile_trs_get allow the PSMILe to send the epios part of a transient out or receive the epio part of a transient in to/from the transformer.

The type of the transient in can differ from the type of the transient out. The transient in type has to be read in the SMIOC by the transformer (in order for the transformer to store the transient in in the right format even if the corresponding psmile_trs_get has not been posted). Therefore, it has to be precise in the temporary input file scc_api (see below).

II. Examples of an input scc_api file

Some new examples have been implemented to illustrate the use of the new psmile_trs routines.

The psmile developer has to keep in mind that any put or get could not be done as long as there are psmile_trs declarations to do to the transformer. This action would be insured through the PRISM_Enddef in the components, but in the example, it is artificially done (sleep).

The psmile developer should also consider that the epio_id defined in the source component process has to be given to the target component process to be used in the psmile_trs_get routines on the target side. This task could be insured under the exchanges of envelops in the PSMILE.

The scc_api has also changed because of the new approach used in the inetarfce. The following scc_api (New_ex3) illustrates this purpose.

```
1 ! execution mode (1 for MPI1 and 2 for MPI2)
```

```
2 ! number of applications
```

```
! appli_name exe_name nb_host nb_comp nb_arg
driver      ../../bin/prismdrv_main  1 1 1
atm         ./atmosphere             1 1 1
oce         ./ocean                   1 1 1
```

```
! for each appli:
```

```
! arg
```

```
! host and nb_pes
```

```
! comp_name and nb of rank sets
```

```
test1
```

```
aral      1
```

```
atm       1
```

```
test2
```

```
aral      1
```

```
oce       1
```

```
! rank sets for all comp ie min1 max1 increment1 min2 max2 increment2
```

```
(1:1:1)
```

```
(1:1:1)
```

```
2 ! nb of grids in the coupling system
```

```
1 ! comp_id
```

```
2801 ! type logically rectangular
```

```
128 ! x size
```

```
64 ! y size
```

```
1 ! z size
```

```
2810 ! poles on this grid
```

```
2813 ! no overlap
```

```
2 ! comp_id
```

```
2801 ! type logically rectangular
```

```
182 ! x size
```

```
149 ! y size
```

```

1 ! z size
2810 ! poles on this grid
2813 ! no overlap

=====
3 ! nb of communications (transient_out-transient_in)

1 ! transient out id
1 ! source component
1 ! transient in id
2 ! target component
3003 ! transient in type (int, real or double)
2015 ! PRISM_3d for the interpolation type
2003 ! PRISM_Bilinear for the interpolation method
4 ! For the number of neighbors
0 ! ...

2 ! transient out id
1 ! source component
2 ! transient in id
1 ! target component
3002 ! transient in type (int, real or double)
2015 ! PRISM_3d for the interpolation type
2001 ! PRISM_Bilinear for the interpolation method
4 ! For the number of neighbors
0 ! ...

3 ! transient out id
2 ! source component
3 ! transient in id
1 ! target component
3001 ! transient in type (int, real or double)
2015 ! PRISM_3d for the interpolation type
2001 ! PRISM_Bilinear for the interpolation method
4 ! For the number of neighbors
0 ! ...

```

There are no modifications in the first part of the file. What we are interested in is the part previously called 'field' that has been re-called 'communications'. A communication represents here the description of an exchange of data between a source and a target component.

We thus have to give:

- the transient out id (global id),
- the source component,
- the transient in id (global id)
- the target component,
- the type for the transient in (3001=int, 3002=real, 3003=double). This will change as soon as I'll merge the include files,
- The interpolation type (2015=3d interpolation),
- The interpolation method (2001 for the nearest neighbor method, 2003 for the bilinear method...)
- The number of neighbors,
- Anything... not really important for the moment.

Once again, this file is here to get the information we should collect from the xml extraction tools. This is not a final element of the coupler, so please, even if some parameters are useless, forget them...


```

! $Id: prismdrv_trs_loop.F90,v 1.8 2003/12/29 13:29:14 deplat Exp $
! $Author: deplat $
!-----
!
! Local declarations
!
CHARACTER(LEN=len_cvs_string), SAVE :: mycvs = &
  '$Id: prismdrv_trs_loop.F90,v 1.8 2003/12/29 13:29:14 deplat Exp $'

!
LOGICAL                :: ll_loop
!
INTEGER, PARAMETER     :: nerrp=3
INTEGER                :: ierrp(nerrp)
!
INTEGER, DIMENSION(PSMILe_trans_Header_length) :: ila_loop
!
INTEGER :: il_status(MPI_STATUS_SIZE)

!-----
!
PRINT *, '| Enter PRISMTrs_Loop'
CALL PSMILe_Flushstd

!
! 1. In the loop
!
ila_loop = 280177
ll_loop = .true.
DO WHILE (ll_loop)

  PRINT *, '| |'
  PRINT *, '| | Trs ready to receive '
  CALL PSMILe_Flushstd

! 1.1. Perform the receptions in the loop
CALL MPI_Recv (ila_loop, PSMILe_trans_Header_length, MPI_Integer, &
  MPI_ANY_SOURCE, MPI_ANY_TAG, comm_drv_trans, &
  il_status, id_err)

IF ( id_err /= MPI_SUCCESS ) THEN
  ierrp (1) = id_err
  ierrp (2) = PRISM_root
  ierrp (3) = PSMILe_Init_tag
  id_err = PRISM_Error_Recv
  CALL PSMILe_Error ( id_err, 'MPI_Recv', &
    ierrp, 3, __FILE__, __LINE__ )
  RETURN
ENDIF

! 1.2. Treat the message
PRINT *, '| | Trs receives : ',ila_loop(1), &
  ' from global rank ', ila_loop(2)
CALL PSMILe_Flushstd

SELECT CASE (ila_loop(1))

! 1.2.3. Set the source epio information

```

CASE (PSMILe_trans_Set_src_epio_info)

```
IF (ila_loop(10).eq. PSMILe_3D) THEN
  PRINT *, &
  '| Trs updates 3d src epio '
  CALL PSMILe_Flushstd
ELSE IF (ila_loop(10).eq. PSMILe_2D1D) THEN
  PRINT *, &
  '| Trs updates 2d1d src epio ', &
  '. Size latlon ', ila_loop(8), ' and size z ', ila_loop(9)
  CALL PSMILe_Flushstd
ELSE
  PRINT *, &
  '| Trs updates 1d1d1d src epio ', &
  '. Size ', ila_loop(8), ' x ', ila_loop(9), ' x ', ila_loop(10)
  CALL PSMILe_Flushstd
END IF
```

```
IF (ila_loop(3).eq. PRISM_Double_Precision) THEN
  CALL PRISMTrs_Set_src_epio_dble(ila_loop(2), &
    ila_loop(4), &
    ila_loop(5), &
    ila_loop(8), &
    ila_loop(9), &
    ila_loop(10), &
    ila_loop(11), &
    id_err)
```

```
ELSE IF (ila_loop(3).eq. PRISM_Real) THEN
  CALL PRISMTrs_Set_src_epio_real(ila_loop(2), &
    ila_loop(4), &
    ila_loop(5), &
    ila_loop(8), &
    ila_loop(9), &
    ila_loop(10), &
    ila_loop(11), &
    id_err)
END IF
```

! 1.2.3. Set the source epio information

CASE (PSMILe_trans_Set_tgt_epio_info)

```
IF (ila_loop(9).eq. PSMILe_3D) THEN
  PRINT *, &
  '| Trs updates 3d tgt epio '
  CALL PSMILe_Flushstd
ELSE IF (ila_loop(9).eq. PSMILe_2D1D) THEN
  PRINT *, &
  '| Trs updates 2d1d tgt epio ', &
  '. Size latlon ', ila_loop(7), ' and size z ', ila_loop(8)
  CALL PSMILe_Flushstd
ELSE
  PRINT *, &
  '| Trs updates 1d1d1d tgt epio ', &
  '. Size ', ila_loop(7), ' x ', ila_loop(8), ' x ', ila_loop(9)
  CALL PSMILe_Flushstd
END IF
```

```

IF (ila_loop(3) .eq. PRISM_Double_Precision) THEN
  CALL PRISMTrs_Set_tgt_epio_dble(ila_loop(2),    &
    ila_loop(6),    &
    ila_loop(4),    &
    ila_loop(5),    &
    ila_loop(7),    &
    ila_loop(8),    &
    ila_loop(9),    &
    ila_loop(10),   &
    id_err)

```

```

ELSE IF (ila_loop(3) .eq. PRISM_Real) THEN
  CALL PRISMTrs_Set_tgt_epio_real(ila_loop(2),   &
    ila_loop(6),   &
    ila_loop(4),   &
    ila_loop(5),   &
    ila_loop(7),   &
    ila_loop(8),   &
    ila_loop(9),   &
    ila_loop(10),  &
    id_err)

```

```

END IF

```

! 1.2.3. Set the epio_id in the exchange structure

```

CASE (PSMILe_trans_Set_triple_links)

```

```

  CALL PRISMTrs_Set_triple_links(ila_loop(3),   &
    ila_loop(4),   &
    ila_loop(5),   &
    id_err)

```

! 1.2.4. Set the neighbors information

```

CASE (PSMILe_trans_Set_neighbors_info)

```

```

  PRINT *, &
    '| | Trs updates its neighbors info for epio ', ila_loop(3)
  CALL PSMILe_Flushstd

```

```

  CALL PRISMTrs_Set_neighbors(ila_loop(2),     &
    ila_loop(3),     &
    ila_loop(4),     &
    ila_loop(5),     &
    id_err)

```

! 1.2.5. Receive a field

```

CASE (PSMILe_trans_Put)

```

```

  PRINT *, '| | Trs receives the field ', &
    ila_loop(3),' from epio ', ila_loop(4)
  CALL PSMILe_Flushstd

```

```

IF (ila_loop(5) .eq. PRISM_Integer) THEN
  CALL PRISMTrs_Mind_int(ila_loop(2),   &
    ila_loop(3),   &
    ila_loop(4),   &
    id_err)

```

```

ELSE IF (ila_loop(5) .eq. PRISM_Real) THEN
  CALL PRISMTrs_Mind_real(ila_loop(2),   &

```

```

        ila_loop(3), &
        ila_loop(4), &
        id_err)
ELSE IF (ila_loop(5).eq. PRISM_Double_Precision) THEN
    CALL PRISMTrs_Mind_dble(ila_loop(2), &
        ila_loop(3), &
        ila_loop(4), &
        id_err)
END IF

```

! 1.2.6. Send a part of a field to the target component
CASE (PSMILe_trans_Get)

```

PRINT *, '| Trs is asked to send the field ', &
    ila_loop(3),' from epio ', ila_loop(4)
CALL PSMILe_Flushstd

IF (ila_loop(5).eq. PRISM_Integer) THEN
    CALL PRISMTrs_Target_int(ila_loop(2), &
        ila_loop(3), &
        ila_loop(4), &
        id_err)
ELSE IF (ila_loop(5).eq. PRISM_Real) THEN
    CALL PRISMTrs_Target_real(ila_loop(2), &
        ila_loop(3), &
        ila_loop(4), &
        id_err)
ELSE IF (ila_loop(5).eq. PRISM_Double_Precision) THEN
    CALL PRISMTrs_Target_dble(ila_loop(2), &
        ila_loop(3), &
        ila_loop(4), &
        id_err)
END IF

```

! 1.2.7. Finalize the transformer context
CASE (PSMILe_trans_Finalize)

```

PRINT *, '| Trs is asked to finalize by process ', &
    ila_loop(2)
ig_stop = ig_stop + 1
PRINT *, '| ', ig_stop, ' pes on ', ig_nb_tot_pes, &
    ' have already asked to finalize'
CALL PSMILe_Flushstd

IF (ig_stop.eq. ig_nb_tot_pes) THEN
    ll_loop = .false.
END IF

```

CASE DEFAULT

```

PRINT *, &
    '| Trs is not intelligent enough to understand'
CALL PSMILe_Flushstd

```

END SELECT

END DO

```
!  
PRINT *, '| Quit PRISMTrs_Loop'  
PRINT *, '|'  
CALL PSMILe_Flushstd  
  
END SUBROUTINE PRISMTrs_Loop
```

IV. A short example of interactions

A short example of the interactions between the transformer and two mono-process components that exchange a field defined on different source and target grids is given in annex 2.

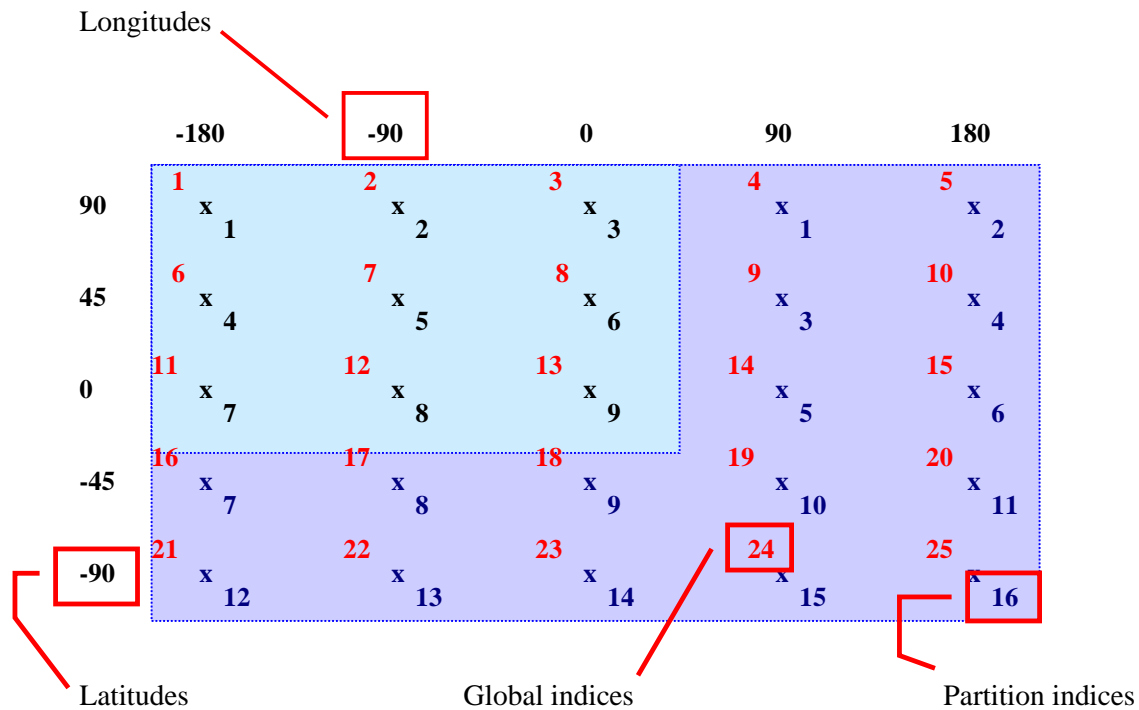
Annex 1: Example of EPIOs

Application 1

The first application is distributed on two processes.

The sizes of the partitions are 9 for process 0 and 16 for process 1.

The following scheme gives the latitudes and longitudes coordinates of the different points:

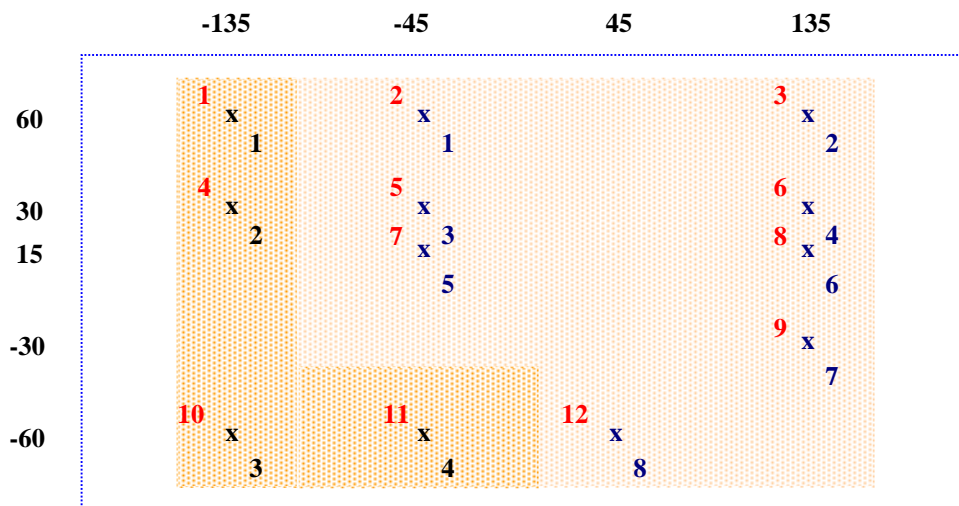


Application 2

The first application is distributed on two processes.

The sizes of the partitions are 4 for process 0 and 8 for process 1.

The following scheme gives the latitudes and longitudes coordinates of the different points:



Description of the elements involved in the interpolation

The intersection of the partition 0 of model 1 and partition 0 of model 2 gives 2 points. These points are point 1 and 2 in the local index space of partition 0 of model 2. Thus:

$$\mathbf{EPIOT(0,0) = (1, 2)}$$

The points of model 1 that will be used in the interpolation operation for this two points of model 2 are point 1, 2, 4, 5, 7, 8 in the local index space of partition 0 of model 1. Thus:

$$\mathbf{EPIOS(0,0) = (1, 2, 4, 5, 7, 8)}$$

$$\text{EPIOT}(0,0) = (1, 2)$$

$$\text{EPIOS}(0,0) = (1, 2, 4, 5, 7, 8)$$

$$\text{EPIOT}(0,1) = (1, 3, 5)$$

$$\text{EPIOS}(0,1) = (2, 3, 5, 6, 8, 9)$$

$$\text{EPIOT}(1,0) = (3, 4)$$

$$\text{EPIOS}(1,0) = (7, 8, 9, 12, 13, 14)$$

$$\text{EPIOT}(1,1) = (2, 4, 6, 7, 8)$$

$$\text{EPIOS}(1,1) = (1, 2, 3, 4, 5, 6, 9)$$

The four neighbors of point 1 (that belongs to EPIOT(0,0)) for the partition 0 of model 2 are 1, 2, 3, 4 in the local index space of the corresponding EPIOS(0,0).

The four neighbors of point 2 (that belongs to EPIOT(0,0)) for the partition 0 of model 2 are 3, 4, 5, 6 in the local index space of the corresponding EPIOS(0,0).

Thus:

$$\mathbf{Neighbors_indices(0,0) = (/1, 2, 3, 4/ ; / 3, 4, 5, 6/)}$$

$$\text{Neighbors_indices}(0,0) = (/1, 2, 3, 4/ ; / 3, 4, 5, 6/)$$

$$\text{Neighbors_indices}(0,1) = (/1, 2, 3, 4/ ; / 3, 4, 5, 6/ ; / 3, 4, 5, 6/)$$

$$\text{Neighbors_indices}(1,0) = (/1, 2, 3, 4/ ; / 3, 4, 5, 6/ ; / 3, 4, 5, 6/ ; / 5, 6, 8, 9/ ; / 7, 8, 10, 11/)$$

$$\text{Neighbors_indices}(1,1) = (/1, 2, 4, 5/ ; / 2, 3, 5, 6/)$$


```

!  

! Netcdf variables  

!  

CHARACTER(LEN=128) :: cla_file_name  

INTEGER           :: il_grid1_nc_file_id  

INTEGER           :: il_grid1_size  

INTEGER           :: il_grid1_height  

INTEGER           :: il_nc_gridsize_id  

INTEGER           :: il_grid2_nc_file_id  

INTEGER           :: il_grid2_size  

INTEGER           :: il_grid2_height  

!  

! grid arrays that corresponds to partitions  

!  

INTEGER, DIMENSION(2) :: ila_grid1_dim  

INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid1_mask  

INTEGER, DIMENSION(:,:), ALLOCATABLE :: ila_neighbors  

INTEGER, DIMENSION(:,:), ALLOCATABLE :: ila_neighbors2  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_lat  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_lon  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_z  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_corner_lat  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_corner_lon  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_field  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid1_lat  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid1_lon  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid1_z  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid1_field  

INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid1_field  

!  

INTEGER, DIMENSION(2) :: ila_grid2_dim  

INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid2_mask  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_lat  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_lon  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_z  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_corner_lat  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_corner_lon  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_field  

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_field_int  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_lat  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_lon  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_z  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_field  

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_field_int  

INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid2_field  

INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid2_field_int  

!  

INTEGER :: il_nb_neighbors  

!  

INTEGER :: il_epio_id, il_epio_id2, il_epio_id3  

!
```

```

!=====
!  

!1. Initialization  

!  

model_name = 'atm'  
  

call PRISM_Init_comp (comp_id, model_name, ierror )  

if (ierror /= 0) n_errors = n_errors + 1  
  

!  

!1.1. Get local communicator  

!  

call PRISM_Get_localcomm (comp_id, localComm, ierror )  

if (ierror /= 0) n_errors = n_errors + 1  
  

call MPI_Comm_Size ( localComm, npes, ierror )  

call MPI_Comm_Rank ( localComm, mype, ierror )  
  

write ( *, * ) 'I am the ', trim(model_name), ' comp ', comp_id, &  

' local rank ', mype  

if (mype == 0) call prism_version  
  

!  

!1.2. Initialize the model  

!  

call PRISM_Initialized (flag, ierror)  

if (ierror /= 0) n_errors = n_errors + 1  

if (.not. flag) then  

write (*,*) model_name, "PRISM is NOT initialized"  

n_errors = n_errors + 1  

endif  

!  

!=====
!  

!2. netcdf extraction for the source grid  

!  

!2.1. Init file for the netcdf context  

!  

cla_file_name = '../NC_TOOLS/Grids/Grid_AT42REGU_Msk.nc'  

call dd_init_file(cla_file_name, &  

il_grid1_nc_file_id, &  

ierror)  

WRITE(12,*) 'dd : ../NC_TOOLS/Grids/Grid_AT42REGU_Msk.nc', &  

il_grid1_nc_file_id  

CALL flush(12)  
  

!  

!2.2. get the size of the arrays  

!  

CALL dd_extract_grid_size(il_grid1_nc_file_id, &  

il_grid1_size, &  

ierror)  

WRITE(12,*) 'dd : dd_extract_grid_size ', il_grid1_size  

CALL flush(12)  
  

!  

!2.3. Allocate the source arrays  

!  


```

```

ALLOCATE(dla_grid1_lat(il_grid1_size), stat = ierror)
ALLOCATE(dla_grid1_lon(il_grid1_size), stat = ierror)
ALLOCATE(dla_grid1_z(il_grid1_size), stat = ierror)
ALLOCATE(dla_grid1_corner_lat(il_grid1_size*4), stat = ierror)
ALLOCATE(dla_grid1_corner_lon(il_grid1_size*4), stat = ierror)
ALLOCATE(ila_grid1_mask(il_grid1_size), stat = ierror)
ALLOCATE(dla_grid1_field(il_grid1_size), stat = ierror)

ALLOCATE(rla_grid1_lat(il_grid1_size), stat = ierror)
ALLOCATE(rla_grid1_lon(il_grid1_size), stat = ierror)
ALLOCATE(rla_grid1_z(il_grid1_size), stat = ierror)
ALLOCATE(rla_grid1_field(il_grid1_size), stat = ierror)

!
! 2.4. extract the source grid arrays (lat, lon, corners, mask)
!
CALL dd_extract_grid_array(il_grid1_nc_file_id, &
    il_grid1_size, &
    ila_grid1_dim, &
    dla_grid1_lon, &
    dla_grid1_lat, &
    dla_grid1_corner_lon, &
    dla_grid1_corner_lat, &
    ila_grid1_mask, &
    ierror)
WRITE(12,*) 'dd : dd_extract_grid_array with x = ', &
    ila_grid1_dim(1), ' and y = ', ila_grid1_dim(2)
CALL flush(12)

DO ib = 1, il_grid1_size
  IF (ila_grid1_mask(ib) .eq. 0) THEN
    ila_grid1_mask(ib) = 1
  ELSE IF (ila_grid1_mask(ib) .eq. 1) THEN
    ila_grid1_mask(ib) = 0
  END IF
END DO

dla_grid1_z(:) = 1

rla_grid1_lon = REAL(dla_grid1_lon)
rla_grid1_lat = REAL(dla_grid1_lat)
rla_grid1_z = REAL(dla_grid1_z)

!
! 2.5. Close the netcdf context
!
CALL dd_close_file(il_grid1_nc_file_id, &
    ierror)
WRITE(12,*) 'dd : apres dd_close_file'
CALL flush(12)

!
!=====
!
! 3. netcdf extraction for the target grid
!
! rem: this part would be in the real case replaced by the exchange of
! envelops

```

```

!  

! 3.1. Init file  

!  

  cla_file_name = '././NC_TOOLS/Grids/Grid_ORCA.nc'  

  call dd_init_file(cla_file_name,      &  

                   il_grid2_nc_file_id, &  

                   ierror)  

  WRITE(12,*) 'dd : ././NC_TOOLS/Grids/Grid_ORCA.nc', &  

             il_grid1_nc_file_id  

  CALL flush(12)  

!  

! 3.2. get the size of the arrays  

!  

  CALL dd_extract_grid_size(il_grid2_nc_file_id,  &  

                            il_grid2_size,      &  

                            ierror)  

  WRITE(12,*) 'dd : dd_extract_grid_size ', il_grid2_size  

  CALL flush(12)  

!  

! 3.3. Allocate the target arrays  

!  

  il_nb_neighbors = 4  

  ALLOCATE(dla_grid2_lat(il_grid2_size), stat = ierror)  

  ALLOCATE(dla_grid2_lon(il_grid2_size), stat = ierror)  

  ALLOCATE(dla_grid2_z(il_grid2_size), stat = ierror)  

  ALLOCATE(dla_grid2_corner_lat(il_grid2_size*4), stat = ierror)  

  ALLOCATE(dla_grid2_corner_lon(il_grid2_size*4), stat = ierror)  

  ALLOCATE(ila_grid2_mask(il_grid2_size), stat = ierror)  

  ALLOCATE(dla_grid2_field(il_grid2_size), stat = ierror)  

  ALLOCATE(dla_grid2_field_int(il_grid2_size), stat = ierror)  

  ALLOCATE(rla_grid2_lat(il_grid2_size), stat = ierror)  

  ALLOCATE(rla_grid2_lon(il_grid2_size), stat = ierror)  

  ALLOCATE(rla_grid2_z(il_grid2_size), stat = ierror)  

  ALLOCATE(rla_grid2_field_int(il_grid2_size), stat = ierror)  

  ALLOCATE(ila_grid2_field_int(il_grid2_size), stat = ierror)  

  ALLOCATE(ila_neighbors(il_grid2_size,il_nb_neighbors), stat = ierror)  

  ALLOCATE(ila_neighbors2(il_grid2_size,il_nb_neighbors), stat = ierror)  

!  

! 3.4. extract the target grid arrays  

!  

  CALL dd_extract_grid_array(il_grid2_nc_file_id,  &  

                             il_grid2_size,      &  

                             ila_grid2_dim,      &  

                             dla_grid2_lon,      &  

                             dla_grid2_lat,      &  

                             dla_grid2_corner_lon, &  

                             dla_grid2_corner_lat, &  

                             ila_grid2_mask,      &  

                             ierror)  

  WRITE(12,*) 'dd : apres dd_extract_grid_array with x = ', &  

             ila_grid2_dim(1), ' and y = ', ila_grid2_dim(2)

```

```

CALL flush(12)

DO ib = 1, il_grid2_size
  IF (ila_grid2_mask(ib) .eq. 0) THEN
    ila_grid2_mask(ib) = 1
  ELSE IF (ila_grid2_mask(ib) .eq. 1) THEN
    ila_grid2_mask(ib) = 0
  END IF
END DO

dla_grid2_z(:) = 1

rla_grid2_lon = REAL(dla_grid2_lon)
rla_grid2_lat = REAL(dla_grid2_lat)
rla_grid2_z = REAL(dla_grid2_z)

!
! 2.5. Close the netcdf context
!
CALL dd_close_file(il_grid2_nc_file_id,      &
                  ierror)

!
!=====
!
! 4. extract the source and target field
!
CALL dd_extract_field_array('fielda_1',    &
                           il_grid1_size, &
                           dla_grid1_field, &
                           ierror)

rla_grid1_field = DBLE(dla_grid1_field)

CALL dd_extract_field_array('fieldo_1',    &
                           il_grid2_size, &
                           dla_grid2_field, &
                           ierror)
WRITE(12,*) 'dd : apres dd_extract_field_array'
CALL flush(12)

!
!=====
!
! 5. Distwght search neighboring
!
WRITE(12,*) 'dd : avant la recherche des voisins '
call flush(12)
CALL psmile_trs_dist_srch_neigh2d (      &
  ila_grid1_dim, il_grid1_size, ila_grid1_mask,      &
  dla_grid1_lon, dla_grid1_lat,                      &
  ila_grid2_dim, il_grid2_size, ila_grid2_mask,      &
  dla_grid2_lon, dla_grid2_lat,                      &
  10, 1, 4, ila_neighbors,                        &
  ierror)
CALL psmile_trs_bili_srch_neigh2d (      &
  ila_grid1_dim, il_grid1_size, ila_grid1_mask,      &
  dla_grid1_lon, dla_grid1_lat,                      &

```

```

    ila_grid2_dim, il_grid2_size, ila_grid2_mask,      &
    dla_grid2_lon, dla_grid2_lat,                    &
    10, 1, ila_neighbors2,                          &
    ierror)
WRITE(12,*) 'dd : apres la recherche des voisins '
call flush(12)

!
!=====
!
! 6. Set the interpolation arays with double
!
! 6.1. The source epioS
CALL PSMILe_Trns_set_src_epio3d_dble(il_epio_id, il_grid1_size, &
    dla_grid1_lat, dla_grid1_lon, dla_grid1_z, 1, ila_grid1_mask, &
    ierror)
WRITE(12,*) 'dd : PSMILe_Trns_set_epio2d'
CALL flush(12)

! 6.2. The target epioT
CALL PSMILe_Trns_set_tgt_epio3d_dble(il_epio_id, il_grid2_size, &
    dla_grid2_lat, dla_grid2_lon, dla_grid2_z, 1, ila_grid2_mask, &
    ierror)
WRITE(12,*) 'dd : PSMILe_Trns_set_epio2d'
CALL flush(12)

! 6.3. The neighbors
CALL PSMILe_Trns_give_neighbors3d(il_epio_id, il_grid2_size, &
    il_nb_neighbors, ila_neighbors2, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_give_neighbors2d'
CALL flush(12)

! 6.4. The triple links
CALL PSMILe_Trns_set_triple_links(1, 1, il_epio_id, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_set_triple_links'
CALL flush(12)

!
! 6. Set the interpolation arays with real
!
! 6.1. The source epioS
CALL PSMILe_Trns_set_src_epio3d_real(il_epio_id2, il_grid1_size, &
    rla_grid1_lat, rla_grid1_lon, rla_grid1_z, 1, ila_grid1_mask, &
    ierror)
WRITE(12,*) 'dd : PSMILe_Trns_set_epio2d'
CALL flush(12)

! 6.2. The target epioT
CALL PSMILe_Trns_set_tgt_epio3d_real(il_epio_id2, il_grid2_size, &
    rla_grid2_lat, rla_grid2_lon, rla_grid2_z, 1, ila_grid2_mask, &
    ierror)
WRITE(12,*) 'dd : PSMILe_Trns_set_epio2d'
CALL flush(12)

! 6.3. The neighbors
CALL PSMILe_Trns_give_neighbors3d(il_epio_id2, il_grid2_size, &
    il_nb_neighbors, ila_neighbors2, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_give_neighbors2d'

```

```

CALL flush(12)

! 6.4. The triple links
CALL PSMILe_Trns_set_triple_links(2, 2, il_epio_id2, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_set_triple_links'
CALL flush(12)

!
!=====
!
! 7. Send the field to/from the transformer with dble
!
il_epio_id3 = 3
CALL sleep(20)
!
CALL PSMILe_Trns_put_dble(1, il_epio_id, &
    il_grid1_size, dla_grid1_field, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_put'
CALL flush(12)

!
! 7. Send and Recv the field to/from the transformer with real
!
CALL PSMILe_Trns_put_real(2, il_epio_id2, &
    il_grid1_size, rla_grid1_field, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_put'
CALL flush(12)

CALL PSMILe_Trns_get_real(2, il_epio_id2, &
    il_grid2_size, rla_grid2_field_int, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_get'
CALL flush(12)

!
! 7. Recv the field to/from the transformer with real and int
!
CALL PSMILe_Trns_get_int(3, il_epio_id3, &
    il_grid2_size, ila_grid2_field_int, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_get'
CALL flush(12)

!
!=====
!
! 8. Set the error file with real
!
CALL dd_error_set('error_real.nc', ila_grid2_dim(1), ila_grid2_dim(2), 1, &
    dla_grid2_field, DBLE(rla_grid2_field_int), ierror)
WRITE(13,*) 'dd : apres dd_error_set'
CALL flush(13)

!
! 8. Set the error file with int
!
CALL dd_error_set('error_int.nc', ila_grid2_dim(1), ila_grid2_dim(2), 1, &
    dla_grid2_field, DBLE(ila_grid2_field_int), ierror)
WRITE(13,*) 'dd : apres dd_error_set'
CALL flush(13)

!
!=====
!

```

```

! 9. Finalize
!
  call PRISM_Terminate ( ierror )

END PROGRAM atmosphere

!-----
! BOP
!
!! PROGRAM ocean
!
!! INTERFACE

PROGRAM ocean

!
!! USES:
!
  USE PRISM

  IMPLICIT NONE

  INCLUDE 'mpif.h'

!! DESCRIPTION
! PROGRAM "ocean" simulates the oceanic component
!
!! REVISED HISTORY
! Date Programmer Description
!-----
! 20/10/2003 D. Declat Creation
!
! EOP
!-----
! $Id: ocean.F90,v 1.2 2003/12/29 13:26:56 declat Exp $
! $Author: declat $
!-----
!
! 0. Local declarations
!
! Define precision of variables
!
  INTEGER, PARAMETER :: wp = SELECTED_REAL_KIND(12,307) ! double
!
! Local variables
!
  INTEGER :: ierror, n_errors = 0, t_errors
  INTEGER :: i, j, n, il_count, ib
!
! Init Component
!
  CHARACTER(LEN=128) :: model_name
  INTEGER :: comp_id
!
! get MPI communicator and Initialized
!

```

```

INTEGER :: localComm, npes, mype
LOGICAL :: flag

!
! Netcdf variables
!
CHARACTER(LEN=128) :: cla_file_name
INTEGER          :: il_grid1_nc_file_id
INTEGER          :: il_grid1_size
INTEGER          :: il_grid1_height
INTEGER          :: il_nc_gridsize_id
INTEGER          :: il_grid2_nc_file_id
INTEGER          :: il_grid2_size
INTEGER          :: il_grid2_height

!
! grid arrays that corresponds to partitions
!
INTEGER, DIMENSION(2) :: ila_grid1_dim
INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid1_mask
INTEGER, DIMENSION(:, :), ALLOCATABLE :: ila_neighbors
INTEGER, DIMENSION(:, :), ALLOCATABLE :: ila_neighbors2

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_lat
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_lon
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_z
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_corner_lat
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_corner_lon
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid1_field

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid1_lat
REAL, DIMENSION(:), ALLOCATABLE :: rla_grid1_lon
REAL, DIMENSION(:), ALLOCATABLE :: rla_grid1_z
REAL, DIMENSION(:), ALLOCATABLE :: rla_grid1_field

INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid1_field
!
INTEGER, DIMENSION(2) :: ila_grid2_dim
INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid2_mask

DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_lat
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_lon
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_z
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_corner_lat
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_corner_lon
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_field
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: dla_grid2_field_int

REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_lat
REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_lon
REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_z
REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_field
REAL, DIMENSION(:), ALLOCATABLE :: rla_grid2_field_int

INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid2_field
INTEGER, DIMENSION(:), ALLOCATABLE :: ila_grid2_field_int
!
INTEGER :: il_nb_neighbors

```



```

!  

! 2.3. Allocate the source arrays  

!  

  ALLOCATE(dla_grid1_lat(il_grid1_size), stat = ierror)  

  ALLOCATE(dla_grid1_lon(il_grid1_size), stat = ierror)  

  ALLOCATE(dla_grid1_z(il_grid1_size), stat = ierror)  

  ALLOCATE(dla_grid1_corner_lat(il_grid1_size*4), stat = ierror)  

  ALLOCATE(dla_grid1_corner_lon(il_grid1_size*4), stat = ierror)  

  ALLOCATE(ila_grid1_mask(il_grid1_size), stat = ierror)  

  ALLOCATE(dla_grid1_field(il_grid1_size), stat = ierror)  
  

  ALLOCATE(rla_grid1_lat(il_grid1_size), stat = ierror)  

  ALLOCATE(rla_grid1_lon(il_grid1_size), stat = ierror)  

  ALLOCATE(rla_grid1_z(il_grid1_size), stat = ierror)  

  ALLOCATE(rla_grid1_field(il_grid1_size), stat = ierror)  
  

!  

! 2.4. extract the source grid arrays (lat, lon, corners, mask)  

!  

  CALL dd_extract_grid_array(il_grid1_nc_file_id, &  

     il_grid1_size, &  

     ila_grid1_dim, &  

     dla_grid1_lon, &  

     dla_grid1_lat, &  

     dla_grid1_corner_lon, &  

     dla_grid1_corner_lat, &  

     ila_grid1_mask, &  

     ierror)  

  WRITE(12,*) 'dd : dd_extract_grid_array with x = ', &  

    ila_grid1_dim(1), ' and y = ', ila_grid1_dim(2)  

  CALL flush(12)  
  

  DO ib = 1, il_grid1_size  

    IF (ila_grid1_mask(ib).eq. 0) THEN  

      ila_grid1_mask(ib) = 1  

    ELSE IF (ila_grid1_mask(ib).eq. 1) THEN  

      ila_grid1_mask(ib) = 0  

    END IF  

  END DO  
  

  dla_grid1_z(:) = 1  
  

  rla_grid1_lon = REAL(dla_grid1_lon)  

  rla_grid1_lat = REAL(dla_grid1_lat)  

  rla_grid1_z = REAL(dla_grid1_z)  
  

!  

! 2.5. Close the netcdf context  

!  

  CALL dd_close_file(il_grid1_nc_file_id, &  

     ierror)  

  WRITE(12,*) 'dd : apres dd_close_file'  

  CALL flush(12)  
  

!  

!=====
!  

!
```



```

        ila_grid2_mask,      &
        ierror)
WRITE(12,*) 'dd : apres dd_extract_grid_array with x = ', &
  ila_grid2_dim(1), ' and y = ', ila_grid2_dim(2)
CALL flush(12)

DO ib = 1, il_grid2_size
  IF (ila_grid2_mask(ib) .eq. 0) THEN
    ila_grid2_mask(ib) = 1
  ELSE IF (ila_grid2_mask(ib) .eq. 1) THEN
    ila_grid2_mask(ib) = 0
  END IF
END DO

dla_grid2_z(:) = 1

rla_grid2_lon = REAL(dla_grid2_lon)
rla_grid2_lat = REAL(dla_grid2_lat)
rla_grid2_z = REAL(dla_grid2_z)

!
! 2.5. Close the netcdf context
!
CALL dd_close_file(il_grid2_nc_file_id,      &
  ierror)

!
!=====
!
! 4. extract the source and target field
!
CALL dd_extract_field_array('fielda_1',      &
  il_grid1_size, &
  dla_grid1_field, &
  ierror)

rla_grid1_field = DBLE(dla_grid1_field)

CALL dd_extract_field_array('fieldo_1',      &
  il_grid2_size, &
  dla_grid2_field, &
  ierror)
WRITE(12,*) 'dd : apres dd_extract_field_array'
CALL flush(12)

!
!=====
!
! 5. Distwght search neighboring
!
WRITE(12,*) 'dd : avant la recherche des voisins '
call flush(12)
CALL psmile_trs_dist_srch_neigh2d (          &
  ila_grid1_dim, il_grid1_size, ila_grid1_mask,      &
  dla_grid1_lon, dla_grid1_lat,                      &
  ila_grid2_dim, il_grid2_size, ila_grid2_mask,      &
  dla_grid2_lon, dla_grid2_lat,                      &
  10, 1, 4, ila_neighbors,                        &

```

```

ierror)
CALL psmile_trs_bili_srch_neigh2d (           &
ila_grid1_dim, il_grid1_size, ila_grid1_mask, &
dla_grid1_lon, dla_grid1_lat,               &
ila_grid2_dim, il_grid2_size, ila_grid2_mask, &
dla_grid2_lon, dla_grid2_lat,             &
10, 1, ila_neighbors2,                   &
ierror)
WRITE(12,*) 'dd : apres la recherche des voisins '
call flush(12)

!
!=====
!
! 6. Set the interpolation arrays with int and epio real and double
!
! break to insure that this epio_id is the 3rd
call sleep(10)
il_epio_id = 1
il_epio_id2 = 2

! 6.1. The source epioS
CALL PSMILE_Trns_set_src_epio3d_dble(il_epio_id3, il_grid1_size, &
dla_grid1_lat, dla_grid1_lon, dla_grid1_z, 1, ila_grid1_mask, &
ierror)
WRITE(12,*) 'dd : PSMILE_Trns_set_epio2d'
CALL flush(12)

! 6.2. The target epioT
CALL PSMILE_Trns_set_tgt_epio3d_real(il_epio_id3, il_grid2_size, &
rla_grid2_lat, rla_grid2_lon, rla_grid2_z, 1, ila_grid2_mask, &
ierror)
WRITE(12,*) 'dd : PSMILE_Trns_set_epio2d'
CALL flush(12)

! 6.3. The neighbors
CALL PSMILE_Trns_give_neighbors3d(il_epio_id3, il_grid2_size, &
il_nb_neighbors, ila_neighbors2, ierror)
WRITE(12,*) 'dd : PSMILE_Trns_give_neighbors2d'
CALL flush(12)

! 6.4. The triple links
CALL PSMILE_Trns_set_triple_links(3, 3, il_epio_id3, ierror)
WRITE(12,*) 'dd : PSMILE_Trns_set_triple_links'
CALL flush(12)

!
!=====
!
! 7. Recv the field to/from the transformer with dble
!
CALL PSMILE_Trns_get_dble(1, il_epio_id, &
il_grid2_size, dla_grid2_field_int, ierror)
WRITE(12,*) 'dd : PSMILE_Trns_get'
CALL flush(12)
!
! 7. Send and Recv the field to/from the transformer with real
!

```

```

CALL PSMILe_Trns_put_real(2, il_epio_id2, &
  il_grid1_size, rla_grid1_field, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_put'
CALL flush(12)

CALL PSMILe_Trns_get_real(2, il_epio_id2, &
  il_grid2_size, rla_grid2_field_int, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_get'
CALL flush(12)
!
! 7. Send the field to/from the transformer with real and int
!
CALL PSMILe_Trns_put_real(3, il_epio_id3, &
  il_grid1_size, rla_grid1_field, ierror)
WRITE(12,*) 'dd : PSMILe_Trns_put'
CALL flush(12)
!
!=====
!
! 8. Set the error file with dble
!
CALL dd_error_set('error_double.nc', ila_grid2_dim(1), ila_grid2_dim(2), 1, &
  dla_grid2_field, dla_grid2_field_int, ierror)
WRITE(13,*) 'dd : apres dd_error_set'
CALL flush(13)
!
!=====
!
! 9. Finalize
!
call PRISM_Terminate ( ierror )

END PROGRAM ocean

```