Interaction between the transformer
and the PSMILE library

Damien Declat – CERFACS – 20/10/03
TR/CMGC/03/103

The aim of this document is to detail the interactions between the PSMILE and the transformer. First we will detail the different routines that should be used inside the PSMILE routines, then the transformer logic and finally an example of interaction between the transformer and two component PSMILE.

I - The PSMILE_trs routines

The PSMILE_trs routines should be used inside the PSMILE to communicate informations to the transformer or to search for neighbors considering a target and a source epios (Annex 1 gives a short description of the ‘epio’ notion).

Note: we realize that it would be necessary to take into account the ids of the source and target grids in the psmile_trs call to insure uniqueness in the transformer tables. These parameters will be added later.

A - The set-up routines

The following routines correspond to the set-up and should be used below the PRISM_Enddef routine.

A.1 – Transfer of EPIO

Subroutine \texttt{PSMILE\_Trs\_set\_epio2d}
Subroutine \texttt{PSMILE\_Trs\_set\_epio2d1d}
Subroutine \texttt{PSMILE\_Trs\_set\_epio3d}

\begin{verbatim}
Subroutine \texttt{PSMILE\_Trs\_set\_epio2d} ( il_epio_type, il_corres_comp_id, il_corres_process, il_size, ila_lat, ila_lon, ila_mask, il_err)
\end{verbatim}

\textbf{Purpose:} this routine transfers the EPIO (the points involved in an interpolation computation between one process of the source component and one process of the target component) that has been calculated inside the PSMILE. This EPIO can be a source EPIO or a target EPIO. As we deal with the 2D routines, the transferred arrays are the latitudes, the longitudes and the mask.

\textbf{Arguments:}
- Integer, Intent (In) :: il_epio_type
This integer indicates if the EPIO is a source EPIO or a target EPIO
- Integer, Intent (In) :: il_corres_comp_id
  Global Id of the corresponding source/target component in the interpolation computation
- Integer, Intent (In) :: il_corres_process
  Rank of the corresponding process
- Integer, Intent (In) :: il_size
  Size of the epio
- Real, Dimension(il_size), Intent (In) :: ila_lat
  Latitudes of the epio points
- Real, Dimension(il_size), Intent (In) :: ila_lon
  Longitudes of the epio points
- Integer, Dimension(il_size), Intent (In) :: ila_mask
  Mask array for the epio
- Integer, Intent (Out) :: il_err
  Error code

**Example:** let’s consider a data exchange from process 1 of component 8 to process 0 of component 12. Let’s also consider that an interpolation is required. The ‘envelop’ computation will determine what points are involved in the interpolation process on the source side (component 8) and on the target side (component 12). Once this computation has been done inside the PSMILe interface, the following call are done to inform the transformer:

- **Component 8:**
  call PSMILe_Trs_set_epio2d ( PRISM_Source, 12, 0, size_epio_source, latitude, longitude, mask, error)

- **Component 12:**
  call PSMILe_Trs_set_epio2d ( PRISM_Target, 8, 1, size_epio_target, latitude, longitude, mask, error)

End Subroutine PSMILe_Trs_set_epio2d

Subroutine PSMILe_Trs_set_epio2d1d (il_epio_type, &
  il_corres_comp_id, &
  il_corres_process, &
  il_size, &
  ila_lat, &
  ila_lon, &
  ila_mask, &
  il_height, &
  ila_z, &
  il_err)

**Purpose:** this routine transfers the EPIO (the points involved in an interpolation computation between one process of the source component and one process of the target component) that has been calculated inside the PSMILe. This EPIO can be a source EPIO or a target EPIO. As we deal with the 2D1D routines, the transferred arrays are the latitudes, the longitudes and the mask, but also the vector that gives the $z$ coordinates. The 2D1D grids are sets of identical horizontal 2d subgrids.

**Arguments:**
- Integer, Intent (In) :: il_epio_type
  This integer indicates if the EPIO is a source EPIO or a target EPIO
- Integer, Intent (In) :: il_corres_comp_id
  Global Id of the corresponding source/target component in the interpolation computation
- Integer, Intent (In) :: il_corres_process
  Rank of the corresponding process.
- Integer, Intent (In) :: il_size
  Size of the 2D horizontal arrays (latitudes, longitudes, mask)
- Real, Dimension(il_size), Intent (In) :: ila_lat
Latitudes of the points on the horizontal level
- Real, Dimension(il_size), Intent (In) :: ila_lon
Longitudes of the points on the horizontal level
- Integer, Dimension(il_size), Intent (In) :: ila_mask
Mask array for the points on the horizontal level
- Integer, Intent (In) :: il_height
  Size of the vector that gives the height of the horizontal levels
- Real, Dimension(il_size), Intent (In) :: ila_z
Height of the horizontal levels
- Integer, Intent (Out) :: il_err
  Error code

Example: let’s consider a data exchange from process 1 of component 8 to process 6 of component 12. Let’s also consider that an interpolation is required. The ‘envelop’ computation will determine what points are involved in the interpolation process on the source side (component 8) and on the target side (component 12). Once this computation has been done inside the PSMILe interface, the following calls are done to inform the transformer:

- **Component 8:**
call PSMILe_Tr_set_epi2d1d (PRISM_Source, 12, 6, size_epi_source,
  latitude, longitude, mask,
  nb_of_horizontal_level_source, z, error)

- **Component 12:**
call PSMILe_Tr_set_epi2d1d (PRISM_Target, 8, 1, size_epi_target,
  latitude, longitude, mask,
  nb_of_horizontal_level_target, z, error)

End Subroutine PSMILe_Tr_set_epi2d1d

Subroutine PSMILe_Tr_set_epi3d ( il_epi_type, &
  il_corres_comp_id, &
  il_corres_process, &
  il_size, &
  ila_lat, &
  ila_lon, &
  ila_z, &
  ila_mask, &
  il_err)

Purpose: this routine transfers the EPIO (the points involved in an interpolation computation between one process of the source component and one process of the target component) that has been calculated inside the PSMILe. This EPIO can be a source EPIO or a target EPIO. As we deal with the 3D routines, the transferred arrays are the latitudes, the longitudes, the z coordinates and the mask (for all the points and not only horizontal points).

Arguments:
- Integer, Intent (In) :: il_epi_type
  This integer indicates if the EPIO is a source EPIO or a target EPIO
- Integer, Intent (In) :: il_tgt_comp_id
  Global Id of the corresponding source/target component in the interpolation computation
- Integer, Intent (In) :: il_tgt_process
  Rank of the corresponding process.
- Integer, Intent (In) :: il_size
  Size of the epio.
- Real, Dimension(il_size), Intent (In) :: ila_lat
  Latitudes of the epio points
- Real, Dimension(il_size), Intent (In) :: ila_lon
  Longitudes of the epio points
- Real, Dimension(il_size), Intent (In) :: ila_z
  Z coordinates of the epio points
- Integer, Dimension(il_size), Intent (In) :: ila_mask
  Mask array for the epio
- Integer, Intent (Out) :: il_err
  Error code

**Example:** let’s consider a data exchange from process 1 of component 8 to process 0 of component 12. Let’s also consider that an interpolation is required. The ‘envelop’ computation will determine what points are involved in the interpolation process on the source side (component 8) and on the target side (component 12). Once this computation has been done inside the PSMiLe interface, the following call are done to inform the transformer:

- **Component 8:**
  call PSMiLe_Trs_set_epio2d (PRISM_Source, 12, 0, size_epio_source, latitude, longitude, mask, error)

- **Component 12:**
  call PSMiLe_Trs_set_epio2d (PRISM_Target, 8, 1, size_epio_target, latitude, longitude, mask, error)

End Subroutine PSMiLe_Trs_set_epio3d

### A.2 – Transfer of neighbors

Subroutine PSMiLe_Trs_give_neighbors2d
Subroutine PSMiLe_Trs_give_neighbors2d1d
Subroutine PSMiLe_Trs_give_neighbors3d

Subroutine PSMiLe_Trs_give_neighbors2d(il_src_field_id, il_epiot_size, &
    il_num_neighbors, ila_neighbors, il_tgt_comp_id, il_tgt_process, il_err)

**Purpose:** this routine transfers the neighboring points (for one target point, the neighboring points are the source points involved in the interpolation computation) computed inside the PSMiLe for 2D interpolations. This routine is called by the source processes only.

**Arguments:**
- Integer, Intent (In) :: il_src_field_id
  Local Id of the field which interpolation computation needs these neighbors informations.
- Integer, Intent (In) :: il_epiot_size
  Size of the target epio
- Integer, Intent (In) :: il_num_neighbors
  Number of neighbors (it depends on the method)
- Integer, Dimension(il_epiot_size,il_num_neighbors), Intent (In) :: ila_neighbors
  Addresses of the neighbors, sorted following the target epio.
- Integer, Intent (In) :: il_tgt_comp_id
  Id of the target component in the interpolation computation
- Integer, Intent (In) :: il_tgt_process
  Rank of the target process.
- Integer, Intent (Out) :: il_err
  Error code.

End Subroutine PSMiLe_Trs_give_neighbors2d

Subroutine PSMiLe_Trs_give_neighbors2d1d(il_src_field_id, il_epiot_size, &
    il_num_neighbors, ila_neighbors, il_tgt_comp_id, il_tgt_process, il_err)
**Purpose:** this routine transfers the neighboring points (for one target point, the neighboring points are the source points involved in the interpolation computation) computed inside the PSMILe for 2D1D interpolations. The total number of informations for the neighbors for one target point is obtained combining the number of neighbors on one vertical level and 2 other real that represents the upper and downer vertical levels. This routine is called by the source processes only.

**Arguments:**
- Integer, Intent (In) :: il_src_field_id
  Local Id of the field which interpolation computation needs these neighbors informations.
- Integer, Intent (In) :: il_epiot_size
  Size of the target epio
- Integer, Intent (In) :: il_num_neighbors
  Number of neighbors (it depends on the method. Here, we speak about the number of neighbors on a single horizontal level)
- Integer, Dimension(il_epiot_size,il_num_neighbors+2), Intent(In) :: ila_neighbors
  Addresses of the neighbors, sorted following the target epio
- Integer, Intent (In) :: il_tgt_comp_id
  Id of the target component in the interpolation computation
- Integer, Intent (In) :: il_tgt_process
  Rank of the target process
- Integer, Intent (Out) :: il_err
  Error code

End Subroutine **PSMILe_Trs_give_neighbors2d1d**

Subroutine **PSMILe_Trs_give_neighbors3d**(il_src_field_id, il_epiot_size, & il_num_neigbors, ila_neighbors, il_tgt_comp_id, il_tgt_process, il_err)

**Purpose:** this routine transfers the neighboring points (for one target point, the neighboring points are the source points involved in the interpolation computation) computed inside the PSMILe for 3D interpolations. This routine is called by the source processes only.

**Arguments:**
- Integer, Intent (In) :: il_src_field_id
  Local Id of the field which interpolation computation needs these neighbors informations.
- Integer, Intent (In) :: il_epiot_size
  Size of the target epio
- Integer, Intent (In) :: il_num_neighbors
  Number of neighbors (it depends on the method)
- Integer, Dimension(il_epiot_size,il_num_neighbors), Intent (In) :: ila_neighbors
  Adresses of the neighbors, sorted following the target epio
- Integer, Intent (In) :: il_tgt_comp_id
  Id of the target component in the interpolation computation
- Integer, Intent (In) :: il_tgt_process
  Rank of the target process
- Integer, Intent (Out) :: il_err
  Error code

End Subroutine **PSMILe_Trs_give_neighbors3d**

**B- The exchange routines**
Under the PRISM_Put and the PRISM_Get primitive, if a global transformation is required, the PSMILE will send/receive the data field to/from the transformer process.

Subroutine **PSMILE_Trs_put**(il_field_id, il_size, ila_field, & il_tgt_comp_id, il_tgt_process, il_err)

**Purpose:** this routine sends the data field (the part of the field that belongs to the calling process) to the transformer if a transformation is required in the transformer. The field sent to the transformer is the source epio field.

**Arguments:**
- Integer, Intent (In) :: il_field_id
  Local Id of the field
- Integer, Intent (In) :: il_size
  Size of the field
- Real, Dimension(il_size), Intent (In) :: ila_field
  Field array to send to the transformer
- Integer, Intent (In) :: il_tgt_comp_id
  Id of the target component
- Integer, Intent (In) :: il_tgt_process
  Rank of the target process
- Integer, Intent (Out) :: il_err
  Error code

End Subroutine **PSMILE_Trs_put**

Subroutine **PSMILE_Trs_get**(il_field_id, il_size, ila_field, & il_src_comp_id, il_src_process, il_err)

**Purpose:** this routine receive the data field (the part of the field that belongs to the calling process) form the transformer if a transformation has been required. The field received from the transformer corresponds to a target epio distribution or a grid distribution.

**Arguments:**
- Integer, Intent (In) :: il_field_id
  Local Id of the field
- Integer, Intent (In) :: il_size
  Size of the field
- Real, Dimension(il_size), Intent (In) :: ila_field
  Field array to recv from the transformer
- Integer, Intent (In) :: il_src_comp_id
  Id of the source component
- Integer, Intent (In) :: il_src_process
  Rank of the source process
- Integer, Intent (Out) :: il_err
  Error code

End Subroutine **PSMILE_Trs_get**
C – The finalization routines

As the transformer is now part of the driver process, the different components inform the transformer that they finalize through a call to PSMILe_Trs_finalize, below PRISM_Terminate( ).

Subroutine PSMILe_Trs_finalize(il_err)

D - The search routines

These routines could be used below the PRISM_Enddef( ) primitive to compute the neighbor(s) addresses considering a source epio and a target epio.

The following methods are implemented:
- 2d: nearest neighbor(s), bilinear an conservative method.
- 2d1d: nearest neighbor(s), bilinear on the horizontal subgrid and linear on the vertical direction.
- 3d: nearest neighbor(s), tri-linear method.

The arguments correspond to the different characteristics of the epios (sizes, latitudes, longitudes, masks, z coordinates, etc…) but also to the description of the restriction searches. The restriction consist on dividing the source epio into bins of latitudes (LATITUDE option) or bins of latitudes and longitudes (LATLON option) in order to reduced the source points to consider for one target point.

Subroutine PSMILe_Trs_dist_srch_neigh2d ( &
ila_grid1_dims, il_grid1_size, ila_grid1_mask, &
ila_grid1_center_lon, ila_grid1_center_lat, &
ila_grid2_dims, il_grid2_size, ila_grid2_mask, &
ila_grid2_center_lon, ila_grid2_center_lat, &
il_num_srch_bins, il_restrict_type, &
il_num_neighbors, ila_neighbors, il_err)

Integer, Dimension(2), Intent (In) :: ila_grid1_dims
Integer, Dimension(2), Intent (In) :: ila_grid2_dims
Integer, Intent (In) :: il_grid1_size
Integer, Intent (In) :: il_grid2_size
Integer, Dimension(il_grid1_size), Intent (In) :: ila_grid1_mask
Integer, Dimension(il_grid2_size), Intent (In) :: ila_grid2_mask
Real, Dimension(il_grid1_size), Intent (In) :: ila_grid1_center_lon
Real, Dimension(il_grid1_size), Intent (In) :: ila_grid1_center_lat
Real, Dimension(il_grid2_size), Intent (In) :: ila_grid2_center_lon
Real, Dimension(il_grid2_size), Intent (In) :: ila_grid2_center_lat
Integer, Intent (In) :: il_num_srch_bins
Integer, Intent (In) :: il_restrict_type
Integer, Intent (In) :: il_num_neighbors
Integer, Dimension(il_grid1_size,il_num_neighbors), Intent (Out) :: & ila_neighbors
Integer, Intent (Out) :: il_err

End Subroutine PSMILe_Trs_dist_srch_neigh2d

Subroutine PSMILe_Trs_bili_srch_neigh2d ( &
Subroutine PSMILe_Trs_bili_srch_neigh2d ( &
ILA_grid1_dims, ila_grid1_size, ila_grid1_mask, &
ila_grid1_center_lon, ila_grid1_center_lat, &
ila_grid2_dims, ila_grid2_mask, &
ila_grid2_center_lon, ila_grid2_center_lat, &
il_num_srch_bins, ila_neighbors, ila_err)

Integer, Dimension(2), Intent (In) :: ila_grid1_dims
Integer, Dimension(2), Intent (In) :: ila_grid2_dims
Integer, Intent (In) :: ila_grid1_size
Integer, Intent (In) :: ila_grid2_size
Integer, Dimension ila_grid1_mask
Integer, Dimension ila_grid2_mask
Real, Dimension ila_grid1_center_lon
Real, Dimension ila_grid1_center_lat
Real, Dimension ila_grid2_center_lon
Real, Dimension ila_grid2_center_lat
Integer, Intent (In) :: ila_num_srch_bins
Integer, Intent (In) :: ila_neighbors
Integer, Intent (Out) :: ila_err

End Subroutine PSMILe_Trs_bili_srch_neigh2d

Subroutine PSMILe_Trs_dist_srch_neigh2d1d ( &
ila_grid1_dims, ila_grid1_size, ila_grid1_mask, &
ila_grid1_center_lon, ila_grid1_center_lat, &
ila_grid1_height, ila_grid1_z, &
ila_grid2_dims, ila_grid2_mask, &
ila_grid2_center_lon, ila_grid2_center_lat, &
ila_grid2_height, ila_grid2_z, &
il_num_srch_bins, ila_restrict_type, ila_neighbors, ila_err)

Integer, Dimension(2), Intent (In) :: ila_grid1_dims
Integer, Dimension(2), Intent (In) :: ila_grid2_dims
Integer, Intent (In) :: ila_grid1_size
Integer, Intent (In) :: ila_grid2_size
Integer, Dimension ila_grid1_mask
Integer, Dimension ila_grid2_mask
Real, Dimension ila_grid1_center_lon
Real, Dimension ila_grid1_center_lat
Real, Dimension ila_grid1_z
Real, Dimension ila_grid2_center_lon
Real, Dimension ila_grid2_center_lat
Real, Dimension ila_grid2_z
Integer, Intent (In) :: ila_grid1_height
Integer, Intent (In) :: ila_grid2_height
Integer, Intent (In) :: ila_restrict_type
Integer, Intent (In) :: ila_neighbors
Integer, Dimension ila_grid1_z
Integer, Dimension ila_grid2_z
Integer, Dimension ila_grid1_center_lon
Integer, Dimension ila_grid2_center_lat
Integer, Intent (In) :: ila_grid1_height
Integer, Intent (In) :: ila_grid2_height
Real, Dimension ila_grid1_z
Real, Dimension ila_grid2_z
Integer, Intent (In) :: ila_num_srch_bins
Integer, Intent (In) :: ila_restrict_type
Integer, Intent (In) :: ila_neighbors
Integer, Intent (In) :: ila_neighbors
Integer, Intent (Out) :: ila_neighbors
Integer, Intent (Out) :: ila_err

End Subroutine PSMILe_Trs_dist_srch_neigh2d1d

Subroutine PSMILe_Trs_bili_srch_neigh2d1d ( &
ila_grid1_dims, ila_grid1_size, ila_grid1_mask, &
ila_grid1_center_lon, ila_grid1_center_lat, &
ila_grid2_dims, ila_grid2_mask, &
ila_grid2_center_lon, ila_grid2_center_lat, &
il_num_srch_bins, ila_neighbors, ila_err)

Integer, Dimension(2), Intent (In) :: ila_grid1_dims
Integer, Dimension(2), Intent (In) :: ila_grid2_dims
Integer, Intent (In) :: ila_grid1_size
Integer, Intent (In) :: ila_grid2_size
Integer, Dimension ila_grid1_mask
Integer, Dimension ila_grid2_mask
Real, Dimension ila_grid1_center_lon
Real, Dimension ila_grid1_center_lat
Real, Dimension ila_grid2_center_lon
Real, Dimension ila_grid2_center_lat
Integer, Intent (In) :: ila_num_srch_bins
Integer, Intent (In) :: ila_neighbors
Integer, Intent (Out) :: ila_neighbors
Integer, Intent (Out) :: ila_err

End Subroutine PSMILe_Trs_bili_srch_neigh2d1d
Subroutine PSMILE_Trs_bili_srch_neigh2d1d

Integer, Dimension(2), Intent (In) :: ila_grid1_dims
Integer, Dimension(2), Intent (In) :: ila_grid2_dims
Integer, Intent (In) :: il_grid1_size
Integer, Intent (In) :: il_grid2_size
Integer, Dimension(il_grid1_size), Intent (In) :: ila_grid1_mask
Integer, Dimension(il_grid2_size), Intent (In) :: ila_grid2_mask
Real, Dimension(il_grid1_size), INTENT (Inout) :: ila_grid1_center_lon
Real, Dimension(il_grid1_size), INTENT (Inout) :: ila_grid1_center_lat
Real, Dimension(il_grid1_size), INTENT (Inout) :: ila_grid1_center_z
Real, Dimension(il_grid2_size), INTENT (Inout) :: ila_grid2_center_lon
Real, Dimension(il_grid2_size), INTENT (Inout) :: ila_grid2_center_lat
Real, Dimension(il_grid2_size), INTENT (Inout) :: ila_grid2_center_z
Integer, Intent (In) :: il_grid1_height
Integer, Intent (In) :: il_grid2_height
Real, Dimension(il_grid1_height), Intent (In) :: ila_grid1_z
Real, Dimension(il_grid2_height), Intent (In) :: ila_grid2_z
Integer, Intent (In) :: il_num_srch_bins
Integer, Intent (In) :: il_restrict_type
Integer, Intent (Out) :: ila_neighbors
Integer, Intent (Out) :: il_err

End Subroutine PSMILE_Trs_bili_srch_neigh2d1d

Subroutine PSMILE_Trs_dist_srch_neigh3d

Integer, Dimension(3), Intent (In) :: ila_grid1_dims
Integer, Dimension(3), Intent (In) :: ila_grid2_dims
Integer, Intent (In) :: il_grid1_size
Integer, Intent (In) :: il_grid2_size
Integer, Dimension(il_grid1_size), Intent (In) :: ila_grid1_mask
Integer, Dimension(il_grid2_size), Intent (In) :: ila_grid2_mask
Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_lon
Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_lat
Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_z
Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_lon
Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_lat
Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_z
Integer, Intent (In) :: il_grid1_height
Integer, Intent (In) :: il_grid2_height
Real, Dimension(il_grid1_height), Intent (In) :: ila_grid1_z
Real, Dimension(il_grid2_height), Intent (In) :: ila_grid2_z
Integer, Intent (In) :: il_num_srch_bins
Integer, Intent (In) :: il_restrict_type
Integer, Intent (Out) :: ila_neighbors
Integer, Intent (Out) :: il_err

End Subroutine PSMILE_Trs_dist_srch_neigh3d
Subroutine PSMILe_Trs_trili_srch_neigh3d ( &
   ila_grid1_dims, il_grid1_size, ila_grid1_mask, &
   ila_grid1_center_lon, ila_grid1_center_lat, ila_grid1_center_z, &
   ila_grid2_dims, il_grid2_size, ila_grid2_mask, &
   ila_grid2_center_lon, ila_grid2_center_lat, ila_grid2_center_z, &
   il_num_srch_bin, il_restrict_type, ila_neighbors, il_err)

   Integer, Dimension(3), Intent (In) :: ila_grid1_dims
   Integer, Dimension(3), Intent (In) :: ila_grid2_dims
   Integer, Intent (In) :: il_grid1_size
   Integer, Intent (In) :: il_grid2_size
   Integer, Dimension(il_grid1_size), Intent (In) :: ila_grid1_mask
   Integer, Dimension(il_grid2_size), Intent (In) :: ila_grid2_mask
   Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_lon
   Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_lat
   Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_z
   Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_lon
   Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_lat
   Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_z
   Integer, Intent (In) :: il_num_srch_bin
   Integer, Intent (In) :: il_restrict_type
   Integer, Dimension(il_grid2_size,8), Intent (Out) :: ila_neighbors
   Integer, Intent (Out) :: il_err

End Subroutine PSMILe_Trs_trili_srch_neigh3d

Subroutine PSMILe_Trs_trili_srch_neigh2d ( &
   il_grid1_size, ila_grid1_mask, &
   ila_grid1_center_lon, ila_grid1_center_lat, &
   ila_grid1_corner_lon, ila_grid1_corner_lat, &
   ila_grid1_frac, ila_grid1_area, &
   il_grid2_size, ila_grid2_mask, &
   ila_grid2_center_lon, ila_grid2_center_lat, &
   ila_grid2_corner_lon, ila_grid2_corner_lat, &
   ila_grid2_frac, ila_grid2_area, &
   il_num_srch_bins, il_max_links_map1, il_num_links_map1, &
   ila_grid1_add, ila_grid2_add, ila_weights, il_err)

   Integer, Intent (In) :: il_grid1_size
   Integer, Intent (In) :: il_grid2_size
   Integer, Dimension(il_grid1_size), Intent (In) :: ila_grid1_mask
   Integer, Dimension(il_grid2_size), Intent (In) :: ila_grid2_mask
   Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_lon
   Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_lat
   Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_center_z
   Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_lon
   Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_lat
   Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_center_z
   Real, Dimension(il_grid1_size,4), Intent (Inout) :: ila_grid1_corner_lon
   Real, Dimension(il_grid1_size,4), Intent (Inout) :: ila_grid1_corner_lat
   Real, Dimension(il_grid2_size,4), Intent (Inout) :: ila_grid2_corner_lon
   Real, Dimension(il_grid2_size,4), Intent (Inout) :: ila_grid2_corner_lat
   Real, Dimension(il_grid1_size,4), Intent (Inout) :: ila_grid1_frac
   Real, Dimension(il_grid2_size,4), Intent (Inout) :: ila_grid2_frac
   Real, Dimension(il_grid1_size), Intent (Inout) :: ila_grid1_area
   Real, Dimension(il_grid2_size), Intent (Inout) :: ila_grid2_area
   Integer, Intent (In) :: il_num_srch_bins
II - The Transformer interactions

As described in previous documents, the transformer reacts according to messages sent by the component PSMILe through the use of the PSMILe_trs routines. Indeed, the first message automatically sent below a psmile_trs routine call is a set of integers that describes the action to perform.

This message is received in the transformer inside a loop over MPI receptions. The 1st integer of the 10 integers array gives the action to perform. The following integers will be interpreted considering the selected action, and could be argument of the transformer routines. Thus the transformer analyzes the integers message and reacts, starting predefined sequences of actions that match with the ones defined on the PSMILe side. Therefore, exchanges of information (latitudes, longitudes, etc…) or data fields are possible.

Once an application has finished its execution and reached the PRISM_Terminate primitive, each process of this application sends through the PSMILe_trs_terminate a signal to the transformer. A global integer is incremented and once all the processes have sent this signal, the transformer exits from the loop.

!------------------------------------------------------------------------!
SUBROUTINE PRISMTrs_Loop(il_err)
!
!USES:
!
USE PRISMDrv, dummy_interface => PRISMTrs_Loop
USE PSMILe
!
IMPLICIT NONE
!

!! RETURN VALUE
!

INTEGER, INTENT (Out) :: il_err ! error value
Subroutine "PRISMTrs_Loop" receives the different signals coming from the models PSMIl to react and perform the transformations.

Date Programmer Description
----------  ----------  ----- 
06/10/2003  D. Declat Creation from PRISMTrs_loop

Local declarations

CHARACTER(LEN=len_cvs_string), SAVE :: mycvs = '$Id$

LOGICAL :: ll_loop

INTEGER, PARAMETER :: nerrp=3
INTEGER :: ierrp(nerrp)

INTEGER, DIMENSION(PRISM_Header_length) :: ila_loop

INTEGER :: il_status(MPI_STATUS_SIZE)

WRITE(TRS_OUT,*), '| Enter PRISMTrs_Loop'

ila_loop = 280177
ll_loop = .true.
DO WHILE (ll_loop)

WRITE(TRS_OUT,*), '| | |
WRITE(TRS_OUT,*), '| Trs ready to receive'

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

IF ( il_err /= MPI_SUCCESS ) THEN
ierrp (1) = il_err
ierrp (2) = PRISM_root
ierrp (3) = PSMILe_Init_tag

il_err = PRISM_Error_Recv

CALL PSMILe_Error ( il_err, 'MPI_Recv', &
   ierrp, 3, __FILE__, __LINE__ )
RETURN
ENDIF

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

SELECT CASE (ila_loop(1))

! 1.2.3. Set the grid information
CASE (PRISM_Set_grid_info)

  WRITE(TRS_OUT,*) &
  ' | | Trs updates its grid info for comp ', &
  ila_loop(3), 'grid ', ila_loop(4)

  CALL PRISMTrs_Set_grid_info(ila_loop(2), &
   ila_loop(3), &
   ila_loop(4), &
   ila_loop(5), &
   il_err)

! 1.2.3bis. Set the epio information
CASE (PRISM_Set_epio_info)

  IF (ila_loop(9) .eq. PRISM_Unset) THEN
    WRITE(TRS_OUT,*) &
    ' | | Trs updates its 2d epio (,ila_loop(5),,,ila_loop(7), &
    ') for comp ', ila_loop(4), ' and ', ila_loop(6), '. Size ', &
    ila_loop(8)
  ELSE IF (ila_loop(9) .eq. PRISM_3d) THEN
    WRITE(TRS_OUT,*) &
    ' | | Trs updates its 3d epio (,ila_loop(5),,,ila_loop(7), &
    ') for comp ', ila_loop(4), ' and ', ila_loop(6), '. Size ', &
    ila_loop(8)
  ELSE
    WRITE(TRS_OUT,*) &
    ' | | Trs updates its 2d1d epio (,ila_loop(5),,,ila_loop(7), &
    ') for comp ', ila_loop(4), ' and ', ila_loop(6), '. Size ', &
    ila_loop(8), ' Height ', ila_loop(9)
  END IF

  CALL PRISMTsr_set_grid_info(ila_loop(2), &
   ila_loop(3), &
   ila_loop(4), &
! 1.2.4. Set the neighbors information

CASE (PRISM_Set_neighbors_info)

WRITE(TRS_OUT,*), &
' | | Trs updates its neighbors info for comp ', &
ila_loop(4), ' and ', ila_loop(6), ' for epio (' , ila_loop(5), &
' ', ila_loop(7), ')' 

CALL PRISMTrs_Set_neighbors_info ila_loop(2), &
ila_loop(3), &
ila_loop(4), &
ila_loop(5), &
ila_loop(6), &
ila_loop(7), &
ila_loop(8), &
il_err)

! 1.2.5. Receive a field

CASE (PRISM_Put)

WRITE(TRS_OUT,*), &
'| | Trs receives the field ', &
ila_loop(3), ' from comp ', ila_loop(4), ' process ', ila_loop(5), &
' size ', ila_loop(6)

CALL PRISMTrs_Mind ila_loop(2), &
ila_loop(3), &
ila_loop(4), &
ila_loop(5), &
ila_loop(6), &
ila_loop(7), &
ila_loop(8), &
il_err)

! 1.2.6. Send a field to the target model

CASE (PRISM_Get)

WRITE(TRS_OUT,*), &
'| | Trs is asked to send the field ', &
ila_loop(3), ' to comp ', ila_loop(4), ' process ', ila_loop(5)
WRITE(TRS_OUT,*), &
'| | This interpolated field is coming from comp', &
ila_loop(6), ' process ', ila_loop(7)

CALL PRISMTrs_Target ila_loop(2), &
ila_loop(3), &
III – 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

A - 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:

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

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:

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

\[ \begin{align*}
\text{EPIOT (0,0)} &= (1, 2) & \text{EPIOS (0,0)} &= (1, 2, 4, 5, 7, 8) \\
\text{EPIOT (0,1)} &= (1, 3, 5) & \text{EPIOS (0,1)} &= (2, 3, 5, 6, 8, 9) \\
\text{EPIOT (1,0)} &= (3, 4) & \text{EPIOS (1,0)} &= (7, 8, 9, 12, 13, 14) \\
\text{EPIOT (1,1)} &= (2, 4, 6, 7, 8) & \text{EPIOS (1,1)} &= (1, 2, 3, 4, 5, 6, 9) 
\end{align*} \]

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:

\[ \text{Neighbors\_indices (0,0)} = (/1, 2, 3, 4/ ; /3, 4, 5, 6/) \]

\[ \begin{align*}
\text{Neighbors\_indices (0,0)} &= (/1, 2, 3, 4/ ; /3, 4, 5, 6/) \\
\text{Neighbors\_indices (0,1)} &= (/1, 2, 3, 4/ ; /3, 4, 5, 6/ ; /3, 4, 5, 6/) \\
\text{Neighbors\_indices (1,0)} &= (/1, 2, 3, 4/ ; /3, 4, 5, 6/ ; /5, 6, 8, 9/ ; /7, 8, 10, 11/) \\
\text{Neighbors\_indices (1,1)} &= (/1, 2, 4, 5/ ; /2, 3, 5, 6/) 
\end{align*} \]
Annex 2: Source code of simple atmospheric and oceanic components

These routines can also be found in Bench TRS/Ex_7 in the psmile CVS distribution. The dd_* routines allow to open or close netcdf files, write or read data into netcdf files, etc. They can be found in Bench TRS/NC_TOOLS.

```
!------------------------------------------------------------------------
! BOP
!
!! PROGRAM atmosphere
!
!!INTERFACE
PROGRAM atmosphere
!
!!USES:
!
USE PRISM
IMPLICIT NONE
INCLUDE 'mpif.h'

!!DESCRIPTION
!! PROGRAM "atmosphere" simulates the atmospheric component
!
!!REVISED HISTORY
!! Date        Programmer     Description
!! ---------        ----------        -----------
!! 20/10/2003     D. Declat         Creation
!
!! EOP
!------------------------------------------------------------------------

integer, parameter :: wp = SELECTED_REAL_KIND(12,307) ! double
!
integer :: ierror, n_errors = 0, t_errors
!
character(len=128) :: model_name
integer :: comp_id
```
! get MPI communicator and Initialized
!
integer :: localComm, npes, mype
logical :: flag
!
! Netcdf test
!
CHARACTER(LEN=128) :: cla_file_name
!
INTEGER                         :: il_grid1_nc_file_id
INTEGER                         :: il_grid1_size
INTEGER                         :: il_grid1_height
!
INTEGER                         :: il_grid2_nc_file_id
INTEGER                         :: il_grid2_size
INTEGER                         :: il_grid2_height
!
INTEGER                         :: il_nc_gridsize_id
!
! Grid arrays
!
INTEGER, DIMENSION(2) :: ila_grid1_dim
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_lat
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_lon
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_corner_lat
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_corner_lon
INTEGER, DIMENSION(:,,:), ALLOCATABLE :: ila_neighbors
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_field
!
INTEGER, DIMENSION(2) :: ila_grid2_dim
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid2_lat
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid2_lon
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid2_corner_lat
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid2_corner_lon
REAL, DIMENSION(:,,:), ALLOCATABLE :: grid2_corner_lat
REAL, DIMENSION(:,,:), ALLOCATABLE :: grid2_corner_lon
INTEGER, DIMENSION(:,), ALLOCATABLE :: ila_grid2_mask
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid2_field
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid2_field_alg
!
INTEGER :: i, j, n, il_count
!
INTEGER :: nb_neighbors
!
!================================================================
!
! 1. Initialization
call PRISM_Init_comp (comp_id, model_name, ierr)
if (ierr /= 0) n_errors = n_errors + 1
model_name = atm

! 1.1. Get local communicator
!
call PRISM_Get_localcomm (comp_id, localComm, ierr)
if (ierr /= 0) n_errors = n_errors + 1

call MPI_Comm_Size (localComm, npes, ierr)
call MPI_Comm_Rank (localComm, mype, ierr)

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, ierr)
if (ierr /= 0) n_errors = n_errors + 1
if (.not. flag) then
    write (*, *) model_name, "PRISM is NOT initialized"
endif
!
!=================================================================
!
! 2. netcdf extraction for the source grid
!
! 2.1. Init file for the netcdf context
!
cla_file_name = '../../NC_TOOLS/Grids/Grid_AT42REGU.nc'
call dd_init_file(cla_file_name, il_grid1_nc_file_id, ierr)
!
! 2.2. get the size of the arrays
!
CALL dd_extract_grid_size(il_grid1_nc_file_id, il_grid1_size, ierr)
!
! 2.3. Allocate the source arrays
!
ALLOCATE(ila_grid1_lat(il_grid1_size), stat = ierr)
ALLOCATE(ila_grid1_lon(il_grid1_size), stat = ierr)
ALLOCATE(ila_grid1_corner_lat(il_grid1_size*4), stat = ierr)
ALLOCATE(ila_grid1_corner_lon(il_grid1_size*4), stat = ierr)
ALLOCATE(ila_grid1_mask(il_grid1_size), stat = ierr)
ALLOCATE(ila_grid1_field(il_grid1_size), stat = ierr)
!
! 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, &
    ila_grid1_lon, &
    ila_grid1_lat, &
    ila_grid1_corner_lon, &
    ila_grid1_corner_lat, &
    ila_grid1_mask, &
    ila_grid1_field, &
    ila_neighbors, &
    ila_grid1_size, &
    ila_grid1_dim, &
    ila_grid1_lon, &
    ila_grid1_lat, &
    ila_grid1_corner_lon, &
    ila_grid1_corner_lat, &
    ila_grid1_mask, &
ierror)
!
! 2.5. Close the netcdf context
!
   CALL dd_close_file(il_grid1_nc_file_id, ierror)
!
!===================================================================================================
!
! 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/Grads/Grid_ORCA.nc'
   call dd_init_file(cla_file_name, il_grid2_nc_file_id, ierror)
!
! 3.2. get the size of the arrays
!
   CALL dd_extract_grid_size(il_grid2_nc_file_id, il_grid2_size, ierror)
!
! 3.3. Allocate the target arrays
!
   nb_neighbors = 4
   ALLOCATE(ila_grid2_lat(il_grid2_size), stat = ierror)
   ALLOCATE(ila_grid2_lon(il_grid2_size), stat = ierror)
   ALLOCATE(ila_grid2_corner_lat(il_grid2_size*4), stat = ierror)
   ALLOCATE(ila_grid2_corner_lon(il_grid2_size*4), stat = ierror)
   ALLOCATE(grid2_corner_lat(il_grid2_size,4), stat = ierror)
   ALLOCATE(grid2_corner_lon(il_grid2_size,4), stat = ierror)
   ALLOCATE(ila_grid2_mask(il_grid2_size), stat = ierror)
   ALLOCATE(ila_grid2_field(il_grid2_size), stat = ierror)
   ALLOCATE(ila_neighbors(il_grid2_size,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, &
    ila_grid2_lon, &
CALL dd_close_file(il_grid2_nc_file_id, ierr)

! 4. Extract the source source field
CALL dd_extract_field_array('fielda_1', il_grid1_size, ila_grid1_field, ierr)

! 5. Bilinear or Distwght search neighboring
CALL psmile_trs_dist_srch_neigh2d(il_grid1_dim, il_grid1_size, ila_grid1_mask, ila_grid1_lon, ila_grid1_lat, il_grid2_dim, il_grid2_size, ila_grid2_mask, ila_grid2_lon, ila_grid2_lat, 10, 1, nb_neighbors, ila_neighbors, ierr)

! 6. Set the interpolation arrays
! 6.1. The source epioS
CALL PSMILE_Trs_set_epi2d(2820, 2, 0, il_grid1_size, ila_grid1_lat, ila_grid1_lon, ila_grid1_mask, ierr)

! 6.2. The neighbors
CALL PSMILE_Trs_give_neighbors2d(1, il_grid2_size, nb_neighbors, ila_neighbors, 2, 0, ierr)

! 7. Send the field to the transformer
CALL PSMILE_Trs_put(1, il_grid1_size, ila_grid1_field, 2, 0, ierr)

! 8. Finalize
call PRISM_Message ('vor PRISM_Finalize')
call PRISM_Terminate (ierr)
END PROGRAM atmosphere

!------------------------------------------------------------------------
!        BOP
!------------------------------------------------------------------------

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
!------------------------------------------------------------------------

0. Local declarations
!
! Define precision of variables
!
integer, parameter :: wp = SELECTED_REAL_KIND(12,307) ! double
!
! Local variables
!
integer :: ierr, n_errors = 0, t_errors
!
! Init Component
integer :: comp_id
! character(len=128) :: model_name
! get MPI communicator and Initialized
!
integer :: localComm, npes, mype
logical :: flag
!
! Netcdf test
!
CHARACTER(LEN=128) :: cla_file_name
INTEGER :: il_grid1_nc_file_id
INTEGER :: il_grid1_size
INTEGER :: il_grid1_height
!
INTEGER :: il_grid2_nc_file_id
INTEGER :: il_grid2_size
INTEGER :: il_grid2_height
!
INTEGER :: il_nc_gridsize_id
!
! Grid arrays
!
INTEGER, DIMENSION(2) :: ila_grid1_dim
!
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_lat
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_lon
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_corner_lat
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_corner_lon
INTEGER, DIMENSION(:,), ALLOCATABLE :: ila_grid1_mask
INTEGER, DIMENSION(:,,:), ALLOCATABLE :: ila_neighbors
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_field
REAL, DIMENSION(:,), ALLOCATABLE :: ila_grid1_field_alg
!
INTEGER :: i, j, n, il_count
!
INTEGER :: nb_neighbors
!
!===============================================
!
! 1. Initialization
!
model_name = 'oce'
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_ORCA.nc'
call dd_init_file(cla_file_name, il_grid1_nc_file_id, ierror)
!
! 2.2. get the size of the arrays
!
CALL dd_extract_grid_size(il_grid1_nc_file_id, il_grid1_size, ierror)
!
! 2.3. Allocate the source arrays
!
ALLOCATE(ila_grid1_lat(il_grid1_size), stat = ierror)
ALLOCATE(ila_grid1_lon(il_grid1_size), stat = ierror)
ALLOCATE(ila_grid1_corner_lat(il_grid1_size*4), stat = ierror)
ALLOCATE(ila_grid1_corner_lon(il_grid1_size*4), stat = ierror)
ALLOCATE(ila_grid1_mask(il_grid1_size), stat = ierror)
ALLOCATE(ila_grid1_field(il_grid1_size), stat = ierror)
ALLOCATE(ila_grid1_field_alg(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, &
  ila_grid1_dim, &
  ila_grid1_lon, &
  ila_grid1_lat, &
  ila_grid1_corner_lon, &
  ila_grid1_corner_lat, &
ierror)
! ila_grid1_mask, &
! 2.5. Close the netcdf context
! CALL dd_close_file(il_grid1_nc_file_id, ierror)
!
!====================================
!====================================
!
! 3. Set the interpolation arrays
!
! 3.1. The target epioT
! CALL PSMILe_Trs_set_epio2d(2821, 1, 0, il_grid1_size, ila_grid1_lat, ila_grid1_lon, &
ila_grid1_mask, ierror)
!
!=================================================================
!
! 4. Get the field from the transformer
!
! CALL PSMILe_Trs_get(1, il_grid1_size, ila_grid1_field, 1, 0, ierror)
!
!=================================================================
!
! 5. extract the target field
!
! CALL dd_extract_field_array('fieldo_1', il_grid1_size, ila_grid1_field_alg, ierror)
!
!=================================================================
!
! 6. Set the error file
!
! CALL dd_error_set ila_grid1_dim(1), ila_grid1_dim(2), 1, &
ila_grid1_field_alg, ila_grid1_field, ierror)
!
!======================================================================
!
! 7. Finalize
!
! call PRISM_Message ('vor PRISM_Finalize')
call PRISM_Terminate ( ierror )

END PROGRAM ocean