

SIREN

Rev:

17058

Generated by Doxygen 1.8.5

Fri Dec 6 2019 09:18:35

Contents

Chapter 1

About

SIREN is a software to set up regional configuration with [NEMO](#).

Actually SIREN creates the input files you need to run a NEMO regional configuration.

SIREN allows you to create your own regional configuration embedded in a wider one.

To know how to install SIREN see [Download](#).

You could find a tutorial for a quick start with SIREN in [How To Use \(Quick Start\)](#).

For more information about how to use each component of SIREN

- see [create_coord.f90](#) to create fine grid coordinate file
- see [create_bathy.f90](#) to create fine grid bathymetry
- see [merge_bathy.f90](#) to merge fine grid bathymetry
- see [create_meshmask.f90](#) to create mesh mask or domain_cfg file.
- see [create_layout.f90](#) to create/compute the domain layout of your configuration.
- see [create_restart.f90](#) to create initial state file, or other fields.
- see [create_boundary.F90](#) to create boundary condition

- [Download](#)
- [How To Use \(Quick Start\)](#)
- [Support](#)
- [Coding Rules](#)
- [Change log](#)
- [Todo List](#)

Chapter 2

Download

Download NEMO

to install SIREN, you should first download NEMO. see [NEMO quick start guide](#)

Compile SIREN

when NEMO is installed, you just have to compile SIREN codes:

1. go to `./NEMOGCM/TOOLS`
2. run `maketools` (ex: `./maketools -n SIREN -m ifort_mpi_beaufix`)

Note

to get help on `maketools`: `./maketools -h`

Fortran Compiler

SIREN codes were successfully tested with :

- ifort (version 18.0.1 20171018)
- gfortran (version 4.8.5 20150623)

- [About](#)
- [How To Use \(Quick Start\)](#)
- [Support](#)
- [Coding Rules](#)
- [Change log](#)
- [Todo List](#)

Chapter 3

How To Use (Quick Start)

SIREN is a software to set up regional configuration with [NEMO](#).

Actually SIREN creates all the input files you need to run a NEMO regional configuration.

SIREN is composed of a set of Fortran programs :

- [create_coord.f90](#) to create regional grid coordinates.
- [create_bathy.f90](#) to create regional grid bathymetry.
- [merge_bathy.f90](#) to merge regional grid bathymetry with wider grid bathymetry at boundaries.

Note

the goal of this step is to avoid break in Bathymetry. This break may cause inconsistency between forcing fields at boundary and regional fields.

- [create_meshmask.f90](#) to create meshmask or domain_cfg [file\(s\)](#) which contain(s) all the ocean domain informations.
- [create_layout.f90](#) to create/compute the domain layout of your configuration.
- [create_restart.f90](#) to create initial state file from coarse grid restart or standard outputs.

Note

this program could also be used to refined other input fields from a wider configurations (as runoff, chlorophyll etc...)

- [create_boundary.F90](#) to create boundaries conditions from wider configurations output fields.

Warning

SIREN can not:

- create global configuration.
- create configuration around or close to North pole.
- change number of vertical level.

Here after we briefly describe how to use each programs, and so how to create your own regional configuration.

Note

As demonstrator for a first start a set of GLORYS files (global reanalysis on *ORCA025* grid), as well as examples of namelists are available [here](#).

3.1 Create coordinates file

To create your own configuration, you first have to create a coordinates file on your domain of study.

SIREN allows you to create this coordinates file from a wider coordinates file.

The coordinates file created could simply be an extraction, or a refinement of the wide grid.

To create this new coordinates file, you have to run :

```
./SIREN/create_coord.exe create_coord.nam
```

Here after is an example of namelist for *create_coord.exe*.

In this example, you create a coordinates file named *coord_fine.nc*.

This new coordinates file is refined from an extraction of *coordinates_ORCA025.nc*.

```
&namlog
/

&namcfg
  cn_varcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/variable.cfg"
  cn_dimcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/dimension.cfg"
  cn_dumcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/dummy.cfg"
/

&namsrc
  cn_coord0 = "PATH/coordinates_ORCA025.nc"
  in_perio0 = 4
/

&namvar
/

&namnst
  in_imin0 = 1070
  in_imax0 = 1072
  in_jmin0 = 607
  in_jmax0 = 609

  in_rhoi = 2
  in_rhoj = 3
/

&namout
  cn_fileout = "PATH/coord_fine.nc"
/
```

Note

you could define sub domain with coarse grid indices or with coordinates.

Let's get describe this namelist.

First we have the **namlog** sub-namelist. This sub-namelist set parameters of the log file.

All the parameters of this sub-namelist have default value, so you could let it empty, as done here.

This will create a log file named *create_coord.log*

namcfg

The **namcfg** sub-namelist defines where found SIREN configuration files.

- The variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.
Obviously, you could add other variables to those already list, in this file.
- The dimension configuration file defines dimensions allowed.

- The dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

Note

You could find the generic version of those configuration files in the directory *SIREN/cfg*.

namsrc

The **namsrc** sub-namelist set parameters of the coarse/source coordinates file name, path to find it, and the NEMO periodicity of the grid.

Note

the NEMO periodicity could be choose between 0 to 6:

in_perio=0 standard regional model

in_perio=1 east-west cyclic model

in_perio=2 model with symmetric boundary condition across the equator

in_perio=3 regional model with North fold boundary and T-point pivot

in_perio=4 global model with a T-point pivot.

example: ORCA2, ORCA025, ORCA12

in_perio=5 regional model with North fold boundary and F-point pivot

in_perio=6 global model with a F-point pivot

example: ORCA05

See Also

For more information see [NEMO periodicity](#)

namvar

The **namvar** sub-namelist lists variables to be used.

By default all the variables of the coarse/source coordinates file are used to create the new coordinates file.

The interpolation methods to be used are defined in the configuration variables file (see above). So you do not need to fill this sub-namelist too.

namnst

The **namnst** sub-namelist defines the subdomain to be used as well as refinement factor.

- you could define sub domain with coarse grid indices

```
&namnst
  in_imin0 = 1070
  in_imax0 = 1072
  in_jmin0 = 607
  in_jmax0 = 609
/
```

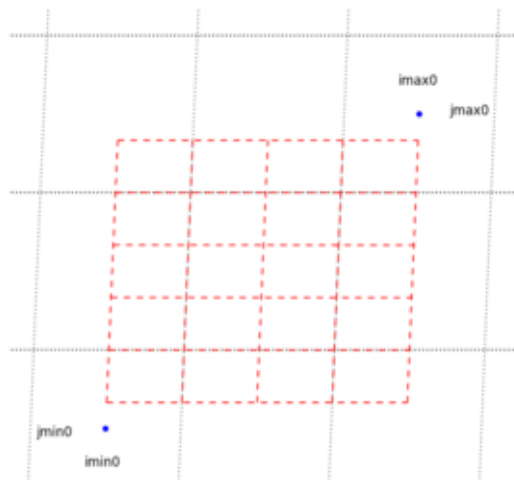
- or with coordinates

```
&namnst
  rn_lonmin0 = -97.9
  rn_lonmax0 = -62.3
  rn_latmin0 = 7.7
  rn_latmax0 = 30.8
/
```

- you can select area quite every where (excepted too close from the North pole), and use the refinement factor you want.

```
&namnst
  in_imin0 = 1070
  in_imax0 = 1072
  in_jmin0 = 607
  in_jmax0 = 609

  in_rhoi = 2
  in_rhoj = 3
/
```



- you can select area crossing the east-west overlap of the global ORCA grid.

```
&namnst
  in_imin0 = 1402
  in_imax0 = 62
  in_jmin0 = 490
  in_jmax0 = 570

  in_rhoi = 5
  in_rhoj = 5
/
```



- you can select east-west cyclic area.

```
&namnst
  in_imin0 = 0
  in_imax0 = 0
  in_jmin0 = 390
  in_jmax0 = 450

  in_rhoi = 1
  in_rhoj = 1
/
```



Finally the **namout** sub-namelist defines the output file.

Note

All the output files created by SIREN include information about NEMO periodicity, as well as source file, indices and refinement used.

See Also

For more information and options to create coordinates, see [create_coord.f90](#)

3.2 Create bathymetry file

Then you need a Bathymetry file.

SIREN allows you to create a Bathymetry extracted or refined from a wider Bathymetry grid.

To create this new bathymetry, you have to run :

```
./SIREN/create_bathy.exe create_bathy.nam
```

Here after is an example of namelist for *create_bathy.exe*.

In this example, you create a bathymetry file named *bathy_fine.nc*.

This new bathymetry file is refined from an extraction of *bathy_meter_ORCA025.nc*.

Moreover a minimum value of 5m is imposed to the output Bathymetry.

```
&namlog
/

&namcfg
  cn_varcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/variable.cfg"
  cn_dimcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/dimension.cfg"
/

&namsrc
  cn_coord0 = "PATH/coordinates_ORCA025.nc"
  in_perio0 = 4
/

&namtgt
  cn_coord1 = "PATH/coord_fine.nc"
/

&namvar
  cn_varfile = "Bathymetry:PATH/bathy_meter_ORCA025.nc"
  cn_varinfo = "Bathymetry: min=5"
/

&namnst
```

```

    in_rhoi = 2
    in_rhoj = 3
/

&namout
  cn_fileout = "PATH/bathy_fine.nc"
/

```

Let's get describe this namelist.

namlog, namcfg

First as previously, we have the **namlog** and **namcfg** sub-namelist (see above for more explanation).

namsrc

Then the **namsrc** sub-namelist set parameters of the source/wide coordinates file.

Note

in all SIREN namelist:
0 referred to the coarse/wide grid.
1 referred to the fine grid.

namtgt

In the same way, the **namtgt** sub-namelist set parameters of the target/fine coordinates file.

Note

in this namelist example, there is no need to set the variable *in_perio1* to define the NEMO periodicity of the fine grid. Indeed, if this variable is not inform, SIREN tries to read it in the global attributes of the file. So if you created the fine coordinates with SIREN, you do not have to fill it. In other case, you should add it to the namelist.

namvar

The **namvar** sub-namelist lists variables to be used:

cn_varfile defines the variable name ("Bathymetry" here) and the input file associated with.

Warning

The domain of the input Bathymetry have to be larger than the output domain.

Note

- if the input file is at coarse grid resolution (same than *cn_coord0*), the output Bathymetry will be refined on fine grid.
- if the input file is a wider bathymetry (already at fine grid resolution), the output Bathymetry will be extracted from this one.

cn_varinfo defines user's requests for a variable.

Note

Default interpolation method for the Bathymetry, is *cubic* interpolation.
 So you may want to specify a minimum value to avoid negative value, or to change interpolation method.
 example:

- `cn_varinfo="Bathymetry: min=1"`
- `cn_varinfo="Bathymetry: int=linear"`

namnst

The **namnst** sub-namelist defines the subdomain refinement factor.

Of course those refinement factors have to be convenient with refinement from coarse grid *cn_coord0* to fine grid *cn_coord1*.

Note

subdomain indices are automatically deduced from fine and coarse grid coordinates.

namout

Finally, this **namout** sub-namelist defines the output file.

Note

All the output files create by SIREN include information about source file, indices, refinement and interpolation method used.

See Also

For more information and options to create bathymetry, see [create_bathy.f90](#)

3.3 Merge bathymetry file

The Bathymetry you build, may differ from the wider one.

To avoid issue with boundaries forcing fields, you should merge fine and coarse Bathymetry on boundaries.

SIREN allows you to do this.

To create this merged bathymetry, you have to run :

```
./SIREN/merge_bathy.exe merge_bathy.nam
```

Here after is an example of namelist for *merge_bathy.exe*.

```
&namlog
/

&namcfg
  cn_varcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/variable.cfg"
  cn_dimcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/dimension.cfg"
/

&namsrc
  cn_bathy0 = "PATH/bathy_meter_ORCA025.nc"
  in_perio0 = 4
/

&namtgt
  cn_bathy1 = "PATH/bathy_fine.nc"
/

&namnst
  in_rhoi = 3
  in_rhoj = 3
/

&nambdy
/

&namout
  cn_fileout = "PATH/bathy_merged.nc"
/
```

namlog, namcfg

In this namelist, you find again the **namlog**, **namcfg** describe above.

namsrc

Then the **namsrc** sub-namelist sets parameters of the source/wide grid. However this time, this is the source/wide grid Bathymetry wich have to be informed.

namtgt

The **namtgt** sub-namelist defines parameters of the target/fine grid Bathymetry.

Note

here again you could add the *in_perio1* parameter if need be i.e. if your fine grid Bathymetry was not created by SIREN.

namnst

The **namnst** sub-namelist defines the subdomain refinement factor.

nambdy

The **nambdy** sub-namelist defines the subdomain boundaries.

By default SIREN tries to create boundaries for each side. Boundary exist if there is at least one sea point on the second row of each side. So you could let this namelist empty.

See Also

For more information about boundaries, see [Create boundaries conditions](#)

namout

Finally, this **namout** sub-namelist defines the output file.

See Also

For more information and options to merge bathymetry, see [merge_bathy.f90](#)

3.4 Create meshmask (ocean domain informations)

Depending on the vertical grid you choose to use, NEMO may not see the bathymetry exactly as you defined it just before. To get the ocean domain informations as seen by NEMO, SIREN allows you to create the meshmask [file\(s\)](#) which contain(s) all those informations.

Morevoer SIREN allows you to create the *domain_cfg* file which is the new input file for NEMO (release 3.7 and upper).

To create the meshmask [file\(s\)](#), you have to run :

```
./SIREN/create_meshmask.exe create_meshmask.nam
```


Here after is an example of namelist for *create_meshmask.exe*.

In this example, you create one meshmask file named *mesh_mask.nc*.

It uses coordinates file *coord_fine.nc* to define horizontal grid. and defines z-coordinate with partial steps. The minimum depth of the final Bathymetry is 10m.

```

&namlog
/

&namcfg
  cn_varcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/variable.cfg"
  cn_dimcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/dimension.cfg"
/

&namsrc
  cn_bathy = "PATH/bathy_merged.nc"
  cn_coord = "PATH/coord_fine.nc"
  in_perio = 4
/

&namhgr
  in_mshhgr = 0
/

&namzgr
  ln_zps = .TRUE.
  in_nlevel= 75
/

&namdmin
  dn_hmin=10.
/

&namzco
  dn_ppsur = -3958.951371276829
  dn_ppa0 = 103.953009600000
  dn_ppa1 = 2.415951269000
  dn_ppkth = 15.351013700000
  dn_ppacr = 7.000000000000
  dn_ppdzmin = 6.
  dn_pphmax = 5750.
  ln_dbletanh= .TRUE.
  dn_ppa2 = 100.760928500000
  dn_ppkth2 = 48.029893720000
  dn_ppacr2 = 13.000000000000
/

&namzps
  dn_e3zps_min = 25.
  dn_e3zps_rat = 0.2
/

&namsco
/

&namlbc
/

&namwd
/

&namgrd
/

&namout
  in_msh = 1
/

```

Let's get describe this namelist more accurately.

namlog, namcfg

As previously, we have the **namlog** and **namcfg** describe above.

namsrc

The **namsrc** sub-namelist defines the Bathymetry to be used. Mainly SIREN need Bathymetry to create meshmask. Here we also read coordinates directly on a file.

Note

1. here again you could add the *in_perio* parameter if need be i.e. if your Bathymetry was not created by SIREN.
2. by default SIREN suppress closed sea/lake from the ORCA domain.

namhgr

The **namhgr** sub-namelist defines the horizontal grid. the type of horizontal mesh is choose between :

- *in_mshhgr=0* : curvilinear coordinate on the sphere read in coordinate.nc
- *in_mshhgr=1* : geographical mesh on the sphere with regular grid-spacing
- *in_mshhgr=2* : f-plane with regular grid-spacing
- *in_mshhgr=3* : beta-plane with regular grid-spacing
- *in_mshhgr=4* : Mercator grid with T/U point at the equator
- *in_mshhgr=5* : beta-plane with regular grid-spacing and rotated domain (GYRE configuration)

namzgr

The **namzgr** sub-namelist allows to choose the type of vertical grid (z-coordinate full steps, partial steps, sigma or hybrid coordinates) and the number of level.

namdmin

The **namdmin** sub-namelist defines the minimum ocean depth. It could be defines in meter (>0) or in number of level (<0).

namzco

The **namzco** sub-namelist defines parameters to compute z-coordinate vertical grid (**needed for all type of vertical grid**)

namzps

The **namzps** sub-namelist defines extra parameters needed to define z-coordinates partial steps.

namSCO

The **namSCO** sub-namelist defines extra parameters needed to define sigma or hybrid coordinates (not needed here).

namlbc

The **namlbc** sub-namelist defines lateral boundary conditions at the coast. It is needed to modify the fmask.

namwd

The **namwd** sub-namelist defines the wetting and drying parameters if activated (see `namzgr` sub-namelist)

namgrd

The **namgrd** sub-namelist allows to use configuration 1D or to choose vertical scale factors (`e3.=dk` or old definition).

namout

Finally, this **namout** sub-namelist defines the number output [file\(s\)](#).

Note

To create the `domain_cfg` file, you should put `in_msh=0`.

See Also

For more information about how to create meshmask, see [create_meshmask.f90](#)

3.5 Create layout (domain layout)

To run faster your configuration you may need to run it on multiprocessor. To do so you first need to know on which domain layout and so on how many processor you could do it. .

To create/compute the domain layout, you have to run :

```
./SIREN/create_layout.exe create_layout.nam
```

Here after is an example of namelist for `create_layout.exe`.

In this example, you compute the domain layout on 40 processors.

```
&namlog
/

&namcfg
  cn_varcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/variable.cfg"
  cn_dimcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/dimension.cfg"
/

&namvar
  cn_varfile = "Bathymetry:PATH/bathy_merged.nc"
/

&namout
  in_nproc = 40
/
```

Let's get describe this namelist more accurately.

namlog, namcfg

As previously, we have the **namlog** and **namcfg** describe above.

namvar

The **namvar** sub-namelist lists variables to be used. Mainly SIREN need Bathymetry to compute the domain layout, or at least the mask of your domain.

namout

The **namout** sub-namelist defines the number of processor you want to work on.

See Also

For more information and options to create layout, see [create_layout.f90](#)

3.6 Create initial state

To run your configuration you need an initial state of the ocean.

You could start from a restart file (with all NEMO variables fields at one time step). Or you could start from "partial" information about ocean state (Temperature and Salinity for example).

SIREN allows you to create both of those initial state.

To create the initial state, you have to run:

```
./SIREN/create_restart.exe create_restart.nam
```

Here after is an example of namelist for *create_restart.exe*.

In this example, you create an initial state split on 81 "processors", and named restart_out.nc.

The initial state is composed of temperature and salinity refined from an extraction of GLORYS fields.

```
&namlog
/

&namcfg
  cn_varcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/variable.cfg"
  cn_dimcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/dimension.cfg"
/

&namsrc
  cn_coord0 = "PATH/coordinates_ORCA025.nc"
  in_perio0 = 4
/

&namtgt
  cn_coord1 = "PATH/coord_fine.nc"
  cn_bathyl = "PATH/bathy_merged.nc"
/

&namzgr
/

&namzps
/

&namvar
  cn_varfile = "votemper:GLORYS_gridT.nc",
              "vosaline:GLORYS_gridS.nc"
/

&namnst
  in_rhoi = 3
  in_rhoj = 3
/

&namout
  cn_fileout = "PATH/restart_out.nc"
  in_nproc = 81
/
```

Let's get describe this namelist more accurately.

namlog, namcfg

As previously, we have the **namlog** and **namcfg** sub-namelists, as well as

namsrc

the **namsrc** sub-namelist to set parameters of the source/wide coordinates file (see above for more explanation).

namtgt

Then the **namtgt** sub-namelist set parameters of the target/fine grid coordinates and bathymetry.

namzgr, namzps

The **namzgr** and **namzps** sub-namelists define respectively parameters for vertical grid and partial step.

By default, those parameters are defined the same way than in GLORYS (i.e. 75 vertical levels).

So you could let it empty.

Note

If you use forcing fields other than GLORYS, you have to be sure it uses the same vertical grid. In other case, you need to get information about the parametrisation use, and to put it in those sub-namelist (see [create_restart.f90](#)).

namvar

the **namvar** sub-namelist lists variables to be used.

Here we use *votemper* (temperature) get from *GLORYS_gridT.nc* file, and *vosaline* (salinity) get from *GLORYS_gridS.nc* file.

Note

To get all variables of a restart file. You have to use:

```
cn_varfile = "all:PATH/restart.dimg"
```

namnst

The **namnst** sub-namelist defines the subdomain refinement factor, as seen previously.

namout

Finally, this **namout** sub-namelist defines the output files.

Here we ask for output on 81 processors, with *restart_out.nc* as file "basename".

So SIREN computes the optimal layout for 81 processors available,

and split restart on output files named *restart_out_num.nc*, where *num* is the proc number.

Note

SIREN could also create the other fields you may need for your configuration.

To do so, you just have to run *create_restart.exe* with other variable(s) from other input [file\(s\)](#).

For example, to get runoff fields, you could use:

```
cn_varfile = "sorunoff:PATH/runoff_GLORYS.nc"
...
cn_fileout = "PATH/runoff_out.nc"
```

See Also

For more information and options to create initial state or other fields, see [create_restart.f90](#)

3.7 Create boundaries conditions

Finally to force your configuration, you may need boundaries conditions.

NEMO read physical boundaries conditions from temperature, salinity, currents, and sea surface height.

To create the boundaries condition with SIREN, you have to run:

```
./SIREN/create_boundary.exe create_boundary.nam
```

Here after is an example of namelist for *create_boundary.exe*.

In this example, you create boundaries conditions named *boundary_out.nc* on each side of the domain.

The boundaries contain information about temperature, salinity, currents and sea surface height refined from an extraction of GLORYS fields.

```
&namlog
/

&namcfg
  cn_varcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/variable.cfg"
  cn_dimcfg = "PATH/NEMOGCM/TOOLS/SIREN/cfg/dimension.cfg"
/

&namsrc
  cn_coord0 = "PATH/coordinates_ORCA025.nc"
  in_perio0 = 4
/

&namtgt
  cn_coord1 = "PATH/coord_fine.nc"
  cn_bathy1 = "PATH/bathy_fine.nc"
/

&namzgr
/

&namzps
/

&namvar
  cn_varfile="votemper:GLORYS_gridT.nc",
             "vosaline:GLORYS_gridS.nc",
             "vozocrtx:GLORYS_gridU.nc",
             "vomecrtx:GLORYS_gridV.nc",
             "sossheig:GLORYS_grid2D.nc"
/

&namnst
  in_rhoi = 3
  in_rhoj = 3
/

&nambdy
/

&namout
  cn_fileout = "PATH/boundary_out.nc"
/
```

Let's get describe this namelist more accurately.

namlog, namcfg

As previously, we have the **namlog** and **namcfg** sub-namelists, as well as

namsrc

the **namsrc** sub-namelist to set parameters of the source/wide coordinates file (see above for more explanation).

namtgt

Then the **namtgt** sub-namelist set parameters of the target/fine grid coordinates and bathymetry.

namzgr, namzps

The **namzgr** and **namzps** sub-namelists define respectively parameters for vertical grid and partial step.

By default, those parameters are defined the same way than in GLORYS (i.e. 75 vertical levels).

So you could let it empty.

Note

If you use forcing fields other than GLORYS, you have to be sure it uses the same vertical grid. In other case, you need to get information about the parametrisation use, and to put it in those sub-namelist (see [create_boundary.F90](#)).

namvar

the **namvar** sub-namelist lists variables to be used.

Here we get *votemper* (temperature) from *GLORYS_gridT.nc* file, *vosaline* (salinity) from *GLORYS_gridS.nc* file, *vozocrtx* (zonal velocity) from *GLORYS_gridU.nc*, *vomecrtx* (meridional velocity) from *GLORYS_gridV.nc*, and *sosshg* (sea surface height) from *GLORYS_grid2D.nc*.

namnst

The **namnst** sub-namelist defines the subdomain refinement factor.

nambdy

The **nambdy** sub-namelist defines the subdomain boundaries.

By default SIREN tries to create boundaries for each side (Boundary is created if sea point exist on the second row of each side).

So you could let this namelist empty.

Note

SIREN allows you to place boundaries away from the side of the domain. To do so you have to define your boundary.

That means you have to give on fine grid the index of the boundary (how far from the border your boundary is), the width of your boundary, and finally first and last point of your boundary (the length of your boundary).

So to define a north boundary, you have to add in the sub-namelist *nambdy*, the parameter:

```
cn_north="index,first:last (width) "
```

namout

Finally, this **namout** sub-namelist defines the output files.

Here we ask for output with *boundary_out.nc* as file "basename".

So SIREN creates output files named *boundary_out_west.nc*, *boundary_out_east.nc*, *boundary_out_north.nc*, and *boundary_out_south.nc* depending if boundary exist or not.

See Also

For more information and options to create boundaries condition, see [create_boundary.F90](#)

3.8 Create and run NEMO configuration

So now you created all the input files you need for your physical configuration, you have to create the "NEMO configuration".

To do so, go to the directory *NEMOGCM/CONFIG/*, and run:

```
./makenemo -n MY_CONFIG -d "OPA_SRC"
```

This creates your configuration "MY_CONFIG" in the directory *NEMOGCM/CONFIG*.

you could check the *cpp_keys* used in file *cpp_MY_CONFIG.fcm*, and re-run *makenemo* if need be.

Once *makenemo* has run successfully, the *opa* executable is available in directory *NEMOGCM/CONFIG/MY_CONFIG/EXP00*.

Then you just have to put all your input files in this directory, fill the namelist *namelist_cfg*, and run:

```
mpirun ./opa
```

Note

no surface forcing here. weighted function needed to do interpolation on the fly, could be created by WEIGHT tools already inside NEMO.

See Also

For more information about how to create NEMO configuration see [NEMO Quick Start Guide](#).

- [About](#)
- [Download](#)
- [How To Use \(Quick Start\)](#)
- [Support](#)
- [Coding Rules](#)
- [Change log](#)
- [Todo List](#)

Chapter 4

Support

How to get support

If you have questions regarding the use of SIREN, please have a look at the [NEMO configuration manager forum](#).

If you don't find an answer in the archives, feel free to register and post your question.

How to Help

The development of SIREN highly depends on your input!

If you are trying SIREN let me know what you think of it (do you miss certain features?). Even if you decide not to use it, please let me know why.

How to report a bug

If you believe you have found a new bug, please report it.

Before submitting a new bug, first search through the database if the same bug has already been submitted by others

If you send only a (vague) description of a bug, you are usually not very helpful and it will cost much more time to figure out what you mean. In the worst-case your bug report may even be completely ignored.

- [About](#)
- [Download](#)
- [How To Use \(Quick Start\)](#)
- [Support](#)
- [Coding Rules](#)
- [Change log](#)
- [Todo List](#)

Chapter 5

Coding Rules

The conventions used in SIREN coding are based on the NEMO coding rules (see [NEMO coding conventions](#)).

However some modifications were added to improve readability of the code.

Some of the NEMO coding rules are reminded here, and extensions are described.

5.1 Fortran Standard

SIREN software adhere to strict **FORTRAN 95** standard.

There is only one exception. The use of functions *COMMAND_ARGUMENT_COUNT* and *GET_COMMAND_ARGUMENT*.

There exist no equivalent for those Fortran 03 intrinsic functions in Fortran 95.

At least none convenient for compilers tested (see [Download](#)).

5.2 Free Form Source

Free Form Source will be used, however a self imposed limit of 80 should enhance readability.

5.3 Indentation

Code as well as comments lines will be indented 3 characters for readability.

Indentation should be write without hard tabs.

Example for vi :

```
:set expandtab tabstop=3 shiftwidth=3
```

5.4 Naming conventions : variable

All variables should be named as explicitly as possible.

The naming conventions concerns prefix letters of these name, in order to identify the variable type and status.

It must be composed of two letters defining type and status follow by an underscore.

table below list the starting letters to be used for variable naming, depending on their type and status.

Type / Status	byte (integer(1)) b	short (integer(2)) s	integer(4) i	integer(8) k	real(4) r	real(8) d	logical l	character c	complex y	structure t
global g	bg_	sg_	ig_	kg_	rg_	dg_	lg_	cg_	yg_	tg_
global parameter p	bp_	sp_	ip_	kp_	rp_	dp_	lp_	cp_	yp_	tp_
module m	bm_	sm_	im_	km_	rm_	dm_	lm_	cm_	ym_	tm_
namelist n	bn_	sn_	in_	kn_	rn_	dn_	ln_	cn_	yn_	tn_
dummy argument d	bd_	sd_	id_	kd_	rd_	dd_	ld_	cd_	yd_	td_
local l	bl_	sl_	il_	kl_	rl_	dl_	ll_	cl_	yl_	tl_
function result f	bf_	sf_	if_	kf_	rf_	df_	lf_	cf_	yf_	tf_
loop control			j?							

5.5 Naming conventions : structure

The structure name should be written in capital letter, and start with **T**

Example: TTRACER

Variables inside the structure should be named as explicitly as possible.

For those variables, the prefix naming conventions only concern the type of variable.

It must be composed of one letter defining type follows by an underscore.

see table of variable conventions.

Example: **tl_type%i_year**

year is an integer(4) variable in a local structure named *type*.

5.6 Naming conventions : function-subroutine

Functions or Subroutines are defined in a module.

Their name should start with the module name then with their "functional" name. So it will be easy to find it.

Example:

a function to realise addition written in a module **math** should be called **math_add**.

PUBLIC function or subroutine should use one underscore: *math_add*

PRIVATE function or subroutine should use two underscores: *math__add*

5.7 Precision

All variables should make use of kinds.

Numerical constant need to have a suffix of **kindvalue**

5.8 Declaration for global variable and constants

All global data must be accompanied with a comment field on the same line.

Note that using doxygen (see [header](#)), we will use symbol !< instead of !: as separator

5.9 Implicit none

All subroutines and functions will include an IMPLICIT NONE statement.

5.10 Header

SIREN use **doxygen auto documentation** tool.

Information could be find on [doxygen](#) web page.

Some basic tag are described [here](#).

- [About](#)
- [Download](#)
- [How To Use \(Quick Start\)](#)
- [Support](#)
- [Change log](#)
- [Todo List](#)

Chapter 6

Change log

6.1 Release 2019-12-03

New features

- M Siren/src/iom_cdf.f90 :
 - write netcdf file as netcdf4

Release 2019-11-05

New features

- M Siren/src/function.f90
- M [src/create_bathy.f90](#) :
 - add help and version optional arguments
 - update header for usage
- M [src/create_boundary.F90](#) :
 - add help and version optional arguments
 - update header for usage
- M [src/create_coord.f90](#) :
 - add help and version optional arguments
 - update header for usage
- M [src/create_layout.f90](#) :
 - add help and version optional arguments
 - update header for usage
- M [src/create_meshmask.f90](#) :
 - add help and version optional arguments
 - update header for usage
- M [src/create_restart.f90](#) :
 - add help and version optional arguments

- update header for usage
- M [src/merge_bathy.f90](#) :
 - add help and version optional arguments
 - update header for usage
- M [src/function.f90](#) :
 - add help and version functions
- M [src/global.f90](#) :
 - add parameter for version, author, and date
 - set SVN keyword Revision, Author, and Date

6.2 Release 2019-10-18

New Features

- A [templates/README](#)
- A [templates/create_templates.py](#) :
 - script to create template of namelist from Siren sources

Changes

- M [templates/addline.nam](#) :
 - update template of namelist (with default value)
- M [templates/create_bathy.nam](#) :
 - update template of namelist (with default value)
- M [templates/create_boundary.nam](#) :
 - update template of namelist (with default value)
- M [templates/create_coord.nam](#) :
 - update template of namelist (with default value)
- M [templates/create_restart.nam](#) :
 - update template of namelist (with default value)
- M [templates/merge_bathy.nam](#) :
 - update template of namelist (with default value)

6.3 Release 2019-08-12

Changes

- [src/create_bathy.f90](#) :
 - use periodicity read from namelist, and store in multi structure
- [src/create_boundary.F90](#) :

- use periodicity read from namelist, and store in multi structure
- [src/create_restart.f90](#) :
 - use periodicity read from namelist, and store in multi structure
- [src/mpp.f90](#) :
 - change print format
- [src/multi.f90](#) :
 - use periodicity read from namelist, and store in multi structure
- [templates/addline.nam](#) :
 - rename sub namelist namcrs to namsrc
- [templates/create_bathy.nam](#) :
 - rename sub namelist namcrs to namsrc
 - rename sub namelist namfin to namtgt
- [templates/create_boundary.nam](#) :
 - rename sub namelist namcrs to namsrc
- rename sub namelist namfin to namtgt
- [templates/create_coord.nam](#) :
 - rename sub namelist namcrs to namsrc
- [templates/create_restart.nam](#) :
 - rename sub namelist namcrs to namsrc
 - rename sub namelist namfin to namtgt

Bug fixes

- [src/create_layout.f90](#) :
 - add missing variable in logger namelist
- [src/multi.f90](#) :
 - rewrite function to subroutine
 - output filename string contains only filename (no more periodicity if given)

6.4 Release 2019-05-21

New Features

- [src/date.f90](#) :
 - add date_time subroutine

Changes

- [src/create_restart.f90](#) :
 - force number of proc to one by default
 - create and clean file structure to avoid memory leaks
- [src/file.f90](#) :
 - add option to check dimension axis but not length
- [src/grid.f90](#) :
 - do not use latitude variable to get pivot
 - read only grid boundaries to handle huge file
 - define as module variable `im_max_overlap`
- [src/iom.f90](#) :
 - permit to write header and variable independantly
 - split `iom_write_file` into `write_header` and `write_var`
- [src/iom_cdf.f90](#) :
 - permit to write header and variable independantly
 - split `iom_cdf_write_file` into `write_header` and `write_var`
 - add dimension structure as optional argument
 - do not check variable dimension if dimension forced
- [src/iom_dom.f90](#) :
 - copy variable struct without array of value, then read array of value
- [src/iom_mpp.f90](#) :
 - handle use of domain decomposition for monoproc file
- [src/iom_rstdimg.f90](#) :
 - handle use of domain decomposition for monoproc file
 - split `iom_rstdimg_write_file` into `write_header` and `write_var`
- [src/merge_bathy.f90](#) :
 - create and clean file structure to avoid memory leaks
- [src/mpp.f90](#) :
 - cosmetic change
 - use mpp decomposition for writing netcdf
 - force to use domain decomposition to enhance read of monoproc file
 - add `mpp__add_proc_arr`
 - handle use of domain decomposition for monoproc file
- [src/variable.f90](#) :
 - permit to copy variable structure without value

Bug fixes

- src/math.f90 :
 - use the correct loop index to look for element bigger than pivot point
- src/multi.f90 :
 - compare each elt of cl_tabfile to cl_file

6.5 Release 2019-05-15

New Features

- src/addline_deg.f9 :
 - new program to add line to all variable of the input file
- [src/create_layout.f90](#) :
 - new program to create/compute the domain layout
- [src/create_meshmask.f90](#) :
 - new program to create meshmask or domain.cfg
- src/grid_hgr.f90 :
 - new module needed to create meshmask
- src/grid_zgr.f90 :
 - new module needed to create meshmask
- src/lbc.f90 :
 - new module needed to create meshmask
- [src/create_bathy.f90](#) :
 - optionally, add random value to a flat bathymetry
- src/global.f90 :
 - add svn keyword properties
- src/math.f90 :
 - add function math_ortho
 - add function math_euclid
- src/multi.f90 :
 - add function multi__get_perio
- src/variable.f90 :
 - add function var_chg_name to rename variable
 - add output name to variable structure

Changes

- [src/create_bathy.f90](#) :
 - add url path to global attributes of output [file\(s\)](#)
 - create and clean file structure to avoid memory leaks
 - check dimension of matrix for 'handmade' bathymetry
 - check name and standard name for longitude and latitude
 - rewrite header (doc)
 - rename sub namelist namcrs to namsrc
 - rename sub namelist namfin to namtgt
 - change NEMO path
- [src/create_boundary.F90](#) :
 - add url path to global attributes of output [file\(s\)](#)
 - explain how to fill Lateral Boundary Condition in NEMO namelist
 - create and clean file structure to avoid memory leaks
 - rewrite header (doc)
 - rename sub namelist namcrs to namsrc
 - rename sub namelist namfin to namtgt
 - change NEMO path
- [src/create_coord.f90](#) :
 - add url path to global attributes of output [file\(s\)](#)
 - create and clean file structure to avoid memory leaks
 - rewrite header (doc)
 - rename sub namelist namcrs to namsrc
- [src/create_restart.f90](#) :
 - add url path to global attributes of output [file\(s\)](#)
 - check name and standard name for longitude and latitude
 - rewrite header (doc)
 - rename sub namelist namcrs to namsrc
 - rename sub namelist namfin to namtgt
 - change NEMO path
- [src/global.f90](#) :
 - define svn url
- [src/merge_bathy.f90](#) :
 - add url path to global attributes of output [file\(s\)](#)
 - rewrite header (doc)
 - rename sub namelist namcrs to namsrc
 - rename sub namelist namfin to namtgt
 - change NEMO path
- [src/create_meshmask.f90](#) :
 - change NEMO path
- [src/mpp.f90](#) :

- copy file periodicity to mpp structure
- clean file, variable, and attributes structures
- deallocate file structure whatever happens
- do not split variable on domain decomposition, if only one processor
- nullify array in layout structure
- nullify file structure inside mpp structure
- src/attribute.f90 :
 - nullify array inside attribute structure
- src/boundary.f90 :
 - nullify segment structure inside boundary structure
- src/date.f90 :
 - check time units CF convention, raise error if not
- src/dimension.f90 :
 - do not reshape array already order
- src/file.f90 :
 - clean variable, attributes, and dimension structures
 - nullify variable, attributes structures inside file structure
 - netcdf4 files identify as netcdf file
- src/function.f90 :
 - permit sign as first character
- src/grid.f90 :
 - do not use silicalim, or silicamax to get pivot point
 - check name and standard name for latitude
- src/iom_cdf.f90 :
 - clean variable, attributes, and dimension structures
 - read array in netcdf file, level by level, and time step by time step
 - apply scale factor and offset, level by level, and time step by time step
 - check attribute array is allocated, before use it
- src/iom_rstdimg.f90 :
 - clean dimension structure
- src/multi.f90 :
 - create and clean file structure to avoid memory leaks
 - fill value read from array of variable structure
 - nullify mpp structure in multi file structure
 - deallocate mpp structure whatever happens
 - print periodicity
- src/phycst.f90 :
 - half reduce epsilon value
- src/variable.f90 :

- use scalar instead of array, as transitional variable
- nullify attributes structure inside variable structure
- decompose array conversion on each dimension
- decompose array copy on each dimension
- deallocate attribute structure whatever happens
- clean attribute structure
- nullify array inside variable structure
- write fill value on array, level by level

- src/docsrc/main.dox :

- add create_meshmask to SIREN tools list
- add create_layout to SIREN tools list

- src/docsrc/1_install.md :

- change NEMO path

- src/docsrc/2_quickstart.md :

- change link to SIREN inputs
- add description of create_meshmask
- add description of create_layout

Bug fixes

- src/function.f90 :

- permit negative exponent

- src/iom_cdf.f90 :

- use dimid to look for the index of the dimension, and not as dimension index

- src/mpp.f90 :

- netcdf proc indices from zero to N-1

- src/variable.f90 :

- check if tg_varextra is allocated before clean it
- add case for units in hours
- read number of element for each dummy array in configuration file

- src/attributes.f90 :

- read and use number of element for each dummy array in configuration file

- src/multi.f90 :

- specify format output

- src/dimension.f90 :

- read number of element for each dimension allowed in configuration file
- read number of element for each dummy array in configuration file

6.6 Release 2016-11-16

New Features

- [create_meshmask.f90](#) program to create meshmask from coordinates and bathymetry files.
- [create_meshmask.f90](#) allows to write domain_cfg file.
- [merge_bathy.f90](#):
 - allow to choose the number of boundary point with coarse grid value.
- [dimension.f90](#):
 - dimension allowed read in configuration file.
- [variable.f90](#):
 - allow to add scalar value.
- [create_meshmask.f90](#):
 - choose vertical scale factors (e3.=dk[depth] or old definition).

Changes

- [create_coord.f90](#):
 - allow to define sub domain with coarse grid indices or coordinates.
- [grid.f90](#): -grid__get_closest_str:
 - add function to get closest grid point using coarse grid coordinates structure.
- [iom_cdf.f90](#): -iom_cdf__get_info:
 - define cdf4 as cdf.
- [variable.f90](#):
 - add subroutine to clean global array of extra information, and define logical for variable to be used.
- [create_coord.f90](#):
 - dimension to be used select from configuration file.
- [create_bathy.f90](#):
 - dimension to be used select from configuration file.
- [merge_bathy.f90](#):
 - dimension to be used select from configuration file.
- [create_boundary.f90](#):
 - dimension to be used select from configuration file.
- [create_restart.f90](#):
 - dimension to be used select from configuration file.

Bug fixes

- boundary.f90: -boundary_check:
 - take into account that boundaries are compute on T point, but expressed on U,V point.
- grid.f90: -grid__get_closest_str:
 - use max of zero and east-west overlap instead of east-west overlap.
- mpp.f90:
 - compare index to td_lay number of proc instead of td_mpp (bug fix) .
- iom_cdf.f90 :
 - check if attribute cl_value is not bug (in read file)

6.7 Release 2016-07-01

Changes

New Features

Bug fixes

- correct check of boundary indices

6.8 Initial Release 2016-04-11

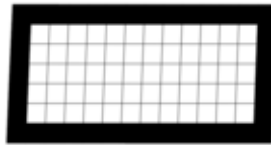
- [About](#)
- [Download](#)
- [How To Use \(Quick Start\)](#)
- [Support](#)
- [Coding Rules](#)
- [Todo List](#)

Chapter 7

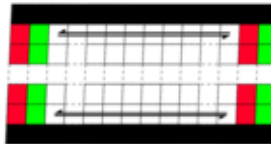
NEMO periodicity

NEMO periodicity is defined as follow :

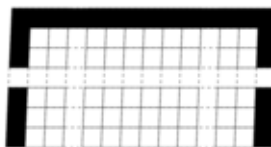
closed boundary (in_perio=0) ghost cells (solid walls) are imposed at all model boundaries.



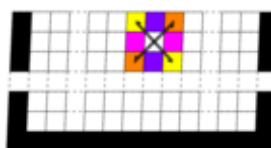
cyclic east-west boundary (in_perio=1) first and last rows are closed, whilst the first column is set to the value of the last-but-one column and the last column to the value of the second one.



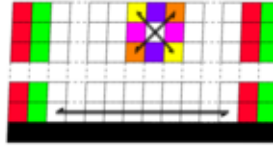
symmetric boundary condition across the equator. (in_perio=2) last row, and first and last columns are closed.



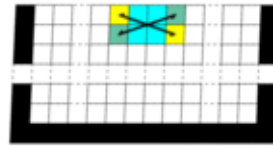
North fold boundary with a T-point pivot (in_perio=3) first row, and first and last columns are closed.



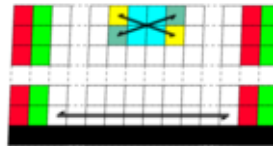
North fold boundary with a T -point pivot and cyclic east-west boundary (in_perio=4) first row is closed. The first column is set to the value of the last-but-one column and the last column to the value of the second one.



North fold boundary with a F -point pivot (in_perio=5) first row, and first and last columns are closed.



North fold boundary with a F -point pivot and cyclic east-west boundary (in_perio=6) first row is closed. The first column is set to the value of the last-but-one column and the last column to the value of the second one.



See Also

For more information about NEMO periodicity, see *Model Boundary Condition* chapter in [NEMO documentation](#))

Chapter 8

Todo List

Type **boundary**

add schematic to boundary structure description

File **create_bathy.f90**

check tl_multi is not empty

File **create_boundary.F90**

rewrite using meshmask instead of bathymetry and coordinates files.

Subprogram **create_coord_interp** (td_var, id_rho, id_offset, id_iext, id_jext)

check if mask is really needed

check if mask is really needed

File **create_restart.f90**

rewrite using meshmask instead of bathymetry and coordinates files

Type **date**

see calendar.f90 and select Gregorian, NoLeap, or D360 calendar

Type **extrap**

create module for each extrapolation method

- smooth extrapolated points

Subprogram **extrap::extrap_add_extrabands** (td_var, id_isize, id_jsize)

invalid special case for grid with north fold

- invalid special case for grid with north fold

Type **iom**

see lbc_ink

- see goup netcdf4

Type **kind**

check i8 max value

Type **mpp**

ECRIRE ET TESTER add_proc_array pour optimiser codes (voir old/MO_mpp.f90)

Type **var**

var_copy_value qui copie le tableau de valeur mais verifie que tous les attribut sont egaux

Type **vgrid**

fusionner vgrid et [grid_zgr](#)

Subprogram **vgrid::vgrid_zgr_bat** (dd_bathy, dd_gdepw, dd_hmin, dd_fill)

add subroutine description

add subroutine description

Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

att		
	This module manage attribute of variable or file	??
att::att_clean		??
att::att_copy		??
att::att_init		??
att::att_print		??
boundary		
	This module manage boundary	??
boundary::boundary_clean		??
boundary::boundary_copy		??
boundary::boundary_init		??
boundary::boundary_print		??
date		
	This module provide the calculation of Julian dates, and do many manipulations with dates	??
date::date_init		??
dim		
	This module manage dimension and how to change order of those dimension	??
dim::dim_clean		??
dim::dim_copy		??
dim::dim_print		??
dim::dim_reorder_2xyzt		??
dim::dim_reorder_xyzt2		??
dim::dim_reshape_2xyzt		??
dim::dim_reshape_xyzt2		??
dom		
	This module manage domain computation	??
dom::dom_copy		??
dom::dom_init		??
extrap		
	This module manage extrapolation	??
extrap::extrap_detect		??
extrap::extrap_fill_value		??
fct		
	This module groups some basic useful function	??
fct::fct_str		??
file		
	This module manage file structure	??
file::file_clean		??

file::file_copy	??
file::file_del_att	??
file::file_del_var	??
file::file_rename	??
filter	
This module is filter manager	??
filter::filter_fill_value	??
global	
This module defines global variables and parameters	??
grid	
This module is grid manager	??
grid::grid_get_closest	??
grid::grid_get_coarse_index	??
grid::grid_get_ew_overlap	??
grid::grid_get_fine_offset	??
grid::grid_get_ghost	??
grid::grid_get_info	??
grid::grid_get_perio	??
grid::grid_get_pivot	??
grid_hgr	
This module manage Horizontal grid	??
grid_zgr	
This module manage Vertical grid	??
interp	
This module manage interpolation on regular grid	??
interp_cubic	
This module manage cubic interpolation on regular grid	??
interp::interp_detect	??
interp::interp_fill_value	??
interp_linear	
This module manage linear interpolation on regular grid	??
interp_nearest	
This module manage nearest interpolation on regular grid	??
iom	
Input/Output manager : Library to read input files	??
iom_cdf	
NETCDF Input/Output manager : Library to read Netcdf input files	??
iom_cdf::iom_cdf_fill_var	??
iom_cdf::iom_cdf_read_att	??
iom_cdf::iom_cdf_read_dim	??
iom_cdf::iom_cdf_read_var	??
iom_dom	
This module allow to read domain (defined as domain structure) in a mpp files	??
iom_dom::iom_dom_read_var	??
iom_mpp	
This module manage massively parallel processing Input/Output manager. Library to read/write mpp files	??
iom_mpp::iom_mpp_read_var	??
iom::iom_read_att	??
iom::iom_read_dim	??
iom::iom_read_var	??
iom_rstdimg	
This module is a library to read/write dimg file	??
iom_rstdimg::iom_rstdimg_read_dim	??
iom_rstdimg::iom_rstdimg_read_var	??
kind	
This module defines the F90 kind parameter for common data types	??

<code>lbc</code>		
	This module groups lateral boundary conditions subroutine	??
<code>lbc::lbc__hide_nfd</code>		??
<code>lbc::lbc_hide</code>		??
<code>lbc::lbc_lnk</code>		??
<code>lbc::lbc_nfd</code>		??
<code>logger</code>		
	This module manage log file	??
<code>math</code>		
	This module groups some useful mathematical function	??
<code>math::math_mean</code>		??
<code>math::math_median</code>		??
<code>math::math_mwe</code>		??
<code>mpp</code>		
	This module manage massively parallel processing	??
<code>mpp::mpp__add_proc</code>		??
<code>mpp::mpp__check_dim</code>		??
<code>mpp::mpp__del_proc</code>		??
<code>mpp::mpp_clean</code>		??
<code>mpp::mpp_copy</code>		??
<code>mpp::mpp_del_att</code>		??
<code>mpp::mpp_del_var</code>		??
<code>mpp::mpp_get_use</code>		??
<code>mpp::mpp_init</code>		??
<code>multi</code>		
	This module manage multi file structure	??
<code>multi::multi_copy</code>		??
<code>date::operator(+)</code>		??
<code>date::operator(-)</code>		??
<code>fct::operator(/)</code>		??
<code>phycst</code>		
	This module defines physical constant	??
<code>boundary::seg_clean</code>		??
<code>boundary::seg_copy</code>		??
<code>att::tatt</code>		??
<code>boundary::tbody</code>		
	Boundary structure	??
<code>date::tdate</code>		??
<code>dim::tdim</code>		??
<code>dom::tdom</code>		??
<code>file::tfile</code>		??
<code>interp::tinterp</code>		??
<code>mpp::tlay</code>		
	Domain layout structure	??
<code>logger::tlogger</code>		??
<code>mpp::tmpp</code>		??
<code>multi::tmulti</code>		??
<code>grid_hgr::tnamh</code>		??
<code>grid_zgr::tnamz</code>		??
<code>boundary::tseg</code>		??
<code>var::tvar</code>		??
<code>var</code>		
	This module manage variable structure	??
<code>var::var_add_att</code>		??
<code>var::var_add_dim</code>		??
<code>var::var_add_value</code>		??
<code>var::var_clean</code>		??
<code>var::var_copy</code>		??

var::var_del_att	??
var::var_init	??
var::var_print	??
vgrid	
This module manage vertical grid	??

Chapter 10

File Index

10.1 File List

Here is a list of all documented files with brief descriptions:

src/addline_deg.f90	This program add line to all variables of the input file	??
src/create_bathy.f90	This program creates fine grid bathymetry file	??
src/create_boundary.F90	This program creates boundary files	??
src/create_coord.f90	This program creates fine grid coordinate file	??
src/create_layout.f90	This program creates/computes the domain layout for you configuration	??
src/create_meshmask.f90	This program creates the NetCDF file(s) which contain(s) all the ocean domain informations. it also permits to create the domain_cfg.nc file (needed to run NEMO v4.0 and upper), or the mesh_mask file(s)	??
src/create_restart.f90	This program creates restart file or initial state	??
src/merge_bathy.f90	This program merges bathymetry file at boundaries	??

Chapter 11

Class Documentation

11.1 att Module Reference

This module manage attribute of variable or file.

Data Types

- interface [att_clean](#)
- interface [att_copy](#)
- interface [att_init](#)
- interface [att_print](#)
- type [tatt](#)

Public Member Functions

- integer(i4) function, public [att_get_index](#) (td_att, cd_name)
- integer(i4) function, public [att_get_id](#) (td_att, cd_name)
This function return attribute id, read from a file.
- subroutine, public [att_get_dummy](#) (cd_dummy)
- logical function, public [att_is_dummy](#) (td_att)
This function check if attribute is defined as dummy attribute in configuraton file.

11.1.1 Detailed Description

This module manage attribute of variable or file.

```
define type TATT:<br/>
```

```
TYPE(tatt) :: tl_att
```

the attribute value inside attribute structure will be character or real(8) 1D array.

However the attribute value could be initialized with:

- character
- scalar (real(4), real(8), integer(4) or integer(8))
- array 1D (real(4), real(8), integer(4) or integer(8))

to initialize an attribute structure :

```
t1_att=att_init('attname',value)
```

- value is a character, scalar value or table of value

to print attribute information of one or array of attribute structure:

```
CALL att_print(td_att)
```

to clean attribute structure:

```
CALL att_clean(td_att)
```

to copy attribute structure in another one (using different memory cell):

```
t1_att2=att_copy(t1_att1)
```

Note

as we use pointer for the value array of the attribute structure, the use of the assignment operator (=) to copy attribute structure create a pointer on the same array. This is not the case with this copy function.

to get attribute index, in an array of attribute structure:

```
il_index=att_get_index( td_att, cd_name )
```

- td_att array of attribute structure
- cd_name attribute name

to get attribute id, read from a file:

```
il_id=att_get_id( td_att, cd_name )
```

- td_att array of attribute structure
- cd_name attribute name

to get attribute name

- t1_att%c_name

to get character length or the number of value store in attribute

- t1_att%i_len

to get attribute value:

- t1_att%c_value (for character attribute)
- t1_att%d_value(i) (otherwise)

to get the type number (based on NETCDF type constants) of the attribute:

- t1_att%i_type

to get attribute id (read from file):

- t1_att%i_id

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- Fix memory leaks bug

September, 2015

- manage useless (dummy) attributes

May, 2019

- read number of element for each dummy array in configuration file

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.1.2 Member Function/Subroutine Documentation**11.1.2.1 subroutine, public att::att_get_dummy (character(len=*), intent(in) *cd_dummy*)****Parameters**

<i>in</i>	<i>cd_dummy</i>
-----------	-----------------

Author

J.Paul

Date

September, 2015 - Initial Version

Marsh, 2016

- close file (bugfix)

May, 2019

- read number of dummy element

Parameters

<i>in</i>	<i>cd_dummy</i>	dummy configuration file
-----------	-----------------	--------------------------

11.1.2.2 integer(i4) function, public att::att_get_id (type(tatt), dimension(:), intent(in) *td_att*, character(len=*), intent(in) *cd_name*)

This function return attribute id, read from a file.

Parameters

<i>in</i>	<i>td_att</i>
-----------	---------------

if attribute name do not exist, return 0.

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- bug fix with use of id read from attribute structure

Parameters

in	<i>td_att</i>	array of attribute structure
in	<i>cd_name</i>	attribute name

Returns

attribute id

11.1.2.3 integer(i4) function, public att::att_get_index (type(tatt), dimension(:), intent(in) *td_att*, character(len=*) , intent(in) *cd_name*)

Parameters

in	<i>td_att</i>	array of attribute structure
in	<i>cd_name</i>	attribute name

Returns

attribute index

11.1.2.4 logical function, public att::att_is_dummy (type(tatt), intent(in) *td_att*)

This function check if attribute is defined as dummy attribute in configuraton file.

Parameters

in	<i>td_att</i>	
----	---------------	--

Author

J.Paul

Date

September, 2015 - Initial Version

, May, 2019

- use number of dummy elt in do-loop

Parameters

<i>in</i>	<i>td_att</i>	attribute structure
-----------	---------------	---------------------

Returns

true if attribute is dummy attribute

The documentation for this module was generated from the following file:

- src/attribute.f90

11.2 att::att_clean Interface Reference

Public Member Functions

- subroutine [att_clean_unit](#) (*td_att*)
- subroutine [att_clean_arr](#) (*td_att*)

11.2.1 Member Function/Subroutine Documentation

11.2.1.1 subroutine att::att_clean::att_clean_arr (type(tatt), dimension(:), intent(inout) *td_att*)

Parameters

<i>in, out</i>	<i>td_att</i>	attribute structure
----------------	---------------	---------------------

11.2.1.2 subroutine att::att_clean::att_clean_unit (type(tatt), intent(inout) *td_att*)

Parameters

<i>in, out</i>	<i>td_att</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version
January, 2019

- nullify array inside attribute structure

Parameters

<i>in, out</i>	<i>td_att</i>	attribute structure
----------------	---------------	---------------------

The documentation for this interface was generated from the following file:

- src/attribute.f90

11.3 att::att_copy Interface Reference

Public Member Functions

- type(**tatt**) function [att_copy_unit](#) (td_att)
This subroutine copy an attribute structure in another one.
- type(**tatt**) function, dimension(size(td_att(:))) [att_copy_arr](#) (td_att)

11.3.1 Member Function/Subroutine Documentation

11.3.1.1 type(**tatt**) function, dimension(size(td_att(:))) att::att_copy::att_copy_arr (type(**tatt**), dimension(:), intent(in) td_att)

Parameters

in	td_att	array of attribute structure
----	--------	------------------------------

Returns

copy of input array of attribute structure

11.3.1.2 type(**tatt**) function att::att_copy::att_copy_unit (type(**tatt**), intent(in) td_att)

This subroutine copy an attribute structure in another one.

Parameters

in	td_att	
----	--------	--

attribute value are copied in a temporary array, so input and output attribute structure value do not point on the same "memory cell", and so on are independant.

Warning

do not use on the output of a function who create or read an attribute (ex: tl_att=[att_copy](#)(att_init()) is forbidden). This will create memory leaks.

to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- use function instead of overload assignment operator (to avoid memory leak)

Parameters

in	td_att	attribute structure
----	--------	---------------------

Returns

copy of input attribute structure

The documentation for this interface was generated from the following file:

- src/attribute.f90

11.4 att::att_init Interface Reference

Public Member Functions

- type(tatt) function [att__init_c](#) (cd_name, cd_value)
- type(tatt) function [att__init_dp](#) (cd_name, dd_value, id_type)
- type(tatt) function [att__init_dp_0d](#) (cd_name, dd_value, id_type)
- type(tatt) function [att__init_sp](#) (cd_name, rd_value, id_type)
- type(tatt) function [att__init_sp_0d](#) (cd_name, rd_value, id_type)
- type(tatt) function [att__init_i1](#) (cd_name, bd_value, id_type)
- type(tatt) function [att__init_i1_0d](#) (cd_name, bd_value, id_type)
- type(tatt) function [att__init_i2](#) (cd_name, sd_value, id_type)
- type(tatt) function [att__init_i2_0d](#) (cd_name, sd_value, id_type)
- type(tatt) function [att__init_i4](#) (cd_name, id_value, id_type)
- type(tatt) function [att__init_i4_0d](#) (cd_name, id_value, id_type)

This function initialize an attribute structure with integer(4) value.
- type(tatt) function [att__init_i8](#) (cd_name, kd_value, id_type)
- type(tatt) function [att__init_i8_0d](#) (cd_name, kd_value, id_type)

11.4.1 Member Function/Subroutine Documentation

11.4.1.1 type(tatt) function att::att_init::att__init_c (character(len=*), intent(in) *cd_name*, character(len=*), intent(in) *cd_value*)

Parameters

in	<i>cd_name</i>	attribute name
in	<i>cd_value</i>	attribute value

Returns

attribute structure

11.4.1.2 type(tatt) function att::att_init::att__init_dp (character(len=*), intent(in) *cd_name*, real(dp), dimension(:), intent(in) *dd_value*, integer(i4), intent(in), optional *id_type*)

Parameters

in	<i>cd_name</i>	attribute name
in	<i>dd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.3 type(tatt) function att::att_init::att__init_dp_0d (character(len=*), intent(in) *cd_name*, real(dp), intent(in) *dd_value*, integer(i4), intent(in), optional *id_type*)

Parameters

in	<i>cd_name</i>	attribute name
in	<i>dd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.4 `type(tatt) function att::att_init::att_init_i1 (character(len=*), intent(in) cd_name, integer(i1), dimension(:), intent(in) bd_value, integer(i4), intent(in), optional id_type)`

Parameters

in	<i>cd_name</i>	attribute name
in	<i>bd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.5 `type(tatt) function att::att_init::att_init_i1_0d (character(len=*), intent(in) cd_name, integer(i1), intent(in) bd_value, integer(i4), intent(in), optional id_type)`

Parameters

in	<i>cd_name</i>	attribute name
in	<i>bd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.6 `type(tatt) function att::att_init::att_init_i2 (character(len=*), intent(in) cd_name, integer(i2), dimension(:), intent(in) sd_value, integer(i4), intent(in), optional id_type)`

Parameters

in	<i>cd_name</i>	attribute name
in	<i>sd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.7 `type(tatt) function att::att_init::att_init_i2_0d (character(len=*), intent(in) cd_name, integer(i2), intent(in) sd_value, integer(i4), intent(in), optional id_type)`

Parameters

in	<i>cd_name</i>	attribute name
in	<i>sd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.8 `type(tatt) function att::att_init::att_init_i4 (character(len=*), intent(in) cd_name, integer(i4), dimension(:), intent(in) id_value, integer(i4), intent(in), optional id_type)`

Parameters

in	<i>cd_name</i>	attribute name
in	<i>id_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.9 `type(tatt) function att::att_init::att_init_i4_0d (character(len=*), intent(in) cd_name, integer(i4), intent(in) id_value, integer(i4), intent(in), optional id_type)`

This function initialize an attribute structure with integer(4) value.

Parameters

in	<i>cd_name</i>	
----	----------------	--

Optionaly you could specify the type of the variable to be saved.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>cd_name</i>	attribute name
in	<i>id_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.10 `type(tatt) function att::att_init::att_init_i8 (character(len=*), intent(in) cd_name, integer(i8), dimension(:), intent(in) kd_value, integer(i4), intent(in), optional id_type)`

Parameters

in	<i>cd_name</i>	attribute name
in	<i>kd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.11 **type(tatt) function att::att_init::att__init_i8_0d** (character(len=*), intent(in) *cd_name*, integer(i8), intent(in) *kd_value*, integer(i4), intent(in), optional *id_type*)

Parameters

in	<i>cd_name</i>	attribute name
in	<i>kd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.12 **type(tatt) function att::att_init::att__init_sp** (character(len=*), intent(in) *cd_name*, real(sp), dimension(:), intent(in) *rd_value*, integer(i4), intent(in), optional *id_type*)

Parameters

in	<i>cd_name</i>	attribute name
in	<i>rd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

11.4.1.13 **type(tatt) function att::att_init::att__init_sp_0d** (character(len=*), intent(in) *cd_name*, real(sp), intent(in) *rd_value*, integer(i4), intent(in), optional *id_type*)

Parameters

in	<i>cd_name</i>	attribute name
in	<i>rd_value</i>	attribute value
in	<i>id_type</i>	type of the variable to be saved

Returns

attribute structure

The documentation for this interface was generated from the following file:

- src/attribute.f90

11.5 att::att_print Interface Reference

Public Member Functions

- subroutine [att__print_unit](#) (td_att)
- subroutine [att__print_arr](#) (td_att)

This subroutine print informations of an array of attribute.

11.5.1 Member Function/Subroutine Documentation

11.5.1.1 subroutine att::att_print::att__print_arr (type(tatt), dimension(:), intent(in) td_att)

This subroutine print informations of an array of attribute.

Parameters

in	td_att
----	--------

Author

J.Paul

Date

June, 2014 - Initial Version

Parameters

in	td_att	array of attribute structure
----	--------	------------------------------

11.5.1.2 subroutine att::att_print::att__print_unit (type(tatt), intent(in) td_att)

Parameters

in	td_att	attribute structure
----	--------	---------------------

The documentation for this interface was generated from the following file:

- src/attribute.f90

11.6 boundary Module Reference

This module manage boundary.

Data Types

- interface [boundary_clean](#)
- interface [boundary_copy](#)
- interface [boundary_init](#)
- interface [boundary_print](#)
- interface [seg__clean](#)
- interface [seg__copy](#)
- type [tbody](#)
boundary structure
- type [tseg](#)

Public Member Functions

- character(len=lc) function, public [boundary_set_filename](#) (cd_file, cd_card, id_seg, cd_date)
- subroutine, public [boundary_get_indices](#) (td_bdy, td_var, ld_oneseg)
- subroutine, public [boundary_check_corner](#) (td_bdy, td_var)
- subroutine, public [boundary_check](#) (td_bdy, td_var)
- subroutine, public [boundary_swap](#) (td_var, td_bdy)

11.6.1 Detailed Description

This module manage boundary.

```
define type TBDY:<br/>
```

```
TYPE(tbdy) :: tl_bdy<br/>
```

to initialise boundary structure:

```
tl_bdy=boundary_init(td_var, [ld_north,] [ld_south,] [ld_east,] [ld_west,]
[cd_north,] [cd_south,] [cd_east,] [cd_west,] [ld_oneseg])
```

- td_var is variable structure
- ld_north is logical to force used of north boundary [optional]
- ld_south is logical to force used of north boundary [optional]
- ld_east is logical to force used of north boundary [optional]
- ld_west is logical to force used of north boundary [optional]
- cd_north is string character description of north boundary [optional]
- cd_south is string character description of south boundary [optional]
- cd_east is string character description of east boundary [optional]
- cd_west is string character description of west boundary [optional]
- ld_oneseg is logical to force to use only one segment for each boundary [optional]

to get boundary cardinal:

- tl_bdy%c_card

to know if boundary is use:

- tl_bdy%l_use

to know if boundary come from namelist (cn_north,..):

- tl_bdy%l_nam

to get the number of segment in boundary:

- tl_bdy%i_nseg

to get array of segment in boundary:

- `tl_bdy%t_seg(:)`

to get orthogonal segment index of north boundary:

- `tl_bdy%t_seg(jp_north)%`

to get segment width of south boundary:

- `tl_bdy%t_seg(jp_south)%`

to get segment first indice of east boundary:

- `tl_bdy%t_seg(jp_east)%`

to get segment last indice of west boundary:

- `tl_bdy%t_seg(jp_west)%`

to print information about boundary:

```
CALL boundary_print (td_bdy)
```

- `td_bdy` is boundary structure or a array of boundary structure

to clean boundary structure:

```
CALL boundary_clean (td_bdy)
```

to get indices of each semgent for each boundary:

```
CALL boundary_get_indices( td_bdy, td_var, ld_oneseg)
```

- `td_bdy` is boundary structure
- `td_var` is variable structure
- `ld_oneseg` is logical to force to use only one segment for each boundary [optional]

to check boundary indices and corner:

```
CALL boundary_check (td_bdy, td_var)
```

- `td_bdy` is boundary structure
- `td_var` is variable structure

to check boundary corner:

```
CALL boundary_check_corner (td_bdy, td_var)
```

- `td_bdy` is boundary structure
- `td_var` is variable structure

to create filename with cardinal name inside:

```
cl_filename=boundary_set_filename(cd_file, cd_card)
```

- `cd_file` = original file name
- `cd_card` = cardinal name

to swap array for east and north boundary:

```
CALL boundary_swap( td_var, td_bdy )
```

- `td_var` is variable structre
- `td_bdy` is boundary structre

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add boundary description

November, 2014

- Fix memory leaks bug

February, 2015

- Do not change indices read from namelist
- Change string character format of boundary read from namelist, see `boundary__get_info`

Todo add schematic to boundary structure description

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.6.2 Member Function/Subroutine Documentation

11.6.2.1 subroutine, public `boundary::boundary_check (type(tbdy), dimension(ip_ncard), intent(inout) td_bdy, type(tvar), intent(in) td_var)`

Parameters

<code>in, out</code>	<code><i>td_bdy</i></code>
----------------------	----------------------------

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2016

- Bug fix: take into account that boundaries are compute on T point, but expressed on U,V point

Parameters

in, out	<i>td_bdy</i>	boundary structure
in	<i>td_var</i>	variable structure

11.6.2.2 subroutine, public boundary::boundary_check_corner (type(tbdy), dimension(ip_ncard), intent(inout) *td_bdy*, type(tvar), intent(in) *td_var*)

Parameters

in, out	<i>td_bdy</i>	boundary structure
in	<i>td_var</i>	variable structure

11.6.2.3 subroutine, public boundary::boundary_get_indices (type(tbdy), dimension(ip_ncard), intent(inout) *td_bdy*, type(tvar), intent(in) *td_var*, logical, intent(in), optional *ld_onseg*)

Parameters

in, out	<i>td_bdy</i>	boundary structure
in	<i>td_var</i>	variable structure
in	<i>ld_onseg</i>	use only one sgment for each boundary

11.6.2.4 character(len=lc) function, public boundary::boundary_set_filename (character(len=*), intent(in) *cd_file*, character(len=*), intent(in) *cd_card*, integer(i4), intent(in), optional *id_seg*, character(len=*), intent(in), optional *cd_date*)

Parameters

in	<i>cd_file</i>	file name
in	<i>cd_card</i>	cardinal name
in	<i>id_seg</i>	segment number
in	<i>cd_date</i>	file date (format: y????m??d??)

Returns

file name with cardinal name inside

11.6.2.5 subroutine, public boundary::boundary_swap (type(tvar), intent(inout) *td_var*, type(tbdy), intent(in) *td_bdy*)

Parameters

in, out	<i>td_var</i>	variable strucutre
in	<i>td_bdy</i>	boundary strucutre

The documentation for this module was generated from the following file:

- src/boundary.f90

11.7 boundary::boundary_clean Interface Reference

Public Member Functions

- subroutine [boundary_clean_unit](#) (td_bdy)
- subroutine [boundary_clean_arr](#) (td_bdy)

11.7.1 Member Function/Subroutine Documentation

11.7.1.1 subroutine `boundary::boundary_clean::boundary__clean_arr` (`type(tbdy)`, `dimension(:)`, `intent(inout) td_bdy`)

Parameters

<code>in, out</code>	<code>td_bdy</code>	boundary strucutre
----------------------	---------------------	--------------------

11.7.1.2 subroutine `boundary::boundary_clean::boundary__clean_unit` (`type(tbdy)`, `intent(inout) td_bdy`)

Parameters

<code>in, out</code>	<code>td_bdy</code>	boundary strucutre
----------------------	---------------------	--------------------

The documentation for this interface was generated from the following file:

- `src/boundary.f90`

11.8 `boundary::boundary_copy` Interface Reference

Public Member Functions

- `type(tbdy)` function `boundary__copy_unit` (`td_bdy`)
- `type(tbdy)` function, `dimension(size(td_bdy(:)))` `boundary__copy_arr` (`td_bdy`)

11.8.1 Member Function/Subroutine Documentation

11.8.1.1 `type(tbdy)` function, `dimension(size(td_bdy(:)))` `boundary::boundary_copy::boundary__copy_arr` (`type(tbdy)`, `dimension(:)`, `intent(in) td_bdy`)

Parameters

<code>in</code>	<code>td_bdy</code>	array of boundary structure
-----------------	---------------------	-----------------------------

Returns

copy of input array of boundary structure

11.8.1.2 `type(tbdy)` function `boundary::boundary_copy::boundary__copy_unit` (`type(tbdy)`, `intent(in) td_bdy`)

Parameters

<code>in</code>	<code>td_bdy</code>	boundary structure
-----------------	---------------------	--------------------

Returns

copy of input boundary structure

The documentation for this interface was generated from the following file:

- `src/boundary.f90`

11.9 boundary::boundary_init Interface Reference

Public Member Functions

- type([tbody](#)) function, dimension(ip_ncard) [boundary__init_wrapper](#) (td_var, ld_north, ld_south, ld_east, ld_west, cd_north, cd_south, cd_east, cd_west, ld_oneseg)

11.9.1 Member Function/Subroutine Documentation

11.9.1.1 type([tbody](#)) function, dimension(ip_ncard) [boundary::boundary_init::boundary__init_wrapper](#) (type([tvar](#)), intent(in) *td_var*, logical, intent(in), optional *ld_north*, logical, intent(in), optional *ld_south*, logical, intent(in), optional *ld_east*, logical, intent(in), optional *ld_west*, character(len=lc), intent(in), optional *cd_north*, character(len=lc), intent(in), optional *cd_south*, character(len=lc), intent(in), optional *cd_east*, character(len=lc), intent(in), optional *cd_west*, logical, intent(in), optional *ld_oneseg*)

Parameters

in	<i>td_var</i>	variable structure
in	<i>ld_north</i>	use north boundary or not
in	<i>ld_south</i>	use south boundary or not
in	<i>ld_east</i>	use east boundary or not
in	<i>ld_west</i>	use west boundary or not
in	<i>cd_north</i>	north boundary description
in	<i>cd_south</i>	south boundary description
in	<i>cd_east</i>	east boundary description
in	<i>cd_west</i>	west boundary description
in	<i>ld_oneseg</i>	force to use only one segment for each boundary

Returns

boundary structure

The documentation for this interface was generated from the following file:

- src/boundary.f90

11.10 boundary::boundary_print Interface Reference

Public Member Functions

- subroutine [boundary__print_unit](#) (td_bdy)
- subroutine [boundary__print_arr](#) (td_bdy)

11.10.1 Member Function/Subroutine Documentation

11.10.1.1 subroutine [boundary::boundary_print::boundary__print_arr](#) (type([tbody](#)), dimension(:), intent(in) *td_bdy*)

Parameters

in	<i>td_bdy</i>	boundary structure
----	---------------	--------------------

11.10.1.2 subroutine `boundary::boundary_print::boundary__print_unit` (`type(tbdy)`, `intent(in) td_bdy`)

Parameters

<code>in</code>	<code>td_bdy</code>	boundary structure
-----------------	---------------------	--------------------

The documentation for this interface was generated from the following file:

- `src/boundary.f90`

11.11 date Module Reference

This module provide the calculation of Julian dates, and do many manipulations with dates.

Data Types

- interface `date_init`
- interface `operator(+)`
- interface `operator(-)`
- type `tdate`

Public Member Functions

- character(`len=lc`) function, public `date_print` (`td_date`, `cd_fmt`)
This function print the date and time with format YYYY/MM/DD hh:mm:ss.
- logical function, public `date_leapyear` (`td_date`)
This function check if year is a leap year.
- type(`tdate`) function, public `date_now` ()
- subroutine, public `date_time` ()
- type(`tdate`) function, public `date_today` ()

11.11.1 Detailed Description

This module provide the calculation of Julian dates, and do many manipulations with dates.

Actually we use Modified Julian Dates, with 17 Nov 1858 at 00:00:00 as origin.

define type TDATE:

```
TYPE(tdate) :: t1_date1
```

default date is 17 Nov 1858 at 00:00:00

to intialise date :

- from date of the day at 12:00:00 :

```
t1_date1=date_today()
```

- from date and time of the day :

```
t1_date1=date_now()
```

- from julian day :

```
t1_date1=date_init(dd_jd)
```

- dd_jd julian day (double precision)

- from number of second since julian day origin :

```
t1_date1=date_init(kd_nsec)
```

- kd_nsec number of second (integer 8)

- from year month day :

```
t1_date1=date_init(2012,12,10)
```

- from string character formatted date :

```
t1_date1=date_init(cd_fmtdate)
```

- cd_fmtdate date in format YYYY-MM-DD hh:mm:ss

to print date in format YYYY-MM-DD hh:mm:ss

CHARACTER(LEN=lc) :: cl_date

```
cl_date=date_print(t1_date1)
print *, trim(cl_date)
```

to print date in another format (only year, month, day):

```
cl_date=date_print(t1_date1, cd_fmt)
print *, trim(cl_date)
```

- cd_fmt output format (ex: cd_fmt="(y',i0.4,'m',i0.2,'d',i0.2)")

to print day of the week:

```
print *, "dow ", t1_date1\%i_dow
```

to print last day of the month:

```
print *, "last day ", t1_date1\%i_lday
```

to know if year is a leap year:

```
ll_isleap=date_leapyear(t1_date1)
```

- ll_isleap is logical

to compute number of days between two dates:

```
t1_date2=date_init(2010,12,10)
dl_diff=t1_date1-t1_date2
```

- dl_diff is the number of days between date1 and date2 (double precision)

to add or subtract nday to a date:

```
t1_date2=t1_date1+2.
t1_date2=t1_date1-2.6
```

- number of day (double precision)

to print julian day:

```
print *, " julian day",t1_date1\%r_jd
```

to print CNES julian day (origin 1950-01-01 00:00:00)

```
print *, " CNES julian day",t1_date1\%r_jc
```

to create pseudo julian day with origin date_now:

```
t1_date1=date_init(2012,12,10,td_dateo=date_now())
```

Note

you erase CNES julian day when doing so

to print julian day in seconds:

```
print *, t1_date1\%k_jdsec
```

to print CNES or new julian day in seconds:

```
print *, t1_date1\%k_jcsec
```

Author

J.Paul

Date

November, 2013 - Initial Version

Note

This module is based on Perderabo's date calculator (ksh)
Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

Todo • see calendar.f90 and select Gregorian, NoLeap, or D360 calendar

11.11.2 Member Function/Subroutine Documentation

11.11.2.1 logical function, public date::date_leapyear (type(tdate), intent(in) *td_date*)

This function check if year is a leap year.

Parameters

<i>in</i>	<i>td_date</i>
-----------	----------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_date</i>	date structcutre
-----------	----------------	------------------

Returns

true if year is leap year

11.11.2.2 type(tdate) function, public date::date_now ()

This function return the current date and time.

Author

J.Paul

Date

November, 2013 - Initial Version

Returns

current date and time in a date structure

11.11.2.3 character(len=lc) function, public date::date_print (type(tdate), intent(in) *td_date*, character(len=*), intent(in), optional *cd_fmt*)

This function print the date and time with format YYYY/MM/DD hh:mm:ss.

Parameters

<i>in</i>	<i>td_date</i>
-----------	----------------

Optionally, you could specify output format. However it will be only apply to year, month, day.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_date</i>	date structutre
in	<i>cd_fmt</i>	ouput format (only for year,month,day)

Returns

date in format YYYY-MM-DD hh:mm:ss

11.11.2.4 subroutine, public date::date_time ()

This subroutine print the current date and time in milliseconds.

Author

J.Paul

Date

August, 2017 - Initial Version

11.11.2.5 type(tdate) function, public date::date_today ()

This function return the date of the day at 12:00:00.

Author

J.Paul

Date

November, 2013 - Initial Version

Returns

date of the day at 12:00:00 in a date structure

The documentation for this module was generated from the following file:

- src/date.f90

11.12 date::date_init Interface Reference**Public Member Functions**

- type(tdate) function [date__init_jd](#) (dd_jd, td_dateo)
This function initialized date structure from julian day.
- type(tdate) function [date__init_nsec](#) (kd_nsec, td_dateo)
This function initialized date structure from number of second since julian day origin.
- type(tdate) function [date__init_ymd](#) (id_year, id_month, id_day, id_hour, id_min, id_sec, td_dateo)
This function initialized date structure form year month day and optionnaly hour min sec.
- type(tdate) function [date__init_fmtdate](#) (cd_datetime, td_dateo)
This function initialized date structure from a character date with format YYYY-MM-DD hh:mm:ss.

11.12.1 Member Function/Subroutine Documentation

11.12.1.1 `type(tdate) function date::date_init::date__init_fmtdate (character(len=*), intent(in) cd_datetime, type(tdate), intent(in), optional td_dateo)`

This function initialized date structure from a character date with format YYYY-MM-DD hh:mm:ss.

Parameters

<i>in</i>	<i>cd_datetime</i>
-----------	--------------------

Optionally create pseudo julian day with new origin.

julian day origin is 17 Nov 1858 at 00:00:00

Author

J.Paul

Date

November, 2013 - Initial Version

April, 2019

- check time units CF convention, raise error if not

Parameters

<i>in</i>	<i>cd_date</i>	date in format YYYY-MM-DD hh:mm:ss
<i>in</i>	<i>td_dateo</i>	new date origin for pseudo julian day

Returns

date structure

11.12.1.2 `type(tdate) function date::date_init::date__init_jd (real(dp), intent(in) dd_jd, type(tdate), intent(in), optional td_dateo)`

This function initialized date structure from julian day.

Parameters

<i>in</i>	<i>dd_jd</i>
-----------	--------------

Optionally create pseudo julian day with new origin.

julian day origin is 17 Nov 1858 at 00:00:00

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>dd_jd</i>	julian day
<i>in</i>	<i>td_dateo</i>	new date origin for pseudo julian day

Returns

date structure of julian day

11.12.1.3 `type(tdate) function date::date_init::date__init_nsec (integer(i8), intent(in) kd_nsec, type(tdate), intent(in), optional td_dateo)`

This function initialized date structure from number of second since julian day origin.

Parameters

<i>in</i>	<i>kd_nsec</i>
-----------	----------------

Optionally create pseudo julian day with new origin.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>kd_nsec</i>	number of second since julian day origin
<i>in</i>	<i>td_dateo</i>	new date origin for pseudo julian day

Returns

date structure of julian day

11.12.1.4 `type(tdate) function date::date_init::date__init_ymd (integer(i4), intent(in) id_year, integer(i4), intent(in) id_month, integer(i4), intent(in) id_day, integer(i4), intent(in), optional id_hour, integer(i4), intent(in), optional id_min, integer(i4), intent(in), optional id_sec, type(tdate), intent(in), optional td_dateo)`

This function initialized date structure form year month day and optionnaly hour min sec.

Parameters

<i>in</i>	<i>id_year</i>
-----------	----------------

Optionally create pseudo julian day with new origin.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>id_year</i>	
<i>in</i>	<i>id_month</i>	
<i>in</i>	<i>id_day</i>	
<i>in</i>	<i>id_hour</i>	
<i>in</i>	<i>id_min</i>	
<i>in</i>	<i>id_sec</i>	
<i>in</i>	<i>td_dateo</i>	new date origin for pseudo julian day

Returns

date structure of year month day

The documentation for this interface was generated from the following file:

- src/date.f90

11.13 dim Module Reference

This module manage dimension and how to change order of those dimension.

Data Types

- interface [dim_clean](#)
- interface [dim_copy](#)
- interface [dim_print](#)
- interface [dim_reorder_2xyzt](#)
- interface [dim_reorder_xyzt2](#)
- interface [dim_reshape_2xyzt](#)
- interface [dim_reshape_xyzt2](#)
- type [tdim](#)

Public Member Functions

- integer(i4) function, public [dim_get_index](#) (td_dim, cd_name, cd_sname)
This function returns dimension index, given dimension name or short name.
- integer(i4) function, public [dim_get_id](#) (td_dim, cd_name, cd_sname)
- type([tdim](#)) function, public [dim_init](#) (cd_name, id_len, ld_uld, cd_sname, ld_use)
This function initialize a dimension structure with given name.
- type([tdim](#)) function, dimension(ip_maxdim), public [dim_fill_unused](#) (td_dim)
This function fill unused dimension of an array of dimension and return a 4 elts array of dimension structure.
- subroutine, public [dim_reorder](#) (td_dim, cd_dimorder)
- subroutine, public [dim_disorder](#) (td_dim)
- subroutine, public [dim_get_dummy](#) (cd_dummy)
- logical function, public [dim_is_dummy](#) (td_dim)
This function check if dimension is defined as dummy dimension in configuraton file.
- subroutine, public [dim_def_extra](#) (cd_file)

11.13.1 Detailed Description

This module manage dimension and how to change order of those dimension.

```
define type TDIM:<br/>
```

```
TYPE(tdim) :: t1_dim
```

to initialize a dimension structure:

```
t1_dim=dim_init( cd_name, [id_len,] [ld_uld,] [cd_sname])
```

- cd_name is the dimension name
- id_len is the dimension size [optional]
- ld_uld is true if this dimension is the unlimited one [optional]
- cd_sname is the dimension short name ('x','y','z','t') [optional]

to clean dimension structure:

```
CALL dim_clean(tl_dim)
```

- `tl_dim` : dimension structure or array of dimension structure

to print information about dimension structure:

```
CALL dim_print(tl_dim)
```

to copy dimension structure in another one (using different memory cell):

```
tl_dim2=dim_copy(tl_dim1)
```

to get dimension name:

- `tl_dim%c_name`

to get dimension short name:

- `tl_dim%c_sname`

to get dimension length:

- `tl_dim%i_len`

to know if dimension is the unlimited one:

- `tl_dim%i_uld`

to get dimension id (for variable or file dimension):

- `tl_dim%i_id`

to know if dimension is used (for variable or file dimension):

- `tl_dim%i_use`

Former function or information concern only one dimension. However variables as well as files use usually 4 dimensions.

To easily work with variable we want they will be all 4D and ordered as following: ('x','y','z','t').

Functions and subroutines below, allow to reorder dimension of variable.

Suppose we defined the array of dimension structure below:

```
TYPE(tdim), DIMENSION(4) :: tl_dim
tl_dim(1)=dim_init('X', id_len=10)
tl_dim(2)=dim_init('T', id_len=3, ld_uld=.true.)
```

to reorder dimension (default order: ('x','y','z','t')):

```
CALL dim_reorder(tl_dim(:))
```

This subroutine filled dimension structure with unused dimension, then switch from "disordered" dimension to "ordered" dimension.

The dimension structure return will be:

```
tl_dim(1) => 'X', i_len=10, l_use=T, l_uld=F
```

```
tl_dim(2) => 'Y', i_len=1, l_use=F, l_uld=F
```

```
tl_dim(3) => 'Z', i_len=1, l_use=F, l_uld=F
```

```
tl_dim(4) => 'T', i_len=3, l_use=T, l_uld=T
```

After using subroutine `dim_reorder` you could use functions and subroutine below.

to use another dimension order.

```
CALL dim_reorder(tl_dim(:), cl_neworder)
```

- `cl_neworder` : character(len=4) (example: 'yxzt')

to switch dimension array from ordered dimension to disordered dimension:

```
CALL dim_disorder(tl_dim(:))
```

to fill unused dimension of an array of dimension structure.

```
tl_dimout(:)=dim_fill_unused(tl_dimin(:))
```

- `tl_dimout(:)` : 1D array (4elts) of dimension structure
- `tl_dimin(:)` : 1D array (<=4elts) of dimension structure

to reshape array of value in "ordered" dimension:

```
CALL dim_reshape_2xyzt(tl_dim(:), value(:, :, :, :))
```

- `value` must be a 4D array of real(8) value "disordered"

to reshape array of value in "disordered" dimension:

```
CALL dim_reshape_xyzt2(tl_dim(:), value(:, :, :, :))
```

- `value` must be a 4D array of real(8) value "ordered"

to reorder a 1D array of 4 elements in "ordered" dimension:

```
CALL dim_reorder_2xyzt(tl_dim(:), tab(:))
```

- `tab` must be a 1D array with 4 elements "disordered". It could be composed of character, integer(4), or logical

to reorder a 1D array of 4 elements in "disordered" dimension:

```
CALL dim_reorder_xyzt2(tl_dim(:), tab(:))
```

- `tab` must be a 1D array with 4 elements "ordered". It could be composed of character, integer(4), or logical

to get dimension index from a array of dimension structure, given dimension name or short name :

```
index=dim_get_index( tl_dim(:), [cl_name, cl_sname] )
```

- `tl_dim(:)` : array of dimension structure
- `cl_name` : dimension name [optional]
- `cl_sname`: dimension short name [optional]

to get dimension id used in an array of dimension structure, given dimension name or short name :

```
id=dim_get_id( tl_dim(:), [cl_name, cl_sname] )
```

- `tl_dim(:)` : array of dimension structure
- `cl_name` : dimension name [optional]
- `cl_sname`: dimension short name [optional]

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2015

- manage useless (dummy) dimension

October, 2016

- dimension allowed read in configuration file

May, 2019

- read number of element for each dimension allowed in configuration file
- read number of element for each dummy array in configuration file

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.13.2 Member Function/Subroutine Documentation

11.13.2.1 subroutine, public `dim::dim_def_extra (character(len=*), intent(in) cd_file)`

Parameters

<code>in</code>	<code>cd_file</code>	input file (dimension configuration file)
-----------------	----------------------	---

11.13.2.2 subroutine, public `dim::dim_disorder (type(tdim), dimension(:), intent(inout) td_dim)`

Parameters

<code>in, out</code>	<code>td_dim</code>	array of dimension structure
----------------------	---------------------	------------------------------

11.13.2.3 `type(tdim)` function, `dimension(ip_maxdim)`, public `dim::dim_fill_unused (type(tdim), dimension(:), intent(in), optional td_dim)`

This function fill unused dimension of an array of dimension and return a 4 elts array of dimension structure.

Parameters

<i>in</i>	<i>td_dim</i>
-----------	---------------

output dimensions 'x','y','z' and 't' are all informed.

Note

without input array of dimension, return a 4 elts array of dimension structure all unused (case variable 0d)

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- Bug fix: use order to disorder table (see `dim_init`)

Parameters

<i>in</i>	<i>td_dim</i>	array of dimension structure
-----------	---------------	------------------------------

Returns

4elts array of dimension structure

11.13.2.4 subroutine, public `dim::dim_get_dummy (character(len=*), intent(in) cd_dummy)`

Parameters

<i>in</i>	<i>cd_dummy</i>
-----------	-----------------

Author

J.Paul

Date

September, 2015 - Initial Version

May, 2019

- read number of dummy element

Parameters

<i>in</i>	<i>cd_dummy</i>	dummy configuration file
-----------	-----------------	--------------------------

11.13.2.5 integer(i4) function, public `dim::dim_get_id (type(tdim), dimension(:), intent(in) td_dim, character(len=*), intent(in) cd_name, character(len=*), intent(in), optional cd_sname)`

Parameters

in	<i>td_dim</i>	dimension structure
in	<i>cd_name</i>	dimension name or short name
in	<i>cd_sname</i>	dimension short name

Returns

dimension id

11.13.2.6 `integer(i4)` function, public `dim::dim_get_index (type(tdim), dimension(:), intent(in) td_dim, character(len=*), intent(in) cd_name, character(len=*), intent(in), optional cd_sname)`

This function returns dimension index, given dimension name or short name.

Parameters

in	<i>td_dim</i>
----	---------------

the function check dimension name, in the array of dimension structure. dimension could be used or not.

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- do not check if dimension used

Parameters

in	<i>td_dim</i>	array of dimension structure
in	<i>cd_name</i>	dimension name
in	<i>cd_sname</i>	dimension short name

Returns

dimension index

11.13.2.7 `type(tdim)` function, public `dim::dim_init (character(len=*), intent(in) cd_name, integer(i4), intent(in), optional id_len, logical, intent(in), optional id_uld, character(len=*), intent(in), optional cd_sname, logical, intent(in), optional id_use)`

This function initialize a dimension structure with given name.

Parameters

in	<i>cd_name</i>
----	----------------

Optionally length could be inform, as well as short name and if dimension is unlimited or not.

By default, define dimension is supposed to be used. Optionally you could force a defined dimension to be unused.

Author

J.Paul

Date

November, 2013 - Initial Version

February, 2015

- add optional argument to define dimension unused

July, 2015

- Bug fix: inform order to disorder table instead of disorder to order table

May, 2019

- use number of element for each dimation allowed, instead of while loop

Parameters

in	<i>cd_name</i>	dimension name
in	<i>id_len</i>	dimension length
in	<i>ld_uld</i>	dimension unlimited
in	<i>cd_sname</i>	dimension short name
in	<i>ld_use</i>	dimension use or not

Returns

dimension structure

11.13.2.8 logical function, public dim::dim_is_dummy (type(tdim), intent(in) td_dim)

This function check if dimension is defined as dummy dimension in configuraton file.

Parameters

in	<i>td_dim</i>	
----	---------------	--

Author

J.Paul

Date

September, 2015 - Initial Version

, May, 2019

- use number of dummy elt in do-loop

Parameters

in	<i>td_dim</i>	dimension structure
----	---------------	---------------------

Returns

true if dimension is dummy dimension

11.13.2.9 subroutine, public dim::dim_reorder (type(tdim), dimension(:), intent(inout) td_dim, character(len=ip_maxdim), intent(in), optional cd_dimorder)

Parameters

in, out	td_dim
---------	--------

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- allow to choose ordered dimension to be output

Parameters

in, out	td_dim	array of dimension structure
in	cd_dimorder	dimension order to be output

The documentation for this module was generated from the following file:

- src/dimension.f90

11.14 dim::dim_clean Interface Reference

Public Member Functions

- subroutine [dim__clean_unit](#) (td_dim)
- subroutine [dim__clean_arr](#) (td_dim)

11.14.1 Member Function/Subroutine Documentation

11.14.1.1 subroutine [dim::dim_clean::dim__clean_arr](#) (type(tdim), dimension(:), intent(inout) td_dim)

Parameters

in	td_dim	array of dimension strucutre
----	--------	------------------------------

11.14.1.2 subroutine [dim::dim_clean::dim__clean_unit](#) (type(tdim), intent(inout) td_dim)

Parameters

in	td_dim	dimension strucutre
----	--------	---------------------

The documentation for this interface was generated from the following file:

- src/dimension.f90

11.15 dim::dim_copy Interface Reference

Public Member Functions

- type(tdim) function [dim__copy_unit](#) (td_dim)
This subroutine copy an dimension structure in another one.
- type(tdim) function, dimension(size(td_dim(:))) [dim__copy_arr](#) (td_dim)

11.15.1 Member Function/Subroutine Documentation

11.15.1.1 `type(tdim) function, dimension(size(td_dim(:))) dim::dim_copy::dim__copy_arr (type(tdim), dimension(:), intent(in) td_dim)`

Parameters

<code>in</code>	<code>td_dim</code>	array of dimension structure
-----------------	---------------------	------------------------------

Returns

copy of input array of dimension structure

11.15.1.2 `type(tdim) function dim::dim_copy::dim__copy_unit (type(tdim), intent(in) td_dim)`

This subroutine copy an dimension structure in another one.

Parameters

<code>in</code>	<code>td_dim</code>	
-----------------	---------------------	--

dummy function to get the same use for all structure

Warning

do not use on the output of a function who create or read an structure (ex: `tl_dim=dim_copy(dim_init())` is forbidden). This will create memory leaks.

to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2014 - Initial Version

Parameters

<code>in</code>	<code>td_dim</code>	dimension structure
-----------------	---------------------	---------------------

Returns

copy of input dimension structure

The documentation for this interface was generated from the following file:

- `src/dimension.f90`

11.16 dim::dim_print Interface Reference

Public Member Functions

- subroutine `dim__print_unit` (td_dim)
- subroutine `dim__print_arr` (td_dim)

11.16.1 Member Function/Subroutine Documentation

11.16.1.1 `subroutine dim::dim_print::dim__print_arr (type(tdim), dimension(:), intent(in) td_dim)`

Parameters

<i>in</i>	<i>td_dim</i>	array of dimension structure
-----------	---------------	------------------------------

11.16.1.2 subroutine dim::dim_print::dim__print_unit (type(tdim), intent(in) *td_dim*)

Parameters

<i>in</i>	<i>td_dim</i>	dimension structure
-----------	---------------	---------------------

The documentation for this interface was generated from the following file:

- src/dimension.f90

11.17 dim::dim_reorder_2xyzt Interface Reference

Public Member Functions

- integer(i4) function, dimension(ip_maxdim) [dim__reorder_2xyzt_i4](#) (td_dim, id_arr)
- character(len=lc) function, dimension(ip_maxdim) [dim__reorder_2xyzt_c](#) (td_dim, cd_arr)
- logical function, dimension(ip_maxdim) [dim__reorder_2xyzt_l](#) (td_dim, ld_arr)

11.17.1 Member Function/Subroutine Documentation

11.17.1.1 character(len=lc) function, dimension(ip_maxdim) dim::dim_reorder_2xyzt::dim__reorder_2xyzt_c (type(tdim), dimension(:), intent(in) *td_dim*, character(len=*) , dimension(:), intent(in) *cd_arr*)

Parameters

<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>cd_arr</i>	array of value to reordered

Returns

array of value reordered

11.17.1.2 integer(i4) function, dimension(ip_maxdim) dim::dim_reorder_2xyzt::dim__reorder_2xyzt_i4 (type(tdim), dimension(:), intent(in) *td_dim*, integer(i4), dimension(:), intent(in) *id_arr*)

Parameters

<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>id_arr</i>	array of value to reshape

Returns

array of value reshaped

11.17.1.3 logical function, dimension(ip_maxdim) dim::dim_reorder_2xyzt::dim__reorder_2xyzt_l (type(tdim), dimension(:), intent(in) *td_dim*, logical, dimension(:), intent(in) *ld_arr*)

Parameters

in	<i>td_dim</i>	array of dimension structure
in	<i>ld_arr</i>	array of value to reordered

Returns

array of value reordered

The documentation for this interface was generated from the following file:

- src/dimension.f90

11.18 dim::dim_reorder_xyz2 Interface Reference**Public Member Functions**

- integer(i4) function, dimension(ip_maxdim) [dim__reorder_xyz2_i4](#) (td_dim, id_arr)
- character(len=lc) function, dimension(ip_maxdim) [dim__reorder_xyz2_c](#) (td_dim, cd_arr)
- logical function, dimension(ip_maxdim) [dim__reorder_xyz2_l](#) (td_dim, ld_arr)

11.18.1 Member Function/Subroutine Documentation

11.18.1.1 character(len=lc) function, dimension(ip_maxdim) [dim::dim_reorder_xyz2::dim__reorder_xyz2_c](#) (type(td_dim), dimension(:), intent(in) *td_dim*, character(len=*) , dimension(:), intent(in) *cd_arr*)

Parameters

in	<i>td_dim</i>	array of dimension structure
in	<i>cd_arr</i>	array of value to reordered

Returns

array of value reordered

11.18.1.2 integer(i4) function, dimension(ip_maxdim) [dim::dim_reorder_xyz2::dim__reorder_xyz2_i4](#) (type(td_dim), dimension(:), intent(in) *td_dim*, integer(i4), dimension(:), intent(in) *id_arr*)

Parameters

in	<i>td_dim</i>	array of dimension structure
in	<i>id_arr</i>	array of value to reshape

Returns

array of value reshaped

11.18.1.3 logical function, dimension(ip_maxdim) [dim::dim_reorder_xyz2::dim__reorder_xyz2_l](#) (type(td_dim), dimension(:), intent(in) *td_dim*, logical, dimension(:), intent(in) *ld_arr*)

Parameters

<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>ld_arr</i>	array of value to reordered

Returns

array of value reordered

The documentation for this interface was generated from the following file:

- src/dimension.f90

11.19 dim::dim_reshape_2xyzt Interface Reference

Public Member Functions

- `real(dp) function, dimension(td_dim(1)%i_len,td_dim(2)%i_len,td_dim(3)%i_len,td_dim(4)%i_len) dim__reshape_2xyzt_dp (td_dim, dd_value)`

11.19.1 Member Function/Subroutine Documentation

11.19.1.1 `real(dp) function, dimension(td_dim(1)%i_len, td_dim(2)%i_len, td_dim(3)%i_len, td_dim(4)%i_len) dim::dim_reshape_2xyzt::dim__reshape_2xyzt_dp (type(tdim), dimension(:), intent(in) td_dim, real(dp), dimension(:,:,:), intent(in) dd_value)`

Parameters

<i>in</i>	<i>td_dim</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version
January, 2019

- do not reshape array already order

Parameters

<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>dd_value</i>	array of value to reshape

Returns

array of value reshaped

The documentation for this interface was generated from the following file:

- src/dimension.f90

11.20 dim::dim_reshape_xyz2 Interface Reference

Public Member Functions

- `real(dp)` function, `dimension(td_dim(td_dim(1)%i_xyz2)%i_len,td_dim(td_dim(2)%i_xyz2)%i_len,td_dim(td_dim(3)%i_xyz2)%i_len,td_dim(td_dim(4)%i_xyz2)%i_len)` [dim__reshape_xyz2_dp](#) (`td_dim`, `dd_value`)

11.20.1 Member Function/Subroutine Documentation

- 11.20.1.1 `real(dp)` function, `dimension(td_dim(td_dim(1)%i_xyz2)%i_len, td_dim(td_dim(2)%i_xyz2)%i_len, td_dim(td_dim(3)%i_xyz2)%i_len, td_dim(td_dim(4)%i_xyz2)%i_len)` `dim::dim_reshape_xyz2::dim__reshape_xyz2_dp` (`type(tdim)`, `dimension(:)`, `intent(in) td_dim`, `real(dp)`, `dimension(:, :, :, :)`, `intent(in) dd_value`)

Parameters

<code>in</code>	<code>td_dim</code>	array of dimension structure
<code>in</code>	<code>dd_value</code>	array of value to reshape

Returns

array of value reshaped

The documentation for this interface was generated from the following file:

- `src/dimension.f90`

11.21 dom Module Reference

This module manage domain computation.

Data Types

- interface [dom_copy](#)
- interface [dom_init](#)
- type [tdom](#)

Public Member Functions

- subroutine, public [dom_print](#) (`td_dom`)
This subroutine print some information about domain strucutre.
- `type(tdom)` function [dom__init_file](#) (`td_file`, `id_imin`, `id_imax`, `id_jmin`, `id_jmax`, `cd_card`)
This function intialise domain structure, given open file structure, and sub domain indices.
- subroutine, public [dom_add_extra](#) (`td_dom`, `id_iext`, `id_jext`)
This subroutine add extra bands to coarse domain to get enough point for interpolation...
- subroutine, public [dom_clean_extra](#) (`td_dom`)
This subroutine clean coarse grid domain structure. it remove extra point added.
- subroutine, public [dom_del_extra](#) (`td_var`, `td_dom`, `id_rho`, `ld_coord`)
This subroutine delete extra band, from fine grid variable value, and dimension, taking into account refinement factor.
- subroutine, public [dom_clean](#) (`td_dom`)
This subroutine clean domain structure.

11.21.1 Detailed Description

This module manage domain computation.

```
define type TDOM:<br/>
```

```
TYPE(tdom) :: tl_dom
```

to initialize domain structure:

```
tl_dom=dom_init(td_mpp, [id_imin,] [id_imax,] [id_jmin,] [id_jmax],[cd_card])
```

- `td_mpp` is mpp structure of an opened file.
- `id_imin` is i-direction sub-domain lower left point indice
- `id_imax` is i-direction sub-domain upper right point indice
- `id_jmin` is j-direction sub-domain lower left point indice
- `id_jmax` is j-direction sub-domain upper right point indice
- `cd_card` is the cardinal name (for boundary case)

to get global domain dimension:

- `tl_dom%t_dim0`

to get NEMO periodicity index of global domain:

- `tl_dom%i_perio0`

to get NEMO pivot point index F(0),T(1):

- `tl_dom%i_pivot`

to get East-West overlap of global domain:

- `tl_dom%i_ew0`

to get selected sub domain dimension:

- `tl_dom%t_dim`

to get NEMO periodicity index of sub domain:

- `tl_dom%i_perio`

to get East-West overlap of sub domain:

- `tl_dom%i_ew`

to get i-direction sub-domain lower left point indice:

- `tl_dom%i_imin`

to get i-direction sub-domain upper right point indice:

- `tl_dom%i_imax`

to get j-direction sub-domain lower left point indice:

- `tl_dom%i_jmin`

to get j-direction sub-domain upper right point indice:

- `tl_dom%i_jmax`

to get size of i-direction extra band:

- `tl_dom%i_iextra`

to get size of j-direction extra band:

- `tl_dom%i_jextra`

to get i-direction ghost cell number:

- `tl_dom%i_ighost`

to get j-direction ghost cell number:

- `tl_dom%i_jghost`

to get boundary index:

- `tl_dom%i_bdy`
 - 0 = no boundary
 - 1 = north
 - 2 = south
 - 3 = east
 - 4 = west

to clean domain structure:

```
CALL dom_clean(td_dom)
```

- `td_dom` is domain structure

to print information about domain structure:

```
CALL dom_print(td_dom)
```

to get East-West overlap (if any):

```
il_ew=dom_get_ew_overlap(td_lon)
```

- `td_lon` : longitude variable structure

to add extra bands to coarse grid domain (for interpolation):

```
CALL dom_add_extra( td_dom, id_iext, id_jext )
```

- `td_dom` is domain structure
- `id_iext` is i-direction size of extra bands
- `id_jext` is j-direction size of extra bands

to remove extra bands from fine grid (after interpolation):

```
CALL dom_del_extra( td_var, td_dom, id_rho )
```

- `td_var` is variable structure to be changed
- `td_dom` is domain structure
- `id_rho` is a array of refinement factor following i- and j-direction

to reset coarse grid domain without extra bands:

```
CALL dom_clean_extra( td_dom )
```

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header
- use zero indice to defined cyclic or global domain

October, 2014

- use mpp file structure instead of file

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.21.2 Member Function/Subroutine Documentation

11.21.2.1 `type(tdom) function dom::dom__init_file (type(tfile), intent(in) td_file, integer(i4), intent(in), optional id_imin, integer(i4), intent(in), optional id_imax, integer(i4), intent(in), optional id_jmin, integer(i4), intent(in), optional id_jmax, character(len=*), intent(in), optional cd_card)`

This function initialise domain structure, given open file structure, and sub domain indices.

Parameters

<code>in</code>	<code>td_file</code>
-----------------	----------------------

sub domain indices are computed, taking into account coarse grid periodicity, pivot point, and East-West overlap.

Author

J.Paul

Date

June, 2013 - Initial Version
September, 2014

- add boundary index
- add ghost cell factor

Parameters

in	<i>td_file</i>	file structure
in	<i>id_perio</i>	grid periodicity
in	<i>id_imin</i>	i-direction sub-domain lower left point indice
in	<i>id_imax</i>	i-direction sub-domain upper right point indice
in	<i>id_jmin</i>	j-direction sub-domain lower left point indice
in	<i>id_jmax</i>	j-direction sub-domain upper right point indice
in	<i>cd_card</i>	name of cardinal (for boundary)

Returns

domain structure

11.21.2.2 subroutine, public dom::dom_add_extra (type(tdom), intent(inout) *td_dom*, integer(i4), intent(in), optional *id_iext*, integer(i4), intent(in), optional *id_jext*)

This subroutine add extra bands to coarse domain to get enough point for interpolation...

Parameters

in, out	<i>td_dom</i>
---------	---------------

- domain periodicity is take into account.
- domain indices are changed, and size of extra bands are saved.
- optionally, i- and j- direction size of extra bands could be specify (default=im_minext)

Author

J.Paul

Date

November, 2013 - Initial version
September, 2014

- take into account number of ghost cell

February, 2016

- number of extra point is the MAX (not the MIN) of zero and ases value.

Parameters

<i>in, out</i>	<i>td_dom</i>	domain strcuture
<i>in</i>	<i>id_jext</i>	i-direction size of extra bands (default=im_minext)
<i>in</i>	<i>id_jext</i>	j-direction size of extra bands (default=im_minext)

11.21.2.3 subroutine, public dom::dom_clean (type(tdom), intent(inout) *td_dom*)

This subroutine clean domain structure.

Parameters

<i>in, out</i>	<i>td_dom</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_dom</i>	domain strcuture
----------------	---------------	------------------

11.21.2.4 subroutine, public dom::dom_clean_extra (type(tdom), intent(inout) *td_dom*)

This subroutine clean coarse grid domain structure. it remove extra point added.

Parameters

<i>in, out</i>	<i>td_dom</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_dom</i>	domain strcuture
----------------	---------------	------------------

11.21.2.5 subroutine, public dom::dom_del_extra (type(tvar), intent(inout) *td_var*, type(tdom), intent(in) *td_dom*, integer(i4), dimension(:), intent(in), optional *id_rho*, logical, intent(in), optional *id_coord*)

This subroutine delete extra band, from fine grid variable value, and dimension, taking into account refinement factor.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

Note

This subroutine should be used before clean domain structure.

Warning

if work on coordinates grid, do not remove all extra point. save value on ghost cell.

Author

J.Paul

Date

November, 2013 - Initial version

September, 2014

- take into account boundary for one point size domain

December, 2014

- add special case for coordinates file.

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
<i>in</i>	<i>td_dom</i>	domain structure
<i>in</i>	<i>id_rho</i>	array of refinement factor
<i>in</i>	<i>ld_coord</i>	work on coordinates file or not

11.21.2.6 subroutine, public dom::dom_print (type(tdom), intent(in) *td_dom*)

This subroutine print some information about domain structure.

Parameters

<i>in</i>	<i>td_dom</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_dom</i>	dom structure
----------------	---------------	---------------

The documentation for this module was generated from the following file:

- src/domain.f90

11.22 dom::dom_copy Interface Reference

Public Member Functions

- `type(tdom)` function `dom__copy_unit` (`td_dom`)
This subroutine copy an domain structure in another one.

11.22.1 Member Function/Subroutine Documentation

11.22.1.1 `type(tdom)` function `dom::dom_copy::dom__copy_unit` (`type(tdom)`, `intent(in) td_dom`)

This subroutine copy an domain structure in another one.

Parameters

<code>in</code>	<code>td_dom</code>
-----------------	---------------------

dummy function to get the same use for all structure

Warning

do not use on the output of a function who create or read an structure (ex: `tl_dom=dom_copy(dom_init())` is forbidden). This will create memory leaks.
to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2014 - Initial Version

Parameters

<code>in</code>	<code>td_dom</code>	domain structure
-----------------	---------------------	------------------

Returns

copy of input domain structure

The documentation for this interface was generated from the following file:

- `src/domain.f90`

11.23 dom::dom_init Interface Reference

Public Member Functions

- `type(tdom)` function `dom__init_file` (`td_file`, `id_imin`, `id_imax`, `id_jmin`, `id_jmax`, `cd_card`)
This function intialise domain structure, given open file structure, and sub domain indices.
- `type(tdom)` function `dom__init_mpp` (`td_mpp`, `id_imin`, `id_imax`, `id_jmin`, `id_jmax`, `cd_card`)
This function intialise domain structure, given open file structure, and sub domain indices.

11.23.1 Member Function/Subroutine Documentation

11.23.1.1 `type(tdom) function dom::dom_init::dom__init_file (type(tfile), intent(in) td_file, integer(i4), intent(in), optional id_imin, integer(i4), intent(in), optional id_imax, integer(i4), intent(in), optional id_jmin, integer(i4), intent(in), optional id_jmax, character(len=*), intent(in), optional cd_card)`

This function initialise domain structure, given open file structure, and sub domain indices.

Parameters

<i>in</i>	<i>td_file</i>
-----------	----------------

sub domain indices are computed, taking into account coarse grid periodicity, pivot point, and East-West overlap.

Author

J.Paul

Date

June, 2013 - Initial Version
September, 2014

- add boundary index
- add ghost cell factor

Parameters

<i>in</i>	<i>td_file</i>	file structure
<i>in</i>	<i>id_perio</i>	grid periodicity
<i>in</i>	<i>id_imin</i>	i-direction sub-domain lower left point indice
<i>in</i>	<i>id_imax</i>	i-direction sub-domain upper right point indice
<i>in</i>	<i>id_jmin</i>	j-direction sub-domain lower left point indice
<i>in</i>	<i>id_jmax</i>	j-direction sub-domain upper right point indice
<i>in</i>	<i>cd_card</i>	name of cardinal (for boundary)

Returns

domain structure

11.23.1.2 `type(tdom) function dom::dom_init::dom__init_mpp (type(tmpp), intent(in) td_mpp, integer(i4), intent(in), optional id_imin, integer(i4), intent(in), optional id_imax, integer(i4), intent(in), optional id_jmin, integer(i4), intent(in), optional id_jmax, character(len=*), intent(in), optional cd_card)`

This function initialise domain structure, given open file structure, and sub domain indices.

Parameters

<i>in</i>	<i>td_mpp</i>
-----------	---------------

sub domain indices are computed, taking into account coarse grid periodicity, pivot point, and East-West overlap.

Author

J.Paul

Date

June, 2013 - Initial Version
September, 2014

- add boundary index
- add ghost cell factor

October, 2014

- work on mpp file structure instead of file structure

Parameters

in	<i>td_mpp</i>	mpp structure
in	<i>id_perio</i>	grid periodicity
in	<i>id_imin</i>	i-direction sub-domain lower left point indice
in	<i>id_imax</i>	i-direction sub-domain upper right point indice
in	<i>id_jmin</i>	j-direction sub-domain lower left point indice
in	<i>id_jmax</i>	j-direction sub-domain upper right point indice
in	<i>cd_card</i>	name of cardinal (for boundary)

Returns

domain structure

The documentation for this interface was generated from the following file:

- src/domain.f90

11.24 extrap Module Reference

This module manage extrapolation.

Data Types

- interface [extrap_detect](#)
- interface [extrap_fill_value](#)

Public Member Functions

- subroutine, public [extrap_add_extrabands](#) (td_var, id_ysize, id_jsize)
This subroutine add to the variable (to be extrapolated) an extraband of N points at north,south,east and west boundaries.
- subroutine, public [extrap_del_extrabands](#) (td_var, id_ysize, id_jsize)
This subroutine remove of the variable an extraband of N points at north,south,east and west boundaries.

11.24.1 Detailed Description

This module manage extrapolation.

```
Extrapolation method to be used is specify inside variable
structure, as array of string character.<br/>
- td_var%c_extrap(1) string character is the interpolation name choose between:
  - 'dist_weight'
  - 'min_error'
```

```
@note Extrapolation method could be specify for each variable in namelist _namvar_,
defining string character _cn\_varinfo_. By default _dist_weight_.<br/>
```

Example:

```
- cn_varinfo='varname1:ext=dist_weight', 'varname2:ext=min_error'
```

```
to detect point to be extrapolated:<br/>
```

```
il_detect(:, :, :) = extrap_detect(td_var)
```

- `il_detect(:, :, :)` is 3D array of point to be extrapolated

- `td_var` is coarse grid variable to be extrapolated

to extrapolate variable:

```
CALL extrap_fill_value( td_var, [id_radius])
```

- `td_var` is coarse grid variable to be extrapolated
- `id_radius` is radius of the halo used to compute extrapolation [optional]

to add extraband to the variable (to be extrapolated):

```
CALL extrap_add_extrabands(td_var, [id_isize,] [id_jsize] )
```

- `td_var` is variable structure
- `id_isize` : i-direction size of extra bands [optional]
- `id_jsize` : j-direction size of extra bands [optional]

to delete extraband of a variable:

```
CALL extrap_del_extrabands(td_var, [id_isize,] [id_jsize] )
```

- `td_var` is variable structure
- `id_isize` : i-direction size of extra bands [optional]
- `id_jsize` : j-direction size of extra bands [optional]

Warning

`_FillValue` must not be zero (use `var_chg_FillValue()`)

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header

June, 2015

- extrapolate all land points (`_FillValue`)
- move deriv function to math module

July, 2015

- compute extrapolation from north west to south east, and from south east to north west

Todo

- create module for each extrapolation method
- smooth extrapolated points

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.24.2 Member Function/Subroutine Documentation

11.24.2.1 subroutine, public `extrap::extrap_add_extrabands` (`type(tvar)`, `intent(inout) td_var`, `integer(i4)`, `intent(in)`, optional `id_ysize`, `integer(i4)`, `intent(in)`, optional `id_jsize`)

This subroutine add to the variable (to be extrapolated) an extraband of N points at north,south,east and west boundaries.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

optionally you could specify size of extra bands in i- and j-direction

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_var</i>	variable
<i>in</i>	<i>id_ysize</i>	i-direction size of extra bands (default=im_minext)
<i>in</i>	<i>id_xsize</i>	j-direction size of extra bands (default=im_minext)

Todo • invalid special case for grid with north fold

11.24.2.2 subroutine, public extrap::extrap_del_extrabands (type(tvar), intent(inout) *td_var*, integer(i4), intent(in), optional *id_ysize*, integer(i4), intent(in), optional *id_xsize*)

This subroutine remove of the variable an extraband of N points at north,south,east and west boundaries.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

optionally you could specify size of extra bands in i- and j-direction

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_var</i>	variable
<i>in</i>	<i>id_ysize</i>	i-direction size of extra bands (default=im_minext)
<i>in</i>	<i>id_xsize</i>	j-direction size of extra bands (default=im_minext)

The documentation for this module was generated from the following file:

- src/extrap.f90

11.25 extrap::extrap_detect Interface Reference

Public Member Functions

- integer(i4) function,
dimension(td_var%t_dim(1)%i_len,td_var%t_dim(2)%i_len,td_var%t_dim(3)%i_len) [extrap__detect_wrapper](#)
(td_var)

This function sort variable to be extrapolated, depending on number of dimentsion, then detected point to be extrapolated.

- **detected**
- **point**
- **to**
- **be**
- **extrapolated**

11.25.1 Member Function/Subroutine Documentation

11.25.1.1 integer(i4) function, dimension(td_var%t_dim(1)%i_len, td_var%t_dim(2)%i_len, td_var%t_dim(3)%i_len)
 extrap::extrap_detect::extrap__detect_wrapper (type(tvar), intent(in) td_var)

This function sort variable to be extrapolated, depending on number of dimentsion, then detected point to be extrapolated.

Parameters

in	td_var
----	--------

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- select all land points for extrapolation

Parameters

in	td_var	coarse grid variable to extrapolate
----	--------	-------------------------------------

Returns

3D array of point to be extrapolated

The documentation for this interface was generated from the following files:

- src/extrap.f90

11.26 extrap::extrap_fill_value Interface Reference

Public Member Functions

- subroutine [extrap__fill_value_wrapper](#) (td_var, id_radius)

This subroutine select method to be used for extrapolation. If need be, increase number of points to be extrapolated. Finally launch extrap__fill_value.

- **detected**
- **point**
- **to**
- **be**
- **interpolated**

11.26.1 Member Function/Subroutine Documentation

11.26.1.1 subroutine `extrap::extrap_fill_value::extrap__fill_value_wrapper` (`type(tvar)`, `intent(inout) td_var`, `integer(i4)`, `intent(in)`, optional `id_radius`)

This subroutine select method to be used for extrapolation. If need be, increase number of points to be extrapolated. Finally launch `extrap__fill_value`.

Parameters

<code>in, out</code>	<code>td_var</code>
----------------------	---------------------

optionally, you could specify :

- refinement factor (default 1)
- offset between fine and coarse grid (default compute from refinement factor as $\text{offset}=(\text{rho}-1)/2$)
- number of point to be extrapolated in each direction (default `im_minext`)
- radius of the halo used to compute extrapolation
- maximum number of iteration

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- select all land points for extrapolation

Parameters

<code>in, out</code>	<code>td_var</code>	variable structure
<code>in</code>	<code>id_radius</code>	radius of the halo used to compute extrapolation

The documentation for this interface was generated from the following files:

- `src/extrap.f90`

11.27 fct Module Reference

This module groups some basic useful function.

Data Types

- interface `fct_str`
- interface `operator(//)`

Public Member Functions

- integer(i4) function, public `fct_getunit ()`
- subroutine, public `fct_err (id_status)`
This subroutine handle Fortran status.
- subroutine, public `fct_pause (cd_msg)`
This subroutine create a pause statement.
- pure function, public `fct_concat (cd_arr, cd_sep)`
This function concatenate all the element of a character array in a character string.
- pure function, public `fct_lower (cd_var)`
This function convert string character upper case to lower case.
- pure function, public `fct_upper (cd_var)`

This function convert string character lower case to upper case.

- pure function, public `fct_is_num` (cd_var)

This function check if character is numeric.

- pure function, public `fct_is_real` (cd_var)

This function check if character is real number.

- pure function, public `fct_split` (cd_string, id_ind, cd_sep)

This function split string of character using separator character, by default '|'; and return the element on index ind.

- pure function, public `fct_basename` (cd_string, cd_sep)

This function return basename of a filename.

- pure function, public `fct_dirname` (cd_string, cd_sep)

This function return dirname of a filename.

- subroutine, public `fct_help` (cd_filename, cd_err)

This function show help message.

- subroutine, public `fct_version` (cd_filename)

This function show the version of Siren.

11.27.1 Detailed Description

This module groups some basic useful function.

to get free I/O unit number:

```
il_id=fct_getunit()
```

to convert "numeric" to string character:

```
cl_string=fct_str(numeric)
```

- "numeric" could be integer, real, or logical

to concatenate "numeric" to a string character:

```
cl_str=cd_char//num
```

- cd_char is the string character
- num is the numeric value (integer, real or logical)

to concatenate all the element of a character array:

```
cl_string=fct_concat(cd_arr [,cd_sep])
```

- cd_arr is a 1D array of character
- cd_sep is a separator character to add between each element of cd_arr [optional]

to convert character from lower to upper case:

```
cl_upper=fct_upper(cd_var)
```

to convert character from upper to lower case:

```
cl_lower=fct_lower(cd_var)
```

to check if character is numeric

```
ll_is_num=fct_is_num(cd_var)
```

to check if character is real

```
ll_is_real=fct_is_real(cd_var)
```

to split string into substring and return one of the element:

```
cl_str=fct_split(cd_string ,id_ind [,cd_sep])
```

- cd_string is a string of character
- id_ind is the indice of the lement to extract
- cd_sep is the separator use to split cd_string (default '|')

to get basename (name without path):

```
cl_str=fct_basename(cd_string [,cd_sep])
```

- cd_string is the string filename
- cd_sep is the separator ti be used (default '/')

to get dirname (path of the filename):

```
cl_str=fct_dirname(cd_string [,cd_sep])
```

- cd_string is the string filename
- cd_sep is the separator ti be used (default '/')

to create a pause statement:

```
CALL fct_pause(cd_msg)
```

- cd_msg : message to be added [optional]

to handle fortran error:

```
CALL fct_err(id_status)
```

to show help message:

```
CALL fct_help(cd_filename, cd_err)
```

- cd_filename : file name
- cd_err : error message [optional]

to show Siren's version:

```
CALL fct_version(cd_filename)
```

- cd_filename : file name

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header

October, 2019

- add help and version function

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.27.2 Member Function/Subroutine Documentation

11.27.2.1 pure function, public fct::fct_basename (character(len=*), intent(in) *cd_string*, character(len=*), intent(in), optional *cd_sep*)

This function return basename of a filename.

Parameters

<i>in</i>	<i>cd_string</i>	
-----------	------------------	--

Actually it splits filename using sperarator '/' and return last string character.

Optionally you could specify another separator.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_string</i>	filename
<i>in</i>	<i>cd_sep</i>	separator character

Returns

basename (filename without path)

11.27.2.2 pure function, public fct::fct_concat (character(*), dimension(:), intent(in) *cd_arr*, character(*), intent(in), optional *cd_sep*)

This function concatenate all the element of a character array in a character string.

Parameters

<i>in</i>	<i>cd_arr</i>
-----------	---------------

optionnally a separator could be added between each element.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_arr</i>	array of character
<i>in</i>	<i>cd_sep</i>	separator character

Returns

character

11.27.2.3 pure function, public fct::fct_dirname (character(len=*), intent(in) *cd_string*, character(len=*), intent(in), optional *cd_sep*)

This function return dirname of a filename.

Parameters

<i>in</i>	<i>cd_string</i>
-----------	------------------

Actually it splits filename using sperarator '/' and return all except last string character.

Optionally you could specify another separator.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_string</i>	filename
<i>in</i>	<i>cd_sep</i>	separator character

Returns

dirname (path of the filename)

11.27.2.4 subroutine, public fct::fct_err (integer(i4), intent(in) *id_status*)

This subroutine handle Fortran status.

Parameters

<i>in</i>	<i>id_status</i>
-----------	------------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>id_status</i>
-----------	------------------

11.27.2.5 `integer(i4)` function, public `fct::fct_getunit ()`

This function returns the next available I/O unit number.

Author

J.Paul

Date

November, 2013 - Initial Version

Returns

file id

11.27.2.6 `subroutine`, public `fct::fct_help (character(len=*), intent(in) cd_filename, character(len=*), intent(in), optional cd_err)`

This function show help message.

Parameters

<i>in</i>	<i>cd_filename</i>
-----------	--------------------

Optionally, print error detected

Author

J.Paul

Date

October, 2019 - Initial Version

Parameters

in	<i>cd_filename</i>	file name
in	<i>cd_err</i>	error message

Returns

print help message

11.27.2.7 pure function, public fct::fct_is_num (character(len=*), intent(in) cd_var)

This function check if character is numeric.

Parameters

in	<i>cd_var</i>	
----	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>cd_var</i>	character
----	---------------	-----------

Returns

character is numeric

11.27.2.8 pure function, public fct::fct_is_real (character(len=*), intent(in) cd_var)

This function check if character is real number.

Parameters

in	<i>cd_var</i>	
----	---------------	--

it permits exponential and decimal number exemple : 1e6, 2.3

Author

J.Paul

Date

June, 2015 - Initial Version

April, 2018

- permit negative exposant
- permit sign as first character

Parameters

<i>in</i>	<i>cd_var</i>	character
-----------	---------------	-----------

Returns

character is real number

11.27.2.9 pure function, public fct::fct_lower (character(*), intent(in) *cd_var*)

This function convert string character upper case to lower case.

Parameters

<i>in</i>	<i>cd_var</i>	
-----------	---------------	--

The function IACHAR returns the ASCII value of the character passed as argument. The ASCII code has the uppercase alphabet starting at code 65, and the lower case one at code 101, therefore IACHAR('a')- IACHAR('A') would be the difference between the uppercase and the lowercase codes.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_var</i>	character
-----------	---------------	-----------

Returns

lower case character

11.27.2.10 subroutine, public fct::fct_pause (character(len=*), intent(in), optional *cd_msg*)

This subroutine create a pause statement.

Parameters

<i>in</i>	<i>cd_msg</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2014 - Initial Version

Parameters

<i>in</i>	<i>cd_msg</i>	optional message to be added
-----------	---------------	------------------------------

11.27.2.11 pure function, public fct::fct_split (character(len=*), intent(in) *cd_string*, integer(i4), intent(in) *id_ind*, character(len=*), intent(in), optional *cd_sep*)

This function split string of character using separator character, by default '|', and return the element on index ind.

Parameters

<i>in</i>	<i>cd_string</i>
-----------	------------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_string</i>	string of character
<i>in</i>	<i>id_ind</i>	indice
<i>in</i>	<i>cd_sep</i>	separator character

Returns

return the element of index *id_ind*

11.27.2.12 pure function, public fct::fct_upper (character(*), intent(in) *cd_var*)

This function convert string character lower case to upper case.

Parameters

<i>in</i>	<i>cd_var</i>
-----------	---------------

The function IACHAR returns the ASCII value of the character passed as argument. The ASCII code has the uppercase alphabet starting at code 65, and the lower case one at code 101, therefore IACHAR('a')- IACHAR('A') would be the difference between the uppercase and the lowercase codes.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_var</i>	character
-----------	---------------	-----------

Returns

upper case character

11.27.2.13 subroutine, public fct::fct_version (character(len=*), intent(in) cd_filename)

This function show the version of Siren.

Parameters

in	cd_filename	
----	-------------	--

Author

J.Paul

Date

October, 2019 - Initial Version

Parameters

in	cd_filename	file name
----	-------------	-----------

Returns

print version message

The documentation for this module was generated from the following file:

- src/function.f90

11.28 fct::fct_str Interface Reference

Public Member Functions

- pure function [fct__i1_str](#) (bd_var)
This function convert integer(1) to string character.
- pure function [fct__i2_str](#) (sd_var)
This function convert integer(2) to string character.
- pure function [fct__i4_str](#) (id_var)
This function convert integer(4) to string character.
- pure function [fct__i8_str](#) (kd_var)
This function convert integer(8) to string character.
- pure function [fct__r4_str](#) (rd_var)
This function convert real(4) to string character.
- pure function [fct__r8_str](#) (dd_var)
This function convert real(8) to string character.
- pure function [fct__l_str](#) (ld_var)
This function convert logical to string character.

11.28.1 Member Function/Subroutine Documentation

11.28.1.1 pure function `fct::fct_str::fct__i1_str (integer(i1), intent(in) bd_var)`

This function convert integer(1) to string character.

Parameters

<i>in</i>	<i>bd_var</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>bd_var</i>	integer(1) variable
-----------	---------------	---------------------

Returns

character of this integer variable

11.28.1.2 pure function fct::fct_str::fct_i2_str (integer(i2), intent(in) sd_var)

This function convert integer(2) to string character.

Parameters

<i>in</i>	<i>sd_var</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>sd_var</i>	integer(2) variable
-----------	---------------	---------------------

Returns

character of this integer variable

11.28.1.3 pure function fct::fct_str::fct_i4_str (integer(i4), intent(in) id_var)

This function convert integer(4) to string character.

Parameters

<i>in</i>	<i>id_var</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>id_var</i>	integer(4) variable
-----------	---------------	---------------------

Returns

character of this integer variable

11.28.1.4 pure function fct::fct_str::fct_i8_str (integer(i8), intent(in) *kd_var*)

This function convert integer(8) to string character.

Parameters

<i>in</i>	<i>kd_var</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>kd_var</i>	integer(8) variable
-----------	---------------	---------------------

Returns

character of this integer variable

11.28.1.5 pure function fct::fct_str::fct_l_str (logical, intent(in) *ld_var*)

This function convert logical to string character.

Parameters

<i>in</i>	<i>ld_var</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>ld_var</i>	logical variable
-----------	---------------	------------------

Returns

character of this integer variable

11.28.1.6 pure function fct::fct_str::fct_r4_str (real(sp), intent(in) *rd_var*)

This function convert real(4) to string character.

Parameters

<i>in</i>	<i>rd_var</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>rd_var</i>	real(4) variable
-----------	---------------	------------------

Returns

character of this real variable

11.28.1.7 pure function `fct::fct_str::fct_r8_str (real(dp), intent(in) dd_var)`

This function convert real(8) to string character.

Parameters

<i>in</i>	<i>dd_var</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>dd_var</i>	real(8) variable
-----------	---------------	------------------

Returns

character of this real variable

The documentation for this interface was generated from the following file:

- `src/function.f90`

11.29 file Module Reference

This module manage file structure.

Data Types

- interface [file_clean](#)
- interface [file_copy](#)
- interface [file_del_att](#)
- interface [file_del_var](#)
- interface [file_rename](#)
- type [tfile](#)

Public Member Functions

- type(**tfile**) function, public [file_init](#) (cd_file, cd_type, ld_wrt, id_ew, id_perio, id_pivot, cd_grid)
 - This function initialize file structure.*
- character(len=lc) function, public [file_get_type](#) (cd_file)
- logical function, public [file_check_var_dim](#) (td_file, td_var, ld_chklen)
- subroutine, public [file_add_var](#) (td_file, td_var)
- subroutine, public [file_move_var](#) (td_file, td_var)
- subroutine, public [file_add_att](#) (td_file, td_att)
- subroutine, public [file_move_att](#) (td_file, td_att)
 - This subroutine move a global attribute structure from file structure.*
- subroutine, public [file_add_dim](#) (td_file, td_dim)
 - This subroutine add a dimension structure in file structure. Do not overwrite, if dimension already in file structure.*
- subroutine, public [file_del_dim](#) (td_file, td_dim)
 - This subroutine delete a dimension structure in file structure.*
- subroutine, public [file_move_dim](#) (td_file, td_dim)
 - This subroutine move a dimension structure in file structure.*
- subroutine, public [file_print](#) (td_file)
 - This subroutine print some information about file strucutre.*
- character(len=lc) function, public [file_add_suffix](#) (cd_file, cd_type)
 - This function add suffix to file name.*
- integer(i4) function, public [file_get_id](#) (td_file, cd_name)
 - This function return the file id, in a array of file structure, given file name.*
- integer(i4) function [file_get_unit](#) (td_file)
 - This function get the next unused unit in array of file structure.*

11.29.1 Detailed Description

This module manage file structure.

```
define type TFILE:<br/>
```

```
TYPE(tfile) :: tl_file
```

to initialize a file structure:

```
tl_file=file_init(cd_file [,cd_type] [,ld_wrt] [,cd_grid])
```

- cd_file is the file name
- cd_type is the type of the file ('cdf', 'dimg') [optional]
- ld_wrt file in write mode or not [optional] - cd_grid is the grid type (default 'ARAKAWA-C')

to get file name:

- tl_file%c_name

to get file id (units):

- tl_file%i_id

to get the type of the file (cdf, cdf4, dimg):

- `tl_file%c_type`

to know if file was open in write mode:

- `tl_file%l_wrt`

to get the record length of the file:

- `tl_file%i_recl`

Files variables

to get the number of variable in the file:

- `tl_file%i_nvar`

to get the array of variable structure associated to the file:

- `tl_file%t_var(:)`

Files attributes

to get the number of global attributes of the file:

- `tl_file%i_natt`

to get the array of attributes structure associated to the file:

- `tl_file%t_att(:)`

Files dimensions

to get the number of dimension used in the file:

- `tl_file%i_ndim`

to get the array of dimension structure (4 elts) associated to the file:

- `tl_file%t_dim(:)`

to print information about file structure:

```
CALL file_print(td_file)
```

to clean file structure:

```
CALL file_clean(td_file)
```

to add a global attribute structure in file structure:

```
CALL file_add_att(td_file, td_att)
```

- `td_att` is an attribute structure

to add a dimension structure in file structure:

```
CALL file_add_dim(td_file, td_dim)
```

- `td_dim` is a dimension structure

to add a variable structure in file structure:

```
CALL file_add_var(td_file, td_var)
```

- `td_var` is a variable structure

to delete a global attribute structure in file structure:

```
CALL file_del_att(td_file, td_att)
```

- `td_att` is an attribute structure

to delete a dimension structure in file structure:

```
CALL file_del_dim(td_file, td_dim)
```

- `td_dim` is a dimension structure

to delete a variable structure in file structure:

```
CALL file_del_var(td_file, td_var)
```

- `td_var` is a variable structure

to overwrite one attribute structure in file structure:

```
CALL file_move_att(td_file, td_att)
```

- `td_att` is an attribute structure

to overwrite one dimension structure in file structure:

```
CALL file_move_dim(td_file, td_dim)
```

- `td_dim` is a dimension structure

to overwrite one variable structure in file structure:

```
CALL file_move_var(td_file, td_var)
```

- `td_var` is a variable structure

to check if file and variable structure share same dimension:

```
ll_check_dim = file_check_var_dim(td_file, td_var)
```

- `td_var` is a variable structure

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- Fix memory leaks bug

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.29.2 Member Function/Subroutine Documentation11.29.2.1 subroutine, public file::file_add_att (type(tfile), intent(inout) *td_file*, type(tatt), intent(in) *td_att*)**Parameters**

<i>in, out</i>	<i>td_file</i>	
----------------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- clean attribute structure

Parameters

<i>in, out</i>	<i>td_file</i>	file structure
<i>in</i>	<i>td_att</i>	attribute structure

11.29.2.2 subroutine, public file::file_add_dim (type(tfile), intent(inout) *td_file*, type(tdim), intent(in) *td_dim*)

This subroutine add a dimension structure in file structure. Do not overwrite, if dimension already in file structure.

Parameters

<i>in, out</i>	<i>td_file</i>	
----------------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- do not reorder dimension, before put in file

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
<code>in</code>	<code>td_dim</code>	dimension structure

11.29.2.3 `character(len=lc) function, public file::file_add_suffix (character(len=*), intent(in) cd_file, character(len=*), intent(in) cd_type)`

This function add suffix to file name.

Parameters

<code>in</code>	<code>cd_file</code>	
-----------------	----------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in</code>	<code>td_file</code>	file structure
-----------------	----------------------	----------------

Returns

file name

11.29.2.4 `subroutine, public file::file_add_var (type(tfile), intent(inout) td_file, type(tvar), intent(inout) td_var)`

Parameters

<code>in, out</code>	<code>td_file</code>	
----------------------	----------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add dimension in file if need be
- do not reorder dimension from variable, before put in file

September, 2015

- check variable dimension expected

January, 2019

- clean variable structure

Parameters

in, out	<i>td_file</i>	file structure
in	<i>td_var</i>	variable structure

11.29.2.5 logical function, public file::file_check_var_dim (type(tfile), intent(in) *td_file*, type(tvar), intent(in) *td_var*, logical, intent(in), optional *ld_chklen*)

Parameters

in	<i>td_file</i>	file structure
in	<i>td_var</i>	variable structure
in	<i>ld_chklen</i>	check length

Returns

true if dimension of variable and file structure agree

11.29.2.6 subroutine, public file::file_del_dim (type(tfile), intent(inout) *td_file*, type(tdim), intent(in) *td_dim*)

This subroutine delete a dimension structure in file structure.

Parameters

in, out	<i>td_file</i>	
---------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- clean dimension structure

Parameters

in, out	<i>td_file</i>	file structure
in	<i>td_dim</i>	dimension structure

11.29.2.7 integer(i4) function, public file::file_get_id (type(tfile), dimension(:), intent(in) *td_file*, character(len=*), intent(in) *cd_name*)

This function return the file id, in a array of file structure, given file name.

Parameters

in	<i>td_file</i>	
----	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	array of file structure
in	<i>cd_name</i>	file name

Returns

file id in array of file structure (0 if not found)

11.29.2.8 `character(len=lc)` function, public `file::file_get_type (character(len=*), intent(in) cd_file)`

Parameters

in	<i>cd_file</i>	
----	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- netcdf4 files identify as netcdf file

Parameters

in	<i>cd_file</i>	file name
----	----------------	-----------

Returns

type of file

11.29.2.9 `integer(i4)` function `file::file_get_unit (type(tfile), dimension(:), intent(in) td_file)`

This function get the next unused unit in array of file structure.

Parameters

in	<i>td_file</i>	
----	----------------	--

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

in	<i>td_file</i>	array of file
----	----------------	---------------

11.29.2.10 `type(tfile)` function, public `file::file_init (character(len=*), intent(in) cd_file, character(len=*), intent(in), optional cd_type, logical, intent(in), optional ld_wrt, integer(i4), intent(in), optional id_ew, integer(i4), intent(in), optional id_perio, integer(i4), intent(in), optional id_pivot, character(len=*), intent(in), optional cd_grid)`

This function initialize file structure.

Parameters

<i>in</i>	<i>cd_file</i>
-----------	----------------

If *cd_type* is not specify, check if file name include '.nc' or '.dimg'

Optionally, you could specify:

- write mode (default .FALSE., *ld_wrt*)
- East-West overlap (*id_ew*)
- NEMO periodicity index (*id_perio*)
- NEMO pivot point index F(0),T(1) (*id_pivot*)
- grid type (default: 'ARAKAWA-C')

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_file</i>	file name
<i>in</i>	<i>cd_type</i>	file type ('cdf', 'dimg')
<i>in</i>	<i>ld_wrt</i>	write mode (default .FALSE.)
<i>in</i>	<i>id_ew</i>	east-west overlap
<i>in</i>	<i>id_perio</i>	NEMO periodicity index
<i>in</i>	<i>id_pivot</i>	NEMO pivot point index F(0),T(1)
<i>in</i>	<i>cd_grid</i>	grid type (default 'ARAKAWA-C')

Returns

file structure

11.29.2.11 subroutine, public `file::file_move_att (type(tfile), intent(inout) td_file, type(tatt), intent(in) td_att)`

This subroutine move a global attribute structure from file structure.

Parameters

<i>in, out</i>	<i>td_file</i>
----------------	----------------

Warning

change attribute id in file structure.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
<code>in</code>	<code>td_att</code>	attribute structure

11.29.2.12 subroutine, public file::file_move_dim (type(tfile), intent(inout) *td_file*, type(tdim), intent(in) *td_dim*)

This subroutine move a dimension structure in file structure.

Parameters

<code>in, out</code>	<code>td_file</code>	
----------------------	----------------------	--

Warning

change dimension order in file structure.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
<code>in</code>	<code>td_dim</code>	dimension structure

11.29.2.13 subroutine, public file::file_move_var (type(tfile), intent(inout) *td_file*, type(tvar), intent(in) *td_var*)

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
<code>in</code>	<code>td_var</code>	variable structure

11.29.2.14 subroutine, public file::file_print (type(tfile), intent(in) *td_file*)

This subroutine print some information about file strucutre.

Parameters

in	<i>td_file</i>
----	----------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
----	----------------	----------------

The documentation for this module was generated from the following file:

- src/file.f90

11.30 file::file_clean Interface Reference

Public Member Functions

- subroutine [file__clean_unit](#) (*td_file*)
This subroutine clean file strcuture.
- subroutine [file__clean_arr](#) (*td_file*)
This subroutine clean file array of file strcuture.

11.30.1 Member Function/Subroutine Documentation

11.30.1.1 subroutine file::file_clean::file__clean_arr (type(tfile), dimension(:), intent(inout) *td_file*)

This subroutine clean file array of file strcuture.

Parameters

in, out	<i>td_file</i>
---------	----------------

Author

J.Paul

Date

Marsh, 2014 - Inital version

Parameters

in, out	<i>td_file</i>	array file strcuture
---------	----------------	----------------------

11.30.1.2 subroutine file::file_clean::file__clean_unit (type(tfile), intent(inout) *td_file*)

This subroutine clean file strcuture.

Parameters

<code>in, out</code>	<code>td_file</code>
----------------------	----------------------

Author

J.Paul

Date

November, 2013 - Inital version

January, 2019

- nullify attribute structure inside file structure
- nullify variable structure inside file structure

Parameters

<code>in, out</code>	<code>td_file</code>	file strcuture
----------------------	----------------------	----------------

The documentation for this interface was generated from the following file:

- src/file.f90

11.31 file::file_copy Interface Reference

Public Member Functions

- type(**tfile**) function [file__copy_unit](#) (`td_file`)
This subroutine copy file structure in another one.
- type(**tfile**) function,
dimension(size(`td_file(:)`)) [file__copy_arr](#) (`td_file`)
This subroutine copy a array of file structure in another one.

11.31.1 Member Function/Subroutine Documentation

11.31.1.1 type(**tfile**) function, dimension(size(`td_file(:)`)) `file::file_copy::file__copy_arr` (type(**tfile**), dimension(:), intent(in) `td_file`)

This subroutine copy a array of file structure in another one.

Parameters

<code>in</code>	<code>td_file</code>
-----------------	----------------------

file variable and attribute value are copied in a temporary array, so input and output file structure value do not point on the same "memory cell", and so on are independant.

Note

new file is assume to be closed.

Warning

do not use on the output of a function who create or read an structure (ex: `tl_file=file_copy(file_init())` is forbidden). This will create memory leaks.
to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- use function instead of overload assignment operator (to avoid memory leak)

Parameters

in	<i>td_file</i>	file structure
----	----------------	----------------

Returns

copy of input array of file structure

11.31.1.2 type(tfile) function file::file_copy::file__copy_unit (type(tfile), intent(in) *td_file*)

This subroutine copy file structure in another one.

Parameters

in	<i>td_file</i>	
----	----------------	--

file variable and attribute value are copied in a temporary array, so input and output file structure value do not point on the same "memory cell", and so on are independant.

Note

new file is assume to be closed.

Warning

do not use on the output of a function who create or read an structure (ex: `tl_file=file_copy(file_init())` is forbidden). This will create memory leaks.

to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- use function instead of overload assignment operator (to avoid memory leak)

January, 2019

- clean variable structure

Parameters

in	<i>td_file</i>	file structure
----	----------------	----------------

Returns

copy of input file structure

The documentation for this interface was generated from the following file:

- src/file.f90

11.32 file::file_del_att Interface Reference**Public Member Functions**

- subroutine [file__del_att_name](#) (td_file, cd_name)
This subroutine delete a global attribute structure in file structure, given attribute name.
- subroutine [file__del_att_str](#) (td_file, td_att)
This subroutine delete a global attribute structure from file structure, given attribute structure.

11.32.1 Member Function/Subroutine Documentation

11.32.1.1 subroutine file::file_del_att::file__del_att_name (type(tfile), intent(inout) *td_file*, character(len=*), intent(in) *cd_name*)

This subroutine delete a global attribute structure in file structure, given attribute name.

Parameters

in, out	<i>td_file</i>	
---------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

February, 2015

- define local attribute structure to avoid mistake with pointer

January, 2019

- clean attribute structure

Parameters

in, out	<i>td_file</i>	file structure
in	<i>cd_name</i>	attribute name

11.32.1.2 subroutine file::file_del_att::file__del_att_str (type(tfile), intent(inout) *td_file*, type(tatt), intent(in) *td_att*)

This subroutine delete a global attribute structure from file structure, given attribute structure.

Parameters

in, out	td_file
---------	---------

Author

J.Paul

Date

November, 2013 - Initial Version
January, 2019

- clean attribute structure

Parameters

in, out	td_file	file structure
in	td_att	attribute structure

The documentation for this interface was generated from the following file:

- src/file.f90

11.33 file::file_del_var Interface Reference

Public Member Functions

- subroutine [file__del_var_name](#) (td_file, cd_name)
This subroutine delete a variable structure in file structure, given variable name or standard name.
- subroutine [file__del_var_str](#) (td_file, td_var)
This subroutine delete a variable structure in file structure, given variable structure.

11.33.1 Member Function/Subroutine Documentation

11.33.1.1 subroutine `file::file_del_var::file__del_var_name (type(tfiler), intent(inout) td_file, character(len=*), intent(in) cd_name)`

This subroutine delete a variable structure in file structure, given variable name or standard name.

Parameters

in, out	td_file
---------	---------

Author

J.Paul

Date

November, 2013 - Initial Version
February, 2015

- define local variable structure to avoid mistake with pointer

Parameters

in, out	<i>td_file</i>	file structure
in	<i>cd_name</i>	variable name or standard name

11.33.1.2 subroutine `file::file_del_var::file__del_var_str (type(tfile), intent(inout) td_file, type(tvar), intent(in) td_var)`

This subroutine delete a variable structure in file structure, given variable structure.

Parameters

in, out	<i>td_file</i>	
---------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- clean variable structure

Parameters

in, out	<i>td_file</i>	file structure
in	<i>td_var</i>	variable structure

The documentation for this interface was generated from the following file:

- src/file.f90

11.34 file::file_rename Interface Reference

Public Member Functions

- character(len=lc) function [file__rename_char](#) (*cd_file*, *id_num*)
This function rename file name, given processor number.
- type(tfile) function [file__rename_str](#) (*td_file*, *id_num*)
This function rename file name, given file structure.

11.34.1 Member Function/Subroutine Documentation

11.34.1.1 character(len=lc) function `file::file_rename::file__rename_char (character(len=*) , intent(in) cd_file, integer(i4), intent(in), optional id_num)`

This function rename file name, given processor number.

Parameters

in	<i>cd_file</i>	
----	----------------	--

If no processor number is given, return file name without number If processor number is given, return file name with new number

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>id_num</i>	processor number (start to 1)

Returns

file name

11.34.1.2 `type(tfile) function file::file_rename::file__rename_str (type(tfile), intent(in) td_file, integer(i4), intent(in), optional id_num)`

This function rename file name, given file structure.

Parameters

in	<i>td_file</i>	
----	----------------	--

If no processor number is given, return file name without number I processor number is given, return file name with new number

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>id_num</i>	processor number (start to 1)

Returns

file structure

The documentation for this interface was generated from the following file:

- src/file.f90

11.35 filter Module Reference

This module is filter manager.

Data Types

- interface [filter_fill_value](#)

11.35.1 Detailed Description

This module is filter manager.

Filtering method to be used is specify inside variable structure, as array of string character.

td_var%c_filter(1) string character is the filter name choose between:

- 'hann'
 - $rad < cutoff : filter = 0.5 + 0.5 * COS(\pi * \frac{rad}{cutoff})$
 - $rad > cutoff : filter = 0$
- 'hamming'
 - $rad < cutoff : filter = 0.54 + 0.46 * COS(\pi * \frac{rad}{cutoff})$
 - $rad > cutoff : filter = 0$
- 'blackman'
 - $rad < cutoff : filter = 0.42 + 0.5 * COS(\pi * \frac{rad}{cutoff}) + 0.08 * COS(2\pi * \frac{rad}{cutoff})$
 - $rad > cutoff : filter = 0$
- 'gauss'
 - $filter = exp(-(\alpha * rad^2)/(2 * cutoff^2))$
- 'butterworth'
 - $filter = 1/(1 + (rad^2/cutoff^2)^\alpha)$ with $rad = \sqrt{(dist - radius)^2}$

td_var%c_filter(2) string character is the number of turn to be done

td_var%c_filter(3) string character is the cut-off frequency td_var%c_filter(4) string character is the halo radius (count in number of mesh grid)

td_var%c_filter(5) string character is the alpha parameter (for gauss and butterworth method)

Note

Filter method could be specify for each variable in namelist *namvar*, defining string character *cn_varinfo*. None by default.

Filter method parameters are informed inside bracket.

- α parameter is added for *gauss* and *butterworth* methods

The number of turn is specify using '*' separator.

Example:

- `cn_varinfo='varname1:flt=2*hamming(cutoff, radius)', 'varname2:flt=gauss(cutoff, radius, alpha)'`

to filter variable value:

```
CALL filter_fill_value( td_var )
```

- td_var is variable structure

Author

J.Paul

Date

November, 2013 - Initial Version

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

The documentation for this module was generated from the following file:

- src/filter.f90

11.36 filter::filter_fill_value Interface Reference

Public Member Functions

- subroutine [filter__fill_value_wrapper](#) (td_var)
This subroutine filter variable value.

11.36.1 Member Function/Subroutine Documentation

11.36.1.1 subroutine filter::filter_fill_value::filter__fill_value_wrapper (type(tvar), intent(inout) td_var)

This subroutine filter variable value.

Parameters

in, out	td_var
---------	--------

it checks if filtering method is available, gets parameter value, and launch filter__fill_value

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	td_var	variable structure
---------	--------	--------------------

The documentation for this interface was generated from the following file:

- src/filter.f90

11.37 global Module Reference

This module defines global variables and parameters.

Public Attributes

- integer(i4), parameter, public `ip_maxvar` =200
maximum number of variable
- integer(i4), parameter, public `ip_maxmtx` =50
matrix variable maximum dimension (cf create_bathy)
- integer(i4), parameter, public `ip_maxseg` =10
maximum number of segment for each boundary
- integer(i4), parameter, public `ip_nsep` =2
number of separator listed
- character(1), dimension(ip_nsep),
parameter, public `cp_sep` = ('.', '_')
- list of separator*
- integer(i4), parameter, public `ip_ncom` =2
number of comment character listed
- character(1), dimension(ip_ncom),
parameter, public `cp_com` = ('#', '!')
- list of comment character*
- integer(i4), parameter, public `ip_ghost` =1
number of ghost cell
- integer(i4), parameter, public `ip_ninterp` =3
- character(len=lc), dimension(ip_ninterp),
parameter, public `cp_interp_list` = (/ 'nearest', 'cubic', 'linear' /)
- integer(i4), parameter, public `ip_nextrap` =2
- character(len=lc), dimension(ip_nextrap),
parameter, public `cp_extrap_list` = (/ 'dist_weight', 'min_error' /)
- integer(i4), parameter, public `ip_nfilter` =5
- character(len=lc), dimension(ip_nfilter),
parameter, public `cp_filter_list` = (/ 'butterworth', 'blackman', 'hamming', 'hann', 'gauss' /)
- real(dp), parameter `dp_fill_i1` =NF90_FILL_BYTE
byte fill value
- real(dp), parameter `dp_fill_i2` =NF90_FILL_SHORT
short fill value
- real(dp), parameter `dp_fill_i4` =NF90_FILL_INT
INT fill value.
- real(dp), parameter `dp_fill_sp` =NF90_FILL_FLOAT
real fill value
- real(dp), parameter, public `dp_fill` =NF90_FILL_DOUBLE
double fill value
- integer(i4), parameter, public `ip_npoint` =4
- integer(i4), parameter, public `jp_t` =1
- integer(i4), parameter, public `jp_u` =2
- integer(i4), parameter, public `jp_v` =3
- integer(i4), parameter, public `jp_f` =4
- character(len=1), dimension(ip_npoint),
parameter, public `cp_grid_point` = (/ 'T', 'U', 'V', 'F' /)
- integer(i4), parameter, public `ip_maxdimcfg` =10
maximum dimension in configuration file
- integer(i4), parameter, public `ip_maxdim` =4
- integer(i4), parameter, public `jp_i` =1
- integer(i4), parameter, public `jp_j` =2
- integer(i4), parameter, public `jp_k` =3
- integer(i4), parameter, public `jp_l` =4

- character(len=ip_maxdim),
parameter, public `cp_dimorder` = 'xyzt'
dimension order to output
- integer(i4), parameter, public `ip_ncard` =4
- character(len=lc), dimension(ip_ncard),
parameter, public `cp_card` = (/ 'north', 'south', 'east ', 'west ' /)
- integer(i4), parameter, public `jp_north` =1
- integer(i4), parameter, public `jp_south` =2
- integer(i4), parameter, public `jp_east` =3
- integer(i4), parameter, public `jp_west` =4
- integer(i4), parameter, public `ip_maxdumcfg` = 10
maximum dummy variable, dimension, or attribute in configuration file
- character(len=lc), parameter,
public `cp_url` = "\$URL: https://svn.mercator-ocean.fr/svnroot/mo/REG/SIREN/trunk/Siren/src/global.f90 \$"
svn url
- character(len=lc), parameter `cp_version` = "\$Revision: 17058 \$"
- character(len=lc), parameter `cp_author` = "\$Author: jpaul \$"
- character(len=lc), parameter `cp_date` = "\$Date: 2019-11-05 14:49:57 +0100 (Tue, 05 Nov 2019) \$"

11.37.1 Detailed Description

This module defines global variables and parameters.

Author

J.paul

Date

November, 2013 - Initial Version

September, 2015

- define fill value for each variable type

January, 2019

- define svn URL variable

October, 2019

- define svn Revision, Date, and Author variable

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

The documentation for this module was generated from the following file:

- src/global.f90

11.38 grid Module Reference

This module is grid manager.

Data Types

- interface [grid_get_closest](#)
- interface [grid_get_coarse_index](#)
- interface [grid_get_ew_overlap](#)
- interface [grid_get_fine_offset](#)
- interface [grid_get_ghost](#)
- interface [grid_get_info](#)
- interface [grid_get_perio](#)
- interface [grid_get_pivot](#)

Public Member Functions

- logical function, public [grid_is_north_fold](#) (td_lat)
This subroutine check if there is north fold.
- subroutine, public [grid_check_dom](#) (td_coord, id_imin, id_imax, id_jmin, id_jmax)
This subroutine check domain validity.
- logical function, public [grid_is_global](#) (td_lon, td_lat)
This function check if grid is global or not.
- real(dp) function, dimension(size(dd_lon(:,:), dim=1),size(dd_lon(:,:), dim=2)),
public [grid_distance](#) (dd_lon, dd_lat, dd_lonA, dd_latA)
This function compute the distance between a point A and grid points.
- subroutine, public [grid_check_coincidence](#) (td_coord0, td_coord1, id_imin0, id_imax0, id_jmin0, id_jmax0, id_rho)
This subroutine check fine and coarse grid coincidence.
- subroutine, public [grid_add_ghost](#) (td_var, id_ghost)
This subroutine add ghost cell at boundaries.
- subroutine, public [grid_del_ghost](#) (td_var, id_ghost)
This subroutine delete ghost cell at boundaries.
- integer(i4) function,
dimension(td_var%t_dim(1)%i_len,td_var%t_dim(2)%i_len),
public [grid_split_domain](#) (td_var, id_level)
This subroutine compute closed sea domain.
- subroutine, public [grid_fill_small_dom](#) (td_var, id_mask, id_minsize)
This subroutine fill small closed sea with fill value.
- subroutine, public [grid_fill_small_msk](#) (id_mask, id_minsize)
This subroutine fill small domain inside bigger one.

Public Attributes

- integer(i4), parameter **im_max_overlap** = 5

11.38.1 Detailed Description

This module is grid manager.

```
to get NEMO pivot point index:<br/>
```

```
il_pivot=grid_get_pivot(td_file)
```

- `il_pivot` is NEMO pivot point index F(0), T(1)

- `td_file` is mpp structure

to get NEMO periodicity index:

```
il_perio=grid_get_perio(td_file)
```

- `il_perio` is NEMO periodicity index (0,1,2,3,4,5,6)
- `td_file` is mpp structure

to check domain validity:

```
CALL grid_check_dom(td_coord, id_imin, id_imax, id_jmin, id_jmax)
```

- `td_coord` is coordinates mpp structure
- `id_imin` is i-direction lower left point indice
- `id_imax` is i-direction upper right point indice
- `id_jmin` is j-direction lower left point indice
- `id_jmax` is j-direction upper right point indice

to get closest coarse grid indices of fine grid domain:

```
il_index(:,:)=grid_get_coarse_index(td_coord0, td_coord1,  
                                  [id_rho,] [cd_point])
```

or

```
il_index(:,:)=grid_get_coarse_index(td_lon0, td_lat0, td_coord1,  
                                  [id_rho,] [cd_point])
```

or

```
il_index(:,:)=grid_get_coarse_index(td_coord0, td_lon1, td_lat1,  
                                  [id_rho,] [cd_point])
```

or

```
il_index(:,:)=grid_get_coarse_index(td_lon0, td_lat0, td_lon1, td_lat1,  
                                  [id_rho,] [cd_point])
```

- `il_index(:,:)` is coarse grid indices (/ (/ imin0, imax0 /), (/ jmin0, jmax0 /) /)
- `td_coord0` is coarse grid coordinate mpp structure
- `td_coord1` is fine grid coordinate mpp structure
- `td_lon0` is coarse grid longitude variable structure
- `td_lat0` is coarse grid latitude variable structure
- `td_lon1` is fine grid longitude variable structure
- `td_lat1` is fine grid latitude variable structure
- `id_rho` is array of refinement factor (default 1)
- `cd_point` is Arakawa grid point (default 'T')

to know if grid is global:

```
ll_global=grid_is_global(td_lon, td_lat)
```

- td_lon is longitude variable structure
- td_lat is latitude variable structure

to know if grid contains north fold:

```
ll_north=grid_is_north_fold(td_lat)
```

- td_lat is latitude variable structure

to get coarse grid indices of the closest point from one fine grid point:

```
il_index(:)=grid_get_closest(dd_lon0(:, :), dd_lat0(:, :), dd_lon1, dd_lat1
    [,dd_fill] [,cd_pos])
```

- il_index(:) is coarse grid indices (/ i0, j0 /)
- dd_lon0 is coarse grid array of longitude value (real(8))
- dd_lat0 is coarse grid array of latitude value (real(8))
- dd_lon1 is fine grid longitude value (real(8))
- dd_lat1 is fine grid latitude value (real(8))
- dd_fill
- cd_pos

to compute distance between a point A and grid points:

```
il_dist(:, :)=grid_distance(dd_lon, dd_lat, dd_lona, dd_lata)
```

- il_dist(:, :) is array of distance between point A and grid points
- dd_lon is array of longitude value (real(8))
- dd_lat is array of longitude value (real(8))
- dd_lona is longitude of point A (real(8))
- dd_lata is latitude of point A (real(8))

to get offset between fine grid and coarse grid:

```
il_offset(:, :)=grid_get_fine_offset(td_coord0,
    id_imin0, id_jmin0, id_imax0, id_jmax0,
    td_coord1
    [,id_rho] [,cd_point])
```

or

```
il_offset(:, :)=grid_get_fine_offset(dd_lon0, dd_lat0,
    id_imin0, id_jmin0, id_imax0, id_jmax0,
    td_coord1
    [,id_rho] [,cd_point])
```

or

```
il_offset(:,:)=grid_get_fine_offset(td_coord0,
                                   id_imin0, id_jmin0, id_imax0, id_jmax0,
                                   dd_lon1, dd_lat1
                                   [,id_rho] [,cd_point])
```

or

```
il_offset(:,:)=grid_get_fine_offset(dd_lon0, dd_lat0,
                                   id_imin0, id_jmin0, id_imax0, id_jmax0,
                                   dd_lon1, dd_lat1
                                   [,id_rho] [,cd_point])
```

- `il_offset(:,:)` is offset array (/ (/ `i_offset_left`, `i_offset_right` /), (/ `j_offset_lower`, `j_offset_upper` /) /)
- `td_coord0` is coarse grid coordinate mpp structure
- `dd_lon0` is coarse grid longitude array (real(8))
- `dd_lat0` is coarse grid latitude array (real(8))
- `id_imin0` is coarse grid lower left corner i-indices of fine grid domain
- `id_jmin0` is coarse grid lower left corner j-indices of fine grid domain
- `id_imax0` is coarse grid upper right corner i-indices of fine grid domain
- `id_jmax0` is coarse grid upper right corner j-indices of fine grid domain
- `td_coord1` is fine grid coordinate mpp structure
- `dd_lon1` is fine grid longitude array (real(8))
- `dd_lat1` is fine grid latitude array (real(8))
- `id_rho` is array of refinement factor (default 1)
- `cd_point` is Arakawa grid point (default 'T')

to check fine and coarse grid coincidence:

```
CALL grid_check_coincidence(td_coord0, td_coord1,
                           id_imin0, id_imax0, id_jmin0, id_jmax0
                           ,id_rho)
```

- `td_coord0` is coarse grid coordinate mpp structure
- `td_coord1` is fine grid coordinate mpp structure
- `id_imin0` is coarse grid lower left corner i-indices of fine grid domain
- `id_imax0` is coarse grid upper right corner i-indices of fine grid domain
- `id_jmin0` is coarse grid lower left corner j-indices of fine grid domain
- `id_jmax0` is coarse grid upper right corner j-indices of fine grid domain
- `id_rho` is array of refinement factor

to add ghost cell at boundaries:

```
CALL grid_add_ghost(td_var, id_ghost)
```

- `td_var` is array of variable structure
- `id_ghost` is 2D array of ghost cell factor

to delete ghost cell at boundaries:

```
CALL grid_del_ghost(td_var, id_ghost)
```

- `td_var` is array of variable structure
- `id_ghost` is 2D array of ghost cell factor

to get ghost cell factor (use or not):

```
il_factor(:)= grid_get_ghost( td_var )
```

or

```
il_factor(:)= grid_get_ghost( td_mpp )
```

- `il_factor(:)` is array of ghost cell factor (0 or 1)
- `td_var` is variable structure
- `td_mpp` is mpp structure

to compute closed sea domain:

```
il_mask(:,:)=grid_split_domain(td_var, [id_level])
```

- `il_mask(:,:)` is domain mask
- `td_var` is variable structure
- `id_level` is level to be used [optional]

to fill small closed sea with `_FillValue`:

```
CALL grid_fill_small_dom(td_var, id_mask, [id_minsize])
```

- `td_var` is variable structure
- `id_mask` is domain mask (from `grid_split_domain`)
- `id_minsize` is minimum size of sea to be kept [optional]

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header

October, 2014

- use mpp file structure instead of file

February, 2015

- add function `grid_fill_small_msk` to fill small domain inside bigger one

February, 2016

- improve way to check coincidence (bug fix)

- manage grid cases for T,U,V or F point, with even or odd refinement (bug fix)

April, 2016

- add function to get closest grid point using coarse grid coordinates structure

May, 2019

- define as module variable `im_max_overlap`

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.38.2 Member Function/Subroutine Documentation

11.38.2.1 subroutine, public `grid::grid_add_ghost (type(tvar), intent(inout) td_var, integer(i4), dimension(2,2), intent(in) id_ghost)`

This subroutine add ghost cell at boundaries.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_var</i>	array of variable structure
<i>in</i>	<i>id_ghost</i>	array of ghost cell factor

11.38.2.2 subroutine, public `grid::grid_check_coincidence (type(tmpp), intent(in) td_coord0, type(tmpp), intent(in) td_coord1, integer(i4), intent(in) id_imin0, integer(i4), intent(in) id_imax0, integer(i4), intent(in) id_jmin0, integer(i4), intent(in) id_jmax0, integer(i4), dimension(:), intent(in) id_rho)`

This subroutine check fine and coarse grid coincidence.

Parameters

<i>in</i>	<i>td_coord0</i>
-----------	------------------

Author

J.Paul

Date

November, 2013- Initial Version

October, 2014

- work on mpp file structure instead of file structure

February, 2016

- use F-point to check coincidence for even refinement
- use F-point estimation, if can not read it.

Parameters

in	<i>td_coord0</i>	coarse grid coordinate file structure
in	<i>td_coord1</i>	fine grid coordinate file structure
in	<i>id_imin0</i>	coarse grid lower left corner i-indice of fine grid domain
in	<i>id_imax0</i>	coarse grid upper right corner i-indice of fine grid domain
in	<i>id_jmin0</i>	coarse grid lower left corner j-indice of fine grid domain
in	<i>id_jmax0</i>	coarse grid upper right corner j-indice of fine grid domain
in	<i>id_rho</i>	array of refinement factor

11.38.2.3 subroutine, public grid::grid_check_dom (type(tmpp), intent(in) *td_coord*, integer(i4), intent(in) *id_imin*, integer(i4), intent(in) *id_imax*, integer(i4), intent(in) *id_jmin*, integer(i4), intent(in) *id_jmax*)

This subroutine check domain validity.

Parameters

in	<i>td_coord</i>	
----	-----------------	--

If maximum latitude greater than 88°N, program will stop.

Note

Not able to manage north fold for now.

Author

J.Paul

Date

November, 2013 - Initial Version

October, 2014

- work on mpp file structure instead of file structure

Parameters

in	<i>cd_coord</i>	coordinate file
in	<i>id_imin</i>	i-direction lower left point indice
in	<i>id_imax</i>	i-direction upper right point indice
in	<i>id_jmin</i>	j-direction lower left point indice
in	<i>id_jmax</i>	j-direction upper right point indice

11.38.2.4 subroutine, public grid::grid_del_ghost (type(tvar), intent(inout) *td_var*, integer(i4), dimension(2,2), intent(in) *id_ghost*)

This subroutine delete ghost cell at boundaries.

Parameters

in, out	<i>td_var</i>	
---------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_var</i>	array of variable structure
<i>in</i>	<i>id_ghost</i>	array of ghost cell factor

11.38.2.5 `real(dp)` function, `dimension(size(dd_lon(:,:),dim=1), size(dd_lon(:,:),dim=2))`, `public grid::grid_distance (real(dp), dimension(:,:), intent(in) dd_lon, real(dp), dimension(:,:), intent(in) dd_lat, real(dp), intent(in) dd_lonA, real(dp), intent(in) dd_latA)`

This function compute the distance between a point A and grid points.

Parameters

<i>in</i>	<i>dd_lon</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>dd_lon</i>	grid longitude array
<i>in</i>	<i>dd_lat</i>	grid latitude array
<i>in</i>	<i>dd_lonA</i>	longitude of point A
<i>in</i>	<i>dd_latA</i>	latitude of point A
<i>in</i>	<i>dd_fill</i>	

Returns

array of distance between point A and grid points.

11.38.2.6 `subroutine, public grid::grid_fill_small_dom (type(tvar), intent(inout) td_var, integer(i4), dimension(:,:), intent(in) id_mask, integer(i4), intent(in), optional id_minsize)`

This subroutine fill small closed sea with fill value.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

the minimum size (number of point) of closed sea to be kept could be sepcify with `id_minsize`. By default only the biggest sea is preserve.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
<i>in</i>	<i>id_mask</i>	domain mask (from <code>grid_split_domain</code>)
<i>in</i>	<i>id_minsize</i>	minimum size of sea to be kept

11.38.2.7 subroutine, public `grid::grid_fill_small_msk` (`integer(i4)`, `dimension(:,:)`, `intent(inout) id_mask`, `integer(i4)`, `intent(in) id_minsize`)

This subroutine fill small domain inside bigger one.

Parameters

<i>in, out</i>	<i>id_mask</i>	
----------------	----------------	--

the minimum size (number of point) of domain sea to be kept is specified by `id_minsize`. smaller domain are included in the one they are embedded.

Author

J.Paul

Date

Ferbruary, 2015 - Initial Version

Parameters

<i>in, out</i>	<i>id_mask</i>	domain mask (from <code>grid_split_domain</code>)
<i>in</i>	<i>id_minsize</i>	minimum size of sea to be kept

11.38.2.8 logical function, public `grid::grid_is_global` (`type(tvar)`, `intent(in) td_lon`, `type(tvar)`, `intent(in) td_lat`)

This function check if grid is global or not.

Parameters

<i>in</i>	<i>td_lon</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_lon</i>	longitude structure
<i>in</i>	<i>td_lat</i>	latitude structure

Returns

true if grid is global

11.38.2.9 logical function, public `grid::grid_is_north_fold` (`type(tvar)`, `intent(in) td_lat`)

This subroutine check if there is north fold.

Parameters

<i>in</i>	<i>td_lat</i>
-----------	---------------

check if maximum latitude greater than 88°N

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_lat</i>	latitude variable structure
-----------	---------------	-----------------------------

Returns

true if there is north fold

11.38.2.10 `integer(i4) function, dimension(td_var%t_dim(1)%i_len, td_var%t_dim(2)%i_len), public grid::grid_split_domain (type(tvar), intent(in) td_var, integer(i4), intent(in), optional id_level)`

This subroutine compute closed sea domain.

Parameters

<i>in</i>	<i>td_var</i>
-----------	---------------

to each domain is associated a negative value id (from -1 to ...)

optionally you could specify which level use (default 1)

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_var</i>	variable strucutre
<i>in</i>	<i>id_level</i>	level

Returns

domain mask

The documentation for this module was generated from the following file:

- src/grid.f90

11.39 grid::grid_get_closest Interface Reference

Public Member Functions

- `integer(i4) function, dimension(2) grid__get_closest_str (td_coord0, dd_lon1, dd_lat1, cd_pos, dd_fill)`

This function return grid indices of the closest point from point (lon1,lat1)

- integer(i4) function, dimension(2) `grid__get_closest_arr` (dd_lon0, dd_lat0, dd_lon1, dd_lat1, cd_pos, dd_fill)

This function return grid indices of the closest point from point (lon1,lat1)

11.39.1 Member Function/Subroutine Documentation

11.39.1.1 integer(i4) function, dimension(2) `grid::grid_get_closest::grid__get_closest_arr` (real(dp), dimension(:,:), intent(in) *dd_lon0*, real(dp), dimension(:,:), intent(in) *dd_lat0*, real(dp), intent(in) *dd_lon1*, real(dp), intent(in) *dd_lat1*, character(len=*) , intent(in), optional *cd_pos*, real(dp), intent(in), optional *dd_fill*)

This function return grid indices of the closest point from point (lon1,lat1)

Parameters

<i>in</i>	<i>dd_lon0</i>
-----------	----------------

Note

overlap band should have been already removed from coarse grid array of longitude and latitude, before running this function

if you add *cd_pos* argument, you could choice to return closest point at

- lower left (ll) of the point
- lower right (lr) of the point
- upper left (ul) of the point
- upper right (ur) of the point
- lower (lo) of the point
- upper (up) of the point
- left (le) of the point
- right (ri) of the point

Author

J.Paul

Date

November, 2013 - Initial Version

February, 2015

- change dichotomy method to manage ORCA grid

February, 2016

- add optional use of relative position

Parameters

<i>in</i>	<i>dd_lon0</i>	coarse grid array of longitude
-----------	----------------	--------------------------------

in	<i>dd_lat0</i>	coarse grid array of latitude
in	<i>dd_lon1</i>	fine grid longitude
in	<i>dd_lat1</i>	fine grid latitude
in	<i>cd_pos</i>	relative position of grid point from point
in	<i>dd_fill</i>	fill value

Returns

coarse grid indices of closest point of fine grid point

11.39.1.2 integer(i4) function, dimension(2) grid::grid_get_closest::grid_get_closest_str (type(tmpp), intent(in) *td_coord0*, real(dp), intent(in) *dd_lon1*, real(dp), intent(in) *dd_lat1*, character(len=*) , intent(in), optional *cd_pos*, real(dp), intent(in), optional *dd_fill*)

This function return grid indices of the closest point from point (lon1,lat1)

Parameters

in	<i>td_coord0</i>
----	------------------

Note

overlap band should have been already removed from coarse grid array of longitude and latitude, before running this function

if you add *cd_pos* argument, you could choice to return closest point at

- lower left (ll) of the point
- lower right (lr) of the point
- upper left (ul) of the point
- upper right (ur) of the point
- lower (lo) of the point
- upper (up) of the point
- left (le) of the point
- right (ri) of the point

Author

J.Paul

Date

April, 2016 - Initial Version

October, 2016

- use max of zero and east-west overlap instead of east-west overlap

Parameters

in	<i>td_coord0</i>	coarse grid coordinate mpp structure
in	<i>dd_lon1</i>	fine grid longitude
in	<i>dd_lat1</i>	fine grid latitude
in	<i>cd_pos</i>	relative position of grid point from point
in	<i>dd_fill</i>	fill value

Returns

coarse grid indices of closest point of fine grid point

The documentation for this interface was generated from the following file:

- src/grid.f90

11.40 grid::grid_get_coarse_index Interface Reference

Public Member Functions

- integer(i4) function,
dimension(2, 2) [grid__get_coarse_index_ff](#) (td_coord0, td_coord1, id_rho, cd_point)
This function get closest coarse grid indices of fine grid domain.
- integer(i4) function,
dimension(2, 2) [grid__get_coarse_index_cf](#) (td_lon0, td_lat0, td_coord1, id_rho, cd_point)
This function get closest coarse grid indices of fine grid domain.
- integer(i4) function,
dimension(2, 2) [grid__get_coarse_index_fc](#) (td_coord0, td_lon1, td_lat1, id_rho, cd_point)
This function get closest coarse grid indices of fine grid domain.
- integer(i4) function,
dimension(2, 2) [grid__get_coarse_index_cc](#) (td_lon0, td_lat0, td_lon1, td_lat1, id_rho, cd_point)

11.40.1 Member Function/Subroutine Documentation

11.40.1.1 integer(i4) function, dimension(2,2) [grid::grid_get_coarse_index::grid__get_coarse_index_cc](#) (type(tvar), intent(in) *td_lon0*, type(tvar), intent(in) *td_lat0*, type(tvar), intent(in) *td_lon1*, type(tvar), intent(in) *td_lat1*, integer(i4), dimension(:), intent(in), optional *id_rho*, character(len=*) , intent(in), optional *cd_point*)

Parameters

in	<i>td_lon0</i>
----	----------------

it use coarse and fine grid array of longitude and latitude. optionally, you could specify the array of refinement factor (default 1.) optionally, you could specify on which Arakawa grid point you want to work (default 'T')

Note

do not use ghost cell

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- check grid point
- take into account EW overlap

February, 2016

- use delta (lon or lat)
- manage cases for T,U,V or F point, with even or odd refinement

Parameters

in	<i>td_lon0</i>	coarse grid longitude
in	<i>td_lat0</i>	coarse grid latitude
in	<i>td_lon1</i>	fine grid longitude
in	<i>td_lat1</i>	fine grid latitude
in	<i>id_rho</i>	array of refinement factor
in	<i>cd_point</i>	Arakawa grid point ('T','U','V','F')

Returns

coarse grid indices ((/imin0, imax0/), (/jmin0, jmax0/))

Todo -check case boundary domain on overlap band

11.40.1.2 integer(i4) function, dimension(2,2) grid::grid_get_coarse_index::grid_get_coarse_index_cf (type(tvar), intent(in) *td_lon0*, type(tvar), intent(in) *td_lat0*, type(tmpp), intent(in) *td_coord1*, integer(i4), dimension(:), intent(in), optional *id_rho*, character(len=*), intent(in), optional *cd_point*)

This function get closest coarse grid indices of fine grid domain.

Parameters

in	<i>td_lon0</i>	
----	----------------	--

it use coarse array of longitude and latitude and fine grid coordinates file. optionally, you could specify the array of refinement factor (default 1.) optionally, you could specify on which Arakawa grid point you want to work (default 'T')

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- use grid point to read coordinates variable.

October, 2014

- work on mpp file structure instead of file structure

February, 2015

- use longitude or latitude as standard name, if can not find longitude_T, latitude_T...

Parameters

in	<i>td_longitude0</i>	coarse grid longitude
in	<i>td_latitude0</i>	coarse grid latitude
in	<i>td_coord1</i>	fine grid coordinate mpp structure
in	<i>id_rho</i>	array of refinement factor
in	<i>cd_point</i>	Arakawa grid point (default 'T')

Returns

coarse grid indices (*((imin0, imax0/), (jmin0, jmax0/))*)

11.40.1.3 integer(i4) function, dimension(2,2) `grid::grid_get_coarse_index::grid_get_coarse_index_fc (type(tmpp), intent(in) td_coord0, type(tvar), intent(in) td_lon1, type(tvar), intent(in) td_lat1, integer(i4), dimension(:), intent(in), optional id_rho, character(len=*), intent(in), optional cd_point)`

This function get closest coarse grid indices of fine grid domain.

Parameters

in	<i>td_coord0</i>
----	------------------

it use coarse grid coordinates file and fine grid array of longitude and latitude. optionally, you could specify the array of refinement factor (default 1.) optionally, you could specify on which Arakawa grid point you want to work (default 'T')

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- use grid point to read coordinates variable.

October, 2014

- work on mpp file structure instead of file structure

February, 2015

- use longitude or latitude as standard name, if can not find longitude_T, latitude_T...

Parameters

in	<i>td_coord0</i>	coarse grid coordinate mpp structure
in	<i>td_lon1</i>	fine grid longitude
in	<i>td_lat1</i>	fine grid latitude
in	<i>id_rho</i>	array of refinement factor (default 1.)
in	<i>cd_point</i>	Arakawa grid point (default 'T')

Returns

coarse grid indices (*((imin0, imax0/), (jmin0, jmax0/))*)

11.40.1.4 integer(i4) function, dimension(2,2) `grid::grid_get_coarse_index::grid_get_coarse_index_ff (type(tmpp), intent(in) td_coord0, type(tmpp), intent(in) td_coord1, integer(i4), dimension(:), intent(in), optional id_rho, character(len=*), intent(in), optional cd_point)`

This function get closest coarse grid indices of fine grid domain.

Parameters

in	<i>td_coord0</i>
----	------------------

it use coarse and fine grid coordinates files. optionally, you could specify the array of refinement factor (default 1.) optionally, you could specify on which Arakawa grid point you want to work (default 'T')

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- use grid point to read coordinates variable.

October, 2014

- work on mpp file structure instead of file structure

February, 2015

- use longitude or latitude as standard name, if can not find longitude_T, latitude_T...

Parameters

in	<i>td_coord0</i>	coarse grid coordinate mpp structure
in	<i>td_coord1</i>	fine grid coordinate mpp structure
in	<i>id_rho</i>	array of refinement factor (default 1.)
in	<i>cd_point</i>	Arakawa grid point (default 'T').

Returns

coarse grid indices((/imin0, imax0/), (/jmin0, jmax0/))

The documentation for this interface was generated from the following file:

- src/grid.f90

11.41 grid::grid_get_ew_overlap Interface Reference

Public Member Functions

- integer(i4) function [grid_get_ew_overlap_mpp](#) (td_mpp)
This function get East-West overlap.
- integer(i4) function [grid_get_ew_overlap_file](#) (td_file)
This function get East-West overlap.
- integer(i4) function [grid_get_ew_overlap_var](#) (td_var)
This function get East-West overlap.

11.41.1 Member Function/Subroutine Documentation

11.41.1.1 integer(i4) function [grid::grid_get_ew_overlap::grid_get_ew_overlap_file](#) (type(tfile), intent(inout) *td_file*)

This function get East-West overlap.

Parameters

<i>in, out</i>	<i>td_file</i>
----------------	----------------

If no East-West wrap return -1, else return the size of the ovarlap band. East-West overlap is computed comparing longitude value of the South part of the domain, to avoid north fold boundary.

Author

J.Paul

Date

October, 2014 - Initial Version

October, 2016

- check varid for longitude_T

August, 2017

- read only grid boundaries to handle huge file

Parameters

<i>in</i>	<i>td_file</i>	file structure
-----------	----------------	----------------

Returns

East West overlap

11.41.1.2 `integer(i4) function grid::grid_get_ew_overlap::grid_get_ew_overlap_mpp (type(tmpp), intent(inout) td_mpp)`

This function get East-West overlap.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

If no East-West wrap return -1, else return the size of the ovarlap band. East-West overlap is computed comparing longitude value of the South part of the domain, to avoid north fold boundary.

Author

J.Paul

Date

November, 2013 - Initial Version

October, 2014

- work on mpp file structure instead of file structure

October, 2016

- check varid for longitude_T

August, 2017

- read only grid boundaries to handle huge file

Parameters

<code>in</code>	<code>td_mpp</code>	mpp structure
-----------------	---------------------	---------------

Returns

East West overlap

11.41.1.3 integer(i4) function grid::grid_get_ew_overlap::grid_get_ew_overlap_var (type(tvar), intent(inout) td_var)

This function get East-West overlap.

Parameters

<code>in, out</code>	<code>td_var</code>	
----------------------	---------------------	--

If no East-West wrap return -1, else return the size of the ovarlap band. East-West overlap is computed comparing longitude value of the South part of the domain, to avoid north fold boundary.

Author

J.Paul

Date

November, 2013 - Initial Version

October, 2014

- work on mpp file structure instead of file structure

October, 2016

- check longitude as longname

Parameters

<code>in</code>	<code>td_lon</code>	longitude variable structure
-----------------	---------------------	------------------------------

Returns

East West overlap

The documentation for this interface was generated from the following file:

- src/grid.f90

11.42 grid::grid_get_fine_offset Interface Reference

Public Member Functions

- integer(i4) function, dimension(2, 2) [grid_get_fine_offset_ff](#) (td_coord0, id_imin0, id_jmin0, id_imax0, id_jmax0, td_coord1, id_rho, cd_point)

This function get offset between fine grid and coarse grid.

- integer(i4) function, dimension(2, 2) [grid_get_fine_offset_fc](#) (td_coord0, id_imin0, id_jmin0, id_imax0, id_jmax0, dd_lon1, dd_lat1, id_rho, cd_point)

This function get offset between fine grid and coarse grid.

- integer(i4) function,
dimension(2, 2) [grid__get_fine_offset_cf](#) (dd_lon0, dd_lat0, id_imin0, id_jmin0, id_imax0, id_jmax0, td_coord1, id_rho, cd_point)
This function get offset between fine grid and coarse grid.
- integer(i4) function,
dimension(2, 2) [grid__get_fine_offset_cc](#) (dd_lon0, dd_lat0, id_imin0, id_jmin0, id_imax0, id_jmax0, dd_lon1, dd_lat1, id_rho, cd_point)
This function get offset between fine grid and coarse grid.

11.42.1 Member Function/Subroutine Documentation

11.42.1.1 integer(i4) function, dimension(2,2) [grid::grid_get_fine_offset::grid_get_fine_offset_cc](#) (real(dp), dimension(:,:), intent(in) *dd_lon0*, real(dp), dimension(:,:), intent(in) *dd_lat0*, integer(i4), intent(in) *id_imin0*, integer(i4), intent(in) *id_jmin0*, integer(i4), intent(in) *id_imax0*, integer(i4), intent(in) *id_jmax0*, real(dp), dimension(:,:), intent(in) *dd_lon1*, real(dp), dimension(:,:), intent(in) *dd_lat1*, integer(i4), dimension(:), intent(in) *id_rho*, character(len=*) , intent(in), optional *cd_point*)

This function get offset between fine grid and coarse grid.

Parameters

<i>in</i>	<i>dd_lon0</i>
-----------	----------------

offset value could be 0,1,...,rho-1

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- rename from [grid_get_fine_offset](#)

May, 2015

- improve way to find offset

July, 2015

- manage case close to greenwich meridian

February, 2016

- use [grid_get_closest](#) to assess offset
- use delta (lon or lat)
- manage cases for T,U,V or F point, with even or odd refinement
- check lower left(upper right) fine grid point inside lower left(upper right) coarse grid cell.

Todo check case close from North fold.

Parameters

<i>in</i>	<i>dd_lon0</i>	coarse grid longitude array
<i>in</i>	<i>dd_lat0</i>	coarse grid latitude array
<i>in</i>	<i>id_imin0</i>	coarse grid lower left corner i-indices of fine grid domain

in	<i>id_jmin0</i>	coarse grid lower left corner j-indice of fine grid domain
in	<i>id_imax0</i>	coarse grid upper right corner i-indice of fine grid domain
in	<i>id_jmax0</i>	coarse grid upper right corner j-indice of fine grid domain
in	<i>dd_lon1</i>	fine grid longitude array
in	<i>dd_lat1</i>	fine grid latitude array
in	<i>id_rho</i>	array of refinement factor
in	<i>cd_point</i>	Arakawa grid point

Returns

offset array (/ (/i_offset_left,i_offset_right/),(/j_offset_lower,j_offset_upper/) /)

11.42.1.2 integer(i4) function, dimension(2,2) grid::grid_get_fine_offset::grid_get_fine_offset_cf (real(dp), dimension(:,:), intent(in) *dd_lon0*, real(dp), dimension(:,:), intent(in) *dd_lat0*, integer(i4), intent(in) *id_imin0*, integer(i4), intent(in) *id_jmin0*, integer(i4), intent(in) *id_imax0*, integer(i4), intent(in) *id_jmax0*, type(tmpp), intent(in) *td_coord1*, integer(i4), dimension(:), intent(in), optional *id_rho*, character(len=*), intent(in), optional *cd_point*)

This function get offset between fine grid and coarse grid.

Parameters

in	<i>dd_lon0</i>
----	----------------

optionally, you could specify on which Arakawa grid point you want to work (default 'T') offset value could be 0,1,...,rho-1

Author

J.Paul

Date

September, 2014 - Initial Version
October, 2014

- work on mpp file structure instead of file structure

Parameters

in	<i>dd_lon0</i>	coarse grid longitude array
in	<i>dd_lat0</i>	coarse grid latitude array
in	<i>id_imin0</i>	coarse grid lower left corner i-indice of fine grid domain
in	<i>id_jmin0</i>	coarse grid lower left corner j-indice of fine grid domain
in	<i>id_imax0</i>	coarse grid upper right corner i-indice of fine grid domain
in	<i>id_jmax0</i>	coarse grid upper right corner j-indice of fine grid domain
in	<i>td_coord1</i>	fine grid coordinate
in	<i>id_rho</i>	array of refinement factor
in	<i>cd_point</i>	Arakawa grid point

Returns

offset array (/ (/i_offset_left,i_offset_right/),(/j_offset_lower,j_offset_upper/) /)

11.42.1.3 integer(i4) function, dimension(2,2) grid::grid_get_fine_offset::grid_get_fine_offset_fc (type(tmpp), intent(in) *td_coord0*, integer(i4), intent(in) *id_imin0*, integer(i4), intent(in) *id_jmin0*, integer(i4), intent(in) *id_imax0*, integer(i4), intent(in) *id_jmax0*, real(dp), dimension(:,:), intent(in) *dd_lon1*, real(dp), dimension(:,:), intent(in) *dd_lat1*, integer(i4), dimension(:), intent(in), optional *id_rho*, character(len=*), intent(in), optional *cd_point*)

This function get offset between fine grid and coarse grid.

Parameters

<i>in</i>	<i>td_coord0</i>
-----------	------------------

optionally, you could specify on which Arakawa grid point you want to work (default 'T') offset value could be 0,1,...,rho-1

Author

J.Paul

Date

September, 2014 - Initial Version

October, 2014

- work on mpp file structure instead of file structure

Parameters

<i>in</i>	<i>td_coord0</i>	coarse grid coordinate
<i>in</i>	<i>id_imin0</i>	coarse grid lower left corner i-indice of fine grid domain
<i>in</i>	<i>id_jmin0</i>	coarse grid lower left corner j-indice of fine grid domain
<i>in</i>	<i>id_imax0</i>	coarse grid upper right corner i-indice of fine grid domain
<i>in</i>	<i>id_jmax0</i>	coarse grid upper right corner j-indice of fine grid domain
<i>in</i>	<i>dd_lon1</i>	fine grid longitude array
<i>in</i>	<i>dd_lat1</i>	fine grid latitude array
<i>in</i>	<i>id_rho</i>	array of refinement factor
<i>in</i>	<i>cd_point</i>	Arakawa grid point

Returns

offset array (/ (/i_offset_left,i_offset_right/),(/j_offset_lower,j_offset_upper/) /)

11.42.1.4 integer(i4) function, dimension(2,2) grid::grid_get_fine_offset::grid_get_fine_offset_ff (type(tmpp), intent(in) *td_coord0*, integer(i4), intent(in) *id_imin0*, integer(i4), intent(in) *id_jmin0*, integer(i4), intent(in) *id_imax0*, integer(i4), intent(in) *id_jmax0*, type(tmpp), intent(in) *td_coord1*, integer(i4), dimension(:), intent(in), optional *id_rho*, character(len=*) , intent(in), optional *cd_point*)

This function get offset between fine grid and coarse grid.

Parameters

<i>in</i>	<i>td_coord0</i>
-----------	------------------

optionally, you could specify on which Arakawa grid point you want to work (default 'T') offset value could be 0,1,...,rho-1

Author

J.Paul

Date

September, 2014 - Initial Version

October, 2014

- work on mpp file structure instead of file structure

Parameters

in	<i>td_coord0</i>	coarse grid coordinate
in	<i>id_imin0</i>	coarse grid lower left corner i-indice of fine grid domain
in	<i>id_jmin0</i>	coarse grid lower left corner j-indice of fine grid domain
in	<i>id_imax0</i>	coarse grid upper right corner i-indice of fine grid domain
in	<i>id_jmax0</i>	coarse grid upper right corner j-indice of fine grid domain
in	<i>td_coord1</i>	fine grid coordinate
in	<i>id_rho</i>	array of refinement factor
in	<i>cd_point</i>	Arakawa grid point

Returns

offset array (/ (/i_offset_left,i_offset_right/),(/j_offset_lower,j_offset_upper/) /)

The documentation for this interface was generated from the following file:

- src/grid.f90

11.43 grid::grid_get_ghost Interface Reference

Public Member Functions

- integer(i4) function,
dimension(2, 2) [grid_get_ghost_var](#) (td_var)
This function check if ghost cell are used or not, and return ghost cell factor (0,1) in horizontal plan.
- integer(i4) function,
dimension(2, 2) [grid_get_ghost_mpp](#) (td_mpp)
This function check if ghost cell are used or not, and return ghost cell factor (0,1) in i- and j-direction.

11.43.1 Member Function/Subroutine Documentation

11.43.1.1 integer(i4) function, dimension(2,2) [grid::grid_get_ghost::grid_get_ghost_mpp](#) (type(tmpp), intent(in) *td_mpp*)

This function check if ghost cell are used or not, and return ghost cell factor (0,1) in i- and j-direction.

Parameters

in	<i>td_mpp</i>
----	---------------

get longitude an latitude array, then check if domain is global, and if there is an East-West overlap

Author

J.Paul

Date

September, 2014 - Initial Version

October, 2014

- work on mpp file structure instead of file structure

Parameters

in	<i>td_file</i>	file sturcture
----	----------------	----------------

Returns

array of ghost cell factor

11.43.1.2 integer(i4) function, dimension(2,2) grid::grid_get_ghost::grid__get_ghost_var (type(tvar), intent(in) *td_var*)

This function check if ghost cell are used or not, and return ghost cell factor (0,1) in horizontal plan.

Parameters

in	<i>td_var</i>	
----	---------------	--

check if domain is global, and if there is an East-West overlap.

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

in	<i>td_var</i>	variable sturcture
----	---------------	--------------------

Returns

array of ghost cell factor

The documentation for this interface was generated from the following file:

- src/grid.f90

11.44 grid::grid_get_info Interface Reference

Public Member Functions

- subroutine [grid__get_info_mpp](#) (td_mpp)
This subroutine get information about global domain, given mpp structure.
- subroutine [grid__get_info_file](#) (td_file)
This subroutine get information about global domain, given file strucutre.

11.44.1 Member Function/Subroutine Documentation

11.44.1.1 subroutine grid::grid_get_info::grid__get_info_file (type(tfile), intent(inout) *td_file*)

This subroutine get information about global domain, given file strucutre.

Parameters

<i>in, out</i>	<i>td_file</i>
----------------	----------------

open edge files then:

- compute NEMO pivot point
- compute NEMO periodicity
- compute East West overlap

Note

need all processor files to be there

Author

J.Paul

Date

October, 2014 - Initial Version

Parameters

<i>in, out</i>	<i>td_file</i>	file structure
----------------	----------------	----------------

11.44.1.2 subroutine grid::grid_get_info::grid_get_info_mpp (type(tmpp), intent(inout) *td_mpp*)

This subroutine get information about global domain, given mpp structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

open edge files then:

- compute NEMO pivot point
- compute NEMO periodicity
- compute East West overlap

Note

need all processor files

Author

J.Paul

Date

October, 2014 - Initial Version

Parameters

in	<i>td_mpp</i>	mpp structure
----	---------------	---------------

The documentation for this interface was generated from the following file:

- src/grid.f90

11.45 grid::grid_get_perio Interface Reference

Public Member Functions

- integer(i4) function [grid__get_perio_mpp](#) (td_mpp, id_pivot)
This subroutine search NEMO periodicity given mpp structure and optionally pivot point index.
- integer(i4) function [grid__get_perio_file](#) (td_file, id_pivot)
This subroutine search NEMO periodicity index given file structure, and optionally pivot point index.
- integer(i4) function [grid__get_perio_var](#) (td_var, id_pivot)
This subroutine search NEMO periodicity index given variable structure and pivot point index.

11.45.1 Member Function/Subroutine Documentation

11.45.1.1 integer(i4) function `grid::grid_get_perio::grid__get_perio_file (type(tfile), intent(in) td_file, integer(i4), intent(in), optional id_pivot)`

This subroutine search NEMO periodicity index given file structure, and optionally pivot point index.

Parameters

in	<i>td_file</i>
----	----------------

The variable used must be on T point.

0: closed boundaries 1: cyclic east-west boundary 2: symmetric boundary condition across the equator 3: North fold boundary (with a F-point pivot) 4: North fold boundary (with a F-point pivot) and cyclic east-west boundary 5: North fold boundary (with a T-point pivot) 6: North fold boundary (with a T-point pivot) and cyclic east-west boundary

Warning

pivot point should have been computed before run this script. see [grid_get_pivot](#).

Author

J.Paul

Date

October, 2014 - Initial version
August, 2017

- read only grid boundaries to handle huge file

Parameters

in	<i>td_file</i>	file structure
in	<i>id_pivot</i>	pivot point index

Returns

NEMO periodicity index

11.45.1.2 integer(i4) function grid::grid_get_perio::grid__get_perio_mpp (type(tmpp), intent(in) *td_mpp*, integer(i4), intent(in), optional *id_pivot*)

This subroutine search NEMO periodicity given mpp structure and optionally pivot point index.

Parameters

in	<i>td_mpp</i>	
----	---------------	--

The variable used must be on T point.

0: closed boundaries 1: cyclic east-west boundary 2: symmetric boundary condition across the equator 3: North fold boundary (with a T-point pivot) 4: North fold boundary (with a T-point pivot) and cyclic east-west boundary 5: North fold boundary (with a F-point pivot) 6: North fold boundary (with a F-point pivot) and cyclic east-west boundary

Warning

pivot point should have been computed before run this script. see [grid_get_pivot](#).

Author

J.Paul

Date

October, 2014 - Initial version

August, 2017

- read only grid boundaries to handle huge file

January, 2019

- do not use silicalim, or silicamax to get pivot point

Todo do not check silicalim, or silicamax

Parameters

in	<i>td_mpp</i>	mpp file structure
in	<i>id_pivot</i>	pivot point index

Returns

NEMO periodicity index

11.45.1.3 integer(i4) function grid::grid_get_perio::grid__get_perio_var (type(tvar), intent(in) *td_var*, integer(i4), intent(in), optional *id_pivot*)

This subroutine search NEMO periodicity index given variable structure and pivot point index.

Parameters

<code>in</code>	<code>td_var</code>
-----------------	---------------------

The variable must be on T point.

0: closed boundaries 1: cyclic east-west boundary 2: symmetric boundary condition across the equator 3: North fold boundary (with a T-point pivot) 4: North fold boundary (with a T-point pivot) and cyclic east-west boundary 5: North fold boundary (with a F-point pivot) 6: North fold boundary (with a F-point pivot) and cyclic east-west boundary

Warning

pivot point should have been computed before run this script. see [grid_get_pivot](#).

Author

J.Paul

Date

November, 2013 - Initial version

October, 2014

- work on variable structure instead of file structure

Parameters

<code>in</code>	<code>td_var</code>	variable structure
<code>in</code>	<code>id_pivot</code>	pivot point index

The documentation for this interface was generated from the following file:

- src/grid.f90

11.46 grid::grid_get_pivot Interface Reference

Public Member Functions

- integer(i4) function [grid__get_pivot_mpp](#) (td_mpp)
This function compute NEMO pivot point index from input mpp variable.
- integer(i4) function [grid__get_pivot_file](#) (td_file)
This function compute NEMO pivot point index from input file variable.
- integer(i4) function [grid__get_pivot_var](#) (td_var)
This function compute NEMO pivot point index of the input variable.

11.46.1 Member Function/Subroutine Documentation

11.46.1.1 integer(i4) function grid::grid_get_pivot::grid__get_pivot_file (type(tfile), intent(in) td_file)

This function compute NEMO pivot point index from input file variable.

Parameters

<code>in</code>	<code>td_file</code>	- F-point : 0 • T-point : 1
-----------------	----------------------	--------------------------------

check north points symmetry of a 2D variable (indices jpj to jpj-3), depending on which grid point (T,F,U,V) variable is defined.

Warning

- do not work with ORCA2 grid (T-point)

Author

J.Paul

Date

October, 2014 - Initial version

August, 2017

- if can't find latitude variable, assume there is a north fold
- do not use latitude variable to get pivot (to avoid mistake with regular grid)

Parameters

<i>in</i>	<i>td_file</i>	file structure
-----------	----------------	----------------

Returns

pivot point index

11.46.1.2 integer(i4) function grid::grid_get_pivot::grid_get_pivot_mpp (type(tmpp), intent(in) *td_mpp*)

This function compute NEMO pivot point index from input mpp variable.

Parameters

<i>in</i>	<i>td_mpp</i>	- F-point : 0 • T-point : 1
-----------	---------------	--------------------------------

check north points symmetry of a 2D variable (indices jpj to jpj-3), depending on which grid point (T,F,U,V) variable is defined.

Warning

- do not work with ORCA2 grid (T-point)

Author

J.Paul

Date

October, 2014 - Initial version

August, 2017

- if can't find latitude variable, assume there is a north fold
- do not use latitude variable to get pivot (to avoid mistake with regular grid)

Parameters

<i>in</i>	<i>td_mpp</i>	mpp file structure
-----------	---------------	--------------------

Returns

pivot point index

11.46.1.3 integer(i4) function grid::grid_get_pivot::grid__get_pivot_var (type(tvar), intent(in) *td_var*)

This function compute NEMO pivot point index of the input variable.

Parameters

<i>in</i>	<i>td_var</i>	- F-point : 0 • T-point : 1
-----------	---------------	--------------------------------

check north points of latitude grid (indices *npj* to *npj-3*) depending on which grid point (T,F,U,V) variable is defined

Note

variable must be at least 2D variable, and should not be coordinate variable (i.e lon, lat)

Warning

- do not work with ORCA2 grid (T-point)

Author

J.Paul

Date

November, 2013 - Initial version

September, 2014

- add dummy loop in case variable not over right point.

October, 2014

- work on variable structure instead of file structure

Parameters

<i>in</i>	<i>td_lat</i>	latitude variable structure
<i>in</i>	<i>td_var</i>	variable structure

Returns

pivot point index

The documentation for this interface was generated from the following file:

- src/grid.f90

11.47 grid_hgr Module Reference

This module manage Horizontal grid.

Data Types

- type `tnamh`

Public Member Functions

- subroutine, public `grid_hgr_init` (`jpi`, `npj`, `jpk`, `ld_domcfg`)
This subroutine initialise hgr structure.
- subroutine, public `grid_hgr_clean` (`ld_domcfg`)
This subroutine clean hgr structure.
- type(`tnamh`) function, public `grid_hgr_nam` (`cd_coord`, `id_perio`, `cd_namelist`)
This function initialise hgr namelist structure.
- subroutine, public `grid_hgr_fill` (`td_nam`, `jpi`, `npj`, `ld_domcfg`)
This subroutine fill horizontal mesh (hgr structure)

Public Attributes

- type(`tvar`), save, public `tg_tmask`
- type(`tvar`), save, public `tg_umask`
- type(`tvar`), save, public `tg_vmask`
- type(`tvar`), save, public `tg_fmask`
- type(`tvar`), save, public `tg_ssmask`
- type(`tvar`), save, public `tg_glamt`
- type(`tvar`), save, public `tg_glamu`
- type(`tvar`), save, public `tg_glamv`
- type(`tvar`), save, public `tg_glamf`
- type(`tvar`), save, public `tg_gphit`
- type(`tvar`), save, public `tg_gphiu`
- type(`tvar`), save, public `tg_gphiv`
- type(`tvar`), save, public `tg_gphif`
- type(`tvar`), save, public `tg_e1t`
- type(`tvar`), save, public `tg_e1u`
- type(`tvar`), save, public `tg_e1v`
- type(`tvar`), save, public `tg_e1f`
- type(`tvar`), save, public `tg_e2t`
- type(`tvar`), save, public `tg_e2u`
- type(`tvar`), save, public `tg_e2v`
- type(`tvar`), save, public `tg_e2f`
- type(`tvar`), save, public `tg_ff_t`
- type(`tvar`), save, public `tg_ff_f`
- type(`tvar`), save, public `tg_gcost`
- type(`tvar`), save, public `tg_gcosu`
- type(`tvar`), save, public `tg_gcosv`
- type(`tvar`), save, public `tg_gcosf`
- type(`tvar`), save, public `tg_gshint`
- type(`tvar`), save, public `tg_gsinu`
- type(`tvar`), save, public `tg_gsinv`
- type(`tvar`), save, public `tg_gsinf`

11.47.1 Detailed Description

This module manage Horizontal grid.

**** Purpose :** Compute the geographical position (in degree) of the model grid-points, the horizontal scale factors (in meters) and the Coriolis factor (in s-1).

**** Method :** The geographical position of the model grid-points is defined from analytical functions, fslam and fsphi, the derivatives of which gives the horizontal scale factors e1,e2. Defining two function fslam and fsphi and their derivatives in the two horizontal directions (fse1 and fse2), the model grid-point position and scale factors are given by:

- t-point:
 - $glamt(i,j) = fslam(i, j) \quad e1t(i,j) = fse1(i, j)$
 - $gphit(i,j) = fsphi(i, j) \quad e2t(i,j) = fse2(i, j)$
- u-point:
 - $glamu(i,j) = fslam(i+1/2, j) \quad e1u(i,j) = fse1(i+1/2, j)$
 - $gphiu(i,j) = fsphi(i+1/2, j) \quad e2u(i,j) = fse2(i+1/2, j)$
- v-point:
 - $glamv(i,j) = fslam(i, j+1/2) \quad e1v(i,j) = fse1(i, j+1/2)$
 - $gphiv(i,j) = fsphi(i, j+1/2) \quad e2v(i,j) = fse2(i, j+1/2)$
- f-point:
 - $glamf(i,j) = fslam(i+1/2, j+1/2) \quad e1f(i,j) = fse1(i+1/2, j+1/2)$
 - $gphif(i,j) = fsphi(i+1/2, j+1/2) \quad e2f(i,j) = fse2(i+1/2, j+1/2)$

Where fse1 and fse2 are defined by:

- $fse1(i,j) = ra * rad * \sqrt{(\cos(\phi) \, di(fslam))^{**2} - di(fsphi)^{**2}}(i,j)$
- $fse2(i,j) = ra * rad * \sqrt{(\cos(\phi) \, dj(fslam))^{**2} - dj(fsphi)^{**2}}(i,j)$

The coriolis factor is given at z-point by:

- $ff = 2.*\omega*\sin(gphif)$ (in s-1)

This routine is given as an example, it must be modified following the user s desiderata. nevertheless, the output as well as the way to compute the model grid-point position and horizontal scale factors must be respected in order to insure second order accuracy schemes.

Note

If the domain is periodic, verify that scale factors are also periodic, and the coriolis term again.

**** Action :**

- define glamt, glamu, glamv, glamf: longitude of t-, u-, v- and f-points (in degree)
- define gphit, gphiu, gphiv, gphif: latitude of t-, u-, v- and f-points (in degree)
- define e1t, e2t, e1u, e2u, e1v, e2v, e1f, e2f: horizontal scale factors (in meters) at t-, u-, v-, and f-points.
- define ff: coriolis factor at f-point

References : Marti, Madec and Delecluse, 1992, JGR Madec, Imbard, 1996, Clim. Dyn.

Author

G, Madec

Date

March, 1988 - Original code

January, 1996

- terrain following coordinates

February, 1997

- print mesh informations

November, 1999

- M. Imbard : NetCDF format with IO-IPSL

August, 2000

- D. Ludicone : Reduced section at Bab el Mandeb

September, 2001

- M. Levy : eel config: grid in km, beta-plane

August, 2002

- G. Madec : F90: Free form and module, namelist

January, 2004

- A.M. Treguier, J.M. Molines : Case 4 (Mercator mesh) use of parameters in par_CONFIG-Rxx.h90, not in namelist

May, 2004

- A. Koch-Larrouy : Add Gyre configuration

February, 2011

- G. Madec : add cell surface (e1e2t)

September, 2015

- J, Paul : rewrite to SIREN format from

Id:

domhgr.F90 5506 2015-06-29 15:19:38Z clevy

Date

October, 2016

- J, Paul : update from trunk (revision 6961): add wetting and drying, ice sheet coupling..
- J, Paul : compute coriolis factor at f-point and at t-point
- J, Paul : do not use anymore special case for ORCA grid

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.47.2 Member Function/Subroutine Documentation

11.47.2.1 subroutine, public grid_hgr::grid_hgr_clean (logical, intent(in) *ld_domcfg*)

This subroutine clean hgr structure.

Parameters

in	<i>ld_domcfg</i>
----	------------------

Author

J.Paul

Date

September, 2015 - Initial version

11.47.2.2 subroutine, public grid_hgr::grid_hgr_fill (type(tnamh), intent(in) *td_nam*, integer(i4), intent(in) *jpi*, integer(i4), intent(in) *npj*, logical, intent(in) *ld_domcfg*)

This subroutine fill horizontal mesh (hgr structure)

Parameters

in	<i>td_nam</i>
----	---------------

Author

J.Paul

Date

September, 2015 - Initial version

Parameters

in	<i>td_nam</i>
in	<i>jpi</i>
in	<i>npj</i>

11.47.2.3 subroutine, public grid_hgr::grid_hgr_init (integer(i4), intent(in) *jpi*, integer(i4), intent(in) *npj*, integer(i4), intent(in) *jpj*, logical, intent(in) *ld_domcfg*)

This subroutine initialise hgr structure.

Parameters

in	<i>jpi</i>
----	------------

Author

J.Paul

Date

September, 2015 - Initial version

Parameters

in	jpi	
in	npj	

11.47.2.4 `type(tnamh)` function, public `grid_hgr::grid_hgr_nam` (`character(len=*)`, `intent(in) cd_coord`, `integer(i4)`, `intent(in) id_perio`, `character(len=*)`, `intent(in) cd_namelist`)

This function initialise hgr namelist structure.

Parameters

in	cd_coord	
----	----------	--

Author

J.Paul

Date

September, 2015 - Initial version

Parameters

in	cd_coord	
in	id_perio	
in	cd_namelist	

Returns

hgr namelist structure

The documentation for this module was generated from the following file:

- `src/grid_hgr.f90`

11.48 grid_zgr Module Reference

This module manage Vertical grid.

Data Types

- type `tnamz`

Public Member Functions

- subroutine, public `grid_zgr_init` (`jpi`, `npj`, `jpk`, `ld_sco`)
This subroutine initialise global variable needed to compute vertical mesh.
- subroutine, public `grid_zgr_clean` (`ld_sco`)
This subroutine clean hgr structure.
- type(`tnamz`) function, public `grid_zgr_nam` (`cd_coord`, `id_perio`, `cd_namelist`)
This function initialise zgr namelist structure.
- subroutine, public `grid_zgr_fill` (`td_nam`, `jpi`, `npj`, `jpk`, `td_bathy`, `td_risfdep`)

This subroutine fill vertical mesh.

- subroutine, public [grid_zgr_zps_init](#) (jpi, jpj)

This subroutine initialise global variable needed to compute vertical mesh.

- subroutine, public [grid_zgr_zps_clean](#) ()
- subroutine, public [grid_zgr_sco_init](#) (jpi, jpj)

This subroutine initialise global variable needed to compute vertical mesh.

- subroutine, public [grid_zgr_sco_clean](#) ()
- subroutine, public [grid_zgr_sco_stiff](#) (td_nam, jpi, jpj, jpk)

This subroutine stretch the s-coordinate system.

Public Attributes

- type(tvar), save, public **tg_gdepw_1d**
- type(tvar), save, public **tg_gdept_1d**
- type(tvar), save, public **tg_e3w_1d**
- type(tvar), save, public **tg_e3t_1d**
- type(tvar), save, public **tg_e3tp**
- type(tvar), save, public **tg_e3wp**
- type(tvar), save, public **tg_rx1**
- type(tvar), save, public **tg_mbathy**
- type(tvar), save, public **tg_misfdep**
- type(tvar), save, public **tg_gdept_0**
- type(tvar), save, public **tg_gdepw_0**
- type(tvar), save, public **tg_e3t_0**
- type(tvar), save, public **tg_e3u_0**
- type(tvar), save, public **tg_e3v_0**
- type(tvar), save, public **tg_e3w_0**
- type(tvar), save, public **tg_e3f_0**
- type(tvar), save, public **tg_e3uw_0**
- type(tvar), save, public **tg_e3vw_0**
- type(tvar), save, public **tg_mbkt**
- type(tvar), save, public **tg_mikt**
- type(tvar), save, public **tg_hbatt**
- type(tvar), save, public **tg_hbatu**
- type(tvar), save, public **tg_hbatv**
- type(tvar), save, public **tg_hbatf**
- type(tvar), save, public **tg_gsigt**
- type(tvar), save, public **tg_gsigw**
- type(tvar), save, public **tg_gsi3w**
- type(tvar), save, public **tg_esigt**
- type(tvar), save, public **tg_esigw**

11.48.1 Detailed Description

This module manage Vertical grid.

** Purpose : set the depth of model levels and the resulting vertical scale factors.

** Method :

- reference 1D vertical coordinate (gdep._1d, e3._1d)
- read/set ocean depth and ocean levels (bathy, mbathy)
- vertical coordinate (gdep., e3.) depending on the coordinate chosen :

- In_zco=T z-coordinate
- In_zps=T z-coordinate with partial steps
- In_zco=T s-coordinate

** Action : define gdep., e3., mbathy and bathy

Author

G, Madec

Date

December, 1995 - Original code : s vertical coordinate

July, 1997

- lbc_Ink call

September, 2002

- A. Bozec, G. Madec : F90: Free form and module

September, 2002

- A. de Miranda : rigid-lid + islands

August, 2003

- G. Madec : Free form and module

October, 2005

- A. Beckmann : modifications for hybrid s-coordinates & new stretching function

April, 2006

- R. Benshila, G. Madec : add zgr_zco

June, 2008

- G. Madec : insertion of domzgr_zps.h90 & condng style

July, 2009

- R. Benshila : Suppression of rigid-lid option

November, 2011

- G. Madec : add mbk. arrays associated to the deepest ocean level

August, 2012

- J. Siddorn : added Siddorn and Furner stretching function

December, 2012

- R. Bourdalle-Badie and G. Reffray : modify C1D case

November, 2014

- P. Mathiot and C. Harris : add ice shelf capability

November, 2015

- H. Liu : Modifications for Wetting/Drying

October, 2016

- J, Paul : update from trunk (revision 6961): add wetting and drying, ice sheet coupling..
- J, Paul : do not use anymore special case for ORCA grid.

November, 2016

- J, Paul : vertical scale factors e3. = dk[gdep] or old definition

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.48.2 Member Function/Subroutine Documentation

11.48.2.1 subroutine, public grid_zgr::grid_zgr_clean (logical, intent(in) *ld_sco*)

This subroutine clean hgr structure.

Parameters

in	<i>ld_sco</i>
----	---------------

Author

J.Paul

Date

September, 2015 - Initial version

Parameters

in	<i>ld_sco</i>
----	---------------

11.48.2.2 subroutine, public grid_zgr::grid_zgr_fill (type(tnamz), intent(in) *td_nam*, integer(i4), intent(in) *jpi*, integer(i4), intent(in) *npj*, integer(i4), intent(in) *jpk*, type(tvar), intent(inout) *td_bathy*, type(tvar), intent(inout) *td_risfdep*)

This subroutine fill vertical mesh.

Parameters

in	<i>td_nam</i>
----	---------------

Author

J.Paul

Date

September, 2015 - Initial version

October, 2016

- ice shelf cavity

Parameters

in	<i>td_nam</i>
in	<i>jpi</i>
in	<i>npj</i>
in	<i>jpk</i>
in	<i>td_bathy</i>
in	<i>td_risfdep</i>

11.48.2.3 subroutine, public grid_zgr::grid_zgr_init (integer(i4), intent(in) *jpi*, integer(i4), intent(in) *npj*, integer(i4), intent(in) *jpk*, logical, intent(in) *ld_sco*)

This subroutine initialise global variable needed to compute vertical mesh.

Parameters

in	<i>jpi</i>
----	------------

Author

J.Paul

Date

September, 2015 - Initial version

Parameters

in	<i>jpi</i>	
in	<i>npj</i>	
in	<i>npk</i>	
in	<i>ld_sco</i>	

11.48.2.4 `type(tnamz) function, public grid_zgr::grid_zgr_nam (character(len=*), intent(in) cd_coord, integer(i4), intent(in) id_perio, character(len=*), intent(in) cd_namelist)`

This function initialise zgr namelist structure.

Parameters

in	<i>cd_coord</i>	
----	-----------------	--

Author

J.Paul

Date

September, 2015 - Initial version

Parameters

in	<i>cd_coord</i>	
in	<i>id_perio</i>	
in	<i>cd_namelist</i>	

Returns

hgr namelist structure

11.48.2.5 `subroutine, public grid_zgr::grid_zgr_sco_clean ()`

This subroutine clean structure

Author

J.Paul

Date

September, 2015 - Initial version

11.48.2.6 `subroutine, public grid_zgr::grid_zgr_sco_init (integer(i4), intent(in) jpi, integer(i4), intent(in) npj)`

This subroutine initialise global variable needed to compute vertical mesh.

Parameters

in	<i>jpi</i>
----	------------

Author

J.Paul

Date

September, 2015 - Initial version

Parameters

in	<i>jpi</i>
in	<i>npj</i>

11.48.2.7 subroutine, public grid_zgr::grid_zgr_sco_stiff (type(tnamz), intent(in) *td_nam*, integer(i4), intent(in) *jpi*, integer(i4), intent(in) *npj*, integer(i4), intent(in) *jpj*, integer(i4), intent(in) *jpk*)

This subroutine stretch the s-coordinate system.

Parameters

in	<i>td_nam</i>	** Method : s-coordinate stretch
----	---------------	----------------------------------

Reference : Madec, Lott, Delecluse and Crepon, 1996. JPO, 26, 1393-1408.

Author

J.Paul

Date

September, 2015 - rewrite from domain (dom_stiff)

Parameters

in	<i>td_nam</i>
in	<i>jpi</i>
in	<i>npj</i>
in	<i>jpj</i>
in	<i>jpk</i>

11.48.2.8 subroutine, public grid_zgr::grid_zgr_zps_clean ()

This subroutine clean hgr structure

Author

J.Paul

Date

September, 2015 - Initial version

11.48.2.9 subroutine, public grid_zgr::grid_zgr_zps_init (integer(i4), intent(in) *jpi*, integer(i4), intent(in) *npj*)

This subroutine initialise global variable needed to compute vertical mesh.

Parameters

in	jpi
----	-----

Author

J.Paul

Date

September, 2015 - Initial version

Parameters

in	jpi
in	jpj

The documentation for this module was generated from the following file:

- src/grid_zgr.f90

11.49 interp Module Reference

This module manage interpolation on regular grid.

Data Types

- interface [interp_detect](#)
- interface [interp_fill_value](#)
- type [tinterp](#)

Public Member Functions

- subroutine, public [interp_create_mixed_grid](#) (td_var, td_mix, id_rho)
This subroutine create mixed grid.
- subroutine, public [interp_clean_mixed_grid](#) (td_mix, td_var, id_rho, id_offset)
This subroutine remove points added on mixed grid to compute interpolation. And save interpolated value over domain.

11.49.1 Detailed Description

This module manage interpolation on regular grid.

Interpolation method to be used is specify inside variable structure, as array of string character.

- td_var%c_interp(1) string character is the interpolation name choose between:
 - 'nearest'
 - 'cubic'
 - 'linear'
- td_var%c_interp(2) string character is an operation to be used on interpolated value.
operation have to be multiplication '*' or division '/'.
coefficient have to be refinement factor following i-direction 'rhoi', j-direction 'rhoj', or k-direction 'rhok'.
Examples: '*rhoi', '/rhoj'.

Note

Those informations are read from namelist or variable configuration file (default).
 Interpolation method could be specify for each variable in namelist *namvar*, defining string character *cn_varinfo*.
 Example:

- `cn_varinfo='varname1:int=cubic/rhoi', 'varname2:int=linear'`

to create mixed grid (with coarse grid point needed to compute interpolation):

```
CALL interp_create_mixed_grid( td_var, td_mix [,id_rho] )
```

- `td_var` is coarse grid variable (should be extrapolated)
- `td_mix` is mixed grid variable structure [output]
- `id_rho` is array of refinement factor [optional]

to detected point to be interpolated:

```
il_detect(:, :, :) = interp_detect( td_mix [,id_rho] )
```

- `il_detect(:, :, :)` is 3D array of detected point to be interpolated
- `td_mix` is mixed grid variable
- `id_rho` is array of refinement factor [optional]

to interpolate variable value:

```
CALL interp_fill_value( td_var [,id_rho] [,id_offset] )
```

- `td_var` is variable structure
- `id_rho` is array of refinement factor [optional]
- `id_offset` is array of offset between fine and coarse grid [optional]

to clean mixed grid (remove points added on mixed grid to compute interpolation):

```
CALL interp_clean_mixed_grid( td_mix, td_var, id_rho )
```

- `td_mix` is mixed grid variable structure
- `td_var` is variable structure [output]
- `id_rho` is array of refinement factor [optional]
- `id_offset` is array of offset between fine and coarse grid [optional]

Note

It use to work on ORCA grid, as we work only with grid indices.

Warning

due to the use of second derivative when using cubic interpolation you should add at least 2 extrabands.

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header
- use interpolation method modules

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.49.2 Member Function/Subroutine Documentation

11.49.2.1 subroutine, public `interp::interp_clean_mixed_grid (type(tvar), intent(in) td_mix, type(tvar), intent(out) td_var, integer(i4), dimension(:), intent(in) id_rho, integer(i4), dimension(2,2), intent(in) id_offset)`

This subroutine remove points added on mixed grid to compute interpolation. And save interpolated value over domain.

Parameters

<i>in</i>	<i>td_mix</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- use offset to save useful domain

Parameters

<i>in</i>	<i>td_mix</i>	mixed grid variable structure
<i>out</i>	<i>td_var</i>	variable structure
<i>in</i>	<i>id_rho</i>	array of refinement factor (default 1)
<i>in</i>	<i>id_offset</i>	2D array of offset between fine and coarse grid

11.49.2.2 subroutine, public `interp::interp_create_mixed_grid (type(tvar), intent(in) td_var, type(tvar), intent(out) td_mix, integer(i4), dimension(:), intent(in), optional id_rho)`

This subroutine create mixed grid.

Parameters

<i>in</i>	<i>td_var</i>
-----------	---------------

Created grid is fine resolution grid. First and last point are coarse grid point.

A special case is done for even refinement on ARAKAWA-C grid.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_var</i>	coarse grid variable (should be extrapolated)
out	<i>td_mix</i>	mixed grid variable
in	<i>id_rho</i>	array of refinement factor (default 1)

The documentation for this module was generated from the following file:

- src/interp.f90

11.50 interp_cubic Module Reference

This module manage cubic interpolation on regular grid.

Public Member Functions

- subroutine, public [interp_cubic_fill](#) (dd_value, dd_fill, id_detect, id_rho, ld_even, ld_discont)

This subroutine compute horizontal cubic interpolation on 4D array of value.

11.50.1 Detailed Description

This module manage cubic interpolation on regular grid.

to compute cubic interpolation:

```
CALL interp_cubic_fill(dd_value, dd_fill, id_detect, id_rho, ld_even [,ld_discont] )
```

- dd_value is 2D array of variable value
- dd_fill is the FillValue of variable
- id_detect is 2D array of point to be interpolated (see interp module)
- id_rho is array of refinement factor
- ld_even indicates even refinement or not
- ld_discont indicates longitudinal discontinuity (-180°/180°, 0°/360°) or not

Author

J.Paul

Date

September, 2014 -Initial version
 June, 2015

- use math module

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.50.2 Member Function/Subroutine Documentation

11.50.2.1 subroutine, public `interp_cubic::interp_cubic_fill (real(dp), dimension(:, :, :, :), intent(inout) dd_value, real(dp), intent(in) dd_fill, integer(i4), dimension(:, :, :), intent(inout) id_detect, integer(i4), dimension(:), intent(in) id_rho, logical, dimension(:), intent(in) id_even, logical, intent(in), optional id_discont)`

This subroutine compute horizontal cubic interpolation on 4D array of value.

Parameters

<code>in, out</code>	<code>dd_value</code>
----------------------	-----------------------

Author

J.Paul

Date

September, 2014 - Initial Version
 July, 2015

- reinitialise detect array for each level

Parameters

<code>in, out</code>	<code>dd_value</code>	2D array of variable value
<code>in</code>	<code>dd_fill</code>	FillValue of variable
<code>in, out</code>	<code>id_detect</code>	2D array of point to be interpolated
<code>in</code>	<code>id_rho</code>	array of refinement factor
<code>in</code>	<code>id_even</code>	even refinement or not
<code>in</code>	<code>id_discont</code>	longitudinal discontinuity (-180°/180°, 0°/360°) or not

The documentation for this module was generated from the following file:

- `src/interp_cubic.f90`

11.51 interp::interp_detect Interface Reference**Public Member Functions**

- `integer(i4) function, dimension(td_mix%t_dim(1)%i_len,td_mix%t_dim(2)%i_len,td_mix%t_dim(3)%i_len) interp__detect_wrapper (td_mix, id_rho)`

This function detected point to be interpolated.

11.51.1 Member Function/Subroutine Documentation

11.51.1.1 `integer(i4) function, dimension(td_mix%t_dim(1)%i_len, td_mix%t_dim(2)%i_len, td_mix%t_dim(3)%i_len)`
`interp::interp_detect::interp__detect_wrapper (type(tvar), intent(in) td_mix, integer(i4), dimension(:), intent(in),`
`optional id_rho)`

This function detected point to be interpolated.

Parameters

<code>in</code>	<code>td_mix</code>
-----------------	---------------------

Actually it checks, the number of dimension used for this variable and launch `interp__detect` which detected point to be interpolated.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in</code>	<code>td_mix</code>	mixed grid variable (to interpolate)
<code>in</code>	<code>id_rho</code>	array of refinement factor

Returns

3D array of detected point to be interpolated

The documentation for this interface was generated from the following file:

- `src/interp.f90`

11.52 `interp::interp_fill_value` Interface Reference

Public Member Functions

- subroutine [`interp__fill_value_wrapper`](#) (`td_var`, `id_rho`, `id_offset`)
This subroutine interpolate variable value.

11.52.1 Member Function/Subroutine Documentation

11.52.1.1 subroutine `interp::interp_fill_value::interp__fill_value_wrapper` (`type(tvar)`, `intent(inout) td_var`, `integer(i4)`, `dimension(:)`, `intent(in)`, optional `id_rho`, `integer(i4)`, `dimension(:,:)`, `intent(in)`, optional `id_offset`)

This subroutine interpolate variable value.

Parameters

<code>in, out</code>	<code>td_var</code>
----------------------	---------------------

Actually it checks, the number of dimension used for this variable and launch `interp__fill_value`.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>id_rho</i>	array of refinement factor
in	<i>id_offset</i>	2D array of offset between fine and coarse grid

The documentation for this interface was generated from the following file:

- src/interp.f90

11.53 interp_linear Module Reference

This module manage linear interpolation on regular grid.

Public Member Functions

- subroutine, public [interp_linear_fill](#) (dd_value, dd_fill, id_detect, id_rho, ld_even, ld_discont)
This subroutine compute horizontal linear interpolation on 4D array of value.

11.53.1 Detailed Description

This module manage linear interpolation on regular grid.

to compute linear interpolation:

```
CALL interp_linear_fill(dd_value, dd_fill, id_detect, id_rho, ld_even [,ld_discont] )
```

- dd_value is 2D array of variable value
- dd_fill is the FillValue of variable
- id_detect is 2D array of point to be interpolated (see interp module)
- id_rho is array of refinement factor
- ld_even indicates even refinement or not
- ld_discont indicates longitudinal discontinuity (-180°/180°, 0°/360°) or not

Author

J.Paul

Date

September, 2014 - Initial version

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.53.2 Member Function/Subroutine Documentation

11.53.2.1 subroutine, public `interp_linear::interp_linear_fill (real(dp), dimension(:,:,:), intent(inout) dd_value, real(dp), intent(in) dd_fill, integer(i4), dimension(:,:,:), intent(inout) id_detect, integer(i4), dimension(:), intent(in) id_rho, logical, dimension(:), intent(in) ld_even, logical, intent(in), optional ld_discont)`

This subroutine compute horizontal linear interpolation on 4D array of value.

Parameters

<i>in, out</i>	<i>dd_value</i>
----------------	-----------------

Author

J.Paul

Date

September, 2014 - Initial Version

July, 2015 - reinitialise detect array for each level

Parameters

<i>in, out</i>	<i>dd_value</i>	2D array of variable value
<i>in</i>	<i>dd_fill</i>	FillValue of variable
<i>in, out</i>	<i>id_detect</i>	2D array of point to be interpolated
<i>in</i>	<i>id_rho</i>	array of refinement factor
<i>in</i>	<i>ld_even</i>	even refinement or not
<i>in</i>	<i>ld_discont</i>	longitudinal discontinuity (-180°/180°, 0°/360°) or not

The documentation for this module was generated from the following file:

- src/interp_linear.f90

11.54 interp_nearest Module Reference

This module manage nearest interpolation on regular grid.

Public Member Functions

- subroutine, public [interp_nearest_fill](#) (*dd_value*, *id_detect*, *id_rho*)
This subroutine compute horizontal nearest interpolation on 4D array of value.

11.54.1 Detailed Description

This module manage nearest interpolation on regular grid.

to compute nearest interpolation:

```
CALL interp_nearest_fill(dd_value, dd_fill, id_detect, id_rho, ld_even [,ld_discont] )
```

- *dd_value* is 2D array of variable value
- *dd_fill* is the FillValue of variable
- *id_detect* is 2D array of point to be interpolated (see interp module)
- *id_rho* is array of refinement factor
- *ld_even* indicates even refinement or not
- *ld_discont* indicates longitudinal discontinuity (-180°/180°, 0°/360°) or not

Author

J.Paul

Date

September, 2014 - Initial version

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.54.2 Member Function/Subroutine Documentation

11.54.2.1 subroutine, public `interp_nearest::interp_nearest_fill (real(dp), dimension(:, :, :, :), intent(inout) dd_value, integer(i4), dimension(:, :, :), intent(inout) id_detect, integer(i4), dimension(:), intent(in) id_rho)`

This subroutine compute horizontal nearest interpolation on 4D array of value.

Parameters

<code>in, out</code>	<code>dd_value</code>
----------------------	-----------------------

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

<code>in, out</code>	<code>dd_value</code>	2D array of variable value
<code>in, out</code>	<code>id_detect</code>	2D array of point to be interpolated
<code>in</code>	<code>id_rho</code>	array of refinement factor

The documentation for this module was generated from the following file:

- `src/interp_nearest.f90`

11.55 iom Module Reference

Input/Output manager : Library to read input files.

Data Types

- interface [iom_read_att](#)
- interface [iom_read_dim](#)
- interface [iom_read_var](#)

Public Member Functions

- subroutine, public [iom_open](#) (`td_file`)
This function open a file in read or write mode.

- subroutine, public `iom_create` (td_file)
This subroutine create a file.
- subroutine, public `iom_close` (td_file)
This subroutine close file.
- subroutine, public `iom_write_file` (td_file, cd_dimorder)
This subroutine write file structure in an opened file.
- subroutine, public `iom_write_header` (td_file, cd_dimorder, td_dim)
This subroutine write header from file structure of an opened file.
- subroutine, public `iom_write_var` (td_file, cd_dimorder, id_start, id_count)
This subroutine write variables from file structure in an opened file.

11.55.1 Detailed Description

Input/Output manager : Library to read input files.

to open file:


```
CALL iom_open(td_file)
```

- td_file is file structure

to create file:

```
CALL iom_create(td_file)
```

- td_file is file structure

to write in file:

```
CALL iom_write_file(td_file)
```

to close file:

```
CALL iom_close(tl_file)
```

to read one dimension in file:

```
tl_dim = iom_read_dim(tl_file, id_dimid)
```

or

```
tl_dim = iom_read_dim(tl_file, cd_name)
```

- id_dimid is dimension id
- cd_name is dimension name

to read variable or global attribute in file:

```
tl_att = iom_read_att(tl_file, id_varid, id_attid)
```

or

```
tl_att = iom_read_att(tl_file, id_varid, cd_attname)
```

or

```
tl_att = iom_read_att(tl_file, cd_varname, id_attid)
```

or

```
tl_att = iom_read_att(tl_file, cd_varname, cd_attname)
```

- `id_varid` is variable id
- `id_attid` is attribute id
- `cd_attname` is attribute name
- `cd_varname` is variable name or standard name

to read one variable in file:

```
tl_var = iom_read_var(td_file, id_varid, [id_start, id_count])
```

or

```
tl_var = iom_read_var(td_file, cd_name, [id_start, [id_count,]])
```

- `id_varid` is variable id
- `cd_name` is variable name or standard name.
- `id_start` is a integer(4) 1D array of index from which the data values will be read [optional]
- `id_count` is a integer(4) 1D array of the number of indices selected along each dimension [optional]

Author

J.Paul

Date

November, 2013 - Initial Version
August, 2017

- permit to write header and variable independantly

Todo

- see `lbc_ink`
- see `goup netcdf4`

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.55.2 Member Function/Subroutine Documentation

11.55.2.1 subroutine, public `iom::iom_close (type(tfiler), intent(inout) td_file)`

This subroutine close file.

Parameters

<code>in, out</code>	<code>td_file</code>
----------------------	----------------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
----------------------	----------------------	----------------

11.55.2.2 subroutine, public `iom::iom_create (type(tfile), intent(inout) td_file)`

This subroutine create a file.

Parameters

<code>in, out</code>	<code>td_file</code>
----------------------	----------------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
----------------------	----------------------	----------------

11.55.2.3 subroutine, public `iom::iom_open (type(tfile), intent(inout) td_file)`

This function open a file in read or write mode.

Parameters

<code>in, out</code>	<code>td_file</code>
----------------------	----------------------

If try to open a file in write mode that did not exist, create it.

If file exist, get information about:

- the number of variables
- the number of dimensions
- the number of global attributes
- the ID of the unlimited dimension
- the file format and finally read dimensions.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
----------------------	----------------------	----------------

11.55.2.4 subroutine, public `iom::iom_write_file` (`type(tfile)`, `intent(inout) td_file`, `character(len=*)`, `intent(in)`, optional `cd_dimorder`)

This subroutine write file structure in an opened file.

Parameters

<code>in, out</code>	<code>td_file</code>	
----------------------	----------------------	--

optionally, you could specify dimension order (default 'xyzt')

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015 - add dimension order option

August, 2017

- split in `write_header` and `write_var`

Parameters

<code>in</code>	<code>td_file</code>	file structure
-----------------	----------------------	----------------

11.55.2.5 subroutine, public `iom::iom_write_header` (`type(tfile)`, `intent(inout) td_file`, `character(len=*)`, `intent(in)`, optional `cd_dimorder`, `type(tdim)`, `dimension(ip_maxdim)`, `intent(in)`, optional `td_dim`)

This subroutine write header from file structure of an opened file.

Parameters

<code>in, out</code>	<code>td_file</code>	
----------------------	----------------------	--

optionally, you could specify dimension order (default 'xyzt'), and dimension structure for netcdf case.

Author

J.Paul

Date

August, 2017 - Initial Version

Parameters

in, out	<i>td_file</i>	file structure
in	<i>cd_dimorder</i>	dimension order
in	<i>td_dim</i>	array of dimension structure

11.55.2.6 subroutine, public `iom::iom_write_var` (type(tfile), intent(inout) *td_file*, character(len=*) , intent(in), optional *cd_dimorder*, integer(i4), dimension(:), intent(in), optional *id_start*, integer(i4), dimension(:), intent(in), optional *id_count*)

This subroutine write variables from file structure in an opened file.

Parameters

in, out	<i>td_file</i>	
---------	----------------	--

Author

J.Paul

Date

August, 2017 - Initial Version

Parameters

in, out	<i>td_file</i>	file structure
in	<i>cd_dimorder</i>	dimension order
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

The documentation for this module was generated from the following file:

- src/iom.f90

11.56 iom_cdf Module Reference

NETCDF Input/Output manager : Library to read Netcdf input files.

Data Types

- interface [iom_cdf_fill_var](#)
- interface [iom_cdf_read_att](#)
- interface [iom_cdf_read_dim](#)
- interface [iom_cdf_read_var](#)

Public Member Functions

- subroutine, public [iom_cdf_open](#) (td_file)
This subroutine open a netcdf file in read or write mode.
- subroutine, public [iom_cdf_close](#) (td_file)
This subroutine close netcdf file.
- subroutine, public [iom_cdf_write_header](#) (td_file, cd_dimorder, td_dim)
This subroutine write file header in an opened netcdf file.
- subroutine, public [iom_cdf_write_var](#) (td_file, cd_dimorder, id_start, id_count)
This subroutine write variable(s) in an opened netcdf file.

11.56.1 Detailed Description

NETCDF Input/Output manager : Library to read Netcdf input files.

to open netcdf file:


```
CALL iom_cdf_open(td_file)
```

- `td_file` is file structure (see [file](#))

to write in netcdf file:

```
CALL iom_cdf_write_file(td_file)
```

to close netcdf file:

```
CALL iom_cdf_close(tl_file)
```

to read one dimension in netcdf file:

```
tl_dim = iom_cdf_read_dim(tl_file, id_dimid)
```

or

```
tl_dim = iom_cdf_read_dim(tl_file, cd_name)
```

- `id_dimid` is dimension id
- `cd_name` is dimension name

to read one attribute in netcdf file:

```
tl_att = iom_cdf_read_att(tl_file, id_varid, id_attid)
```

or

```
tl_att = iom_cdf_read_att(tl_file, id_varid, cd_name)
```

- `id_varid` is variable id
- `id_attid` is attribute id
- `cd_name` is attribute name

to read one variable in netcdf file:

```
tl_var = iom_cdf_read_var(td_file, id_varid, [id_start, id_count])
```

or

```
tl_var = iom_cdf_read_var(td_file, cd_name, [id_start, [id_count,]])
```

- `id_varid` is variable id
- `cd_name` is variable name
- `id_start` is a integer(4) 1D array of index from which the data values will be read [optional]
- `id_count` is a integer(4) 1D array of the number of indices selected along each dimension [optional]

Author

J.Paul

Date

November, 2013 - Initial Version

August, 2017

- permit to write header and variable independantly

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.56.2 Member Function/Subroutine Documentation**11.56.2.1 subroutine, public iom_cdf::iom_cdf_close (type(tfile), intent(inout) td_file)**

This subroutine close netcdf file.

Parameters

<i>in, out</i>	<i>td_file</i>
----------------	----------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_file</i>	file structure
----------------	----------------	----------------

11.56.2.2 subroutine, public iom_cdf::iom_cdf_open (type(tfile), intent(inout) td_file)

This subroutine open a netcdf file in read or write mode.

Parameters

<i>in, out</i>	<i>td_file</i>
----------------	----------------

if try to open a file in write mode that did not exist, create it.

if file already exist, get information about0:

- the number of variables
- the number of dimensions
- the number of global attributes
- the ID of the unlimited dimension
- the file format Finally it read dimensions, and 'longitude' variable to compute East-West overlap.

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2018

- write netcdf file as netcdf4

Parameters

<i>in, out</i>	<i>td_file</i>	file structure
----------------	----------------	----------------

11.56.2.3 subroutine, public `iom_cdf::iom_cdf_write_header` (`type(tfile)`, `intent(inout) td_file`, `character(len=*)`, `intent(in)`, optional `cd_dimorder`, `type(tdim)`, `dimension(ip_maxdim)`, `intent(in)`, optional `td_dim`)

This subroutine write file header in an opened netcdf file.

Parameters

<i>in, out</i>	<i>td_file</i>	
----------------	----------------	--

optionally, you could specify dimension order (default 'xyzt'), and/or dimension structure to be used.

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- add dimension order option

August, 2017

- split `write_file` into `write_header` and `write_var`
- add dimension structure as optional argument

September, 2017

- do not check variable dimension if dimension forced

Parameters

<i>in, out</i>	<i>td_file</i>	file structure
----------------	----------------	----------------

11.56.2.4 subroutine, public `iom_cdf::iom_cdf_write_var` (`type(tfile)`, `intent(inout) td_file`, `character(len=*)`, `intent(in)`, optional `cd_dimorder`, `integer(i4)`, `dimension(:)`, `intent(in)`, optional `id_start`, `integer(i4)`, `dimension(:)`, `intent(in)`, optional `id_count`)

This subroutine write variable(s) in an opened netcdf file.

Parameters

<i>in, out</i>	<i>td_file</i>	
----------------	----------------	--

optionally, you could specify dimension order (default 'xyzt')

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- add dimension order option

August, 2017

- add start and count array as optional argument

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
<code>in</code>	<code>td_var</code>	array of variable structure
<code>in</code>	<code>cd_dimorder</code>	dimension order
<code>in</code>	<code>id_start</code>	index in the variable from which the data values will be read
<code>in</code>	<code>id_count</code>	number of indices selected along each dimension

The documentation for this module was generated from the following file:

- `src/iom_cdf.f90`

11.57 iom_cdf::iom_cdf_fill_var Interface Reference

Public Member Functions

- subroutine `iom_cdf_fill_var_id` (`td_file`, `id_varid`, `id_start`, `id_count`)
- subroutine `iom_cdf_fill_var_name` (`td_file`, `cd_name`, `id_start`, `id_count`)
This subroutine fill variable value in an opened netcdf file, given variable name or standard name.
- subroutine `iom_cdf_fill_var_all` (`td_file`, `id_start`, `id_count`)
This subroutine fill all variable value from an opened netcdf file.

11.57.1 Member Function/Subroutine Documentation

11.57.1.1 subroutine `iom_cdf::iom_cdf_fill_var::iom_cdf_fill_var_all` (`type(tfile)`, `intent(inout) td_file`, `integer(i4)`, `dimension(:)`, `intent(in)`, optional `id_start`, `integer(i4)`, `dimension(:)`, `intent(in)`, optional `id_count`)

This subroutine fill all variable value from an opened netcdf file.

Parameters

<code>in, out</code>	<code>td_file</code>
----------------------	----------------------

Optionally, start indices and number of indices selected along each dimension could be specify in a 4 dimension array (`'x','y','z','t'`)

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_file</i>	file structure
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

11.57.1.2 subroutine `iom_cdf::iom_cdf_fill_var::iom_cdf_fill_var_id` (type(tfiler), intent(inout) *td_file*, integer(i4), intent(in) *id_varid*, integer(i4), dimension(:), intent(in), optional *id_start*, integer(i4), dimension(:), intent(in), optional *id_count*)

Parameters

in, out	<i>td_file</i>	file structure
in	<i>id_varid</i>	variable id
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

11.57.1.3 subroutine `iom_cdf::iom_cdf_fill_var::iom_cdf_fill_var_name` (type(tfiler), intent(inout) *td_file*, character(len=*) intent(in) *cd_name*, integer(i4), dimension(:), intent(in), optional *id_start*, integer(i4), dimension(:), intent(in), optional *id_count*)

This subroutine fill variable value in an opened netcdf file, given variable name or standard name.

Parameters

in, out	<i>td_file</i>	
---------	----------------	--

Optionally, start indices and number of indices selected along each dimension could be specify in a 4 dimension array ('x','y','z','t')

look first for variable name. If it doesn't exist in file, look for variable standard name.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_file</i>	file structure
in	<i>cd_name</i>	variable name or standard name
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

The documentation for this interface was generated from the following file:

- `src/iom_cdf.f90`

11.58 iom_cdf::iom_cdf_read_att Interface Reference

Public Member Functions

- type(tatt) function `iom_cdf__read_att_id` (`td_file`, `id_varid`, `id_attid`)
This function read variable or global attribute in an opened netcdf file, given attribute id.
- type(tatt) function `iom_cdf__read_att_name` (`td_file`, `id_varid`, `cd_name`)
This function read variable or global attribute in an opened netcdf file, given attribute name.

11.58.1 Member Function/Subroutine Documentation

11.58.1.1 `type(tatt) function iom_cdf::iom_cdf_read_att::iom_cdf__read_att_id (type(tfiler), intent(in) td_file, integer(i4), intent(in) id_varid, integer(i4), intent(in) id_attid)`

This function read variable or global attribute in an opened netcdf file, given attribute id.

Parameters

<i>in</i>	<i>td_file</i>	
-----------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_file</i>	file structure
<i>in</i>	<i>id_varid</i>	variable id. use NF90_GLOBAL to read global attribute in a file
<i>in</i>	<i>id_attid</i>	attribute id

Returns

attribute structure

11.58.1.2 `type(tatt) function iom_cdf::iom_cdf_read_att::iom_cdf__read_att_name (type(tfiler), intent(in) td_file, integer(i4), intent(in) id_varid, character(len=*), intent(in) cd_name)`

This function read variable or global attribute in an opened netcdf file, given attribute name.

Parameters

<i>in</i>	<i>td_file</i>	
-----------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

November 2017

- check if *cl_value* is not bug

Parameters

<i>in</i>	<i>td_file</i>	file structure
<i>in</i>	<i>id_varid</i>	variable id. use NF90_GLOBAL to read global attribute in a file
<i>in</i>	<i>cd_name</i>	attribute name

Returns

attribute structure

The documentation for this interface was generated from the following file:

- src/iom_cdf.f90

11.59 iom_cdf::iom_cdf_read_dim Interface Reference

Public Member Functions

- type(tdim) function [iom_cdf__read_dim_id](#) (td_file, id_dimid)
This function read one dimension in an opened netcdf file, given dimension id.
- type(tdim) function [iom_cdf__read_dim_name](#) (td_file, cd_name)
This function read one dimension in an opened netcdf file, given dimension name.

11.59.1 Member Function/Subroutine Documentation

11.59.1.1 type(tdim) function [iom_cdf::iom_cdf_read_dim::iom_cdf__read_dim_id](#) (type(tfile), intent(in) *td_file*, integer(i4), intent(in) *id_dimid*)

This function read one dimension in an opened netcdf file, given dimension id.

Parameters

in	<i>td_file</i>	
----	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

February, 2015 - create unused dimension, when reading dimension of length less or equal to zero

Parameters

in	<i>td_file</i>	file structure
in	<i>id_dimid</i>	dimension id

Returns

dimension structure

11.59.1.2 type(tdim) function [iom_cdf::iom_cdf_read_dim::iom_cdf__read_dim_name](#) (type(tfile), intent(in) *td_file*, character(len=*), intent(in) *cd_name*)

This function read one dimension in an opened netcdf file, given dimension name.

Parameters

in	<i>td_file</i>
----	----------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>cd_name</i>	dimension name

Returns

dimension structure

The documentation for this interface was generated from the following file:

- src/iom_cdf.f90

11.60 iom_cdf::iom_cdf_read_var Interface Reference

Public Member Functions

- type(tvar) function [iom_cdf__read_var_id](#) (td_file, id_varid, id_start, id_count)
This function read variable value in an opened netcdf file, given variable id.
- type(tvar) function [iom_cdf__read_var_name](#) (td_file, cd_name, id_start, id_count)
This function read variable value in an opened netcdf file, given variable name or standard name.

11.60.1 Member Function/Subroutine Documentation

11.60.1.1 type(tvar) function `iom_cdf::iom_cdf_read_var::iom_cdf__read_var_id (type(tfile), intent(in) td_file, integer(i4), intent(in) id_varid, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count)`

This function read variable value in an opened netcdf file, given variable id.

Parameters

in	<i>td_file</i>
----	----------------

Optionally, start indices and number of indices selected along each dimension could be specify in a 4 dimension array (*'x','y','z','t'*)

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>id_varid</i>	variable id
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

Returns

variable structure

11.60.1.2 `type(tvar) function iom_cdf::iom_cdf_read_var::iom_cdf__read_var_name (type(tfile), intent(in) td_file, character(len=*), intent(in), optional cd_name, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count)`

This function read variable value in an opened netcdf file, given variable name or standard name.

Parameters

in	<i>td_file</i>
----	----------------

Optionally, start indices and number of indices selected along each dimension could be specify in a 4 dimension array ('x','y','z','t')

look first for variable name. If it doesn't exist in file, look for variable standard name.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>cd_name</i>	variable name or standard name.
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

Returns

variable structure

The documentation for this interface was generated from the following file:

- `src/iom_cdf.f90`

11.61 iom_dom Module Reference

This module allow to read domain (defined as domain structure) in a mpp files.

Data Types

- interface [iom_dom_read_var](#)

Public Member Functions

- subroutine, public [iom_dom_open](#) (td_mpp, td_dom, id_perio, id_ew)
- subroutine, public [iom_dom_close](#) (td_mpp)

11.61.1 Detailed Description

This module allow to read domain (defined as domain structure) in a mpp files.

to read one variable in an mpp files over domain defined as domain structure:


```
tl_var=iom_dom_read_var( td_mpp, id_varid, td_dom )
```

or

```
tl_var=iom_dom_read_var( td_mpp, cd_name, td_dom )
```

- td_mpp is a mpp structure
- id_varid is a variable id
- cd_name is variable name or standard name
- td_dom is a domain structure

Author

J.Paul

Date

October, 2014 - Initial Version

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.61.2 Member Function/Subroutine Documentation

11.61.2.1 subroutine, public [iom_dom::iom_dom_close](#) (type(tmpp), intent(inout) *td_mpp*)

Parameters

in	<i>td_mpp</i>	mpp structure
----	---------------	---------------

11.61.2.2 subroutine, public [iom_dom::iom_dom_open](#) (type(tmpp), intent(inout) *td_mpp*, type(tdom), intent(in) *td_dom*, integer(i4), intent(in), optional *id_perio*, integer(i4), intent(in), optional *id_ew*)

Parameters

in, out	<i>td_mpp</i>	mpp structure
---------	---------------	---------------

The documentation for this module was generated from the following file:

- src/iom_dom.f90

11.62 iom_dom::iom_dom_read_var Interface Reference

Public Member Functions

- type(tvar) function `iom_dom__read_var_id` (td_mpp, id_varid, td_dom)
This function read variable value in opened mpp files, given variable id and domain structure.
- type(tvar) function `iom_dom__read_var_name` (td_mpp, cd_name, td_dom)
This function read variable value in opened mpp files, given variable name or standard name, and domain structure.

11.62.1 Member Function/Subroutine Documentation

11.62.1.1 type(tvar) function `iom_dom::iom_dom_read_var::iom_dom__read_var_id` (type(tmpp), intent(in) *td_mpp*, integer(i4), intent(in) *id_varid*, type(tdom), intent(in) *td_dom*)

This function read variable value in opened mpp files, given variable id and domain structure.

Parameters

<i>in</i>	<i>td_mpp</i>
-----------	---------------

Optionally start indices and number of point to be read could be specify. as well as East West overlap of the global domain.

Author

J.Paul

Date

October, 2014 - Initial Version

Parameters

<i>in</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>id_varid</i>	variable id
<i>in</i>	<i>td_dom</i>	domain structure

Returns

variable structure

11.62.1.2 type(tvar) function `iom_dom::iom_dom_read_var::iom_dom__read_var_name` (type(tmpp), intent(in) *td_mpp*, character(len=*) , intent(in) *cd_name*, type(tdom), intent(in) *td_dom*)

This function read variable value in opened mpp files, given variable name or standard name, and domain structure.

Parameters

<code>in</code>	<code>td_mpp</code>
-----------------	---------------------

Optionally start indices and number of point to be read could be specify. as well as East West overlap of the global domain.

look first for variable name. If it doesn't exist in file, look for variable standard name.

If variable name is not present, check variable standard name.

Author

J.Paul

Date

October, 2014 - Initial Version

May, 2019

- copy variable struct without array of value, then read array of value.

Parameters

<code>in</code>	<code>td_mpp</code>	mpp structure
<code>in</code>	<code>cd_name</code>	variable name
<code>in</code>	<code>td_dom</code>	domain structure

Returns

variable structure

The documentation for this interface was generated from the following file:

- `src/iom_dom.f90`

11.63 iom_mpp Module Reference

This module manage massively parallel processing Input/Output manager. Library to read/write mpp files.

Data Types

- interface [iom_mpp_read_var](#)

Public Member Functions

- subroutine, public [iom_mpp_open](#) (`td_mpp`, `id_perio`, `id_ew`)
This subroutine open files composing mpp structure to be used.
- subroutine, public [iom_mpp_create](#) (`td_mpp`)
This subroutine create files, composing mpp structure to be used, in write mode.
- subroutine, public [iom_mpp_close](#) (`td_mpp`)
This subroutine close files composing mpp structure.
- subroutine, public [iom_mpp_write_file](#) (`td_mpp`, `cd_dimorder`)
This subroutine write files composing mpp structure.
- subroutine [iom_mpp__write_var](#) (`td_mpp`, `cd_dimorder`)
This subroutine write variables from mpp structure in one output file.

11.63.1 Detailed Description

This module manage massively parallel processing Input/Output manager. Library to read/write mpp files.

to open mpp files (only file to be used (see mpp_get_use) will be open):


```
CALL iom_mpp_open(td_mpp)
```

- td_mpp is a mpp structure

to creates mpp files:

```
CALL iom_mpp_create(td_mpp)
```

- td_mpp is a mpp structure

to write in mpp files :

```
CALL iom_mpp_write_file(td_mpp)
```

- td_mpp is a mpp structure

to close mpp files:

```
CALL iom_mpp_close(td_mpp)
```

to read one variable in an mpp files:

```
tl_var=iom_mpp_read_var( td_mpp, id_varid, [id_start, id_count] [,id_ew] )
```

or

```
tl_var=iom_mpp_read_var( td_mpp, cd_name, [id_start, id_count] [,id_ew] )
```

- td_mpp is a mpp structure
- id_varid is a variable id
- cd_name is variable name or standard name
- id_start is a integer(4) 1D array of index from which the data values will be read [optional]
- id_count is a integer(4) 1D array of the number of indices selected along each dimension [optional]
- id_ew East West overlap [optional]

to fill variable value in mpp structure:

```
CALL iom_mpp_fill_var(td_mpp, id_varid, [id_start, id_count] [,id_ew] )
```

or

```
CALL iom_mpp_fill_var(td_mpp, cd_name, [id_start, id_count] [,id_ew] )
```

- td_mpp is mpp structure
- id_varid is variable id

- `cd_name` is variable name or standard name
- `id_start` is a integer(4) 1D array of index from which the data values will be read [optional]
- `id_count` is a integer(4) 1D array of the number of indices selected along each dimension [optional]
- `id_ew` East West overlap [optional]

to fill all variable in mpp structure:

```
CALL iom_mpp_fill_var(td_mpp, [id_start, id_count] [,id_ew] )
```

- `td_mpp` is mpp structure
- `id_start` is a integer(4) 1D array of index from which the data values will be read [optional]
- `id_count` is a integer(4) 1D array of the number of indices selected along each dimension [optional]
- `id_ew` East West overlap

to write files composong mpp strucutre:

```
CALL iom_mpp_write_file(td_mpp)
```

Author

J.Paul

Date

November, 2013 - Initial Version

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.63.2 Member Function/Subroutine Documentation

11.63.2.1 subroutine `iom_mpp::iom_mpp__write_var` (type(tmpp), intent(inout) *td_mpp*, character(len=*) , intent(in), optional *cd_dimorder*)

This subroutine write variables from mpp structure in one output file.

Parameters

<code>in, out</code>	<code>td_mpp</code>
----------------------	---------------------

optionally, you could specify the dimension order (default 'xyzt')

Author

J.Paul

Date

August, 2017 - Initial Version

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>cd_dimorder</i>	dimension order

11.63.2.2 subroutine, public iom_mpp::iom_mpp_close (type(tmpp), intent(inout) *td_mpp*)

This subroutine close files composing mpp structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_mpp</i>	mpp structure
-----------	---------------	---------------

11.63.2.3 subroutine, public iom_mpp::iom_mpp_create (type(tmpp), intent(inout) *td_mpp*)

This subroutine create files, composing mpp structure to be used, in write mode.

Parameters

<i>in, out</i>	<i>td_mpp</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
----------------	---------------	---------------

11.63.2.4 subroutine, public iom_mpp::iom_mpp_open (type(tmpp), intent(inout) *td_mpp*, integer(i4), intent(in), optional *id_perio*, integer(i4), intent(in), optional *id_ew*)

This subroutine open files composing mpp structure to be used.

Parameters

<i>in, out</i>	<i>td_mpp</i>	
----------------	---------------	--

If try to open a file in write mode that did not exist, create it.

If file already exist, get information about:

- the number of variables
- the number of dimensions
- the number of global attributes
- the ID of the unlimited dimension
- the file format and finally read dimensions.

Author

J.Paul

Date

November, 2013 - Initial Version

August, 2017

- handle use of domain decomposition for monoproc file

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
----------------	---------------	---------------

11.63.2.5 subroutine, public `iom_mpp::iom_mpp_write_file (type(tmpp), intent(inout) td_mpp, character(len=*) , intent(in), optional cd_dimorder)`

This subroutine write files composing mpp structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

optionally, you could specify the dimension order (default 'xyzt')

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- add dimension order option

August, 2017

- handle use of domain decomposition for monoproc file

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
----------------	---------------	---------------

in	<i>cd_dimorder</i>	dimension order
----	--------------------	-----------------

The documentation for this module was generated from the following file:

- src/iom_mpp.f90

11.64 iom_mpp::iom_mpp_read_var Interface Reference

Public Member Functions

- type(tvar) function [iom_mpp__read_var_id](#) (td_mpp, id_varid, id_start, id_count)
This function read variable value in opened mpp files, given variable id.
- type(tvar) function [iom_mpp__read_var_name](#) (td_mpp, cd_name, id_start, id_count)
This function read variable value in opened mpp files, given variable name or standard name.

11.64.1 Member Function/Subroutine Documentation

11.64.1.1 type(tvar) function `iom_mpp::iom_mpp_read_var::iom_mpp__read_var_id (type(tmpp), intent(in) td_mpp, integer(i4), intent(in) id_varid, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count)`

This function read variable value in opened mpp files, given variable id.

Parameters

in	<i>td_mpp</i>
----	---------------

Optionally start indices and number of point to be read could be specify. as well as East West overlap of the global domain.

Author

J.Paul

Date

November, 2013 - Initial Version
October, 2014

- use start and count array instead of domain structure.

Parameters

in	<i>td_mpp</i>	mpp structure
in	<i>id_varid</i>	variable id
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

Returns

variable structure

11.64.1.2 type(tvar) function `iom_mpp::iom_mpp_read_var::iom_mpp__read_var_name (type(tmpp), intent(in) td_mpp, character(len=*), intent(in) cd_name, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count)`

This function read variable value in opened mpp files, given variable name or standard name.

Parameters

in	<i>td_mpp</i>
----	---------------

Optionally start indices and number of point to be read could be specify. as well as East West overlap of the global domain.

look first for variable name. If it doesn't exist in file, look for variable standard name.

If variable name is not present, check variable standard name.

Author

J.Paul

Date

November, 2013 - Initial Version

October, 2014

- use start and count array instead of domain structure.

Parameters

in	<i>td_mpp</i>	mpp structure
in	<i>cd_name</i>	variable name
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

Returns

variable structure

The documentation for this interface was generated from the following file:

- src/iom_mpp.f90

11.65 iom::iom_read_att Interface Reference

Public Member Functions

- type(tatt) function [iom__read_att_varname_id](#) (td_file, cd_varname, id_attid)

This function read attribute (of variable or global) in an opened file, given variable name or standard name and attribute id.

- **given**
- **variable**
- **name**
- **or**
- **standard**
- **name**
- **and**
- **attribute**
- **id**

- type(tatt) function [iom__read_att_varid_id](#) (td_file, id_varid, id_attid)

This function read attribute (of variable or global) in an opened file, given variable id and attribute id.

- **given**
- **variable**
- **id**

- **and**
- **attribute**
- **id**
- type(tatt) function [iom__read_att_varname_name](#) (td_file, cd_varname, cd_attname)

This function read attribute (of variable or global) in an opened file, given variable name or standard name, and attribute name.
- **given**
- **variable**
- **name**
- **or**
- **standard**
- **name**
- **and**
- **attribute**
- **name**
- type(tatt) function [iom__read_att_varid_name](#) (td_file, id_varid, cd_attname)

This function read attribute (of variable or global) in an opened file, given variable id and attribute name.
- **given**
- **variable**
- **id**
- **and**
- **attribute**
- **name**

11.65.1 Member Function/Subroutine Documentation

11.65.1.1 type(tatt) function `iom::iom_read_att::iom__read_att_varid_id` (type(tfile), intent(in) *td_file*, integer(i4), intent(in) *id_varid*, integer(i4), intent(in) *id_attid*)

This function read attribute (of variable or global) in an opened file, given variable id and attribute id.

Parameters

<i>in</i>	<i>td_file</i>	
-----------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_file</i>	file structure
<i>in</i>	<i>id_varid</i>	variable id. use NF90_GLOBAL to read global attribute in a file
<i>in</i>	<i>id_attid</i>	attribute id

Returns

attribute structure

11.65.1.2 type(tatt) function `iom::iom_read_att::iom__read_att_varid_name` (type(tfile), intent(in) *td_file*, integer(i4), intent(in) *id_varid*, character(len=*) , intent(in) *cd_attname*)

This function read attribute (of variable or global) in an opened file, given variable id and attribute name.

Parameters

<i>in</i>	<i>td_file</i>	
-----------	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_file</i>	file structure
<i>in</i>	<i>id_varid</i>	variable id. use NF90_GLOBAL to read global attribute in a file
<i>in</i>	<i>cd_attname</i>	attribute name

Returns

attribute structure

11.65.1.3 `type(tatt) function iom::iom_read_att::iom__read_att_varname_id (type(tfile), intent(in) td_file, character(len=lc), intent(in) cd_varname, integer(i4), intent(in) id_attid)`

This function read attribute (of variable or global) in an opened file, given variable name or standard name and attribute id.

Parameters

<i>in</i>	<i>td_file</i>	
-----------	----------------	--

- to get global attribute use 'GLOBAL' as variable name.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_file</i>	file structure
<i>in</i>	<i>cd_varname</i>	variable name. use 'GLOBAL' to read global attribute in a file
<i>in</i>	<i>id_attid</i>	attribute id

Returns

attribute structure

11.65.1.4 `type(tatt) function iom::iom_read_att::iom__read_att_varname_name (type(tfile), intent(in) td_file, character(len=*), intent(in) cd_varname, character(len=*), intent(in) cd_attname)`

This function read attribute (of variable or global) in an opened file, given variable name or standard name, and attribute name.

Parameters

in	<i>td_file</i>
----	----------------

- to get global attribute use 'GLOBAL' as variable name.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>cd_varname</i>	variable name or standard name. use 'GLOBAL' to read global attribute in a file
in	<i>cd_attname</i>	attribute name

Returns

attribute structure

The documentation for this interface was generated from the following files:

- src/iom.f90

11.66 iom::iom_read_dim Interface Reference

Public Member Functions

- type(tdim) function [iom__read_dim_id](#) (td_file, id_dimid)
This function read one dimension in an opened file, given dimension id.
- type(tdim) function [iom__read_dim_name](#) (td_file, cd_name)
This function read one dimension in an opened netcdf file, given dimension name.

11.66.1 Member Function/Subroutine Documentation

11.66.1.1 type(tdim) function `iom::iom_read_dim::iom__read_dim_id (type(tfile), intent(in) td_file, integer(i4), intent(in) id_dimid)`

This function read one dimension in an opened file, given dimension id.

Parameters

in	<i>td_file</i>
----	----------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>id_dimid</i>	dimension id

Returns

dimension structure

11.66.1.2 `type(tdim) function iom::iom_read_dim::iom__read_dim_name (type(tfile), intent(in) td_file, character(len=*), intent(in) cd_name)`

This function read one dimension in an opened netcdf file, given dimension name.

Parameters

in	<i>td_file</i>	
----	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>cd_name</i>	dimension name

Returns

dimension structure

The documentation for this interface was generated from the following file:

- `src/iom.f90`

11.67 iom::iom_read_var Interface Reference

Public Member Functions

- `type(tvar) function iom__read_var_id (td_file, id_varid, id_start, id_count)`
This function read variable value in an opened file, given variable id.
- `type(tvar) function iom__read_var_name (td_file, cd_name, id_start, id_count)`
This function read variable value in an opened file, given variable name or standard name.

11.67.1 Member Function/Subroutine Documentation

11.67.1.1 `type(tvar) function iom::iom_read_var::iom__read_var_id (type(tfile), intent(in) td_file, integer(i4), intent(in) id_varid, integer(i4), dimension(ip_maxdim), intent(in), optional id_start, integer(i4), dimension(ip_maxdim), intent(in), optional id_count)`

This function read variable value in an opened file, given variable id.

Parameters

<i>in</i>	<i>td_file</i>
-----------	----------------

start indices and number of indices selected along each dimension could be specify in a 4 dimension array (/x',y',z',t')

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_file</i>	file structure
<i>in</i>	<i>id_varid</i>	variable id
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension

Returns

variable structure

11.67.1.2 `type(tvar) function iom::iom_read_var::iom__read_var_name (type(tfile), intent(in) td_file, character(len=*) , intent(in) cd_name, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count)`

This function read variable value in an opened file, given variable name or standard name.

Parameters

<i>in</i>	<i>td_file</i>
-----------	----------------

start indices and number of indices selected along each dimension could be specify in a 4 dimension array (/x',y',z',t')

look first for variable name. If it doesn't exist in file, look for variable standard name.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_file</i>	file structure
<i>in</i>	<i>cd_name</i>	variable name or standard name
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension

Returns

variable structure

The documentation for this interface was generated from the following file:

- src/iom.f90

11.68 iom_rstdimg Module Reference

This module is a library to read/write dimg file.

Data Types

- interface [iom_rstdimg_read_dim](#)
- interface [iom_rstdimg_read_var](#)

Public Member Functions

- subroutine, public [iom_rstdimg_open](#) (td_file)
This subroutine open a dimg file in read or write mode.
- subroutine, public [iom_rstdimg_close](#) (td_file)
This subroutine close dimg file.
- subroutine, public [iom_rstdimg_get_mpp](#) (td_file)
This subroutine get sub domain decomposition in a dimg file.
- subroutine, public [iom_rstdimg_write_header](#) (td_file)
This subroutine write header of dimg file from file structure.
- subroutine, public [iom_rstdimg_write_var](#) (td_file)
This subroutine write variable in dimg file from file structure.

11.68.1 Detailed Description

This module is a library to read/write dimg file.

to open dimg file (create file structure):


```
CALL iom_rstdimg_open(td_file)
```

- td_file is file structure (see file.f90)

to write in dimg file:

```
CALL iom_rstdimg_write_file(td_file)
```

to close dimg file:

```
CALL iom_rstdimg_close(tl_file)
```

to read one dimension in dimg file:

```
tl_dim = iom_rstdimg_read_dim(tl_file, id_dimid)
```

or

```
tl_dim = iom_rstdimg_read_dim(tl_file, cd_name)
```

- id_dimid is dimension id
- cd_name is dimension name

to read one variable in dimg file:


```
tl_var = iom_rstdimg_read_var(td_file, id_varid, [id_start, id_count])
```

or

```
tl_var = iom_rstdimg_read_var(td_file, cd_name, [id_start, [id_count]])
```

- `id_varid` is variable id
- `cd_name` is variable name or standard name
- `id_start` is a integer(4) 1D array of index from which the data values will be read [optional]
- `id_count` is a integer(4) 1D array of the number of indices selected along each dimension [optional]

to get sub domain decomposition in a dimg file:

```
CALL iom_rstdimg_get_mpp(td_file)
```

Author

J.Paul

Date

November, 2013 - Initial Version

August, 2017

- handle use of domain decomposition for monoproc file

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.68.2 Member Function/Subroutine Documentation

11.68.2.1 subroutine, public `iom_rstdimg::iom_rstdimg_close (type(tfile), intent(inout) td_file)`

This subroutine close dimg file.

Parameters

<code>in, out</code>	<code>td_file</code>
----------------------	----------------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
----------------------	----------------------	----------------

11.68.2.2 subroutine, public `iom_rstdim::iom_rstdim_get_mpp (type(tfile), intent(inout) td_file)`

This subroutine get sub domain decomposition in a dimg file.

Parameters

<code>in, out</code>	<code><i>td_file</i></code>
----------------------	-----------------------------

domain decomposition informations are saved in attributes.

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2016

- mismatch with "halo" indices

Parameters

<code>in, out</code>	<code><i>td_file</i></code>	file structure
----------------------	-----------------------------	----------------

11.68.2.3 subroutine, public `iom_rstdim::iom_rstdim_open (type(tfile), intent(inout) td_file)`

This subroutine open a dimg file in read or write mode.

Parameters

<code>in, out</code>	<code><i>td_file</i></code>
----------------------	-----------------------------

if try to open a file in write mode that did not exist, create it.

if file already exist, get information about:

- the number of variables
- the number of dimensions
- the number of global attributes
- the ID of the unlimited dimension
- the file format Finally it read dimensions, and 'longitude' variable to compute East-West overlap.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_file</i>	file structure
----------------	----------------	----------------

11.68.2.4 subroutine, public iom_rstdimg::iom_rstdimg_write_header (type(tfile), intent(inout) *td_file*)

This subroutine write header of dimg file from file structure.

Parameters

<i>in, out</i>	<i>td_file</i>	
----------------	----------------	--

dimg file have to be already opened in write mode.

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- use iom_rstdimg__get_rec

August, 2017

- split in write_header and write_var

Parameters

<i>in, out</i>	<i>td_file</i>	file structure
----------------	----------------	----------------

11.68.2.5 subroutine, public iom_rstdimg::iom_rstdimg_write_var (type(tfile), intent(inout) *td_file*)

This subroutine write variable in dimg file from file structure.

Parameters

<i>in, out</i>	<i>td_file</i>	
----------------	----------------	--

dimg file have to be already opened in write mode.

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- use iom_rstdimg__get_rec

August, 2017

- split in write_header and write_var

Parameters

<code>in, out</code>	<code>td_file</code>	file structure
----------------------	----------------------	----------------

The documentation for this module was generated from the following file:

- `src/iom_rstdimg.f90`

11.69 iom_rstdimg::iom_rstdimg_read_dim Interface Reference

Public Member Functions

- `type(tdim)` function `iom_rstdimg__read_dim_id` (`td_file`, `id_dimid`)
This function read one dimension in an opened netcdf file, given dimension id.
- `type(tdim)` function `iom_rstdimg__read_dim_name` (`td_file`, `cd_name`)
This function read one dimension in an opened netcdf file, given dimension name.

11.69.1 Member Function/Subroutine Documentation

11.69.1.1 `type(tdim)` function `iom_rstdimg::iom_rstdimg_read_dim::iom_rstdimg__read_dim_id` (`type(tfile)`, `intent(in) td_file`, `integer(i4)`, `intent(in) id_dimid`)

This function read one dimension in an opened netcdf file, given dimension id.

Parameters

<code>in</code>	<code>td_file</code>	
-----------------	----------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in</code>	<code>td_file</code>	file structure
<code>in</code>	<code>id_dimid</code>	dimension id

Returns

dimension structure

11.69.1.2 `type(tdim)` function `iom_rstdimg::iom_rstdimg_read_dim::iom_rstdimg__read_dim_name` (`type(tfile)`, `intent(in) td_file`, `character(len=*)`, `intent(in) cd_name`)

This function read one dimension in an opened netcdf file, given dimension name.

Parameters

<code>in</code>	<code>td_file</code>	
-----------------	----------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>cd_name</i>	dimension name

Returns

dimension structure

The documentation for this interface was generated from the following file:

- src/iom_rstdimg.f90

11.70 iom_rstdimg::iom_rstdimg_read_var Interface Reference

Public Member Functions

- type(tvar) function [iom_rstdimg__read_var_id](#) (td_file, id_varid, id_start, id_count)
This function read variable value in an opened dimg file, given variable id.
- type(tvar) function [iom_rstdimg__read_var_name](#) (td_file, cd_name, id_start, id_count)
This function read variable value in an opened dimg file, given variable name or standard name.

11.70.1 Member Function/Subroutine Documentation

11.70.1.1 type(tvar) function `iom_rstdimg::iom_rstdimg_read_var::iom_rstdimg__read_var_id (type(tfile), intent(in) td_file, integer(i4), intent(in) id_varid, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count)`

This function read variable value in an opened dimg file, given variable id.

Parameters

in	<i>td_file</i>
----	----------------

Optionally, start indices and number of indices selected along each dimension could be specify in a 4 dimension array ('x','y','z','t')

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>id_varid</i>	variable id
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

Returns

variable structure

11.70.1.2 `type(tvar) function iom_rstdimg::iom_rstdimg_read_var::iom_rstdimg__read_var_name (type(tfile), intent(in) td_file, character(len=*), intent(in) cd_name, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count)`

This function read variable value in an opened dimg file, given variable name or standard name.

Parameters

in	<i>td_file</i>
----	----------------

Optionally, start indices and number of indices selected along each dimension could be specify in a 4 dimension array ('x','y','z','t')

look first for variable name. If it doesn't exist in file, look for variable standard name.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	file structure
in	<i>cd_name</i>	variable name or standard name
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

Returns

variable structure

The documentation for this interface was generated from the following file:

- src/iom_rstdimg.f90

11.71 kind Module Reference

This module defines the F90 kind parameter for common data types.

Public Attributes

- integer, parameter, public `sp = SELECTED_REAL_KIND(6, 37)`
single precision (real 4)

- integer, parameter, public `dp` = `SELECTED_REAL_KIND(12, 307)`
double precision (real 8)
- integer, parameter, public `wp` = `dp`
working precision
- integer, parameter, public `i1` = `SELECTED_INT_KIND(1)`
single precision (integer 1)
- integer, parameter, public `i2` = `SELECTED_INT_KIND(4)`
single precision (integer 2)
- integer, parameter, public `i4` = `SELECTED_INT_KIND(9)`
single precision (integer 4)
- integer, parameter, public `i8` = `SELECTED_INT_KIND(14)`
double precision (integer 8)
- integer, parameter, public `lc` = 256
Length of Character strings.

11.71.1 Detailed Description

This module defines the F90 kind parameter for common data types.

Author

G. Madec

Date

June, 2006 - Initial Version

December, 2012 - G. Madec

- add a standard length of character strings

Todo

- check i8 max value

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

The documentation for this module was generated from the following file:

- `src/kind.f90`

11.72 lbc Module Reference

This module groups lateral boundary conditions subroutine.

Data Types

- interface `lbc__hide_nfd`
- interface `lbc_hide`
- interface `lbc_ink`
- interface `lbc_nfd`

11.72.1 Detailed Description

This module groups lateral boundary conditions subroutine.

Warning

keep only non mpp case

Author

G. Madec

Date

June, 1997 - Original code

September, 2002

- F90: Free form and module

Marsh, 2009

- R. Benshila : External north fold treatment

December, 2012

- S.Mocavero, I. Epicoco : Add 'lbc_bdy_ink' and lbc_obc_ink' routine to optimize the BDY/OBC communications

December, 2012

- R. Bourdalle-Badie and G. Refray : add a C1D case

January, 2015

- J.Paul : rewrite with SIREN coding rules

Marsh, 2015

- J.Paul : add hide subroutine

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

The documentation for this module was generated from the following file:

- src/lbc.f90

11.73 lbc::lbc__hide_nfd Interface Reference

Public Member Functions

- subroutine [lbc__hide_nfd_2d](#) (dd_array, cd_type, id_perio, dd_psgn, dd_fill)

This subroutine manage 2D lateral boundary condition : hide North fold treatment without processor exchanges.

11.73.1 Member Function/Subroutine Documentation

- 11.73.1.1 subroutine `lbc::lbc__hide_nfd::lbc__hide_nfd_2d` (real(dp), dimension(:, :), intent(inout) *dd_array*, character(len=*), intent(in) *cd_type*, integer(i4), intent(in) *id_perio*, real(dp), intent(in) *dd_psgn*, real(dp), intent(in), optional *dd_fill*)

This subroutine manage 2D lateral boundary condition : hide North fold treatment without processor exchanges.

Parameters

in, out	<i>dd_array</i>
---------	-----------------

Warning

keep only non mpp case
do not use additional halos

Author

J.Paul

- Marsh, 2015- initial version

Parameters

in, out	<i>dd_array</i>	2D array
in	<i>cd_type</i>	point grid
in	<i>id_perio</i>	NEMO periodicity of the grid
in	<i>dd_psgn</i>	
in	<i>dd_fill</i>	

The documentation for this interface was generated from the following file:

- src/lbc.f90

11.74 lbc::lbc_hide Interface Reference

Public Member Functions

- subroutine [lbc__hide_ink_2d](#) (*dd_array*, *cd_type*, *id_perio*, *dd_psgn*, *dd_fill*)
This subroutine hide lateral boundary conditions on a 2D array (non mpp case)

11.74.1 Member Function/Subroutine Documentation

11.74.1.1 subroutine `lbc::lbc_hide::lbc__hide_ink_2d` (`real(dp)`, `dimension(:,:)`, `intent(inout) dd_array`, `character(len=*)`, `intent(in) cd_type`, `integer(i4)`, `intent(in) id_perio`, `real(dp)`, `intent(in) dd_psgn`, `real(dp)`, `intent(in)`, `optional dd_fill`)

This subroutine hide lateral boundary conditions on a 2D array (non mpp case)

Parameters

in, out	<i>dd_array</i>	<pre> dd_psign = -1 : change the sign across the north fold = 1 : no change of the sign across the north fold = 0 : no change of the sign across the north fold and strict positivity preserved: use inner row/column for closed boundaries. </pre>
---------	-----------------	---

Author

J.Paul

- Marsh, 2015- initial version

Parameters

in, out	<i>dd_array</i>	2D array
in	<i>cd_type</i>	point grid
in	<i>id_perio</i>	NEMO periodicity of the grid
in	<i>dd_psgn</i>	
in	<i>dd_fill</i>	fillValue

The documentation for this interface was generated from the following file:

- src/lbc.f90

11.75 lbc::lbc_Ink Interface Reference**Public Member Functions**

- subroutine `lbc__Ink_3d` (`dd_array`, `cd_type`, `id_perio`, `dd_psgn`, `dd_fill`)
This subroutine set lateral boundary conditions on a 3D array (non mpp case)
- subroutine `lbc__Ink_2d` (`dd_array`, `cd_type`, `id_perio`, `dd_psgn`, `dd_fill`)
This subroutine set lateral boundary conditions on a 2D array (non mpp case)

11.75.1 Member Function/Subroutine Documentation

11.75.1.1 subroutine `lbc::lbc_Ink::lbc__Ink_2d` (`real(dp)`, `dimension(:,:)`, `intent(inout) dd_array`, `character(len=*)`, `intent(in) cd_type`, `integer(i4)`, `intent(in) id_perio`, `real(dp)`, `intent(in) dd_psgn`, `real(dp)`, `intent(in)`, `optional dd_fill`)

This subroutine set lateral boundary conditions on a 2D array (non mpp case)

Parameters

in, out	<i>dd_array</i>	<pre> dd_psign = -1 : change the sign across the north fold = 1 : no change of the sign across the north fold = 0 : no change of the sign across the north fold and strict positivity preserved: use inner row/column for closed boundaries. </pre>
---------	-----------------	---

Author

J.Paul

- January, 2015- rewrite with SIREN coding rules

Parameters

in, out	<i>dd_array</i>	2D array
in	<i>cd_type</i>	point grid
in	<i>id_perio</i>	NEMO periodicity of the grid
in	<i>dd_psgn</i>	
in	<i>dd_fill</i>	fillValue

11.75.1.2 subroutine lbc::lbc_Ink::lbc_Ink_3d (real(dp), dimension(:,:), intent(inout) *dd_array*, character(len=*), intent(in) *cd_type*, integer(i4), intent(in) *id_perio*, real(dp), intent(in) *dd_psgn*, real(dp), intent(in), optional *dd_fill*)

This subroutine set lateral boundary conditions on a 3D array (non mpp case)

Parameters

in, out	<i>dd_array</i>	<pre> dd_psign = -1 : change the sign across the north fold = 1 : no change of the sign across the north fold = 0 : no change of the sign across the north fold and strict positivity preserved: use inner row/column for closed boundaries. </pre>
---------	-----------------	---

Author

J.Paul

- January, 2015- rewrite with SIREN coding rules

Parameters

in, out	<i>dd_array</i>	3D array
in	<i>cd_type</i>	point grid
in	<i>id_perio</i>	NEMO periodicity of the grid
in	<i>dd_psgn</i>	
in	<i>dd_fill</i>	fillValue

The documentation for this interface was generated from the following file:

- src/lbc.f90

11.76 lbc::lbc_nfd Interface Reference

Public Member Functions

- subroutine [lbc__nfd_3d](#) (*dd_array*, *cd_type*, *id_perio*, *dd_psgn*)
This subroutine manage 3D lateral boundary condition : North fold treatment without processor exchanges.
- subroutine [lbc__nfd_2d](#) (*dd_array*, *cd_type*, *id_perio*, *dd_psgn*)
This subroutine manage 2D lateral boundary condition : North fold treatment without processor exchanges.

11.76.1 Member Function/Subroutine Documentation

11.76.1.1 subroutine lbc::lbc_nfd::lbc_nfd_2d (real(dp), dimension(:,:), intent(inout) *dd_array*, character(len=*), intent(in) *cd_type*, integer(i4), intent(in) *id_perio*, real(dp), intent(in) *dd_psgn*)

This subroutine manage 2D lateral boundary condition : North fold treatment without processor exchanges.

Parameters

<i>in, out</i>	<i>dd_array</i>
----------------	-----------------

Warning

keep only non mpp case
do not use additional halos

Author

J.Paul

- January, 2015- rewrite with SIREN coding rules

Parameters

<i>in, out</i>	<i>dd_array</i>	2D array
<i>in</i>	<i>cd_type</i>	point grid
<i>in</i>	<i>id_perio</i>	NEMO periodicity of the grid
<i>in</i>	<i>dd_psgn</i>	

11.76.1.2 subroutine lbc::lbc_nfd::lbc_nfd_3d (real(dp), dimension(:,:,:), intent(inout) *dd_array*, character(len=*), intent(in) *cd_type*, integer(i4), intent(in) *id_perio*, real(dp), intent(in) *dd_psgn*)

This subroutine manage 3D lateral boundary condition : North fold treatment without processor exchanges.

Parameters

<i>in, out</i>	<i>dd_array</i>
----------------	-----------------

Warning

keep only non mpp case

Author

J.Paul

- January, 2015- rewrite with SIREN coding rules

Parameters

<i>in, out</i>	<i>dd_array</i>	3D array
<i>in</i>	<i>cd_type</i>	point grid
<i>in</i>	<i>id_perio</i>	NEMO periodicity of the grid
<i>in</i>	<i>dd_psgn</i>	

The documentation for this interface was generated from the following file:

- src/lbc.f90

11.77 logger Module Reference

This module manage log file.

Data Types

- type `tlogger`

Public Member Functions

- subroutine, public `logger_open` (cd_file, cd_verbosity, id_maxerror, id_logid)
This subroutine create a log file with default verbosity ('warning').
- subroutine, public `logger_close` ()
- subroutine, public `logger_clean` ()
- subroutine, public `logger_flush` ()
This subroutine flushing output into log file.
- recursive subroutine, public `logger_header` ()
- subroutine, public `logger_footer` ()
- subroutine, public `logger_trace` (cd_msg, ld_flush)
This subroutine write trace message on log file.
- subroutine, public `logger_debug` (cd_msg, ld_flush)
This subroutine write debug message on log file.
- subroutine, public `logger_info` (cd_msg, ld_flush)
This subroutine write info message on log file.
- subroutine, public `logger_warn` (cd_msg, ld_flush)
This subroutine write warning message on log file.
- subroutine, public `logger_error` (cd_msg, ld_flush)
This subroutine write error message on log file.
- recursive subroutine, public `logger_fatal` (cd_msg)
This subroutine write fatal error message on log file, close log file and stop process.

11.77.1 Detailed Description

This module manage log file.

This module create log file and fill it depending of verbosity.

verbosity could be choosen between :

- trace : Most detailed information.
- debug : Detailed information on the flow through the system.
- info : Interesting runtime events (startup/shutdown).
- warning: Use of deprecated APIs, poor use of API, 'almost' errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong".
- error : Other runtime errors or unexpected conditions.
- fatal : Severe errors that cause premature termination.
- none : to not create and write any information in logger file.
in this case only FATAL ERROR will be detected.

Note

default verbosity is warning

If total number of error exceeded maximum number authorized, program stop.

to open/create logger file:

```
CALL logger_open(cd_file, [cd_verbosity,] [id_maxerror,] [id_loggerid])
```

- cd_file is logger file name
- cd_verbosity is verbosity to be used [optional, default 'warning']
- id_loggerid is file id [optional, use only to flush]
- id_maxerror is the maximum number of error authorized before program stop [optional, default 5]

to close logger file:

```
CALL logger_close()
```

to clean logger file:

```
CALL logger_clean()
```

to write header in logger file:

```
CALL logger_header()
```

to write footer in logger file:

```
CALL logger_footer()
```

to flushing output:

```
CALL logger_flush()
```

to write TRACE message in logger file:

```
CALL logger_trace(cd_msg [,ld_flush])
```

- cd_msg is TRACE message
- ld_flush to flush output [optional]

to write DEBUG message in logger file:

```
CALL logger_debug(cd_msg [,ld_flush])
```

- cd_msg is DEBUG message
- ld_flush to flush output [optional]

to write INFO message in logger file:

```
CALL logger_info(cd_msg [,ld_flush])
```

- cd_msg is INFO message

- `ld_flush` to flush output [optional]

to write WARNING message in logger file:

```
CALL logger_warn(cd_msg [,ld_flush])
```

- `cd_msg` is WARNING message
- `ld_flush` to flush output [optional]

to write ERROR message in logger file:

```
CALL logger_error(cd_msg [,ld_flush])
```

- `cd_msg` is ERROR message
- `ld_flush` to flush output [optional]

to write FATAL message in logger file:

```
CALL logger_fatal(cd_msg)
```

- `cd_msg` is FATAL message

Examples :

```
CALL logger_open('loggerfile.txt','info')
```

```
CALL logger_header()
CALL logger_debug('une info de debug')
CALL logger_info('une info')
CALL logger_warn('un warning')
CALL logger_error('une erreur')
CALL logger_footer()
CALL logger_close()
CALL logger_clean()
```

```
CALL logger_open('loggerfile.txt')
```

```
CALL logger_header()
CALL logger_debug('une info de debug')
CALL logger_info('une info')
CALL logger_warn('un warning')
CALL logger_error('une erreur')
CALL logger_footer()
CALL logger_close()
CALL logger_clean()
```

Author

J.Paul

Date

November, 2013 - Initial Version

February, 2015

- check verbosity validity
- add 'none' verbosity level to not used logger file

January, 2016

- add `logger_clean` subroutine

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.77.2 Member Function/Subroutine Documentation

11.77.2.1 subroutine, public logger::logger_clean ()

This subroutine clean a log structure.

Author

J.Paul

Date

January, 2016 - Initial Version

11.77.2.2 subroutine, public logger::logger_close ()

This subroutine close a log file.

Author

J.Paul

Date

November, 2013 - Initial Version

11.77.2.3 subroutine, public logger::logger_debug (character(len=*), intent(in) *cd_msg*, logical, intent(in), optional *ld_flush*)

This subroutine write debug message on log file.

Parameters

<i>in</i>	<i>cd_msg</i>
-----------	---------------

Optionally you could flush output.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_msg</i>	message to write
<i>in</i>	<i>ld_flush</i>	flushing ouput

11.77.2.4 subroutine, public logger::logger_error (character(len=*), intent(in) *cd_msg*, logical, intent(in), optional *ld_flush*)

This subroutine write error message on log file.

Parameters

<i>in</i>	<i>cd_msg</i>
-----------	---------------

Optionally you could flush output.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_msg</i>	message to write
<i>in</i>	<i>ld_flush</i>	flushing ouput

11.77.2.5 recursive subroutine, public logger::logger_fatal (character(len=*), intent(in) *cd_msg*)

This subroutine write fatal error message on log file, close log file and stop process.

Parameters

<i>in</i>	<i>cd_msg</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2015

- stop program for FATAL ERROR if verbosity is none

Parameters

<i>in</i>	<i>cd_msg</i>	message to write
-----------	---------------	------------------

11.77.2.6 subroutine, public logger::logger_flush ()

This subroutine flushing output into log file.

Author

J.Paul

Date

November, 2013 - Initial Version

11.77.2.7 subroutine, public logger::logger_footer ()

This subroutine write footer on log file.

Author

J.Paul

Date

November, 2013 - Initial Version

11.77.2.8 recursive subroutine, public logger::logger_header ()

This subroutine write header on log file.

Author

J.Paul

Date

November, 2013 - Initial Version

11.77.2.9 subroutine, public logger::logger_info (character(len=*), intent(in) *cd_msg*, logical, intent(in), optional *ld_flush*)

This subroutine write info message on log file.

Parameters

<i>in</i>	<i>cd_msg</i>
-----------	---------------

Optionally you could flush output.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_msg</i>	message to write
<i>in</i>	<i>ld_flush</i>	flushing ouput

11.77.2.10 subroutine, public logger::logger_open (character(len=*), intent(in) *cd_file*, character(len=*), intent(in), optional *cd_verbosity*, integer(i4), intent(in), optional *id_maxerror*, integer(i4), intent(in), optional *id_logid*)

This subroutine create a log file with default verbosity ('warning').

Parameters

<i>in</i>	<i>cd_file</i>
-----------	----------------

Optionally verbosity could be change to ('trace','debug','info','warning','error','fatal').

Optionally maximum number of error allowed could be change.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_file</i>	log file name
<i>in</i>	<i>cd_verbosity</i>	log file verbosity
<i>in</i>	<i>id_maxerror</i>	maximum number of error
<i>in</i>	<i>id_logid</i>	log file id (use to flush)

11.77.2.11 subroutine, public logger::logger_trace (character(len=*), intent(in) *cd_msg*, logical, intent(in), optional *id_flush*)

This subroutine write trace message on log file.

Parameters

<i>in</i>	<i>cd_msg</i>
-----------	---------------

Optionally you could flush output.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>cd_msg</i>	message to write
<i>in</i>	<i>id_flush</i>	flushing ouput

11.77.2.12 subroutine, public logger::logger_warn (character(len=*), intent(in) *cd_msg*, logical, intent(in), optional *id_flush*)

This subroutine write warning message on log file.

Parameters

<i>in</i>	<i>cd_msg</i>
-----------	---------------

Optionally you could flush output.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>cd_msg</i>	message to write
in	<i>ld_flush</i>	flushing output

The documentation for this module was generated from the following file:

- src/logger.f90

11.78 math Module Reference

This module groups some useful mathematical function.

Data Types

- interface [math_mean](#)
- interface [math_median](#)
- interface [math_mwe](#)

Public Member Functions

- pure recursive subroutine, public [math_qsortc](#) (dd_array)
This subroutine sort a 1D array.
- pure subroutine, public [math_unwrap](#) (dd_array, dd_discont)
This subroutine correct phase angles to produce smoother phase plots.
- recursive function, public [math_compute](#) (cd_var)
This function compute simple operation.
- pure function, public [math_deriv_1d](#) (dd_value, dd_fill, ld_discont)
This function compute derivative of 1D array.
- real(dp) function, dimension(size(dd_value, dim=1),size(dd_value, dim=2)), public [math_deriv_2d](#) (dd_value, dd_fill, cd_dim, ld_discont)
This function compute derivative of 2D array. you have to specify in which direction derivative have to be computed: first (I) or second (J) dimension.
- pure function, public [math_deriv_3d](#) (dd_value, dd_fill, cd_dim, ld_discont)
This function compute derivative of 3D array. you have to specify in which direction derivative have to be computed: first (I), second (J) or third (K) dimension.
- real(dp) function, public [math_ortho](#) (dd_latm)
This function compute orthodome distance between opposite point of a cell of one degree.
- real(dp) function, public [math_euclid](#) (dd_lonm, dd_latm)
This function compute euclidian distance between opposite point of a cell of one degree, center on (lonm,latm).

11.78.1 Detailed Description

This module groups some useful mathematical function.

to compute the mean of an array:


```
dl_value=math_mean( dl_value, dd_fill )
```

- dl_value is 1D or 2D array

- `dd_fill` is FillValue

to compute the median of an array:

```
dl_value=math_median( dl_value, dd_fill )
```

- `dl_value` is 1D or 2D array
- `dd_fill` is FillValue

to compute the mean without extremum of an array:

```
dl_value=math_mwe( dl_value, id_next, dd_fill )
```

- `dl_value` is 1D or 2D array
- `id_next` is the number of extremum to be removed
- `dd_fill` is FillValue

to sort an 1D array:

```
CALL math_QsortC(dl_value)
```

- `dl_value` is 1D array

to correct phase angles to produce smoother phase:

```
CALL math_unwrap(dl_value, [dl_discont])
```

- `dl_value` is 1D array
- `dl_discont` maximum discontinuity between values, default pi

to compute simple operation

```
dl_res=math_compute( cl_var )
```

- `cl_var` operation to compute (string of character)
- `dl_res` result of the operation, real(dp)

to compute first derivative of 1D array:

```
dl_value(:)=math_deriv_1d( dd_value(:), dd_fill, [ld_discont] )
```

- `dd_value` is 1D array of variable
- `dd_fill` is FillValue of variable
- `ld_discont` is logical to take into account longitudinal East-West discontinuity [optional]

to compute first derivative of 2D array:

```
dl_value(:,:)=math_deriv_2d( dd_value(:,:), dd_fill, cd_dim,
[ld_discont] )
```

- `dd_value` is 2D array of variable

- `dd_fill` is FillValue of variable
- `cd_dim` is character to compute derivative on first (I) or second (J) dimension
- `ld_discont` is logical to take into account longitudinal East-West discontinuity [optional]

to compute first derivative of 3D array:

```
dl_value(:, :, :) = math_deriv_3d( dd_value(:, :, :), dd_fill, cd_dim,
                                  [ld_discont] )
```

- `dd_value` is 3D array of variable
- `dd_fill` is FillValue of variable
- `cd_dim` is character to compute derivative on first (I), second (J), or third (K) dimension
- `ld_discont` is logical to take into account longitudinal East-West discontinuity [optional]

Author

J.Paul

Date

January, 2015 - Initial version

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.78.2 Member Function/Subroutine Documentation

11.78.2.1 recursive function, public `math::math_compute (character(len=*), intent(in) cd_var)`

This function compute simple operation.

Parameters

<code>in</code>	<code><i>cd_var</i></code>
-----------------	----------------------------

- operation should be write as a string of character.
- operators allowed are : +, -, *, /
- to ordered operation you should use parentheses

exemples: '1e6/(16/122)', '(3/2)*(2+1)'

Author

J.Paul

Date

June, 2015 - initial version

Parameters

<i>in</i>	<i>cd_var</i>	operation to compute (string of character)
-----------	---------------	--

Returns

result of the operation, real(dp)

11.78.2.2 pure function, public math::math_deriv_1d (real(dp), dimension(:), intent(in) *dd_value*, real(dp), intent(in) *dd_fill*, logical, intent(in), optional *ld_discont*)

This function compute derivative of 1D array.

Parameters

<i>in</i>	<i>dd_value</i>
-----------	-----------------

optionally you could specify to take into account east west discontinuity (-180° 180° or 0° 360° for longitude variable)

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>dd_value</i>	1D array of variable to be extrapolated
<i>in</i>	<i>dd_fill</i>	FillValue of variable
<i>in</i>	<i>ld_discont</i>	logical to take into account east west discontinuity

11.78.2.3 real(dp) function, dimension(size(*dd_value*,dim=1), size(*dd_value*,dim=2)), public math::math_deriv_2d (real(dp), dimension(:,:), intent(in) *dd_value*, real(dp), intent(in) *dd_fill*, character(len=*), intent(in) *cd_dim*, logical, intent(in), optional *ld_discont*)

This function compute derivative of 2D array. you have to specify in which direction derivative have to be computed: first (I) or second (J) dimension.

Parameters

<i>in</i>	<i>dd_value</i>
-----------	-----------------

optionally you could specify to take into account east west discontinuity (-180° 180° or 0° 360° for longitude variable)

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>dd_value</i>	2D array of variable to be extrapolated
in	<i>dd_fill</i>	FillValue of variable
in	<i>cd_dim</i>	compute derivative on first (I) or second (J) dimension
in	<i>ld_discont</i>	logical to take into account east west discontinuity

11.78.2.4 pure function, public math::math_deriv_3d (real(dp), dimension(:,:,), intent(in) *dd_value*, real(dp), intent(in) *dd_fill*, character(len=*) , intent(in) *cd_dim*, logical, intent(in), optional *ld_discont*)

This function compute derivative of 3D array. you have to specify in which direction derivative have to be computed: first (I), second (J) or third (K) dimension.

Parameters

in	<i>dd_value</i>	
----	-----------------	--

optionaly you could specify to take into account east west discontinuity (-180° 180° or 0° 360° for longitude variable)

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>dd_value</i>	3D array of variable to be extrapolated
in	<i>dd_fill</i>	FillValue of variable
in	<i>cd_dim</i>	compute derivative on first (I) second (J) or third (K) dimension
in	<i>ld_discont</i>	logical to take into account east west discontinuity

11.78.2.5 real(dp) function, public math::math_euclid (real(dp), target *dd_lonm*, real(dp), target *dd_latm*)

This function compute euclidian distance between opposite point of a cell of one degree, center on (lonm,latm).

Parameters

<i>dd_lonm</i>	
----------------	--

Author

J.Paul

Date

April, 2017 - Initial Version

Parameters

in	<i>dd_lonm</i>	mean longitude of the cell
in	<i>dd_latm</i>	mean latitude of the cell

Returns

orthodome distance

11.78.2.6 `real(dp)` function, public `math::math_ortho (real(dp), target dd_latm)`

This function compute orthodome distance between opposite point of a cell of one degree.

Parameters

<i>dd_latm</i>

Author

J.Paul

Date

April, 2017 - Initial Version

Parameters

<i>in</i>	<i>dd_latm</i>	mean latitude of the cell
-----------	----------------	---------------------------

Returns

orthodome distance

11.78.2.7 `pure recursive subroutine`, public `math::math_qsortc (real(dp), dimension(:), intent(inout) dd_array)`

This subroutine sort a 1D array.

Parameters

<i>in, out</i>	<i>dd_array</i>
----------------	-----------------

Recursive Fortran 95 quicksort routine sorts real numbers into ascending numerical order Author: Juli Rew, SCD Consulting (juliana@ucar.edu), 9/03 Based on algorithm from Cormen et al., Introduction to Algorithms, 1997 printing

Author

J.Paul

Date

January, 2015 - Rewrite with SIREN coding rules

Parameters

<i>in, out</i>	<i>dd_array</i>	1D array
----------------	-----------------	----------

11.78.2.8 pure subroutine, public `math::math_unwrap (real(dp), dimension(:), intent(inout) dd_array, real(dp), intent(in), optional dd_discont)`

This subroutine correct phase angles to produce smoother phase plots.

Parameters

<code>in, out</code>	<code>dd_array</code>
----------------------	-----------------------

This code is based on numpy unwrap function

Unwrap by changing deltas between values to 2π complement.

Unwrap radian phase `dd_array` by changing absolute jumps greater than `dd_discont` to their 2π complement.

Note

If the discontinuity in `dd_array` is smaller than π , but larger than `dd_discont`, no unwrapping is done because taking the 2π complement would only make the discontinuity larger.

Author

J.Paul

Date

Marsh, 2015 - Rewrite in fortran, with SIREN coding rules

Parameters

<code>in, out</code>	<code>dd_array</code>	1D array
<code>in</code>	<code>dd_discont</code>	maximum discontinuity between values, default pi

The documentation for this module was generated from the following file:

- `src/math.f90`

11.79 math::math_mean Interface Reference

Public Member Functions

- pure function `math__mean_1d` (`dd_array`, `dd_fill`)
This function compute the mean of a 1D array.
- pure function `math__mean_2d` (`dd_array`, `dd_fill`)
This function compute the mean of a 2D array.

11.79.1 Member Function/Subroutine Documentation

11.79.1.1 pure function `math::math_mean::math__mean_1d (real(dp), dimension(:), intent(in) dd_array, real(dp), intent(in), optional dd_fill)`

This function compute the mean of a 1D array.

Parameters

in	<i>dd_array</i>
----	-----------------

Author

J.Paul

Date

January, 2015 - Initial Version

Parameters

in	<i>dd_array</i>	1D array
in	<i>dd_fill</i>	fillValue

Returns

mean value, real(dp)

11.79.1.2 pure function `math::math_mean::math__mean_2d` (real(dp), dimension(:,:), intent(in) *dd_array*, real(dp), intent(in), optional *dd_fill*)

This function compute the mean of a 2D array.

Parameters

in	<i>dd_array</i>
----	-----------------

Author

J.Paul

Date

January, 2015 - Initial Version

Parameters

in	<i>dd_array</i>	2D array
in	<i>dd_fill</i>	fillValue

Returns

mean value, real(dp)

The documentation for this interface was generated from the following file:

- src/math.f90

11.80 math::math_median Interface Reference

Public Member Functions

- pure function [math__median_1d](#) (dd_array, dd_fill)
This function compute the median of a 1D array.
- pure function [math__median_2d](#) (dd_array, dd_fill)
This function compute the median of a 2D array.

11.80.1 Member Function/Subroutine Documentation

11.80.1.1 pure function `math::math_median::math__median_1d` (`real(dp)`, `dimension(:)`, `intent(in) dd_array`, `real(dp)`, `intent(in)`, optional `dd_fill`)

This function compute the median of a 1D array.

Parameters

<code>in</code>	<code>dd_array</code>	
-----------------	-----------------------	--

Author

J.Paul

Date

January, 2015 - Initial Version

Parameters

<code>in</code>	<code>dd_array</code>	1D array
<code>in</code>	<code>dd_fill</code>	fillValue

Returns

median value, `real(dp)`

11.80.1.2 pure function `math::math_median::math__median_2d` (`real(dp)`, `dimension(:,:)`, `intent(in) dd_array`, `real(dp)`, `intent(in)`, optional `dd_fill`)

This function compute the median of a 2D array.

Parameters

<code>in</code>	<code>dd_array</code>	
-----------------	-----------------------	--

Author

J.Paul

Date

January, 2015 - Initial Version

Parameters

<code>in</code>	<code>dd_array</code>	2D array
<code>in</code>	<code>dd_fill</code>	fillValue

Returns

median value, `real(dp)`

The documentation for this interface was generated from the following file:

- `src/math.f90`

11.81 math::math_mwe Interface Reference

Public Member Functions

- pure function `math__mwe_1d` (`dd_array`, `id_next`, `dd_fill`)
This function compute the mean without extremum of a 1D array.
- pure function `math__mwe_2d` (`dd_array`, `id_next`, `dd_fill`)
This function compute the mean without extremum of a 2D array.

11.81.1 Member Function/Subroutine Documentation

11.81.1.1 pure function `math::math_mwe::math__mwe_1d` (`real(dp)`, `dimension(:)`, `intent(in) dd_array`, `integer(i4)`, `intent(in)`, optional `id_next`, `real(dp)`, `intent(in)`, optional `dd_fill`)

This function compute the mean without extremum of a 1D array.

Parameters

<code>in</code>	<code>dd_array</code>	
-----------------	-----------------------	--

Author

J.Paul

Date

January, 2015 - Initial Version

Parameters

<code>in</code>	<code>dd_array</code>	1D array
<code>in</code>	<code>id_next</code>	number of extremum to be removed
<code>in</code>	<code>dd_fill</code>	fillValue

Returns

median value, `real(dp)`

11.81.1.2 pure function `math::math_mwe::math__mwe_2d` (`real(dp)`, `dimension(:, :)`, `intent(in) dd_array`, `integer(i4)`, `intent(in)`, optional `id_next`, `real(dp)`, `intent(in)`, optional `dd_fill`)

This function compute the mean without extremum of a 2D array.

Parameters

<code>in</code>	<code>dd_array</code>	
-----------------	-----------------------	--

Author

J.Paul

Date

January, 2015 - Initial Version

Parameters

in	<i>dd_array</i>	2D array
in	<i>id_next</i>	number of extremum to be removed
in	<i>dd_fill</i>	fillValue

Returns

median value, real(dp)

The documentation for this interface was generated from the following file:

- src/math.f90

11.82 mpp Module Reference

This module manage massively parallel processing.

Data Types

- interface [mpp__add_proc](#)
- interface [mpp__check_dim](#)
- interface [mpp__del_proc](#)
- interface [mpp_clean](#)
- interface [mpp_copy](#)
- interface [mpp_del_att](#)
- interface [mpp_del_var](#)
- interface [mpp_get_use](#)
- interface [mpp_init](#)
- type [tlay](#)
 - domain layout structure*
- type [tmpp](#)

Public Member Functions

- subroutine, public [mpp_print](#) (td_mpp)
 - This subroutine print some information about mpp structre.*
- subroutine, public [mpp_add_var](#) (td_mpp, td_var)
 - This subroutine add variable in all files of mpp structure.*
- subroutine, public [mpp_move_var](#) (td_mpp, td_var)
 - This subroutine overwrite variable in mpp structure.*
- subroutine, public [mpp_add_dim](#) (td_mpp, td_dim)
 - This subroutine add a dimension structure in a mpp structure. Do not overwrite, if dimension already in mpp structure.*
- subroutine, public [mpp_del_dim](#) (td_mpp, td_dim)
 - This subroutine delete a dimension structure in a mpp structure.*
- subroutine, public [mpp_move_dim](#) (td_mpp, td_dim)
 - This subroutine move a dimension structure in mpp structure.*
- subroutine, public [mpp_add_att](#) (td_mpp, td_att)
 - This subroutine add global attribute to mpp structure.*
- subroutine, public [mpp_move_att](#) (td_mpp, td_att)
 - This subroutine overwrite attribute in mpp structure.*
- subroutine, public [mpp_get_contour](#) (td_mpp)

This subroutine get sub domains which form global domain border.

- integer(i4) function,
dimension(4), public [mpp_get_proc_index](#) (td_mpp, id_procid)

This function return processor indices, without overlap boundary, given processor id.

- integer(i4) function,
dimension(2), public [mpp_get_proc_size](#) (td_mpp, id_procid)

This function return processor domain size, depending of domain decomposition type, given sub domain id.

- subroutine, public [mpp_get_dom](#) (td_mpp)

This subroutine determine domain decomposition type. (full, overlap, nooverlap)

- integer(i4) function, public [mpp_get_index](#) (td_mpp, cd_name)

This function return the mpp id, in a array of mpp structure, given mpp base name.

- type(tvar) function, public [mpp_recombine_var](#) (td_mpp, cd_name)

This function recombine variable splitted in mpp structure.

Public Attributes

- integer(i4) [im_psize](#) = 2000
processor dimension length for huge file
- integer(i4) [im_iumout](#) = 44
- logical [Im_layout](#) = .FALSE.

11.82.1 Detailed Description

This module manage massively parallel processing.

define type TMPP:

```
TYPE(tmpp) :: t1_mpp
```

to initialise a mpp structure:

```
t1_mpp=mpp_init( cd_file, id_mask,
                [id_niproc,] [id_njproc,] [id_nproc,]
                [id_preci,] [id_precj,]
                [cd_type,] [id_ew])
```

or

```
t1_mpp=mpp_init( cd_file, td_var,
                [id_niproc,] [id_njproc,] [id_nproc,]
                [id_preci,] [id_precj,]
                [cd_type] )
```

or

```
t1_mpp=mpp_init( td_file [,id_ew] )
```

- [cd_file](#) is the filename of the global domain file, in which MPP will be done (example: Bathymetry)
- [td_file](#) is the file structure of one processor file composing an MPP
- [id_mask](#) is the 2D mask of global domain [optional]
- [td_var](#) is a variable structure (on T-point) from global domain file. mask of the domain will be computed using FillValue [optional]
- [id_niproc](#) is the number of processor following I-direction to be used [optional]

- `id_njproc` is the number of processor following J-direction to be used [optional]
- `id_nproc` is the total number of processor to be used [optional]
- `id_preci` is the size of the overlap region following I-direction [optional]
- `id_precj` is the size of the overlap region following J-direction [optional]
- `cd_type` is the type of files composing MPP [optional]
- `id_ew` is east-west overlap [optional]

to get mpp name:

- `tl_mpp%c_name`

to get the total number of processor:

- `tl_mpp%i_nproc`

to get the number of processor following I-direction:

- `tl_mpp%i_niproc`

to get the number of processor following J-direction:

- `tl_mpp%i_njproc`

to get the length of the overlap region following I-direction:

- `tl_mpp%i_preci`

to get the length of the overlap region following J-direction:

- `tl_mpp%i_precj`

to get the type of files composing mpp structure:

- `tl_mpp%c_type`

to get the type of the global domain:

- `tl_mpp%c_dom`

MPP dimensions (global domain)

to get the number of dimensions to be used in mpp structure:

- `tl_mpp%i_ndim`

to get the array of dimension structure (4 elts) associated to the mpp structure:

- `tl_mpp%t_dim(:)`

MPP processor (files composing domain)

- `tl_mpp%t_proc(:)`

to clean a mpp structure:

```
CALL mpp_clean(tl_mpp)
```

to print information about mpp:

```
CALL mpp_print(tl_mpp)
```

to add variable to mpp:

```
CALL mpp_add_var(td_mpp, td_var)
```

- `td_var` is a variable structure

to add dimension to mpp:

```
CALL mpp_add_dim(td_mpp, td_dim)
```

- `td_dim` is a dimension structure

to add attribute to mpp:

```
CALL mpp_add_att(td_mpp, td_att)
```

- `td_att` is a attribute structure

to delete variable from mpp:

```
CALL mpp_del_var(td_mpp, td_var)
```

or

```
CALL mpp_del_var(td_mpp, cd_name)
```

- `td_var` is a variable structure
- `cd_name` is variable name or standard name

to delete dimension from mpp:

```
CALL mpp_del_dim(td_mpp, td_dim)
```

- `td_dim` is a dimension structure

to delete attribute from mpp:

```
CALL mpp_del_att(td_mpp, td_att)
```

or

```
CALL mpp_del_att(td_mpp, cd_name)
```

- `td_att` is a attribute structure

- `cd_name` is attribute name

to overwrite variable to mpp:

```
CALL mpp_move_var(td_mpp, td_var)
```

- `td_var` is a variable structure

to overwrite dimension to mpp:

```
CALL mpp_move_dim(td_mpp, td_dim)
```

- `td_dim` is a dimension structure

to overwrite attribute to mpp:

```
CALL mpp_move_att(td_mpp, td_att)
```

- `td_att` is a attribute structure

to determine domain decomposition type:

```
CALL mpp_get_dom(td_mpp)
```

to get processors to be used:

```
CALL mpp_get_use( td_mpp, id_imin, id_imax, &  
& id_jmin, id_jmax )
```

- `id_imin`
- `id_imax`
- `id_jmin`
- `id_jmax`

to get sub domains which form global domain contour:

```
CALL mpp_get_contour( td_mpp )
```

to get global domain indices of one processor:

```
il_ind(1:4)=mpp_get_proc_index( td_mpp, id_procid )
```

- `il_ind(1:4)` are global domain indices (`i1,i2,j1,j2`)
- `id_procid` is the processor id

to get the processor domain size:

```
il_size(1:2)=mpp_get_proc_size( td_mpp, id_procid )
```

- `il_size(1:2)` are the size of domain following I and J
- `id_procid` is the processor id

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- Fix memory leaks bug

October, 2015

- improve way to compute domain layout

January, 2016

- allow to print layout file (use `lm_layout`, hard coded)
- add `mpp__compute_halo` and `mpp__read_halo`

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

Todo • ECRIRE ET TESTER `add_proc_array` pour optimiser codes (voir `old/MO_mpp.f90`)

11.82.2 Member Function/Subroutine Documentation

11.82.2.1 subroutine, public `mpp::mpp_add_att (type(tmpp), intent(inout) td_mpp, type(tatt), intent(in) td_att)`

This subroutine add global attribute to mpp structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>td_att</i>	attribute structure

11.82.2.2 subroutine, public `mpp::mpp_add_dim (type(tmpp), intent(inout) td_mpp, type(tdim), intent(in) td_dim)`

This subroutine add a dimension structure in a mpp structure. Do not overwrite, if dimension already in mpp structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- rewrite the same as way var_add_dim

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>td_dim</i>	dimension structure

11.82.2.3 subroutine, public mpp::mpp_add_var (type(tmpp), intent(inout) *td_mpp*, type(tvar), intent(inout) *td_var*)

This subroutine add variable in all files of mpp structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

Author

J.Paul

Date

November, 2013 - Initial version

January, 2019

- do not split variable on domain decomposition, if only one procesor

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp strcuture
<i>in</i>	<i>td_var</i>	variable strcuture

11.82.2.4 subroutine, public mpp::mpp_del_dim (type(tmpp), intent(inout) *td_mpp*, type(tdim), intent(in) *td_dim*)

This subroutine delete a dimension structure in a mpp structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- rewrite the same as way var_del_dim

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>td_dim</i>	dimension structure

11.82.2.5 subroutine, public mpp::mpp_get_contour (type(tmpp), intent(inout) *td_mpp*)

This subroutine get sub domains which form global domain border.

Parameters

<i>in, out</i>	<i>td_mpp</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp strcuture
----------------	---------------	---------------

11.82.2.6 subroutine, public mpp::mpp_get_dom (type(tmpp), intent(inout) *td_mpp*)

This subroutine determine domain decomposition type. (full, overlap, noverlap)

Parameters

<i>in, out</i>	<i>td_mpp</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp strcuture
----------------	---------------	---------------

11.82.2.7 integer(i4) function, public mpp::mpp_get_index (type(tmpp), dimension(:), intent(in) *td_mpp*, character(len=*), intent(in) *cd_name*)

This function return the mpp id, in a array of mpp structure, given mpp base name.

Parameters

<i>in</i>	<i>td_mpp</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_file</i>	array of file structure
in	<i>cd_name</i>	file name

Returns

file id in array of file structure (0 if not found)

11.82.2.8 integer(i4) function, dimension(4), public mpp::mpp_get_proc_index (type(tmpp), intent(in) *td_mpp*, integer(i4), intent(in) *id_procid*)

This function return processor indices, without overlap boundary, given processor id.

Parameters

in	<i>td_mpp</i>	
----	---------------	--

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

in	<i>td_mpp</i>	mpp strcuture
in	<i>id_procid</i>	processor id

Returns

array of index (/ i1, i2, j1, j2 /)

11.82.2.9 integer(i4) function, dimension(2), public mpp::mpp_get_proc_size (type(tmpp), intent(in) *td_mpp*, integer(i4), intent(in) *id_procid*)

This function return processor domain size, depending of domain decompisition type, given sub domain id.

Parameters

in	<i>td_mpp</i>
----	---------------

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

in	<i>td_mpp</i>	mpp structure
in	<i>id_procid</i>	sub domain id

Returns

array of index (/ isize, jsize /)

11.82.2.10 subroutine, public mpp::mpp_move_att (type(tmpp), intent(inout) *td_mpp*, type(tatt), intent(in) *td_att*)

This subroutine overwrite attribute in mpp structure.

Parameters

in, out	<i>td_mpp</i>
---------	---------------

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

in, out	<i>td_mpp</i>	mpp structure
in	<i>td_att</i>	attribute structure

11.82.2.11 subroutine, public mpp::mpp_move_dim (type(tmpp), intent(inout) *td_mpp*, type(tdim), intent(in) *td_dim*)

This subroutine move a dimension structure in mpp structure.

Parameters

in, out	<i>td_mpp</i>
---------	---------------

Warning

dimension order may have changed

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>td_dim</i>	dimension structure

11.82.2.12 subroutine, public mpp::mpp_move_var (type(tmpp), intent(inout) *td_mpp*, type(tvar), intent(in) *td_var*)

This subroutine overwrite variable in mpp structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>td_var</i>	variable structure

11.82.2.13 subroutine, public mpp::mpp_print (type(tmpp), intent(in) *td_mpp*)

This subroutine print some information about mpp structure.

Parameters

<i>in</i>	<i>td_mpp</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_mpp</i>	mpp structure
-----------	---------------	---------------

11.82.2.14 type(tvar) function, public mpp::mpp_recombine_var (type(tmpp), intent(in) *td_mpp*, character(len=*) *cd_name*)

This function recombine variable splitted in mpp structure.

Parameters

in	<i>td_mpp</i>
----	---------------

Author

J.Paul

Date

October, 2014 - Initial Version

Parameters

in	<i>td_mpp</i>	mpp file structure
in	<i>cd_name</i>	variable name

Returns

variable structre

The documentation for this module was generated from the following file:

- src/mpp.f90

11.83 mpp::mpp__add_proc Interface Reference

Public Member Functions

- subroutine [mpp__add_proc_unit](#) (td_mpp, td_proc)
- subroutine [mpp__add_proc_arr](#) (td_mpp, td_proc)
This subroutine add array of processor to mpp structure.

11.83.1 Member Function/Subroutine Documentation

11.83.1.1 subroutine `mpp::mpp__add_proc::mpp__add_proc_arr (type(tmpp), intent(inout) td_mpp, type(tfile), dimension(:), intent(in) td_proc)`

This subroutine add array of processor to mpp structure.

Parameters

in, out	<i>td_mpp</i>
---------	---------------

Note

mpp structure should be empty

Author

J.Paul

Date

August, 2017 - Initial version

Parameters

in, out	<i>td_mpp</i>	mpp structure
in	<i>td_proc</i>	array of processor structure

11.83.1.2 subroutine `mpp::mpp__add_proc::mpp__add_proc_unit (type(tmpp), intent(inout) td_mpp, type(tfile), intent(in) td_proc)`

Parameters

in, out	<i>td_mpp</i>	mpp structure
in	<i>td_proc</i>	processor structure

The documentation for this interface was generated from the following file:

- `src/mpp.f90`

11.84 mpp::mpp__check_dim Interface Reference

Public Member Functions

- logical function `mpp__check_proc_dim` (`td_mpp`, `td_proc`)
This function check if variable and mpp structure use same dimension.
- **check**
- **if**
- **processor**
- **and**
- **mpp**
- **structure**
- **use**
- **same**
- **dimension**
- logical function `mpp__check_var_dim` (`td_mpp`, `td_var`)
This function check if variable and mpp structure use same dimension.
- **check**
- **if**
- **variable**
- **and**
- **mpp**
- **structure**
- **use**
- **same**
- **dimension**

11.84.1 Member Function/Subroutine Documentation

11.84.1.1 logical function `mpp::mpp__check_dim::mpp__check_proc_dim (type(tmpp), intent(in) td_mpp, type(tfile), intent(in) td_proc)`

This function check if variable and mpp structure use same dimension.

Parameters

<i>in</i>	<i>td_mpp</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>td_proc</i>	processor structure

Returns

dimension of processor and mpp structure agree (or not)

11.84.1.2 logical function `mpp::mpp__check_dim::mpp__check_var_dim (type(tmpp), intent(in) td_mpp, type(tvar), intent(in) td_var)`

This function check if variable and mpp structure use same dimension.

Parameters

<i>in</i>	<i>td_mpp</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

September 2015

- do not check used dimension here

Parameters

<i>in</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>td_var</i>	variable structure

Returns

dimension of variable and mpp structure agree (or not)

The documentation for this interface was generated from the following files:

- src/mpp.f90

11.85 mpp::mpp__del_proc Interface Reference

Public Member Functions

- subroutine [mpp__del_proc_id](#) (td_mpp, id_procid)
This subroutine delete processor in mpp structure, given processor id.
- subroutine [mpp__del_proc_str](#) (td_mpp, td_proc)
This subroutine delete processor in mpp structure, given processor structure.

11.85.1 Member Function/Subroutine Documentation

11.85.1.1 subroutine mpp::mpp__del_proc::mpp__del_proc_id (type(tmpp), intent(inout) *td_mpp*, integer(i4), intent(in) *id_procid*)

This subroutine delete processor in mpp structure, given processor id.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

Author

J.Paul

Date

November, 2013 - Initial version

January, 2019

- clean file structure

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp structure
<i>in</i>	<i>id_procid</i>	processor id

11.85.1.2 subroutine mpp::mpp__del_proc::mpp__del_proc_str (type(tmpp), intent(inout) *td_mpp*, type(tfile), intent(in) *td_proc*)

This subroutine delete processor in mpp structure, given processor structure.

Parameters

<i>in, out</i>	<i>td_mpp</i>
----------------	---------------

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<code>in, out</code>	<code>td_mpp</code>	: mpp structure
<code>in</code>	<code>td_proc</code>	: file/processor structure

The documentation for this interface was generated from the following file:

- `src/mpp.f90`

11.86 mpp::mpp_clean Interface Reference

Public Member Functions

- subroutine `mpp__clean_unit` (`td_mpp`)
This subroutine clean mpp structure.
- subroutine `mpp__clean_arr` (`td_mpp`)
This subroutine clean mpp structure.

11.86.1 Member Function/Subroutine Documentation

11.86.1.1 subroutine `mpp::mpp_clean::mpp__clean_arr` (`type(tmpp)`, `dimension(:)`, `intent(inout) td_mpp`)

This subroutine clean mpp structure.

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<code>in, out</code>	<code>td_mpp</code>	<code>mpp</code> structure
----------------------	---------------------	----------------------------

11.86.1.2 subroutine `mpp::mpp_clean::mpp__clean_unit` (`type(tmpp)`, `intent(inout) td_mpp`)

This subroutine clean mpp structure.

Parameters

<code>in, out</code>	<code>td_mpp</code>	
----------------------	---------------------	--

Author

J.Paul

Date

November, 2013 - Initial version
January, 2019

- nullify file structure inside mpp structure

Parameters

in, out	<i>td_mpp</i>	mpp structure
---------	---------------	---------------

The documentation for this interface was generated from the following file:

- src/mpp.f90

11.87 mpp::mpp_copy Interface Reference

Public Member Functions

- `type(tmpp)` function `mpp__copy_unit` (`td_mpp`)
This subroutine copy mpp structure in another one.
- `type(tmpp)` function, dimension(size(td_mpp(:))) `mpp__copy_arr` (`td_mpp`)
This subroutine copy an array of mpp structure in another one.

11.87.1 Member Function/Subroutine Documentation

11.87.1.1 `type(tmpp)` function, dimension(size(td_mpp(:))) `mpp::mpp_copy::mpp__copy_arr` (`type(tmpp)`, dimension(:), intent(in) `td_mpp`)

This subroutine copy an array of mpp structure in another one.

Parameters

in	<i>td_mpp</i>	
----	---------------	--

mpp file are copied in a temporary array, so input and output mpp structure do not point on the same "memory cell", and so on are independant.

Warning

do not use on the output of a function who create or read an structure (ex: `tl_file=file_copy(file_init())` is forbidden). This will create memory leaks.
to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2013 - Initial Version
November, 2014

- use function instead of overload assignment operator (to avoid memory leak)

Parameters

in	<i>td_mpp</i>	mpp structure
----	---------------	---------------

Returns

copy of input array of mpp structure

11.87.1.2 `type(tmpp)` function `mpp::mpp_copy::mpp__copy_unit` (`type(tmpp)`, intent(in) `td_mpp`)

This subroutine copy mpp structure in another one.

Parameters

<code>in</code>	<code>td_mpp</code>
-----------------	---------------------

mpp file are copied in a temporary array, so input and output mpp structure do not point on the same "memory cell", and so on are independant.

Warning

do not use on the output of a function who create or read an structure (ex: `tl_file=file_copy(file_init())` is forbidden). This will create memory leaks.
to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- use function instead of overload assignment operator (to avoid memory leak)

January, 2019

- clean file structure

Parameters

<code>in</code>	<code>td_mpp</code>	mpp structure
-----------------	---------------------	---------------

Returns

copy of input mpp structure

The documentation for this interface was generated from the following file:

- `src/mpp.f90`

11.88 mpp::mpp_del_att Interface Reference

Public Member Functions

- subroutine [mpp__del_att_name](#) (`td_mpp`, `cd_name`)
This subroutine delete attribute in mpp structure, given attribute name.
- subroutine [mpp__del_att_str](#) (`td_mpp`, `td_att`)
This subroutine delete attribute in mpp structure, given attribute structure.

11.88.1 Member Function/Subroutine Documentation

11.88.1.1 subroutine `mpp::mpp_del_att::mpp__del_att_name` (`type(tmpp)`, `intent(inout) td_mpp`, `character(len=*)`, `intent(in) cd_name`)

This subroutine delete attribute in mpp structure, given attribute name.

Parameters

<code>in, out</code>	<code>td_mpp</code>
----------------------	---------------------

Author

J.Paul

Date

November, 2013 - Initial version

February, 2015

- define local attribute structure to avoid mistake with pointer

January, 2019

- clean attributes structure

Parameters

<code>in, out</code>	<code>td_mpp</code>	mpp strcuture
<code>in</code>	<code>cd_name</code>	attribute name

11.88.1.2 subroutine `mpp::mpp_del_att::mpp__del_att_str` (`type(tmpp)`, `intent(inout) td_mpp`, `type(tatt)`, `intent(in) td_att`)

This subroutine delete attribute in mpp structure, given attribute structure.

Parameters

<code>in, out</code>	<code>td_mpp</code>
----------------------	---------------------

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

<code>in, out</code>	<code>td_mpp</code>	mpp strcuture
<code>in</code>	<code>td_att</code>	attribute structure

The documentation for this interface was generated from the following file:

- `src/mpp.f90`

11.89 mpp::mpp_del_var Interface Reference

Public Member Functions

- subroutine `mpp__del_var_name` (`td_mpp`, `cd_name`)
This subroutine delete variable in mpp structure, given variable name.
- subroutine `mpp__del_var_str` (`td_mpp`, `td_var`)
This subroutine delete variable in mpp structure, given variable structure.
- subroutine `mpp__del_var_mpp` (`td_mpp`)
This subroutine delete all variable in mpp strcuture.

11.89.1 Member Function/Subroutine Documentation

11.89.1.1 subroutine mpp::mpp_del_var::mpp__del_var_mpp (type(tmpp), intent(inout) *td_mpp*)

This subroutine delete all variable in mpp strcuture.

Parameters

<i>in, out</i>	<i>td_mpp</i>	
----------------	---------------	--

Author

J.Paul

Date

October, 2014 - Initial version

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp strcuture
----------------	---------------	---------------

11.89.1.2 subroutine mpp::mpp_del_var::mpp__del_var_name (type(tmpp), intent(inout) *td_mpp*, character(len=*) *cd_name*, intent(in) *cd_name*)

This subroutine delete variable in mpp structure, given variable name.

Parameters

<i>in, out</i>	<i>td_mpp</i>	
----------------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial version

February, 2015

- define local variable structure to avoid mistake with pointer

January, 2019

- clean variable strcuture

Parameters

<i>in, out</i>	<i>td_mpp</i>	mpp strcuture
<i>in</i>	<i>cd_name</i>	variable name

11.89.1.3 subroutine mpp::mpp_del_var::mpp__del_var_str (type(tmpp), intent(inout) *td_mpp*, type(tvar), intent(in) *td_var*)

This subroutine delete variable in mpp structure, given variable structure.

Parameters

in, out	<i>td_mpp</i>
---------	---------------

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

in, out	<i>td_mpp</i>	mpp strcuture
in	<i>td_var</i>	variable strcuture

The documentation for this interface was generated from the following file:

- src/mpp.f90

11.90 mpp::mpp_get_use Interface Reference

Public Member Functions

- subroutine [mpp__get_use_unit](#) (td_mpp, id_imin, id_imax, id_jmin, id_jmax)
This subroutine get sub domains which cover "zoom domain". proc use in "zoom domain".

11.90.1 Member Function/Subroutine Documentation

11.90.1.1 subroutine `mpp::mpp_get_use::mpp__get_use_unit (type(tmpp), intent(inout) td_mpp, integer(i4), intent(in), optional id_imin, integer(i4), intent(in), optional id_imax, integer(i4), intent(in), optional id_jmin, integer(i4), intent(in), optional id_jmax)`

This subroutine get sub domains which cover "zoom domain". proc use in "zoom domain".

Parameters

in, out	<i>td_mpp</i>
---------	---------------

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

in, out	<i>td_mpp</i>	mpp strcuture
in	<i>id_imin</i>	i-direction lower indice

in	<i>id_imax</i>	i-direction upper indice
in	<i>id_jmin</i>	j-direction lower indice
in	<i>id_jmax</i>	j-direction upper indice

The documentation for this interface was generated from the following file:

- src/mpp.f90

11.91 mpp::mpp_init Interface Reference

Public Member Functions

- type(**tmpp**) function [mpp__init_mask](#) (cd_file, id_mask, id_niproc, id_njproc, id_nproc, id_prci, id_prcj, cd_type, id_ew, id_perio, id_pivot, td_dim, ld_usempp)
This function initialise mpp structure, given file name, and optionally mask and number of processor following I and J .
- type(**tmpp**) function [mpp__init_var](#) (cd_file, td_var, id_niproc, id_njproc, id_nproc, id_prci, id_prcj, cd_type, id_perio, id_pivot, ld_usempp)
This function initialise mpp structure, given variable structure and optionally number of processor following I and J .
- type(**tmpp**) function [mpp__init_file](#) (td_file, id_ew, id_perio, id_pivot)
This function initialise a mpp structure given file structure.

11.91.1 Member Function/Subroutine Documentation

- 11.91.1.1 type(**tmpp**) function `mpp::mpp_init::mpp__init_file (type(tfile), intent(in) td_file, integer(i4), intent(in), optional id_ew, integer(i4), intent(in), optional id_perio, integer(i4), intent(in), optional id_pivot)`

This function initialise a mpp structure given file structure.

Parameters

in	<i>td_file</i>
----	----------------

It reads restart dimg files, or some netcdf files.

Warning

netcdf file must contains some attributes:

- DOMAIN_number_total
- DOMAIN_size_global
- DOMAIN_number
- DOMAIN_position_first
- DOMAIN_position_last
- DOMAIN_halo_size_start
- DOMAIN_halo_size_end or the file is assume to be no mpp file.

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2016

- mismatch with "halo" indices, use `mpp__compute_halo`

Marsh, 2017

- netcdf proc indices from zero to N-1
- copy file periodicity to mpp structure

August, 2017

- force to use domain decomposition to enhance read of monoproc file

Parameters

in	<i>td_file</i>	file structure
in	<i>id_ew</i>	east-west overlap
in	<i>id_perio</i>	NEMO periodicity index
in	<i>id_pivot</i>	NEMO pivot point index F(0),T(1)

Returns

mpp structure

11.91.1.2 `type(tmpp) function mpp::mpp_init::mpp__init_mask (character(len=*), intent(in) cd_file, integer(i4), dimension(:,:), intent(in) id_mask, integer(i4), intent(in), optional id_niproc, integer(i4), intent(in), optional id_njproc, integer(i4), intent(in), optional id_nproc, integer(i4), intent(in), optional id_preci, integer(i4), intent(in), optional id_precj, character(len=*), intent(in), optional cd_type, integer(i4), intent(in), optional id_ew, integer(i4), intent(in), optional id_perio, integer(i4), intent(in), optional id_pivot, type(tdim), dimension(ip_maxdim), intent(in), optional td_dim, logical, intent(in), optional ld_usempp)`

This function initialise mpp structure, given file name, and optionally mask and number of processor following I and J .

Parameters

in	<i>cd_file</i>	- If no total number of processor is defined (<i>id_nproc</i>), optimize the domain decomposition (look for the domain decomposition with the most land processor to remove) <ul style="list-style-type: none"> • length of the overlap region (<i>id_preci</i>, <i>id_precj</i>) could be specify in I and J direction (default value is 1)
----	----------------	---

Author

J.Paul

Date

November, 2013 - Initial version

September, 2015

- allow to define dimension with array of dimension structure

January, 2016

- use RESULT to rename output
- mismatch with "halo" indices

Parameters

in	<i>cd_file</i>	file name of one file composing mpp domain
in	<i>id_mask</i>	domain mask
in	<i>id_niproc</i>	number of processors following i
in	<i>id_njproc</i>	number of processors following j
in	<i>id_nproc</i>	total number of processors
in	<i>id_preci</i>	i-direction overlap region
in	<i>id_precj</i>	j-direction overlap region
in	<i>cd_type</i>	type of the files (cdf, cdf4, dimg)
in	<i>id_ew</i>	east-west overlap
in	<i>id_perio</i>	NEMO periodicity index
in	<i>id_pivot</i>	NEMO pivot point index F(0),T(1)
in	<i>td_dim</i>	array of dimension structure

Returns

mpp structure

11.91.1.3 **type(tmpp)** function `mpp::mpp_init::mpp_init_var (character(len=*), intent(in) cd_file, type(tvar), intent(in) td_var, integer(i4), intent(in), optional id_niproc, integer(i4), intent(in), optional id_njproc, integer(i4), intent(in), optional id_nproc, integer(i4), intent(in), optional id_preci, integer(i4), intent(in), optional id_precj, character(len=*), intent(in), optional cd_type, integer(i4), intent(in), optional id_perio, integer(i4), intent(in), optional id_pivot, logical, intent(in), optional id_usempp)`

This function initialise mpp structure, given variable structure and optionally number of processor following I and J .

Parameters

in	<i>cd_file</i>	- If no total number of processor is defined (<i>id_nproc</i>), optimize the domain decomposition (look for the domain decomposition with the most land processor to remove) <ul style="list-style-type: none"> length of the overlap region (<i>id_preci</i>, <i>id_precj</i>) could be specify in I and J direction (default value is 1)
----	----------------	---

Author

J.Paul

Date

November, 2013 - Initial version

Parameters

in	<i>cd_file</i>	file name of one file composing mpp domain
in	<i>td_var</i>	variable structure
in	<i>id_niproc</i>	number of processors following i
in	<i>id_njproc</i>	number of processors following j
in	<i>id_nproc</i>	total number of processors
in	<i>id_preci</i>	i-direction overlap region

in	<i>id_precj</i>	j-direction overlap region
in	<i>cd_type</i>	type of the files (cdf, cdf4, dimg)
in	<i>id_perio</i>	NEMO periodicity index
in	<i>id_pivot</i>	NEMO pivot point index F(0),T(1)

Returns

mpp structure

The documentation for this interface was generated from the following file:

- src/mpp.f90

11.92 multi Module Reference

This module manage multi file structure.

Data Types

- interface [multi_copy](#)
- type [tmulti](#)

Public Member Functions

- type([tmulti](#)) function, public [multi_init](#) (cd_varfile)
This subroutine initialize multi file structure.
- subroutine, public [multi_clean](#) (td_multi)
This subroutine clean multi file strucutre.
- subroutine, public [multi_print](#) (td_multi)
This subroutine print some information about mpp strucutre.

11.92.1 Detailed Description

This module manage multi file structure.

```
define type TMULTI:<br/>
```

```
TYPE(tmulti) :: tl_multi
```

to initialize a multi-file structure:

```
tl_multi=multi_init(cd_varfile(:))
```

- cd_varfile : array of variable with file path ('var1:file1','var2:file2')
file path could be replaced by a matrix of value.
separators used to defined matrix are:
 - ';' for line
 - '/' for row
 - '\' for level
- Example:

```
* 'var1:3,2,3/1,4,5'
* 3,2,3/1,4,5 => ( 3 2 3 )
                  ( 1 4 5 )
```

to get the number of mpp file in mutli file structure:

- `tl_multi%i_nmpp`

to get the total number of variable in mutli file structure:

- `tl_multi%i_nvar`

Note

number of variable and number of file could differ cause several variable could be in the same file.

to get array of mpp structure in mutli file structure:

- `tl_multi%t_mpp(:)`

to print information about multi structure:

```
CALL multi_print(td_multi)
```

to clean multi file strucutre:

```
CALL multi_clean(td_multi)
```

- `td_multi` is multi file structure

Author

J.Paul

Date

November, 2013 - Initial Version

October, 2014

- use mpp file structure instead of file

November, 2014

- Fix memory leaks bug

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.92.2 Member Function/Subroutine Documentation

11.92.2.1 subroutine, public `multi::multi_clean (type(tmulti), intent(inout) td_multi)`

This subroutine clean multi file strucutre.

Parameters

<i>in, out</i>	<i>td_multi</i>
----------------	-----------------

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- nullify mpp structure in multi file structure

Parameters

<i>in</i>	<i>td_multi</i>	multi file structure
-----------	-----------------	----------------------

11.92.2.2 type(tmulti) function, public multi::multi_init (character(len=*), dimension(:), intent(in) *cd_varfile*)

This subroutine initialize multi file structure.

Parameters

<i>in</i>	<i>cd_varfile</i>
-----------	-------------------

if variable name is 'all', add all the variable of the file in mutli file structure. Optionnaly, periodicity could be read behind filename.

Note

if first character of filename is numeric, assume matrix is given as input.
create pseudo file named 'data-*.', with matrix read as variable value.

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- check if variable to be read is in file

January, 2016

- read variable dimensions

July, 2016

- get variable to be read and associated file first

August, 2017

- get perio from namelist

January, 2019

- create and clean file structure to avoid memory leaks
- fill value read from array of variable structure

May, 2019

- compare each elt of *cl_tabfile* to *cl_file*

August, 2019

- use periodicity read from namelist, and store in multi structure

Parameters

in	<i>cd_varfile</i>	variable location information (from namelist)
----	-------------------	---

Returns

multi file structure

11.92.2.3 subroutine, public multi::multi_print (type(tmulti), intent(in) *td_multi*)

This subroutine print some information about mpp strucutre.

Parameters

in	<i>td_multi</i>	
----	-----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- print periodicity

May, 2019

- specify format output

Parameters

in	<i>td_multi</i>	multi file structure
----	-----------------	----------------------

The documentation for this module was generated from the following file:

- src/multi.f90

11.93 multi::multi_copy Interface Reference

Public Member Functions

- type(tmulti) function [multi__copy_unit](#) (td_multi)
This function copy multi mpp structure in another one.

11.93.1 Member Function/Subroutine Documentation

11.93.1.1 type(tmulti) function multi::multi_copy::multi__copy_unit (type(tmulti), intent(in) *td_multi*)

This function copy multi mpp structure in another one.

Parameters

<code>in</code>	<code>td_multi</code>
-----------------	-----------------------

file variable value are copied in a temporary array, so input and output file structure value do not point on the same "memory cell", and so on are independant.

Warning

do not use on the output of a function who create or read an attribute (ex: `tl_att=att_copy(att_init())` is forbidden). This will create memory leaks.
to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2013 - Initial Version
November, 2014

- use function instead of overload assignment operator (to avoid memory leak)

Parameters

<code>in</code>	<code>td_multi</code>	mpp structure
-----------------	-----------------------	---------------

Returns

copy of input multi structure

The documentation for this interface was generated from the following file:

- `src/multi.f90`

11.94 `date::operator(+)` Interface Reference

Public Member Functions

- `type(tdate)` function `date__addnday` (`td_date`, `dd_nday`)
This function add nday to a date: $date2 = date1 + nday$.

11.94.1 Member Function/Subroutine Documentation

11.94.1.1 `type(tdate)` function `date::operator(+):date__addnday` (`type(tdate)`, `intent(in) td_date`, `real(dp)`, `intent(in) dd_nday`)

This function add nday to a date: $date2 = date1 + nday$.

Parameters

<code>in</code>	<code>td_date</code>
-----------------	----------------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_date</i>	date strutcutre
in	<i>dd_nday</i>	number of day

Returns

date strutcutre of date + nday

The documentation for this interface was generated from the following file:

- src/date.f90

11.95 date::operator(-) Interface Reference

Public Member Functions

- type(tdate) function [date__subnday](#) (td_date, dd_nday)
This function substract nday to a date: date2 = date1 - nday.
- real(dp) function [date__diffdate](#) (td_date1, td_date2)
This function compute number of day between two dates: nday= date1 - date2.

11.95.1 Member Function/Subroutine Documentation

11.95.1.1 real(dp) function [date::operator\(-\)::date__diffdate](#) (type(tdate), intent(in) *td_date1*, type(tdate), intent(in) *td_date2*)

This function compute number of day between two dates: nday= date1 - date2.

Parameters

in	<i>td_date1</i>	
----	-----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_date1</i>	first date strutcutre
in	<i>td_date2</i>	second date strutcutre

Returns

nday

11.95.1.2 type(tdate) function [date::operator\(-\)::date__subnday](#) (type(tdate), intent(in) *td_date*, real(dp), intent(in) *dd_nday*)

This function substract nday to a date: date2 = date1 - nday.

Parameters

<code>in</code>	<code>td_date</code>	
-----------------	----------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in</code>	<code>td_date</code>	date strutcutre
<code>in</code>	<code>dd_nday</code>	number of day

Returns

date strutcutre of date - nday

The documentation for this interface was generated from the following file:

- `src/date.f90`

11.96 `fct::operator(//)` Interface Reference

Public Member Functions

- pure function `fct__i1_cat` (`cd_char`, `bd_val`)
This function concatenate character and integer(1) (as character).
- pure function `fct__i2_cat` (`cd_char`, `sd_val`)
This function concatenate character and integer(2) (as character).
- pure function `fct__i4_cat` (`cd_char`, `id_val`)
This function concatenate character and integer(4) (as character).
- pure function `fct__i8_cat` (`cd_char`, `kd_val`)
This function concatenate character and integer(8) (as character).
- pure function `fct__r4_cat` (`cd_char`, `rd_val`)
This function concatenate character and real(4) (as character).
- pure function `fct__r8_cat` (`cd_char`, `dd_val`)
This function concatenate character and real(8) (as character).
- pure function `fct__l_cat` (`cd_char`, `ld_val`)
This function concatenate character and logical (as character).

11.96.1 Member Function/Subroutine Documentation

11.96.1.1 pure function `fct::operator(//)::fct__i1_cat` (`character(len=lc)`, `intent(in) cd_char`, `integer(i1)`, `intent(in) bd_val`)

This function concatenate character and integer(1) (as character).

Parameters

<i>in</i>	<i>cd_char</i>
-----------	----------------

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

<i>in</i>	<i>cd_char</i>	string character
<i>in</i>	<i>bd_val</i>	integer(1) variable value

Returns

string character

11.96.1.2 pure function fct::operator(/)::fct_i2_cat (character(len=lc), intent(in) *cd_char*, integer(i2), intent(in) *sd_val*)

This function concatenate character and integer(2) (as character).

Parameters

<i>in</i>	<i>cd_char</i>
-----------	----------------

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

<i>in</i>	<i>cd_char</i>	string character
<i>in</i>	<i>sd_val</i>	integer(2) variable value

Returns

string character

11.96.1.3 pure function fct::operator(/)::fct_i4_cat (character(len=lc), intent(in) *cd_char*, integer(i4), intent(in) *id_val*)

This function concatenate character and integer(4) (as character).

Parameters

<i>in</i>	<i>cd_char</i>
-----------	----------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>cd_char</i>	string character
in	<i>id_val</i>	integer(4) variable value

Returns

string character

11.96.1.4 pure function `fct::operator(/)::fct_i8_cat (character(len=lc), intent(in) cd_char, integer(i8), intent(in) kd_val)`

This function concatenate character and integer(8) (as character).

Parameters

in	<i>cd_char</i>
----	----------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>cd_char</i>	string character
in	<i>kd_val</i>	integer(8) variable value

Returns

string character

11.96.1.5 pure function `fct::operator(/)::fct_l_cat (character(len=lc), intent(in) cd_char, logical, intent(in) ld_val)`

This function concatenate character and logical (as character).

Parameters

in	<i>cd_char</i>
----	----------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>cd_char</i>	string character
in	<i>ld_val</i>	logical variable value

Returns

string character

11.96.1.6 pure function fct::operator(/)::fct_r4_cat (character(len=lc), intent(in) *cd_char*, real(sp), intent(in) *rd_val*)

This function concatenate character and real(4) (as character).

Parameters

in	<i>cd_char</i>	
----	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>cd_char</i>	string character
in	<i>rd_val</i>	real(4) variable value

Returns

string character

11.96.1.7 pure function fct::operator(/)::fct_r8_cat (character(len=lc), intent(in) *cd_char*, real(dp), intent(in) *dd_val*)

This function concatenate character and real(8) (as character).

Parameters

in	<i>cd_char</i>	
----	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>cd_char</i>	string character
----	----------------	------------------

in	<i>dd_val</i>	real(8) variable value
----	---------------	------------------------

Returns

string character

The documentation for this interface was generated from the following file:

- src/function.f90

11.97 phycst Module Reference

This module defines physical constant.

Public Attributes

- real(dp), public **rday** = 24.*60.*60.
- real(dp), public **rsiyea**
- real(dp), public **rsiday**
- real(dp), parameter, public **dp_pi** = 3.14159274101257_dp
- real(dp), parameter, public **dp_eps** = 0.5 * EPSILON(1._dp)
- real(dp), parameter, public **dp_rearth** = 6371229._dp
- real(dp), parameter, public **dp_deg2rad** = dp_pi/180.0
- real(dp), parameter, public **dp_rad2deg** = 180.0/dp_pi
- real(dp), parameter, public **dp_day** = 24.*60.*60.
- real(dp), parameter, public **dp_siyea** = 365.25_dp * dp_day * 2._dp * dp_pi / 6.283076_dp
- real(dp), parameter, public **dp_siday** = dp_day / (1._dp + dp_day / dp_siyea)
- real(dp), parameter, public **dp_delta** = 1.e-6
- real(dp), parameter, public **dp_omega** = 2._dp * dp_pi / dp_siday

11.97.1 Detailed Description

This module defines physical constant.

Author

J.paul

Date

November, 2013 - Initial Version

September, 2015

- add physical constant to compute meshmask

January, 2019

- half reduce epsilon value

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

The documentation for this module was generated from the following file:

- src/phycst.f90

11.98 boundary::seg_clean Interface Reference

Public Member Functions

- subroutine [seg_clean_unit](#) (td_seg)
- subroutine [seg_clean_arr](#) (td_seg)

11.98.1 Member Function/Subroutine Documentation

11.98.1.1 subroutine boundary::seg_clean::seg_clean_arr (type(tseg), dimension(:), intent(inout) *td_seg*)

Parameters

<i>in, out</i>	<i>td_seg</i>	array of segment structure
----------------	---------------	----------------------------

11.98.1.2 subroutine boundary::seg_clean::seg_clean_unit (type(tseg), intent(inout) *td_seg*)

Parameters

<i>in, out</i>	<i>td_seg</i>	segment structure
----------------	---------------	-------------------

The documentation for this interface was generated from the following file:

- src/boundary.f90

11.99 boundary::seg_copy Interface Reference

Public Member Functions

- type(tseg) function [seg_copy_unit](#) (td_seg)
- type(tseg) function, dimension(size(td_seg(:))) [seg_copy_arr](#) (td_seg)

11.99.1 Member Function/Subroutine Documentation

11.99.1.1 type(tseg) function, dimension(size(td_seg(:))) boundary::seg_copy::seg_copy_arr (type(tseg), dimension(:), intent(in) *td_seg*)

Parameters

<i>in</i>	<i>td_seg</i>	segment structure
-----------	---------------	-------------------

Returns

copy of input array of segment structure

11.99.1.2 type(tseg) function boundary::seg_copy::seg_copy_unit (type(tseg), intent(in) *td_seg*)

Parameters

<code>in</code>	<code>td_seg</code>	segment structure
-----------------	---------------------	-------------------

Returns

copy of input segment structure

The documentation for this interface was generated from the following file:

- `src/boundary.f90`

11.100 `att::tatt` Type Reference

Public Attributes

- character(len=lc) `c_name` = "
attribute name
- integer(i4) `i_id` = 0
attribute id
- integer(i4) `i_type` = 0
attribute type
- integer(i4) `i_len` = 0
number of value store in attribute
- character(len=lc) `c_value` = 'none'
attribute value if type CHAR
- real(dp), dimension(:), pointer `d_value` => NULL()
attribute value if type SHORT,INT,FLOAT or DOUBLE

The documentation for this type was generated from the following file:

- `src/attribute.f90`

11.101 `boundary::tbody` Type Reference

boundary structure

Public Attributes

- character(len=lc) `c_card` = "
boundary cardinal
- logical `l_use` = .FALSE.
boundary use or not
- logical `l_nam` = .FALSE.
boundary get from namelist
- integer(i4) `i_nseg` = 0
number of segment in boundary
- type(`tseg`), dimension(:), pointer `t_seg` => NULL()
array of segment structure

11.101.1 Detailed Description

boundary structure

The documentation for this type was generated from the following file:

- src/boundary.f90

11.102 date::tdate Type Reference

Public Attributes

- integer(i4) `i_year` = 1858
year
- integer(i4) `i_month` = 11
month
- integer(i4) `i_day` = 17
day
- integer(i4) `i_hour` = 0
hour
- integer(i4) `i_min` = 0
min
- integer(i4) `i_sec` = 0
sec
- integer(i4) `i_dow` = 0
day of week
- integer(i4) `i_lday` = 0
last day of the month
- real(dp) `d_jd` = 0
julian day (origin : 1858/11/17 00:00:00)
- real(dp) `d_jc` = 0
CNES julian day or pseudo julian day with new date origin.
- integer(i8) `k_jdsec` = 0
number of seconds since julian day origin
- integer(i8) `k_jcsec` = 0
number of seconds since CNES or pseudo julian day origin

The documentation for this type was generated from the following file:

- src/date.f90

11.103 dim::tdim Type Reference

Public Attributes

- character(len=lc) `c_name` = ""
dimension name
- character(len=lc) `c_sname` = 'u'
dimension short name
- integer(i4) `i_id` = 0

- *dimension id*
integer(i4) `i_len` = 1
dimension length
- logical `L_uld` = .FALSE.
dimension unlimited or not
- logical `L_use` = .FALSE.
dimension used or not
- integer(i4) `i_2xyzt` = 0
indices to reshape array to ('x','y','z','t')
- integer(i4) `i_xyzt2` = 0
indices to reshape array from ('x','y','z','t')

The documentation for this type was generated from the following file:

- src/dimension.f90

11.104 dom::tdom Type Reference

Public Attributes

- type(tdim), dimension(ip_maxdim) `t_dim0`
global domain dimension
- type(tdim), dimension(ip_maxdim) `t_dim`
sub domain dimension
- integer(i4) `i_perio0`
NEMO periodicity index of global domain.
- integer(i4) `i_ew0`
East-West overlap of global domain.
- integer(i4) `i_perio`
NEMO periodicity index of sub domain.
- integer(i4) `i_pivot`
NEMO pivot point index F(0),T(1)
- integer(i4) `i_imin` = 0
i-direction sub-domain lower left point indice
- integer(i4) `i_imax` = 0
i-direction sub-domain upper right point indice
- integer(i4) `i_jmin` = 0
j-direction sub-domain lower left point indice
- integer(i4) `i_jmax` = 0
j-direction sub-domain upper right point indice
- integer(i4) `i_bdy` = 0
boundary index : 0 = no boundary 1 = north 2 = south 3 = east 4 = west
- integer(i4), dimension(2, 2) `i_ghost0` = 0
array of ghost cell factor of global domain
- integer(i4), dimension(2, 2) `i_ghost` = 0
array of ghost cell factor of sub domain
- integer(i4), dimension(2) `i_iextra` = 0
i-direction extra point
- integer(i4), dimension(2) `i_jextra` = 0
j-direction extra point

The documentation for this type was generated from the following file:

- src/domain.f90

11.105 file::tfile Type Reference

Public Attributes

- character(len=lc) `c_name` = ""
file name
- character(len=lc) `c_type` = ""
type of the file (cdf, cdf4, dimg)
- integer(i4) `i_id` = 0
file id
- logical `l_wrt` = .FALSE.
read or write mode
- integer(i4) `i_nvar` = 0
number of variable
- type(tvar), dimension(:), pointer `t_var` => NULL()
file variables
- character(len=lc) `c_grid` = 'ARAKAWA-C'
grid type
- integer(i4) `i_ew` = -1
east-west overlap
- integer(i4) `i_perio` = -1
NEMO periodicity index.
- integer(i4) `i_pivot` = -1
NEMO pivot point index F(0),T(1)
- integer(i4) `i_depthid` = 0
variable id of depth
- integer(i4) `i_timeid` = 0
variable id of time
- integer(i4) `i_ndim` = 0
number of dimensions used in the file
- integer(i4) `i_natt` = 0
number of global attributes in the file
- integer(i4) `i_uldid` = 0
id of the unlimited dimension in the file
- logical `l_def` = .FALSE.
define mode or not
- type(tatt), dimension(:), pointer `t_att` => NULL()
global attributes
- type(tdim), dimension(ip_maxdim) `t_dim`
dimension structure
- integer(i4) `i_recl` = 0
record length (binary file)
- integer(i4) `i_n0d` = 0
number of scalar variable
- integer(i4) `i_n1d` = 0
number of 1D variable
- integer(i4) `i_n2d` = 0
number of 2D variable
- integer(i4) `i_n3d` = 0
number of 3D variable
- integer(i4) `i_rhd` = 0

- record of the header infos (last record)*
- integer(i4) `i_pid` = -1
 - processor id (start to 1)*
- integer(i4) `i_impp` = 0
 - i-indices for mpp-subdomain left bottom*
- integer(i4) `i_jmpp` = 0
 - j-indices for mpp-subdomain left bottom*
- integer(i4) `i_lci` = 0
 - i-dimensions of subdomain*
- integer(i4) `i_lcj` = 0
 - j-dimensions of subdomain*
- integer(i4) `i_ldi` = 0
 - first indoor i-indices*
- integer(i4) `i_ldj` = 0
 - first indoor j-indices*
- integer(i4) `i_lei` = 0
 - last indoor i-indices*
- integer(i4) `i_lej` = 0
 - last indoor j-indices*
- logical `l_ctr` = .FALSE.
 - domain is on border*
- logical `l_use` = .FALSE.
 - domain is used*
- integer(i4) `i_iind` = 0
 - i-direction indices*
- integer(i4) `i_jind` = 0
 - j-direction indices*

The documentation for this type was generated from the following file:

- src/file.f90

11.106 interp::tinterp Type Reference

Public Attributes

- character(len=lc) `c_name` = ""
 - interpolation method name*
- character(len=lc) `c_factor` = ""
 - interpolation factor*
- character(len=lc) `c_divisor` = ""
 - interpolation divisor*

The documentation for this type was generated from the following file:

- src/interp.f90

11.107 mpp::tlay Type Reference

domain layout structure

Public Attributes

- integer(i4) `i_niproc` = 0
number of processors following i
- integer(i4) `i_njproc` = 0
number of processors following j
- integer(i4) `i_nland` = 0
number of land processors
- integer(i4) `i_nsea` = 0
number of sea processors
- integer(i4) `i_mean` = 0
mean sea point per proc
- integer(i4) `i_min` = 0
min sea point per proc
- integer(i4) `i_max` = 0
max sea point per proc
- integer(i4), dimension(:,:),
pointer `i_msk` => NULL()
sea/land processor mask
- integer(i4), dimension(:,:),
pointer `i_impp` => NULL()
i-indexes for mpp-subdomain left bottom
- integer(i4), dimension(:,:),
pointer `i_jmpp` => NULL()
j-indexes for mpp-subdomain left bottom
- integer(i4), dimension(:,:),
pointer `i_lci` => NULL()
i-dimensions of subdomain
- integer(i4), dimension(:,:),
pointer `i_lcj` => NULL()
j-dimensions of subdomain

11.107.1 Detailed Description

domain layout structure

The documentation for this type was generated from the following file:

- `src/mpp.f90`

11.108 logger::tlogger Type Reference

Public Attributes

- integer(i4) `i_id` = 0
log file id
- logical `l_use` = .TRUE.
use logger or not
- character(len=lc) `c_name`
log file name
- character(len=lc) `c_verbosity` = "warning"

- verbosity choose*
- character(len=lc) `c_verb` = ""
array of "verboosities" to used
- integer(i4) `i_nerror` = 0
number of error
- integer(i4) `i_nfatal` = 0
number of fatal error
- integer(i4) `i_maxerror` = 5
maximum number of error before stoping program

The documentation for this type was generated from the following file:

- src/logger.f90

11.109 mpp::tmpp Type Reference

Public Attributes

- character(len=lc) `c_name` = "
base name
- integer(i4) `i_id` = 0
mpp id
- integer(i4) `i_niproc` = 0
number of processors following i
- integer(i4) `i_njproc` = 0
number of processors following j
- integer(i4) `i_nproc` = 0
total number of processors used
- integer(i4) `i_preci` = 1
i-direction overlap region length
- integer(i4) `i_precj` = 1
j-direction overlap region length
- integer(i4) `i_ew` = -1
east-west overlap
- integer(i4) `i_perio` = -1
NEMO periodicity index.
- integer(i4) `i_pivot` = -1
NEMO pivot point index F(0),T(1)
- character(len=lc) `c_type` = "
type of the files (cdf, cdf4, dimg)
- character(len=lc) `c_dom` = "
type of domain (full, noextra, nooverlap)
- integer(i4) `i_ndim` = 0
number of dimensions used in mpp
- type(tdim), dimension(ip_maxdim) `t_dim`
global domain dimension
- type(tfile), dimension(:), pointer `t_proc` => NULL()
files/processors composing mpp
- logical `l_usempp` = .TRUE.
use mpp decomposition for writing netcdf

The documentation for this type was generated from the following file:

- src/mpp.f90

11.110 multi::tmulti Type Reference

Public Attributes

- integer(i4) **i_nmpp** = 0
number of mpp files
- integer(i4) **i_nvar** = 0
total number of variables
- type(tmpp), dimension(:), pointer **t_mpp** => NULL()
mpp files composing multi

The documentation for this type was generated from the following file:

- src/multi.f90

11.111 grid_hgr::tnamh Type Reference

Public Attributes

- character(len=lc) **c_coord**
- integer(i4) **i_perio**
- integer(i4) **i_mshhgr**
- real(dp) **d_ppglam0**
- real(dp) **d_ppgphi0**
- real(dp) **d_ppe1_deg**
- real(dp) **d_ppe2_deg**
- integer(i4) **i_cfg**
- logical **l_bench**

The documentation for this type was generated from the following file:

- src/grid_hgr.f90

11.112 grid_zgr::tnamz Type Reference

Public Attributes

- character(len=lc) **c_coord**
- integer(i4) **i_perio**
- logical **l_zco**
- logical **l_zps**
- logical **l_sco**
- logical **l_isfcav**
- logical **l_iscpl**
- logical **l_wd**
- integer(i4) **i_nlevel**
- real(dp) **d_ppsur**
- real(dp) **d_ppa0**
- real(dp) **d_ppa1**
- real(dp) **d_ppkth**
- real(dp) **d_ppacr**

- real(dp) **d_ppdzmin**
- real(dp) **d_pphmax**
- logical **l_dbletanh**
- real(dp) **d_ppa2**
- real(dp) **d_ppkth2**
- real(dp) **d_ppacr2**
- real(dp) **d_hmin**
- real(dp) **d_isfmin**
- real(dp) **d_e3zps_min**
- real(dp) **d_e3zps_rat**
- logical **l_s_sh94**
- logical **l_s_sf12**
- real(dp) **d_sbot_min**
- real(dp) **d_sbot_max**
- real(dp) **d_rmax**
- real(dp) **d_hc**
- real(dp) **d_theta**
- real(dp) **d_thetb**
- real(dp) **d_bb**
- logical **l_sigcrit**
- real(dp) **d_alpha**
- real(dp) **d_efold**
- real(dp) **d_zs**
- real(dp) **d_zb_a**
- real(dp) **d_zb_b**
- integer(i4) **i_cla**
- real(dp) **d_wdmin1**
- real(dp) **d_wdmin2**
- real(dp) **d_wldd**
- logical **l_c1d**
- logical **l_e3_dep**

The documentation for this type was generated from the following file:

- src/grid_zgr.f90

11.113 boundary::tseg Type Reference

Public Attributes

- integer(i4) **i_index** = 0
segment index
- integer(i4) **i_width** = 0
segment width
- integer(i4) **i_first** = 0
segment first indice
- integer(i4) **i_last** = 0
segment last indices

The documentation for this type was generated from the following file:

- src/boundary.f90

11.114 var::tvar Type Reference

Public Attributes

- character(len=lc) **c_name** = "
 variable name
- character(len=lc) **c_point** = 'T'
 ARAKAWA C-grid point name (T,U,V,F)
- integer(i4) **i_id** = 0
 variable id
- integer(i4) **i_ew** = -1
 east-west overlap
- real(dp), dimension(:,:,:),
 pointer **d_value** => NULL()
 variable value
- integer(i4) **i_type** = 0
 variable type
- integer(i4) **i_natt** = 0
 number of attributes
- integer(i4) **i_ndim** = 0
 number of dimensions
- type(tatt), dimension(:), pointer **t_att** => NULL()
 variable attributes
- type(tdim), dimension(ip_maxdim) **t_dim**
 variable dimension
- logical **l_file** = .FALSE.
 variable read in a file
- logical **l_use** = .TRUE.
 variable to be used
- character(len=lc) **c_stdname** = "
 variable standard name
- character(len=lc) **c_longname** = "
 variable long name
- character(len=lc) **c_units** = "
 variable units
- character(len=lc) **c_axis** = "
 variable axis
- real(dp) **d_scf** = 1.
 scale factor
- real(dp) **d_ofs** = 0.
 offset
- real(dp) **d_fill** = 0.
 fill value ! NF90_FILL_DOUBLE
- real(dp) **d_min** = dp_fill
 minimum value
- real(dp) **d_max** = dp_fill
 maximum value
- character(len=lc) **c_unt** = "
 output variable unit (linked to unit factor)
- real(dp) **d_unf** = 1._dp
 unit factor

- character(len=lc) `c_namout` = "
 output variable name (renamed variable)
- logical `l_contiguous` = .FALSE.
 use contiguous storage or not
- logical `l_shuffle` = .FALSE.
 shuffle filter is turned on or not
- logical `l_fletcher32` = .FALSE.
 fletcher32 filter is turned on or not
- integer(i4) `i_deflvl` = 0
 deflate level from 0 to 9, 0 indicates no deflation is in use
- integer(i4), dimension(ip_maxdim) `i_chunksz` = (/1,1,1,1/)
 chunk size
- integer(i4) `i_rec` = 0
 record number
- character(len=lc), dimension(2) `c_interp` = "
 interpolation method
- character(len=lc), dimension(1) `c_extrap` = "
 extrapolation method
- character(len=lc), dimension(5) `c_filter` = "
 filter method

The documentation for this type was generated from the following file:

- src/variable.f90

11.115 var Module Reference

This module manage variable structure.

Data Types

- type `tvar`
- interface `var_add_att`
- interface `var_add_dim`
- interface `var_add_value`
- interface `var_clean`
- interface `var_copy`
- interface `var_del_att`
- interface `var_init`
- interface `var_print`

Public Member Functions

- type(`tvar`) function, public `var_concat` (td_var1, td_var2, id_dim)
 This function concatenate variable value following id_dim direction.
- subroutine, public `var_move_att` (td_var, td_att)
 This subroutine move an attribute structure from variable structure.
- subroutine, public `var_del_dim` (td_var, td_dim)
 This subroutine delete a dimension structure in a variable structure.
- subroutine, public `var_move_dim` (td_var, td_dim)

- This subroutine move a dimension structure in variable structure.*

 - subroutine, public `var_del_value` (td_var)
- This subroutine remove variable value in a variable structure.*

 - integer(i4) function, public `var_get_index` (td_var, cd_name, cd_stdname)

This function return the variable index, in a array of variable structure, given variable name or standard name.
- integer(i4) function, public `var_get_id` (td_var, cd_name, cd_stdname)

This function return the variable id, given variable name or standard name.
- integer(i4) function,
dimension(td_var%t_dim(1)%i_len,td_var%t_dim(2)%i_len,td_var%t_dim(3)%i_len),
public `var_get_mask` (td_var)

This function return the mask 3D of variable, given variable structure.
- subroutine, public `var_chg_fillvalue` (td_var, dd_fill)

This subroutine change FillValue of the variable to standard NETCDF FillValue.
- subroutine, public `var_def_extra` (cd_file)

This subroutine read variable configuration file. And save global array of variable structure with extra information: tg_varextra.
- subroutine, public `var_chg_extra` (cd_varinfo)

This subroutine add variable information get from namelist in global array of variable structure with extra information: tg_varextra.
- subroutine, public `var_clean_extra` ()
- subroutine, public `var_read_matrix` (td_var, cd_matrix)

This subroutine read matrix value from character string in namelist and fill variable structure value.
- type(tdim) function, dimension(ip_maxdim),
public `var_max_dim` (td_var)

This function search and save the biggest dimensions use in an array of variable structure.
- subroutine, public `var_limit_value` (td_var)

This subroutine forced minimum and maximum value of variable, with value of variable structure attribute d_min and d_max.
- subroutine, public `var_chg_name` (td_var)

This subroutine replace name of the variable,.
- subroutine, public `var_chg_unit` (td_var)

This subroutine replace unit name of the variable, and apply unit factor to the value of this variable.
- subroutine, public `var_check_dim` (td_var)

This subroutine check variable dimension expected, as defined in file 'variable.cfg'.
- subroutine, public `var_reorder` (td_var, cd_dimorder)

This subroutine reshape variable value and dimension in variable structure.
- integer(i4) function, public `var_get_unit` (td_var)

This function get the next unused unit in array of variable structure.
- type(tdate) function, public `var_to_date` (td_var)

This function convert a time variable structure in date structure.
- subroutine, public `var_get_dummy` (cd_dummy)

This subroutine fill dummy variable array.
- logical function, public `var_is_dummy` (td_var)

This function check if variable is defined as dummy variable in configuraton file.

Public Attributes

- type(tvar), dimension(:),
allocatable, public `tg_varextra`

array of variable structure with extra information. fill when running `var_def_extra()`

11.115.1 Detailed Description

This module manage variable structure.

to define type TVAR:

```
TYPE(tvar) :: t1_var
```

Note

the variable value inside structure will always be 4D array of real(8).
However the variable value could be initialised with array of real(4), real(8), integer(4) or integer(8).

to initialise a variable structure:

```
t1_var=var_init( cd_name, [value,] [id_start, [id_count,]] [id_type,] [td_dim,] [td_att]... )
```

- cd_name is the variable name
- value is a 1D,2D,3D or 4D array, see [var_init](#) for more information [optional]
- id_start is a integer(4) 1D array of index from which the data values will be read [optional]
- id_count is a integer(4) 1D array of the number of indices selected along each dimension [optional]
- id_type is the type of the variable to be used [optional]
- td_dim is the array of dimension structure [optional]
- td_att is the array of attribute structure [optional] Note:
- others optionals arguments could be added, see [var_init](#).
- to put scalar variable (OD), use td_dim with all dimension unused (td_dim(:)|_use=.FALSE.)

to print information about variable structure:

```
CALL var_print(td_var [,ld_more])
```

- td_var is the variable structure
- ld_more to print more information about variable

to clean variable structure:

```
CALL var_clean(t1_var)
```

to copy variable structure in another one (using different memory cell):

```
t1_var2=var_copy(t1_var1)
```

Note

as we use pointer for the value array of the variable structure, the use of the assignment operator (=) to copy variable structure create a pointer on the same array. This is not the case with this copy function.

to get variable name:

- t1_var%c_name

to get grid point of the variable

- `tl_var%c_point`

to get EW overlap:

- `tl_var%i_ew`

to get variable value:

- `tl_var%d_value(:,:,:)`

to get the type number (based on NETCDF type constants) of the variable (as define initially or read in file):

- `tl_var%i_type`

to get variable id (read from a file):

- `tl_var%i_id`

Variable dimension

to get the number of dimension used in the variable:

- `tl_var%i_ndim`

to get the array of dimension structure (4 elts) associated to the variable:

- `tl_var%t_dim(:)`

Variable attributes

Note

attribue value are always character or real(8) 1D array.

to get the number of attributes of the variable:

- `tl_var%i_natt`

to get the array of attribute structure associated to the variable:

- `tl_var%t_att(:)`

Some attribute are highlight, to be easily used. to get variable standard name:

- `tl_var%c_stdname`

to get variable longname:

- `tl_var%c_longname`

to get variable units:

- `tl_var%c_units`

to get variable axis:

- `tl_var%c_axis`

to get variable scale factor:

- `tl_var%d_scf`

to get variable add offset:

- `tl_var%d_ofs`

to get variable FillValue:

- `tl_var%d_fill`

to add value to a variable structure:

```
CALL var_add_value(tl_var, value, [id_type,] [id_start, [id_count]])
```

- `value` : 4D array of value (real(4), real(8), integer(1), integer(2), integer(4), integer(8))
- `id_type` is the type of the variable to be used (default is the type of array value)
- `id_start` : 1D array of the index in the variable from which the data values will be read (integer(4), optional)
- `id_count` : 1D array of the number of indices selected along each dimension (integer(4), optional)

to add attribute to a variable structure:

```
CALL var_add_att(tl_var, td_att)
```

- `td_att` is an attribute structure, or array of attribute structure

to add dimension to a variable structure:

```
CALL var_add_dim(tl_var, td_dim)
```

- `td_dim` is a dimension structure, or array of dimension structure

to delete value of a variable structure:

```
CALL var_del_value(tl_var)
```

to delete one attribute of a variable structure:

```
CALL var_del_att(tl_var, td_att)
```

- `td_att` is an attribute structure or

```
CALL var_del_att(tl_var, cd_name)
```

- `cd_name` is attribute name

to delete one dimension of a variable structure:

```
CALL var_del_dim(tl_var, td_dim)
```

- `td_dim` is a dimension structure

to overwrite one attribute structure in variable structure:

```
CALL var_move_att(tl_var, td_att)
```

- `td_att` is an attribute structure

to overwrite one dimension structure in variable structure:

```
CALL var_move_dim(tl_var, td_dim)
```

- `td_dim` is a dimension structure

to get the mask of a variable structure, (based on its FillValue):

```
mask(:, :)=var_get_mask(tl_var)
```

to change FillValue to standard NETCDF Fill Value:

```
CALL var_chg_FillValue(tl_var, [dd_fill])
```

- `dd_fill` is the FillValue to be used [optional]

to concatenate two variables:

```
tl_var=var_concat(tl_var1, tl_var2, [dim])
```

- `tl_var1` : variable structure
- `tl_var2` : variable structure
- `DIM` : number of the dimension following which concatenate (1=>I, 2=>J, 3=>Z, 4=>T) [optional, default=4]

to forced min and max value of a variable:

- define min and max value of the variable:
 - `tl_var%d_min=min`
 - `tl_var%d_max=max`
 - * min and max : real(8) value
- then

```
CALL var_limit_value( tl_var )
```

to get the biggest dimensions use in a array of variable:

```
tl_dim(:)=var_max_dim(tl_var(:))
```

- `tl_var(:)` : array of variable structure
- `tl_dim(:)` : array (4 elts) of dimension structure

to reorder dimension of a variable (default 'x','y','z','t'):

```
CALL var_reorder( td_var, cd_dimorder )
```

- `td_var` is variable structure
- `cd_dimorder` string character(LEN=4) of dimension order to be used (example: 'yxzt') [optional]

to get variable index, in an array of variable structure:

```
il_index=var_get_index( td_var, cd_name )
```

- `td_var` array of variable structure
- `cd_name` variable name

to get variable id, read from a file:

```
il_id=var_get_id( td_var, cd_name )
```

- `td_var` array of variable structure
- `cd_name` variable name

to get free variable unit in an array of variable structure:

```
il_unit=var_get_unit( td_var )
```

- `td_var` array of variable structure

to convert time variable structure in date structure:

```
tl_date=var_to_date( td_var )
```

- `td_var` is time variable structure
- `tl_date` is date structure

to read matrix value from character string in namelist

```
CALL var_read_matrix( td_var, cd_matrix )
```

- `td_var` is variable structure
- `cd_matrix` is matrix value

to read variable configuration file ('variable.cfg') and fill global array of variable structure:

```
CALL var_def_extra( cd_file )
```

- `cd_file` is filename

to add variable information get from namelist, in global array of variable structure:

```
CALL var_chg_extra( cd_varinfo )
```

- `cd_varinfo` is variable information from namelist

to clean global array of variable structure:

```
CALL var_clean_extra( )
```

to check variable dimension expected, as defined in file 'variable.cfg':

```
CALL var_check_dim( td_var )
```

- `td_var` is variable structure

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add `var_reorder`

November, 2014

- Fix memory leaks bug

June, 2015

- change way to get variable information in namelist

July, 2015

- add subroutine `var_chg_unit` to change unit of output variable

Spetember, 2015

- manage useless (dummy) variable

October, 2016

- add subroutine to clean global array of extra information.
- define logical for variable to be used

May, 2019

- read number of element for each dummy array in configuration file

Todo • `var_copy_value` qui copie le tableau de valeur mais verifie que tous les attribut sont egaux

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.115.2 Member Function/Subroutine Documentation

11.115.2.1 subroutine, public `var::var_check_dim (type(tvvar), intent(inout) td_var)`

This subroutine check variable dimension expected, as defined in file 'variable.cfg'.

Parameters

<code>in, out</code>	<code>td_var</code>
----------------------	---------------------

compare dimension used in variable structure with string character axis from configuration file.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_var</code>	variable structure
----------------------	---------------------	--------------------

11.115.2.2 subroutine, public var::var_chg_extra (character(len=*), dimension(:), intent(in) *cd_varinfo*)

This subroutine add variable information get from namelist in global array of variable structure with extra information: `tg_varextra`.

Parameters

<code>in</code>	<code>cd_varinfo</code>
-----------------	-------------------------

string character format must be :

"varname:int=interp;flt=filter;ext=extrap;min=min;max=max"

you could specify only interpolation, filter or extrapolation method, whatever the order. you could find more information about available method in [interp](#), [filter](#), and [extrap](#) module.

Examples: `cn_varinfo='Bathymetry:flt=2*hamming(2,3);min=10.'` `cn_varinfo='votemper:int=cubic;ext=dist_weight;max=40.'`

Warning

variable should be define in `tg_varextra` (ie in configuration file, to be able to add information from namelist

Note

If you do not specify a method which is required, default one is apply.

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- get unit and unit factor (to change unit)

February, 2019

- get variable output name

Parameters

<i>in</i>	<i>cd_varinfo</i>	variable information from namelist
-----------	-------------------	------------------------------------

11.115.2.3 subroutine, public var::var_chg_fillvalue (type(tvar), intent(inout) *td_var*, real(dp), intent(in), optional *dd_fill*)

This subroutine change FillValue of the variable to standard NETCDF FillValue.

Parameters

<i>in, out</i>	<i>td_var</i>	
----------------	---------------	--

optionally, you could specify a dummy *_FillValue* to be used

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- write fill value on array level by level

Parameters

<i>in, out</i>	<i>td_var</i>	array of variable structure
<i>in</i>	<i>dd_fill</i>	<i>_FillValue</i> to be used

11.115.2.4 subroutine, public var::var_chg_name (type(tvar), intent(inout) *td_var*)

This subroutine replace name of the variable,.

Parameters

<i>in, out</i>	<i>td_var</i>	
----------------	---------------	--

output name (namout) is read from the namelist.

Note

the variable value should be already read.

Author

J.Paul

Date

February, 2019 - Initial Version

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
----------------	---------------	--------------------

11.115.2.5 subroutine, public var::var_chg_unit (type(tvar), intent(inout) *td_var*)

This subroutine replace unit name of the variable, and apply unit factor to the value of this variable.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

new unit name (unt) and unit factor (unf) are read from the namelist.

Note

the variable value should be already read.

Author

J.Paul

Date

June, 2015 - Initial Version

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
----------------	---------------	--------------------

11.115.2.6 subroutine, public var::var_clean_extra ()

This subroutine clean global array of variable structure with extra information: tg_varextra.

Author

J.Paul

Date

October, 2016 - Initial Version

January, 2019

- check if tg_varextra is allocated before clean it

11.115.2.7 type(tvar) function, public var::var_concat (type(tvar), intent(in) *td_var1*, type(tvar), intent(in) *td_var2*, integer(i4), intent(in), optional *id_dim*)

This function concatenate variable value following *id_dim* direction.

Parameters

<i>in</i>	<i>td_var1</i>
-----------	----------------

By default variable are concatenate following time dimension. To concatenate following another dimension, specify *id_dim=x* where x is the dimension number (jp_L, jp_J, jp_K, jp_L).

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_var1</i>	variable structure
in	<i>td_var2</i>	variable structure
in	<i>DIM</i>	dimension following which concatenate

Returns

variable structure

11.115.2.8 subroutine, public var::var_def_extra (character(len=*), intent(in) *cd_file*)

This subroutine read variable configuration file. And save global array of variable structure with extra information: *tg_varextra*.

Parameters

in	<i>cd_file</i>	
----	----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- new namelist format to get extra information (interpolation,...)

Parameters

in	<i>cd_file</i>	configuration file of variable
----	----------------	--------------------------------

11.115.2.9 subroutine, public var::var_del_dim (type(tvar), intent(inout) *td_var*, type(tdim), intent(in) *td_dim*)

This subroutine delete a dimension structure in a variable structure.

Parameters

in, out	<i>td_var</i>	
---------	---------------	--

Warning

delete variable value too.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_var</code>	variable structure
<code>in</code>	<code>td_dim</code>	dimension structure

11.115.2.10 subroutine, public var::var_del_value (type(tvar), intent(inout) *td_var*)

This subroutine remove variable value in a variable structure.

Parameters

<code>in, out</code>	<code>td_var</code>	
----------------------	---------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- nullify array inside variable structure

Parameters

<code>in, out</code>	<code>td_var</code>	variable structure
----------------------	---------------------	--------------------

11.115.2.11 subroutine, public var::var_get_dummy (character(len=*), intent(in) *cd_dummy*)

This subroutine fill dummy variable array.

Parameters

<code>in</code>	<code>cd_dummy</code>	
-----------------	-----------------------	--

Author

J.Paul

Date

September, 2015 - Initial Version

May, 2019

- read number of dummy element

Parameters

<code>in</code>	<code>cd_dummy</code>	dummy configuration file
-----------------	-----------------------	--------------------------

11.115.2.12 integer(i4) function, public var::var_get_id (type(tvar), dimension(:), intent(in) *td_var*, character(len=*), intent(in) *cd_name*, character(len=*), intent(in), optional *cd_stdname*)

This function return the variable id, given variable name or standard name.

Parameters

<i>in</i>	<i>td_var</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

July, 2015

- check long name

Parameters

<i>in</i>	<i>td_var</i>	array of variable structure
<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>cd_stdname</i>	variable standard name

Returns

variable id in array of variable structure (0 if not found)

11.115.2.13 `integer(i4) function, public var::var_get_index (type(tvar), dimension(:), intent(in) td_var, character(len=*), intent(in) cd_name, character(len=*), intent(in), optional cd_stdname)`

This function return the variable index, in a array of variable structure, given variable name or standard name.

Parameters

<i>in</i>	<i>td_var</i>
-----------	---------------

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

<i>in</i>	<i>td_var</i>	array of variable structure
<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>cd_stdname</i>	variable standard name

Returns

variable index in array of variable structure (0 if not found)

11.115.2.14 `integer(i4) function, dimension(td_var%t_dim(1)%i_len, td_var%t_dim(2)%i_len, td_var%t_dim(3)%i_len), public var::var_get_mask (type(tvar), intent(in) td_var)`

This function return the mask 3D of variable, given variable structure.

Parameters

<i>in</i>	<i>td_var</i>
-----------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_var</i>	array of variable structure
-----------	---------------	-----------------------------

Returns

variable mask(3D)

11.115.2.15 `integer(i4) function, public var::var_get_unit (type(tvar), dimension(:), intent(in) td_var)`

This function get the next unused unit in array of variable structure.

Parameters

<i>in</i>	<i>td_var</i>
-----------	---------------

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

<i>in</i>	<i>td_var</i>	array of variable structure
-----------	---------------	-----------------------------

Returns

free variable id

11.115.2.16 `logical function, public var::var_is_dummy (type(tvar), intent(in) td_var)`

This function check if variable is defined as dummy variable in configuraton file.

Parameters

<i>in</i>	<i>td_var</i>
-----------	---------------

Author

J.Paul

Date

September, 2015 - Initial Version
, May, 2019

- use number of dummy elt in do-loop

Parameters

<code>in</code>	<code>td_var</code>	variable structure
-----------------	---------------------	--------------------

Returns

true if variable is dummy variable

11.115.2.17 subroutine, public var::var_limit_value (type(tvar), intent(inout) td_var)

This subroutine forced minimum and maximum value of variable, with value of variable structure attribute `d_min` and `d_max`.

Parameters

<code>in, out</code>	<code>td_var</code>	
----------------------	---------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_var</code>	variable structure
----------------------	---------------------	--------------------

11.115.2.18 type(tdim) function, dimension(ip_maxdim), public var::var_max_dim (type(tvar), dimension(:), intent(in) td_var)

This function search and save the biggest dimensions use in an array of variable structure.

Parameters

<code>in</code>	<code>td_var</code>	
-----------------	---------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in</code>	<code>td_var</code>	array of variable structure
-----------------	---------------------	-----------------------------

Returns

array of dimension

11.115.2.19 subroutine, public var::var_move_att (type(tvar), intent(inout) td_var, type(tatt), intent(in) td_att)

This subroutine move an attribute structure from variable structure.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
<i>in</i>	<i>td_att</i>	attribute structure

11.115.2.20 subroutine, public var::var_move_dim (type(tvar), intent(inout) *td_var*, type(tdim), intent(in) *td_dim*)

This subroutine move a dimension structure in variable structure.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

Warning

- dimension order could be changed
- delete variable value

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
<i>in</i>	<i>td_dim</i>	dimension structure

11.115.2.21 subroutine, public var::var_read_matrix (type(tvar), intent(inout) *td_var*, character(len=*) , intent(in) *cd_matrix*)

This subroutine read matrix value from character string in namelist and fill variable structure value.

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

to split matrix, separator use are:

- ';' for line
- '/' for row
- '\' for level

Example:

$$3,2,3/1,4,5 => \begin{pmatrix} 3 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}$$

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
<i>in</i>	<i>cd_matrix</i>	matrix value

11.115.2.22 subroutine, public var::var_reorder (type(tvar), intent(inout) *td_var*, character(len=ip_maxdim), intent(in), optional *cd_dimorder*)

This subroutine reshape variable value and dimension in variable structure.

Parameters

<i>in, out</i>	<i>td_var</i>	
----------------	---------------	--

output dimension will be ordered as defined in input array of dimension Optionally you could specify output dimension order with string character of dimension

Author

J.Paul

Date

August, 2014 - Initial Version
July 2015

- do not use *dim_disorder* anymore

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
<i>in</i>	<i>cd_dimorder</i>	string character of dimension order to be used

11.115.2.23 type(tdate) function, public var::var_to_date (type(tvar), intent(in) *td_var*)

This function convert a time variable structure in date structure.

Parameters

<i>in</i>	<i>td_var</i>	
-----------	---------------	--

Author

J.Paul

Date

November, 2014 - Initial Version
January, 2019

- add case for units in hours

Parameters

<code>in</code>	<code>td_var</code>	time variable structure
-----------------	---------------------	-------------------------

Returns

date structure

The documentation for this module was generated from the following file:

- src/variable.f90

11.116 var::var_add_att Interface Reference**Public Member Functions**

- subroutine [var__add_att_unit](#) (`td_var`, `td_att`)
This subroutine add an attribute structure in a variable structure.
- subroutine [var__add_att_arr](#) (`td_var`, `td_att`)
This subroutine add an array of attribute structure in a variable structure.

11.116.1 Member Function/Subroutine Documentation

11.116.1.1 subroutine `var::var_add_att::var__add_att_arr` (`type(tvar)`, `intent(inout) td_var`, `type(tatt)`, `dimension(:)`, `intent(in) td_att`)

This subroutine add an array of attribute structure in a variable structure.

Parameters

<code>in, out</code>	<code>td_var</code>	
----------------------	---------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add all element of the array in the same time

January, 2019

- deallocate attribute strucure whatever happens

Parameters

<code>in, out</code>	<code>td_var</code>	variable structure
<code>in</code>	<code>td_att</code>	array of attribute structure

11.116.1.2 subroutine `var::var_add_att::var__add_att_unit` (`type(tvar)`, `intent(inout) td_var`, `type(tatt)`, `intent(in) td_att`)

This subroutine add an attribute structure in a variable structure.

Parameters

in, out	td_var
---------	--------

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- use var__add_att_arr subroutine

January, 2019

- clean attribute structure

Parameters

in, out	td_var	variable structure
in	td_att	attribute structure

The documentation for this interface was generated from the following file:

- src/variable.f90

11.117 var::var_add_dim Interface Reference

Public Member Functions

- subroutine [var__add_dim_unit](#) (td_var, td_dim)
This subroutine add one dimension in a variable structure.
- subroutine [var__add_dim_arr](#) (td_var, td_dim)
This subroutine add an array of dimension structure in a variable structure.

11.117.1 Member Function/Subroutine Documentation

11.117.1.1 subroutine var::var_add_dim::var__add_dim_arr (type(tvar), intent(inout) td_var, type(tdim), dimension(:), intent(in) td_dim)

This subroutine add an array of dimension structure in a variable structure.

Parameters

in, out	td_var	- number of dimension in variable can't be greater than 4 • dimension can't be already uses in variable structure
---------	--------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>td_dim</i>	dimension structure

11.117.1.2 subroutine var::var_add_dim::var__add_dim_unit (type(tvar), intent(inout) *td_var*, type(tdim), intent(in) *td_dim*)

This subroutine add one dimension in a variable structure.

Parameters

in, out	<i>td_var</i>	
---------	---------------	--

- number of dimension in variable can't be greater than 4
- dimension can't be already uses in variable structure

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>td_dim</i>	dimension structure

The documentation for this interface was generated from the following file:

- src/variable.f90

11.118 var::var_add_value Interface Reference

Public Member Functions

- subroutine [var__add_value_dp](#) (td_var, dd_value, id_type, id_start, id_count)
This subroutine add a 4D array of real(8) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')
- subroutine [var__add_value_rp](#) (td_var, rd_value, id_type, id_start, id_count)
This subroutine add a 4D array of real(4) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')
- subroutine [var__add_value_i1](#) (td_var, bd_value, id_type, id_start, id_count)
This subroutine add a 4D array of integer(1) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')
- subroutine [var__add_value_i2](#) (td_var, sd_value, id_type, id_start, id_count)
This subroutine add a 4D array of integer(2) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')
- subroutine [var__add_value_i4](#) (td_var, id_value, id_type, id_start, id_count)
This subroutine add a 4D array of integer(4) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')
- subroutine [var__add_value_i8](#) (td_var, kd_value, id_type, id_start, id_count)
This subroutine add a 4D array of integer(8) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')

11.118.1 Member Function/Subroutine Documentation

11.118.1.1 subroutine var::var_add_value::var__add_value_dp (type(tvar), intent(inout) *td_var*, real(dp), dimension(:,:,:,:), intent(in) *dd_value*, integer(i4), intent(in), optional *id_type*, integer(i4), dimension(ip_maxdim), intent(in), optional *id_start*, integer(i4), dimension(ip_maxdim), intent(in), optional *id_count*)

This subroutine add a 4D array of real(8) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

Optionally, you could specify the type of the variable to be used (default real(8)), and indices of the variable where value will be written with start and count array.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in, out</i>	<i>td_var</i>	variable structure
<i>in</i>	<i>dd_value</i>	array of variable value
<i>in</i>	<i>id_type</i>	type of the variable to be used (default real(8))
<i>in</i>	<i>id_start</i>	start indices of the variable where data values will be written
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension

11.118.1.2 subroutine var::var_add_value::var__add_value_i1 (type(tvar), intent(inout) *td_var*, integer(i1), dimension(:,:,:,:), intent(in) *bd_value*, integer(i4), intent(in), optional *id_type*, integer(i4), dimension(ip_maxdim), intent(in), optional *id_start*, integer(i4), dimension(ip_maxdim), intent(in), optional *id_count*)

This subroutine add a 4D array of integer(1) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

Optionally, you could specify the type of the variable to be used (default integer(1)), and indices of the variable where value will be written with start and count array.

Note

variable type is forced to BYTE

Author

J.Paul

Date

November, 2013 - Initial Version
January, 2019

- decompose array conversion on each dimension

Parameters

<i>in, out</i>	<i>td_var</i>	variabele structure
<i>in</i>	<i>bd_value</i>	array of variable value
<i>in</i>	<i>id_type</i>	type of the variable to be used (default integer(1))
<i>in</i>	<i>id_start</i>	start indices of the variable where data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension

11.118.1.3 subroutine `var::var_add_value::var__add_value_i2` (`type(tvar)`, `intent(inout) td_var`, `integer(i2)`, `dimension(:,::,::)`, `intent(in) sd_value`, `integer(i4)`, `intent(in)`, `optional id_type`, `integer(i4)`, `dimension(ip_maxdim)`, `intent(in)`, `optional id_start`, `integer(i4)`, `dimension(ip_maxdim)`, `intent(in)`, `optional id_count`)

This subroutine add a 4D array of integer(2) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

Optionally, you could specify the type of the variable to be used (default integer(2)), and indices of the variable where value will be written with start and count array.

Note

variable type is forced to SHORT

Author

J.Paul

Date

November, 2013 - Initial Version
January, 2019

- decompose array conversion on each dimension

Parameters

<i>in, out</i>	<i>td_var</i>	variabele structure
<i>in</i>	<i>sd_value</i>	array of variable value
<i>in</i>	<i>id_type</i>	type of the variable to be used (default integer(2))
<i>in</i>	<i>id_start</i>	start indices of the variable where data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension

11.118.1.4 subroutine `var::var_add_value::var__add_value_i4` (`type(tvar)`, `intent(inout) td_var`, `integer(i4)`, `dimension(:,::,::)`, `intent(in) id_value`, `integer(i4)`, `intent(in)`, `optional id_type`, `integer(i4)`, `dimension(ip_maxdim)`, `intent(in)`, `optional id_start`, `integer(i4)`, `dimension(ip_maxdim)`, `intent(in)`, `optional id_count`)

This subroutine add a 4D array of integer(4) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')

Parameters

<i>in, out</i>	<i>td_var</i>
----------------	---------------

Optionally, you could specify the type of the variable to be used (default integer(4)), and indices of the variable where value will be written with start and count array.

Note

variable type is forced to INT

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- decompose array conversion on each dimension

Parameters

in, out	<i>td_var</i>	variabele structure
in	<i>id_value</i>	array of variable value
in	<i>id_type</i>	type of the variable to be used (default integer(4))
in	<i>id_start</i>	start indices of the variable where data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

11.118.1.5 subroutine var::var_add_value::var_add_value_i8 (type(tvar), intent(inout) *td_var*, integer(i8), dimension(:,:,:,:), intent(in) *kd_value*, integer(i4), intent(in), optional *id_type*, integer(i4), dimension(ip_maxdim), intent(in), optional *id_start*, integer(i4), dimension(ip_maxdim), intent(in), optional *id_count*)

This subroutine add a 4D array of integer(8) value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')

Parameters

in, out	<i>td_var</i>
---------	---------------

Optionally, you could specify the type of the variable to be used (default integer(4)), and indices of the variable where value will be written with start and count array.

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- decompose array conversion on each dimension

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>kd_value</i>	array of variable value

in	<i>id_type</i>	type of the variable to be used (default integer(8))
in	<i>id_start</i>	start indices of the variable where data values will be read
in	<i>id_count</i>	number of indices selected along each dimension

11.118.1.6 subroutine `var::var_add_value::var__add_value_rp` (`type(tvar)`, `intent(inout) td_var`, `real(sp)`, `dimension(:, :, :, :)`, `intent(in) rd_value`, `integer(i4)`, `intent(in)`, `optional id_type`, `integer(i4)`, `dimension(ip_maxdim)`, `intent(in)`, `optional id_start`, `integer(i4)`, `dimension(ip_maxdim)`, `intent(in)`, `optional id_count`)

This subroutine add a 4D array of `real(4)` value in a variable structure. Dimension of the array must be ordered as ('x','y','z','t')

Parameters

in, out	<i>td_var</i>
---------	---------------

Optionally, you could specify the type of the variable to be used (default `real(4)`), and indices of the variable where value will be written with start and count array.

Author

J.Paul

Date

November, 2013 - Initial Version

January, 2019

- decompose array conversion on each dimension

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>rd_value</i>	array of variable value
in	<i>id_type</i>	type of the variable to be used (default <code>real(4)</code>)
in	<i>id_start</i>	start indices of the variable where data values will be written
in	<i>id_count</i>	number of indices selected along each dimension

The documentation for this interface was generated from the following file:

- `src/variable.f90`

11.119 var::var_clean Interface Reference

Public Member Functions

- subroutine `var__clean_unit` (`td_var`)
This subroutine clean variable structure.
- subroutine `var__clean_arr_1d` (`td_var`)
This subroutine clean 1D array of variable structure.
- subroutine `var__clean_arr_2d` (`td_var`)
This subroutine clean 2D array of variable structure.
- subroutine `var__clean_arr_3d` (`td_var`)
This subroutine clean 3D array of variable structure.

11.119.1 Member Function/Subroutine Documentation

11.119.1.1 subroutine var::var_clean::var__clean_arr_1d (type(tvar), dimension(:), intent(inout) *td_var*)

This subroutine clean 1D array of variable structure.

Parameters

<code>in, out</code>	<code>td_var</code>
----------------------	---------------------

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

<code>in, out</code>	<code>td_var</code>	array of variable strucutre
----------------------	---------------------	-----------------------------

11.119.1.2 subroutine `var::var_clean::var__clean_arr_2d` (`type(tvar)`, `dimension(:,:)`, `intent(inout) td_var`)

This subroutine clean 2D array of variable structure.

Parameters

<code>in, out</code>	<code>td_var</code>
----------------------	---------------------

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

<code>in, out</code>	<code>td_var</code>	array of variable strucutre
----------------------	---------------------	-----------------------------

11.119.1.3 subroutine `var::var_clean::var__clean_arr_3d` (`type(tvar)`, `dimension(:,,:)`, `intent(inout) td_var`)

This subroutine clean 3D array of variable structure.

Parameters

<code>in, out</code>	<code>td_var</code>
----------------------	---------------------

Author

J.Paul

Date

September, 2014 - Initial Version

Parameters

in, out	td_var	array of variable strucutre
---------	--------	-----------------------------

11.119.1.4 subroutine var::var_clean::var__clean_unit (type(tvar), intent(inout) td_var)

This subroutine clean variable structure.

Parameters

in, out	td_var	
---------	--------	--

Author

J.Paul

Date

November, 2013 - Initial Version
January, 2019

- nullify attributes structure inside variable strucutre

Parameters

in, out	td_var	variable strucutre
---------	--------	--------------------

The documentation for this interface was generated from the following file:

- src/variable.f90

11.120 var::var_copy Interface Reference

Public Member Functions

- type(tvar) function `var__copy_unit` (td_var, ld_value)
This subroutine copy variable structure in another one.
- type(tvar) function, dimension(size(td_var(:))) `var__copy_arr` (td_var)
This subroutine copy a array of variable structure in another one.

11.120.1 Member Function/Subroutine Documentation

11.120.1.1 type(tvar) function, dimension(size(td_var(:))) var::var_copy::var__copy_arr (type(tvar), dimension(:), intent(in) td_var)

This subroutine copy a array of variable structure in another one.

Parameters

in	td_var	
----	--------	--

see var__copy_unit

Warning

do not use on the output of a function who create or read an structure (ex: `tl_var=var_copy(var_init())` is forbidden). This will create memory leaks.
to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- use function instead of overload assignment operator (to avoid memory leak)

Parameters

in	<i>td_var</i>	array of variable structure
----	---------------	-----------------------------

Returns

copy of input array of variable structure

11.120.1.2 `type(tvar) function var::var_copy::var__copy_unit (type(tvar), intent(in) td_var, logical, intent(in), optional ld_value)`

This subroutine copy variable structure in another one.

Parameters

in	<i>td_var</i>	
----	---------------	--

variable values are copied in a transitional variable, so input and output variable structure values do not point on the same "memory cell", and so are independant.

Warning

do not use on the output of a function who create or read an structure (ex: `tl_var=var_copy(var_init())` is forbidden). This will create memory leaks.

to avoid infinite loop, do not use any function inside this subroutine

Author

J.Paul

Date

November, 2013 - Initial Version

November, 2014

- use function instead of overload assignment operator (to avoid memory leak)

July, 2017

- permit to copy variable structure without value

January, 2019

- use scalar instead of array, as transitional variable

February, 2019

- copy namout

Parameters

in	<i>td_var</i>	variable structure
in	<i>ld_value</i>	copy variable value (default .TRUE.)

Returns

copy of input variable structure

The documentation for this interface was generated from the following file:

- src/variable.f90

11.121 var::var_del_att Interface Reference**Public Member Functions**

- subroutine [var__del_att_name](#) (td_var, cd_name)
This subroutine delete an attribute from variable structure.
- subroutine [var__del_att_str](#) (td_var, td_att)
This subroutine delete an attribute from variable structure.

11.121.1 Member Function/Subroutine Documentation

11.121.1.1 subroutine var::var_del_att::var__del_att_name (type(tvar), intent(inout) *td_var*, character(len=*) intent(in) *cd_name*)

This subroutine delete an attribute from variable structure.

Parameters

in, out	<i>td_var</i>	
---------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

February, 2015

- define local attribute structure to avoid mistake with pointer

January, 2019

- clean attribute strucure

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>cd_name</i>	attribute name

11.121.1.2 subroutine var::var_del_att::var__del_att_str (type(tvar), intent(inout) *td_var*, type(tatt), intent(in) *td_att*)

This subroutine delete an attribute from variable structure.

Parameters

<code>in, out</code>	<code>td_var</code>
----------------------	---------------------

Author

J.Paul

Date

November, 2013- Initial Version

February, 2015

- delete highlight attribute too, when attribute is deleted

Parameters

<code>in, out</code>	<code>td_var</code>	variable structure
<code>in</code>	<code>td_att</code>	attribute structure

The documentation for this interface was generated from the following file:

- `src/variable.f90`

11.122 `var::var_init` Interface Reference

Public Member Functions

- type(`tvar`) function `var__init` (`cd_name`, `id_type`, `td_dim`, `td_att`, `dd_fill`, `cd_units`, `cd_axis`, `cd_stdname`, `cd_longname`, `cd_point`, `id_id`, `id_ew`, `dd_scf`, `dd_ofs`, `id_rec`, `dd_min`, `dd_max`, `ld_contiguous`, `ld_shuffle`, `ld_fletcher32`, `id_deflvl`, `id_chunksz`, `cd_interp`, `cd_extrap`, `cd_filter`, `cd_unt`, `dd_unf`, `cd_namout`)

This function initialize a variable structure, given variable name.

- type(`tvar`) function `var__init_dp` (`cd_name`, `dd_value`, `id_start`, `id_count`, `id_type`, `td_dim`, `td_att`, `dd_fill`, `cd_units`, `cd_axis`, `cd_stdname`, `cd_longname`, `cd_point`, `id_id`, `id_ew`, `dd_scf`, `dd_ofs`, `id_rec`, `dd_min`, `dd_max`, `ld_contiguous`, `ld_shuffle`, `ld_fletcher32`, `id_deflvl`, `id_chunksz`, `cd_interp`, `cd_extrap`, `cd_filter`, `cd_unt`, `dd_unf`, `cd_namout`)

This function initialize a variable structure, with a real(8) 4D array of value.

- type(`tvar`) function `var__init_1d_dp` (`cd_name`, `dd_value`, `id_start`, `id_count`, `id_type`, `td_dim`, `td_att`, `dd_fill`, `cd_units`, `cd_axis`, `cd_stdname`, `cd_longname`, `cd_point`, `id_id`, `id_ew`, `dd_scf`, `dd_ofs`, `id_rec`, `dd_min`, `dd_max`, `ld_contiguous`, `ld_shuffle`, `ld_fletcher32`, `id_deflvl`, `id_chunksz`, `cd_interp`, `cd_extrap`, `cd_filter`, `cd_unt`, `dd_unf`, `cd_namout`)

This function initialize a variable structure, with a real(8) 1D array of value.

- type(`tvar`) function `var__init_2d_dp` (`cd_name`, `dd_value`, `id_start`, `id_count`, `id_type`, `td_dim`, `td_att`, `dd_fill`, `cd_units`, `cd_axis`, `cd_stdname`, `cd_longname`, `cd_point`, `id_id`, `id_ew`, `dd_scf`, `dd_ofs`, `id_rec`, `dd_min`, `dd_max`, `ld_contiguous`, `ld_shuffle`, `ld_fletcher32`, `id_deflvl`, `id_chunksz`, `cd_interp`, `cd_extrap`, `cd_filter`, `cd_unt`, `dd_unf`, `cd_namout`)

This function initialize a variable structure, with a real(8) 2D array of value. optionally could be added:

- type(`tvar`) function `var__init_3d_dp` (`cd_name`, `dd_value`, `id_start`, `id_count`, `id_type`, `td_dim`, `td_att`, `dd_fill`, `cd_units`, `cd_axis`, `cd_stdname`, `cd_longname`, `cd_point`, `id_id`, `id_ew`, `dd_scf`, `dd_ofs`, `id_rec`, `dd_min`, `dd_max`, `ld_contiguous`, `ld_shuffle`, `ld_fletcher32`, `id_deflvl`, `id_chunksz`, `cd_interp`, `cd_extrap`, `cd_filter`, `cd_unt`, `dd_unf`, `cd_namout`)

This function initialize a variable structure, with a real(8) 3D array of value.

- type(`tvar`) function `var__init_sp` (`cd_name`, `rd_value`, `id_start`, `id_count`, `id_type`, `td_dim`, `td_att`, `rd_fill`, `cd_units`, `cd_axis`, `cd_stdname`, `cd_longname`, `cd_point`, `id_id`, `id_ew`, `dd_scf`, `dd_ofs`, `id_rec`, `dd_min`, `dd_max`, `ld_contiguous`, `ld_shuffle`, `ld_fletcher32`, `id_deflvl`, `id_chunksz`, `cd_interp`, `cd_extrap`, `cd_filter`, `cd_unt`, `dd_unf`, `cd_namout`)

This function initialize a variable structure, with a real(4) 4D array of value.

- type(**tvar**) function `var__init_1d_sp` (cd_name, rd_value, id_start, id_count, id_type, td_dim, td_att, rd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a real(4) 1D array of value.

- type(**tvar**) function `var__init_2d_sp` (cd_name, rd_value, id_start, id_count, id_type, td_dim, td_att, rd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a real(4) 2D array of value.

- type(**tvar**) function `var__init_3d_sp` (cd_name, rd_value, id_start, id_count, id_type, td_dim, td_att, rd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a real(4) 3D array of value.

- type(**tvar**) function `var__init_i1` (cd_name, bd_value, id_start, id_count, id_type, td_dim, td_att, bd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(1) 4D array of value.

- type(**tvar**) function `var__init_1d_i1` (cd_name, bd_value, id_start, id_count, id_type, td_dim, td_att, bd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(1) 1D array of value.

- type(**tvar**) function `var__init_2d_i1` (cd_name, bd_value, id_start, id_count, id_type, td_dim, td_att, bd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(1) 2D array of value.

- type(**tvar**) function `var__init_3d_i1` (cd_name, bd_value, id_start, id_count, id_type, td_dim, td_att, bd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(1) 3D array of value.

- type(**tvar**) function `var__init_i2` (cd_name, sd_value, id_start, id_count, id_type, td_dim, td_att, sd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(2) 4D array of value.

- type(**tvar**) function `var__init_1d_i2` (cd_name, sd_value, id_start, id_count, id_type, td_dim, td_att, sd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(2) 1D array of value.

- type(**tvar**) function `var__init_2d_i2` (cd_name, sd_value, id_start, id_count, id_type, td_dim, td_att, sd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(2) 2D array of value.

- type(**tvar**) function `var__init_3d_i2` (cd_name, sd_value, id_start, id_count, id_type, td_dim, td_att, sd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(2) 3D array of value.

- type(**tvar**) function `var__init_i4` (cd_name, id_value, id_start, id_count, id_type, td_dim, td_att, id_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(4) 4D array of value.

- type(**tvar**) function `var__init_1d_i4` (cd_name, id_value, id_start, id_count, id_type, td_dim, td_att, id_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(4) 1D array of value.

- type(**tvar**) function `var__init_2d_i4` (cd_name, id_value, id_start, id_count, id_type, td_dim, td_att, id_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(4) 2D array of value.

- type(**tvar**) function `var__init_3d_i4` (cd_name, id_value, id_start, id_count, id_type, td_dim, td_att, id_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(4) 3D array of value.

- type(**tvar**) function `var__init_i8` (cd_name, kd_value, id_start, id_count, id_type, td_dim, td_att, kd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(8) 4D array of value.

- type(**tvar**) function `var__init_1d_i8` (cd_name, kd_value, id_start, id_count, id_type, td_dim, td_att, kd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(8) 1D array of value.

- type(**tvar**) function `var__init_2d_i8` (cd_name, kd_value, id_start, id_count, id_type, td_dim, td_att, kd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(8) 2D array of value.

- type(**tvar**) function `var__init_3d_i8` (cd_name, kd_value, id_start, id_count, id_type, td_dim, td_att, kd_fill, cd_units, cd_axis, cd_stdname, cd_longname, cd_point, id_id, id_ew, dd_scf, dd_ofs, id_rec, dd_min, dd_max, ld_contiguous, ld_shuffle, ld_fletcher32, id_deflvl, id_chunksz, cd_interp, cd_extrap, cd_filter, cd_unt, dd_unf, cd_namout)

This function initialize a variable structure, with a integer(8) 3D array of value.

11.122.1 Member Function/Subroutine Documentation

11.122.1.1 `type(tvar) function var::var_init::var__init (character(len=*), intent(in) cd_name, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(dp), intent(in), optional dd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, given variable name.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

Optionally you could add 1D,2D,3D or 4D array of value, see `var__init_1D_dp`, `var__init_2D_dp`... for more information.

you could also add more information with the following optional arguments:

- `id_type` : integer(4) variable type, (as defined by NETCDF type constants).
- `td_dim` : array of dimension structure.
- `td_att` : array of attribute structure.
- `dd_fill` : real(8) variable FillValue. if none NETCDF FillValue will be used.
- `cd_units` : string character of units.
- `cd_axis` : string character of axis expected to be used
- `cd_stdname` : string character of variable standard name.
- `cd_longname` : string character of variable long name.
- `cd_point` : one character for ARAKAWA C-grid point name (T,U,V,F).
- `id_id` : variable id (read from a file).
- `id_ew` : number of point composing east west wrap band.
- `dd_unf` : real(8) value for units factor attribute.
- `dd_scf` : real(8) value for scale factor attribute.
- `dd_ofs` : real(8) value for add offset attribute.
- `id_rec` : record id (for rstdim file).
- `dd_min` : real(8) value for minimum value.
- `dd_max` : real(8) value for maximum value.
- `ld_contiguous` : use contiguous storage or not (for netcdf4).
- `ld_shuffle` : shuffle filter is turned on or not (for netcdf4).
- `ld_fletcher32` : fletcher32 filter is turned on or not (for netcdf4).
- `id_deflvl` : deflate level from 0 to 9, 0 indicates no deflation is in use (for netcdf4).
- `id_chunksz` : chunk size (for netcdf4).
- `cd_interp` : a array of character defining interpolation method.
- `cd_extrap` : a array of character defining extrapolation method.
- `cd_filter` : a array of character defining filtering method.
- `cd_unt` : a string character to define output unit
- `dd_unf` : real(8) factor applied to change unit

Note

most of these optionals arguments will be inform automatically, when reading variable from a file, or using configuration file `variable.cfg`.

Author

J.Paul

Date

November, 2013 - Initial Version

February, 2015

- Bug fix: conversion of the FillValue type (float case)

June, 2015

- add unit factor (to change unit)

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>dd_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	output unit (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.2 `type(tvar) function var::var_init::var__init_1d_dp (character(len=*), intent(in) cd_name, real(dp), dimension(:), intent(in) dd_value, integer(i4), intent(in), optional id_start, integer(i4), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(dp), intent(in), optional dd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvt, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a real(8) 1D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

Optionally could be added:

- dimension structure.
- attribute structure.

Dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrapol, and filter argument

July, 2015

- add unit factor (to change unit)

November, 2016

- allow to add scalar value

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>dd_value</i>	1D array of real(8) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>dd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset

in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.3 `type(tvar) function var::var_init::var__init_1d_i1 (character(len=*), intent(in) cd_name, integer(i1), dimension(:), intent(in) bd_value, integer(i4), intent(in), optional id_start, integer(i4), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i1), intent(in), optional bd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(1) 1D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>bd_value</i>	1D array of integer(1) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>bd_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.4 `type(tvar) function var::var_init::var__init_1d_i2 (character(len=*), intent(in) cd_name, integer(i2), dimension(:), intent(in) sd_value, integer(i4), intent(in), optional id_start, integer(i4), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i2), intent(in), optional sd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvt, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(2) 1D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrapol, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>sd_value</i>	1D array of integer(2) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>sd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset

in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.5 `type(tvar) function var::var_init::var__init_1d_i4 (character(len=*), intent(in) cd_name, integer(i4), dimension(:), intent(in) id_value, integer(i4), intent(in), optional id_start, integer(i4), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i4), intent(in), optional id_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(4) 1D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>id_value</i>	1D array of integer(4) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>id_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.6 `type(tvar) function var::var_init::var__init_1d_i8 (character(len=*), intent(in) cd_name, integer(i8), dimension(:), intent(in) kd_value, integer(i4), intent(in), optional id_start, integer(i4), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i8), intent(in), optional kd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvt, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(8) 1D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrapol, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	: variable name
<i>in</i>	<i>kd_value</i>	: 1D array of integer(8) value
<i>in</i>	<i>id_start</i>	: index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	: number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	: variable type
<i>in</i>	<i>td_dim</i>	: array of dimension structure
<i>in</i>	<i>td_att</i>	: array of attribute structure
<i>in</i>	<i>kd_fill</i>	: fill value
<i>in</i>	<i>cd_units</i>	: units
<i>in</i>	<i>cd_axis</i>	: axis expected to be used
<i>in</i>	<i>cd_stdname</i>	: variable standard name
<i>in</i>	<i>cd_longname</i>	: variable long name
<i>in</i>	<i>cd_point</i>	: point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	: variable id
<i>in</i>	<i>id_ew</i>	: east west wrap
<i>in</i>	<i>dd_scf</i>	: scale factor
<i>in</i>	<i>dd_ofs</i>	: add offset

in	<i>id_rec</i>	: record id (for rstdimg file)
in	<i>dd_min</i>	: minimum value
in	<i>dd_max</i>	: maximum value
in	<i>ld_contiguous</i>	: use contiguous storage or not
in	<i>ld_shuffle</i>	: shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	: fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	: deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	: chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.7 `type(tvar) function var::var_init::var__init_1d_sp (character(len=*), intent(in) cd_name, real(sp), dimension(:), intent(in) rd_value, integer(i4), intent(in), optional id_start, integer(i4), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(sp), intent(in), optional rd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a real(4) 1D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>rd_value</i>	1D array of real(4) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>rd_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.8 `type(tvar) function var::var_init::var__init_2d_dp (character(len=*), intent(in) cd_name, real(dp), dimension(:,:), intent(in) dd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(dp), intent(in), optional dd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional id_contiguous, logical, intent(in), optional id_shuffle, logical, intent(in), optional id_fletcher32, integer(i4), intent(in), optional id_deflvt, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a real(8) 2D array of value. optionally could be added:

Parameters

<i>in</i>	<i>cd_name</i>	- dimension structure. • attribute structure.
-----------	----------------	--

array of 2 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

February, 2015

- bug fix: array initialise with dimension array not only one value

June, 2015

- add interp, extrapol, and filter argument
- Bug fix: dimension array initialise not only one value

July, 2015

- add unit factor (to change unit)

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>dd_value</i>	1D array of real(8) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>dd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset
<i>in</i>	<i>id_rec</i>	record id (for rstdimg file)

in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.9 `type(tvar) function var::var_init::var_init_2d_i1 (character(len=*), intent(in) cd_name, integer(i1), dimension(:,:), intent(in) bd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i1), intent(in), optional bd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(1) 2D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 2 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>bd_value</i>	2D array of integer(1) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>bd_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.10 `type(tvar) function var::var_init::var__init_2d_i2 (character(len=*), intent(in) cd_name, integer(i2), dimension(:,:), intent(in) sd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i2), intent(in), optional sd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional id_contiguous, logical, intent(in), optional id_shuffle, logical, intent(in), optional id_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(2) 2D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 2 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrapol, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>sd_value</i>	2D array of integer(2) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>sd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset

in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.11 `type(tvar) function var::var_init::var__init_2d_i4 (character(len=*), intent(in) cd_name, integer(i4), dimension(:,:), intent(in) id_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i4), intent(in), optional id_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(4) 2D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 2 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y') and we use array size as lentgh dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>id_value</i>	2D array of integer(4) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>id_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.12 `type(tvar) function var::var_init::var__init_2d_i8 (character(len=*), intent(in) cd_name, integer(i8), dimension(:,:), intent(in) kd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i8), intent(in), optional kd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional id_contiguous, logical, intent(in), optional id_shuffle, logical, intent(in), optional id_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(8) 2D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 2 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrapol, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>kd_value</i>	2D array of integer(8) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>kd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset

in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.13 `type(tvar) function var::var_init::var__init_2d_sp (character(len=*), intent(in) cd_name, real(sp), dimension(:,:), intent(in) rd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(sp), intent(in), optional rd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a real(4) 2D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 2 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	: variable name
in	<i>rd_value</i>	: 2D array of real(4) value
in	<i>id_start</i>	: index in the variable from which the data values will be read
in	<i>id_count</i>	: number of indices selected along each dimension
in	<i>id_type</i>	: variable type
in	<i>td_dim</i>	: array of dimension structure
in	<i>td_att</i>	: array of attribute structure
in	<i>rd_fill</i>	: fill value
in	<i>cd_units</i>	: units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	: variable standard name
in	<i>cd_longname</i>	: variable long name
in	<i>cd_point</i>	: point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	: variable id
in	<i>id_ew</i>	: east west wrap
in	<i>dd_scf</i>	: scale factor
in	<i>dd_ofs</i>	: add offset
in	<i>id_rec</i>	: record id (for rstdimg file)
in	<i>dd_min</i>	: minimum value
in	<i>dd_max</i>	: maximum value
in	<i>ld_contiguous</i>	: use contiguous storage or not
in	<i>ld_shuffle</i>	: shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	: fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	: deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	: chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.14 `type(tvar) function var::var_init::var_init_3d_dp (character(len=*), intent(in) cd_name, real(dp), dimension(:, :, :), intent(in) dd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(dp), intent(in), optional dd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional id_contiguous, logical, intent(in), optional id_shuffle, logical, intent(in), optional id_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a real(8) 3D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 3 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>dd_value</i>	1D array of real(8) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>dd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset
<i>in</i>	<i>id_rec</i>	record id (for rstdimg file)
<i>in</i>	<i>dd_min</i>	minimum value

in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.15 `type(tvar) function var::var_init::var_init_3d_i1 (character(len=*), intent(in) cd_name, integer(i1), dimension(:,:,:), intent(in) bd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i1), intent(in), optional bd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(1) 3D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 3 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>bd_value</i>	3D array of integer(1) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>bd_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.16 `type(tvar) function var::var_init::var__init_3d_i2 (character(len=*), intent(in) cd_name, integer(i2), dimension(:,:,:), intent(in) sd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i2), intent(in), optional sd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(2) 3D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 3 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>sd_value</i>	3D array of integer(2) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>sd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset

in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.17 `type(tvar) function var::var_init::var__init_3d_i4 (character(len=*), intent(in) cd_name, integer(i4), dimension(:,:,:), intent(in) id_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i4), intent(in), optional id_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(4) 3D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 3 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>id_value</i>	3D array of integer(4) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>id_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.18 `type(tvar) function var::var_init::var__init_3d_i8 (character(len=*), intent(in) cd_name, integer(i8), dimension(:,:,:), intent(in) kd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i8), intent(in), optional kd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional id_contiguous, logical, intent(in), optional id_shuffle, logical, intent(in), optional id_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(8) 3D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 3 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrapol, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>kd_value</i>	2D array of integer(8) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>kd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset

in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.19 `type(tvar) function var::var_init::var__init_3d_sp (character(len=*), intent(in) cd_name, real(sp), dimension(:,:,:), intent(in) rd_value, integer(i4), dimension(:), intent(in), optional id_start, integer(i4), dimension(:), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(sp), intent(in), optional rd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a real(4) 3D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

array of 3 dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	: variable name
in	<i>rd_value</i>	: 2D array of real(4) value
in	<i>id_start</i>	: index in the variable from which the data values will be read
in	<i>id_count</i>	: number of indices selected along each dimension
in	<i>id_type</i>	: variable type
in	<i>td_dim</i>	: array of dimension structure
in	<i>td_att</i>	: array of attribute structure
in	<i>rd_fill</i>	: fill value
in	<i>cd_units</i>	: units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	: variable standard name
in	<i>cd_longname</i>	: variable long name
in	<i>cd_point</i>	: point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	: variable id
in	<i>id_ew</i>	: east west wrap
in	<i>dd_scf</i>	: scale factor
in	<i>dd_ofs</i>	: add offset
in	<i>id_rec</i>	: record id (for rstdimg file)
in	<i>dd_min</i>	: minimum value
in	<i>dd_max</i>	: maximum value
in	<i>ld_contiguous</i>	: use contiguous storage or not
in	<i>ld_shuffle</i>	: shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	: fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	: deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	: chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.20 `type(tvar) function var::var_init::var_init_dp (character(len=*), intent(in) cd_name, real(dp), dimension(:, :, :, :), intent(in) dd_value, integer(i4), dimension(ip_maxdim), intent(in), optional id_start, integer(i4), dimension(ip_maxdim), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(dp), intent(in), optional dd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional id_contiguous, logical, intent(in), optional id_shuffle, logical, intent(in), optional id_fletcher32, integer(i4), intent(in), optional id_deflvt, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a real(8) 4D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

Dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z','t') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>dd_value</i>	4D array of real(8) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>dd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset
<i>in</i>	<i>id_rec</i>	record id (for rstdimg file)
<i>in</i>	<i>dd_min</i>	minimum value

in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.21 `type(tvar) function var::var_init::var__init_i1 (character(len=*), intent(in) cd_name, integer(i1), dimension(:, :, :, :), intent(in) bd_value, integer(i4), dimension(ip_maxdim), intent(in), optional id_start, integer(i4), dimension(ip_maxdim), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i1), intent(in), optional bd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(1) 4D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

Dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z','t') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>bd_value</i>	4D array of integer(1) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>bd_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.22 `type(tvar) function var::var_init::var__init_i2 (character(len=*), intent(in) cd_name, integer(i2), dimension(:, :, :, :), intent(in) sd_value, integer(i4), dimension(ip_maxdim), intent(in), optional id_start, integer(i4), dimension(ip_maxdim), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i2), intent(in), optional sd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional id_contiguous, logical, intent(in), optional id_shuffle, logical, intent(in), optional id_fletcher32, integer(i4), intent(in), optional id_deflvt, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(2) 4D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

Dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z','t') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrapol, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>sd_value</i>	4D array of integer(2) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>sd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset

in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.23 `type(tvar) function var::var_init::var_init_i4 (character(len=*), intent(in) cd_name, integer(i4), dimension(:, :, :, :), intent(in) id_value, integer(i4), dimension(ip_maxdim), intent(in), optional id_start, integer(i4), dimension(ip_maxdim), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i4), intent(in), optional id_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(4) 4D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

Dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z','t') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>id_value</i>	4D array of integer(4) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>id_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.24 `type(tvar) function var::var_init::var__init_i8 (character(len=*), intent(in) cd_name, integer(i8), dimension(:, :, :, :), intent(in) kd_value, integer(i4), dimension(ip_maxdim), intent(in), optional id_start, integer(i4), dimension(ip_maxdim), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, integer(i8), intent(in), optional kd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional id_contiguous, logical, intent(in), optional id_shuffle, logical, intent(in), optional id_fletcher32, integer(i4), intent(in), optional id_deflvt, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a integer(8) 4D array of value.

Parameters

<i>in</i>	<i>cd_name</i>
-----------	----------------

optionally could be added:

- dimension structure.
- attribute structure.

Dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z','t') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrapol, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

<i>in</i>	<i>cd_name</i>	variable name
<i>in</i>	<i>kd_value</i>	4D array of integer(8) value
<i>in</i>	<i>id_start</i>	index in the variable from which the data values will be read
<i>in</i>	<i>id_count</i>	number of indices selected along each dimension
<i>in</i>	<i>id_type</i>	variable type
<i>in</i>	<i>td_dim</i>	array of dimension structure
<i>in</i>	<i>td_att</i>	array of attribute structure
<i>in</i>	<i>kd_fill</i>	fill value
<i>in</i>	<i>cd_units</i>	units
<i>in</i>	<i>cd_axis</i>	axis expected to be used
<i>in</i>	<i>cd_stdname</i>	variable standard name
<i>in</i>	<i>cd_longname</i>	variable long name
<i>in</i>	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
<i>in</i>	<i>id_id</i>	variable id
<i>in</i>	<i>id_ew</i>	east west wrap
<i>in</i>	<i>dd_scf</i>	scale factor
<i>in</i>	<i>dd_ofs</i>	add offset

in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

11.122.1.25 `type(tvar) function var::var_init::var__init_sp (character(len=*), intent(in) cd_name, real(sp), dimension(:, :, :, :), intent(in) rd_value, integer(i4), dimension(ip_maxdim), intent(in), optional id_start, integer(i4), dimension(ip_maxdim), intent(in), optional id_count, integer(i4), intent(in), optional id_type, type(tdim), dimension(:), intent(in), optional td_dim, type(tatt), dimension(:), intent(in), optional td_att, real(sp), intent(in), optional rd_fill, character(len=*), intent(in), optional cd_units, character(len=*), intent(in), optional cd_axis, character(len=*), intent(in), optional cd_stdname, character(len=*), intent(in), optional cd_longname, character(len=*), intent(in), optional cd_point, integer(i4), intent(in), optional id_id, integer(i4), intent(in), optional id_ew, real(dp), intent(in), optional dd_scf, real(dp), intent(in), optional dd_ofs, integer(i4), intent(in), optional id_rec, real(dp), intent(in), optional dd_min, real(dp), intent(in), optional dd_max, logical, intent(in), optional ld_contiguous, logical, intent(in), optional ld_shuffle, logical, intent(in), optional ld_fletcher32, integer(i4), intent(in), optional id_deflvl, integer(i4), dimension(ip_maxdim), intent(in), optional id_chunksz, character(len=*), dimension(2), intent(in), optional cd_interp, character(len=*), dimension(1), intent(in), optional cd_extrap, character(len=*), dimension(5), intent(in), optional cd_filter, character(len=*), intent(in), optional cd_unt, real(dp), intent(in), optional dd_unf, character(len=*), intent(in), optional cd_namout)`

This function initialize a variable structure, with a real(4) 4D array of value.

Parameters

in	<i>cd_name</i>
----	----------------

optionally could be added:

- dimension structure.
- attribute structure.

Dimension structure is needed to put value in variable structure. If none is given, we assume array is ordered as ('x','y','z','t') and we use array size as length dimension.

indices in the variable where value will be written could be specify if start and count array are given. Dimension structure is needed in that case.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- add interp, extrap, and filter argument

July, 2015

- add unit factor (to change unit)

January, 2019

- decompose array conversion on each dimension

February, 2019

- add output name (to change name)

Parameters

in	<i>cd_name</i>	variable name
in	<i>rd_value</i>	4D array of real(4) value
in	<i>id_start</i>	index in the variable from which the data values will be read
in	<i>id_count</i>	number of indices selected along each dimension
in	<i>id_type</i>	variable type
in	<i>td_dim</i>	array of dimension structure
in	<i>td_att</i>	array of attribute structure
in	<i>rd_fill</i>	fill value
in	<i>cd_units</i>	units
in	<i>cd_axis</i>	axis expected to be used
in	<i>cd_stdname</i>	variable standard name
in	<i>cd_longname</i>	variable long name
in	<i>cd_point</i>	point on Arakawa-C grid (T,U,V,F)
in	<i>id_id</i>	variable id
in	<i>id_ew</i>	east west wrap
in	<i>dd_scf</i>	scale factor
in	<i>dd_ofs</i>	add offset
in	<i>id_rec</i>	record id (for rstdimg file)
in	<i>dd_min</i>	minimum value
in	<i>dd_max</i>	maximum value
in	<i>ld_contiguous</i>	use contiguous storage or not
in	<i>ld_shuffle</i>	shuffle filter is turned on or not
in	<i>ld_fletcher32</i>	fletcher32 filter is turned on or not
in	<i>id_deflvl</i>	deflate level from 0 to 9, 0 indicates no deflation is in use
in	<i>id_chunksz</i>	chunk size
in	<i>cd_interp</i>	interpolation method
in	<i>cd_extrap</i>	extrapolation method
in	<i>cd_filter</i>	filter method
in	<i>cd_unt</i>	new units (linked to units factor)
in	<i>dd_unf</i>	units factor
in	<i>cd_namout</i>	output name (renamed variable)

Returns

variable structure

The documentation for this interface was generated from the following file:

- src/variable.f90

11.123 var::var_print Interface Reference

Public Member Functions

- subroutine [var__print_unit](#) (td_var, Id_more)
This subroutine print variable information.
- subroutine [var__print_arr](#) (td_var)
This subroutine print informations of an array of variables.

11.123.1 Member Function/Subroutine Documentation

11.123.1.1 subroutine var::var_print::var__print_arr (type(tvar), dimension(:), intent(in) td_var)

This subroutine print informations of an array of variables.

Parameters

in	td_var
----	--------

Author

J.Paul

Date

June, 2014 - Initial Version

Parameters

in	td_var	array of variables structure
----	--------	------------------------------

11.123.1.2 subroutine var::var_print::var__print_unit (type(tvar), intent(in) td_var, logical, intent(in), optional Id_more)

This subroutine print variable information.

Parameters

in	td_var
----	--------

If Id_more is TRUE (default), print information about variable dimensions and variable attributes.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	td_var	variable structure
in	Id_more	print more information about variable

The documentation for this interface was generated from the following file:

- src/variable.f90

11.124 vgrid Module Reference

This module manage vertical grid.

Public Member Functions

- subroutine, public [vgrid_zgr_z](#) (dd_gdepw, dd_gdept, dd_e3w, dd_e3t, dd_e3w_1d, dd_e3t_1d, dd_ppkth, dd_ppkth2, dd_ppacr, dd_ppacr2, dd_ppdzmin, dd_pphmax, dd_pp_to_be_computed, dd_ppa0, dd_ppa1, dd_ppa2, dd_ppsur)
This subroutine set the depth of model levels and the resulting vertical scale factors.
- subroutine [vgrid_zgr_bat](#) (dd_bathy, dd_gdepw, dd_hmin, dd_fill)
This subroutine.
- subroutine, public [vgrid_zgr_zps](#) (id_mbathy, dd_bathy, id_jpkmax, dd_gdepw, dd_e3t, dd_e3zps_min, dd_e3zps_rat, dd_fill)
This subroutine set the depth and vertical scale factor in partial step z-coordinate case.
- subroutine, public [vgrid_zgr_bat_ctl](#) (id_mbathy, id_jpkmax, id_jpk)
This subroutine check the bathymetry in levels.
- type(tvar) function, dimension(ip_npoint),
public [vgrid_get_level](#) (td_bathy, cd_namelist, td_dom, id_nlevel)
This function compute bathy level in T,U,V,F point, and return them as array of variable structure.

11.124.1 Detailed Description

This module manage vertical grid.

to set the depth of model levels and the resulting vertical scale

factors:

```
CALL vgrid_zgr_z(dd_gdepw(:), dd_gdept(:), dd_e3w(:), dd_e3t(:),
               dd_ppkth, dd_ppkth2, dd_ppacr, dd_ppacr2,
               dd_ppdzmin, dd_pphmax, dd_pp_to_be_computed,
               dd_ppa0, dd_ppa1, dd_ppa2, dd_ppsur)
```

- dd_gdepw is array of depth value on W point
- dd_gdept is array of depth value on T point
- dd_e3w is array of vertical mesh size on W point
- dd_e3t is array of vertical mesh size on T point
- dd_ppkth see NEMO documentation
- dd_ppkth2 see NEMO documentation
- dd_ppacr see NEMO documentation
- dd_ppdzmin see NEMO documentation
- dd_pphmax see NEMO documentation

- `dd_pp_to_be_computed` see NEMO documentation
- `dd_ppa1` see NEMO documentation
- `dd_ppa2` see NEMO documentation
- `dd_ppa0` see NEMO documentation
- `dd_ppsur` see NEMO documentation

to set the depth and vertical scale factor in partial step z-coordinate

case:

```
CALL vgrid_zgr_zps(id_mbathy(:, :), dd_bathy(:, :), id_jpkmax, dd_gdepw(:),
                 dd_e3t(:), dd_e3zps_min, dd_e3zps_rat)
```

- `id_mbathy` is array of bathymetry level
- `dd_bathy` is array of bathymetry
- `id_jpkmax` is the maximum number of level to be used
- `dd_gdepw` is array of vertical mesh size on W point
- `dd_e3t` is array of vertical mesh size on T point
- `dd_e3zps_min` see NEMO documentation
- `dd_e3zps_rat` see NEMO documentation

to check the bathymetry in levels:

```
CALL vgrid_zgr_bat_ctl(id_mbathy, id_jpkmax, id_jpk)
```

- `id_mbathy` is array of bathymetry level
- `id_jpkmax` is the maximum number of level to be used
- `id_jpk` is the number of level

to compute bathy level in T,U,V,F point from Bathymetry file:

```
tl_level(:)=vgrid_get_level(td_bathy, [cd_namelist,] [td_dom,] [id_nlevel])
```

- `td_bathy` is Bathymetry file structure
- `cd_namelist` is namelist [optional]
- `td_dom` is domain structure [optional]
- `id_nlevel` is number of level to be used [optional]

Author

J.Paul

Date

November, 2013 - Initial Version

Spetember, 2014

- add header

June, 2015 - update subroutine with NEMO 3.6

Todo

- fusionner vgrid et [grid_zgr](#)

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

11.124.2 Member Function/Subroutine Documentation

11.124.2.1 `type(tvar) function, dimension(ip_npoint), public vgrid::vgrid_get_level (type(tmpp), intent(in) td_bathy, character(len=*) , intent(in), optional cd_namelist, type(tdom), intent(in), optional td_dom, integer(i4), intent(in), optional id_nlevel)`

This function compute bathy level in T,U,V,F point, and return them as array of variable structure.

Parameters

<i>in</i>	<i>td_bathy</i>
-----------	-----------------

Bathymetry is read on Bathymetry file, then bathy level is computed on T point, and finally fit to U,V,F point.

you could specify :

- namelist where find parameter to set the depth of model levels (default use GLORYS 75 levels parameters)
- domain structure to specify on e area to work on
- number of level to be used

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<i>in</i>	<i>td_bathy</i>	Bathymetry file structure
<i>in</i>	<i>cd_namelist</i>	namelist
<i>in</i>	<i>td_dom</i>	domain structure
<i>in</i>	<i>id_nlevel</i>	number of lelvel to be used

Returns

array of level on T,U,V,F point (variable structure)

11.124.2.2 `subroutine vgrid::vgrid_zgr_bat (real(dp), dimension(:, :), intent(inout) dd_bathy, real(dp), dimension(:), intent(in) dd_gdepw, real(dp), intent(in) dd_hmin, real(dp), intent(in), optional dd_fill)`

This subroutine.

Parameters

in, out	<i>dd_bathy</i>	todo 25.
in, out	<i>dd_bathy</i>	
in	<i>dd_gdepw</i>	
in	<i>dd_hmin</i>	
in	<i>dd_fill</i>	

11.124.2.3 subroutine, public vgrid::vgrid_zgr_bat_ctl (integer(i4), dimension(:,:), intent(inout) *id_mbathy*, integer(i4), intent(inout) *id_jpkmax*, integer(i4), intent(inout) *id_jpk*)

This subroutine check the bathymetry in levels.

Parameters

in, out	<i>id_mbathy</i>
---------	------------------

** Method : The array mbathy is checked to verified its consistency with the model options. in particular: mbathy must have at least 1 land grid-points ($mbathy \leq 0$) along closed boundary. mbathy must be cyclic IF $jperio=1$. mbathy must be lower or equal to $jpk-1$. isolated ocean grid points are suppressed from mbathy since they are only connected to remaining ocean through vertical diffusion. C A U T I O N : mbathy will be modified during the initialization phase to become the number of non-zero w-levels of a water column, with a minimum value of 1.

** Action : - update mbathy: level bathymetry (in level index)

- update bathy : meter bathymetry (in meters)

Author

G.Madec

Date

Marsh, 2008 - Original code

Parameters

in	<i>id_mbathy</i>
in	<i>id_jpkmax</i>
in	<i>id_jpk</i>

11.124.2.4 subroutine, public vgrid::vgrid_zgr_z (real(dp), dimension(:), intent(inout) *dd_gdepw*, real(dp), dimension(:), intent(inout) *dd_gdept*, real(dp), dimension(:), intent(inout) *dd_e3w*, real(dp), dimension(:), intent(inout) *dd_e3t*, real(dp), dimension(:), intent(inout) *dd_e3w_1d*, real(dp), dimension(:), intent(inout) *dd_e3t_1d*, real(dp), intent(in) *dd_ppkth*, real(dp), intent(in) *dd_ppkth2*, real(dp), intent(in) *dd_ppacr*, real(dp), intent(in) *dd_ppacr2*, real(dp), intent(in) *dd_ppdzmin*, real(dp), intent(in) *dd_pphmax*, real(dp), intent(in) *dd_pp_to_be_computed*, real(dp), intent(in) *dd_ppa0*, real(dp), intent(in) *dd_ppa1*, real(dp), intent(in) *dd_ppa2*, real(dp), intent(in) *dd_ppsur*)

This subroutine set the depth of model levels and the resulting vertical scale factors.

Parameters

in, out	<i>dd_gdepw</i>
---------	-----------------

** Method : z-coordinate system (use in all type of coordinate) The depth of model levels is defined from an analytical function the derivative of which gives the scale factors. both depth and scale factors only depend on k (1d arrays).

w-level: $gdepw = fsdep(k)$

$e3w(k) = dk(fsdep)(k) = fse3(k)$

t-level: $gdept = fsdep(k+0.5)$

$e3t(k) = dk(fsdep)(k+0.5) = fse3(k+0.5)$

** Action : - $gdept, gdepw$: depth of T- and W-point (m)

- $e3t, e3w$: scale factors at T- and W-levels (m)

Author

G. Madec

Date

Marsh,2008 - F90: Free form and module

Note

Reference : Marti, Madec & Delecluse, 1992, JGR, 97, No8, 12,763-12,766.

Parameters

in, out	<i>dd_gdepw</i>	
in, out	<i>dd_gdept</i>	
in, out	<i>dd_e3w</i>	
in, out	<i>dd_e2t</i>	
in	<i>dd_ppkth</i>	
in	<i>dd_ppkth2</i>	
in	<i>dd_ppacr</i>	
in	<i>dd_ppacr2</i>	
in	<i>dd_ppdzmin</i>	
in	<i>dd_pphmax</i>	
in	<i>dd_pp_to_be_ - computed</i>	
in	<i>dd_ppa1</i>	
in	<i>dd_ppa2</i>	
in	<i>dd_ppa0</i>	
in	<i>dd_ppsur</i>	

11.124.2.5 subroutine, public vgrid::vgrid_zgr_zps (integer(i4), dimension(:,:), intent(out) *id_mbathy*, real(dp), dimension(:,:), intent(inout) *dd_bathy*, integer(i4), intent(inout) *id_jpkmax*, real(dp), dimension(:), intent(in) *dd_gdepw*, real(dp), dimension(:), intent(in) *dd_e3t*, real(dp), intent(in) *dd_e3zps_min*, real(dp), intent(in) *dd_e3zps_rat*, real(dp), intent(in), optional *dd_fill*)

This subroutine set the depth and vertical scale factor in partial step z-coordinate case.

Parameters

out	<i>id_mbathy</i>
-----	------------------

** Method : Partial steps : computes the 3D vertical scale factors of T-, U-, V-, W-, UW-, VW and F-points that are associated with a partial step representation of bottom topography.

The reference depth of model levels is defined from an analytical function the derivative of which gives the reference vertical scale factors. From depth and scale factors reference, we compute there new value with partial steps on 3d arrays (i, j, k).

w-level:

- $gdepw_ps(i,j,k) = fsdep(k)$
- $e3w_ps(i,j,k) = dk(fsdep)(k) = fse3(i,j,k)$ t-level:
- $gdept_ps(i,j,k) = fsdep(k+0.5)$
- $e3t_ps(i,j,k) = dk(fsdep)(k+0.5) = fse3(i,j,k+0.5)$

With the help of the bathymetric file (bathymetry_depth_ORCA_R2.nc), we find the mbathy index of the depth at each grid point. This leads us to three cases:

- $bathy = 0 \Rightarrow mbathy = 0$
- $1 < mbathy < jpkm1$
- $bathy > gdepw(jpk) \Rightarrow mbathy = jpkm1$

Then, for each case, we find the new depth at t- and w- levels and the new vertical scale factors at t-, u-, v-, w-, uw-, vw- and f-points.

This routine is given as an example, it must be modified following the user s desiderata. nevertheless, the output as well as the way to compute the model levels and scale factors must be respected in order to insure second order accuracy schemes.

Warning

- gdept, gdepw and e3 are positives
- gdept_ps, gdepw_ps and e3_ps are positives

Author

A. Bozec, G. Madec

Date

February, 2009 - F90: Free form and module

February, 2009

- A. de Miranda : rigid-lid + islands

Note

Reference : Pacanowsky & Gnanadesikan 1997, Mon. Wea. Rev., 126, 3248-3270.

Parameters

<i>in, out</i>	<i>id_mbathy</i>	
<i>in, out</i>	<i>dd_bathy</i>	
<i>in, out</i>	<i>id_jpkmax</i>	
<i>in</i>	<i>dd_gdepw</i>	
<i>in</i>	<i>dd_e3t</i>	
<i>in</i>	<i>dd_e3zps_min</i>	
<i>in</i>	<i>dd_e3zps_rat</i>	
<i>in</i>	<i>dd_fill</i>	

The documentation for this module was generated from the following file:

- `src/vgrid.f90`

Chapter 12

File Documentation

12.1 src/addline_deg.f90 File Reference

This program add line to all variables of the input file.

Functions/Subroutines

- program **addline_deg**
- type(tvar) function [add_line](#) (td_var, td_mpp, td_coord)
This function add line to variable and return variable structure.

12.1.1 Detailed Description

This program add line to all variables of the input file.

12.1.2 how to

to add line to file:

```
./SIREN/bin/addline addline.nam
```

the namelist file (**addline.nam**) sets up program parameters.

to set up program parameters, you just have to fill the namelist file (**add_line.nam**).

Note

you could find a template of the namelist in templates directory.

create_bathy.nam comprise 4 namelists:

- **namlog** to set logger parameters
- **namcfg** to set configuration file parameters
- **namsrc** to set source grid parameters
- **namout** to set output parameters

here after, each sub-namelist parameters is detailed.

Note

default values are specified between brackets

12.1.2.1 namlog

the logger sub-namelist parameters are :

- **cn_logfile** [*addline.log*]
logger filename
- **cn_verbosity** [*warning*]
verbosity level, choose between :
 - trace
 - debug
 - info
 - warning
 - error
 - fatal
 - none
- **in_maxerror** [*5*]
maximum number of error allowed

12.1.2.2 namcfg

the configuration sub-namelist parameters are :

- **cn_varcfg** [*./cfg/variable.cfg*]
path to the variable configuration file.
the variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.
- **cn_dimcfg** [*./cfg/dimension.cfg*]
path to the dimension configuration file.
the dimension configuration file defines dimensions allowed.
- **cn_dumcfg** [*./cfg/dummy.cfg*]
path to the useless (dummy) configuration file.
the dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

12.1.2.3 namsrc

the source/coarse grid sub-namelist parameters are :

- **cn_coord0** []
path to the coordinate file
- **in_perio0** []
NEMO periodicity index
the NEMO periodicity could be choose between 0 to 6:

- in_perio=0** standard regional model
- in_perio=1** east-west cyclic model
- in_perio=2** model with symmetric boundary condition across the equator
- in_perio=3** regional model with North fold boundary and T-point pivot
- in_perio=4** global model with a T-point pivot.
example: ORCA2, ORCA025, ORCA12
- in_perio=5** regional model with North fold boundary and F-point pivot
- in_perio=6** global model with a F-point pivot
example: ORCA05

See Also

For more information see [NEMO periodicity](#) and Model Boundary Condition paragraph in the [NEMO documentation](#)

12.1.2.4 namvar

the variable sub-namelist parameters are :

- **cn_varfile []**

list of variable, and associated file

cn_varfile is the path and filename of the file where find variable.

Note

cn_varfile could be a matrix of value, if you want to handwrite variable value.
the variable array of value is split into equal subdomain.
each subdomain is filled with the corresponding value of the matrix.
separators used to defined matrix are:

- ‘;’ for line
- ‘/’ for row Example:

$$3,2,3/1,4,5 \Rightarrow \begin{pmatrix} 3 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}$$

Examples:

- ‘Bathymetry:gridT.nc’

Note

Optionnaly, NEMO periodicity could be added following the filename. the periodicity must be separated by ‘;’

Example:

- ‘Bathymetry:gridT.nc ; perio=4’

- **cn_varinfo []**

list of variable and extra information about request(s) to be used

each elements of *cn_varinfo* is a string character (separated by ‘,’).

it is composed of the variable name follow by ‘:’, then request(s) to be used on this variable.

request could be:

- int = interpolation method
- ext = extrapolation method
- flt = filter method

- min = minimum value
 - max = maximum value
 - unt = new units
 - unf = unit scale factor (linked to new units)
- requests must be separated by ','.
order of requests does not matter.

informations about available method could be find in [interp](#), [extrap](#) and [filter](#) modules.

Example:

- 'Bathymetry: flt=2*hamming(2,3); min=0'

Note

If you do not specify a method which is required, default one is apply.

12.1.2.5 namout

the output sub-namelist parameter is :

- **cn_fileout** [*addline_deg.nc*]
output filename
- **In_extrap** [.FALSE.]
extrapolate extra line
- **In_copy** [.FALSE.]
copy extra line from above
- **in_nproc** [*1*]
number of processor to be used
- **in_niproc** [*1*]
i-direction number of processor
- **in_njproc** [*1*]
j-direction numebr of processor

Author

J.Paul

Date

October, 2015 - Initial Version

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

12.1.3 Function/Subroutine Documentation

12.1.3.1 `type(tvar) function addline_deg::add_line (type(tvar), intent(in) td_var, type(tmpp), intent(in) td_mpp, type(tmpp), intent(in) td_coord)`

This function add line to variable and return variable structure.

Parameters

in	td_var
----	--------

Author

J.Paul

Date

October, 2015 - Initial Version

Parameters

in	td_var	variable structure
in	td_mpp	mpp file structure
in	td_coord	coordinate file structure

Returns

variable structure

12.2 src/create_bathy.f90 File Reference

This program creates fine grid bathymetry file.

Functions/Subroutines

- program **create_bathy**
- type(tvar) function [create_bathy_matrix](#) (td_var, td_coord, ld_rand)
This function create variable, filled with matrix value.
- type(tvar) function [create_bathy_extract](#) (td_var, td_mpp, td_coord)
This function extract variable from file over coordinate domain and return variable structure.
- type(tvar) function [create_bathy_get_var](#) (td_var, td_mpp, id_imin, id_jmin, id_imax, id_jmax, id_offset, id_rho)
This function get coarse grid variable, interpolate variable, and return variable structure over fine grid.
- subroutine [create_bathy_interp](#) (td_var, id_rho, id_offset, id_iext, id_jext)
This subroutine interpolate variable.
- subroutine [create_bathy_check_depth](#) (td_mpp, td_depth)
This subroutine get depth variable value in an open mpp structure and check if agree with already input depth variable.
- subroutine [create_bathy_check_time](#) (td_mpp, td_time)
This subroutine get date and time in an open mpp structure and check if agree with date and time already read.

12.2.1 Detailed Description

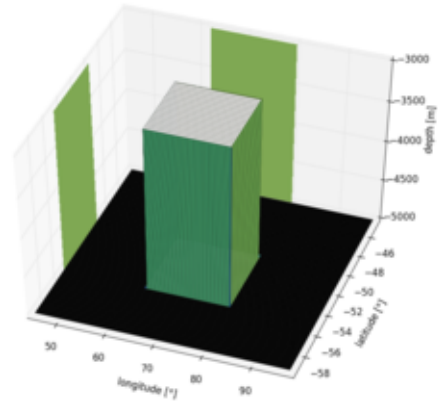
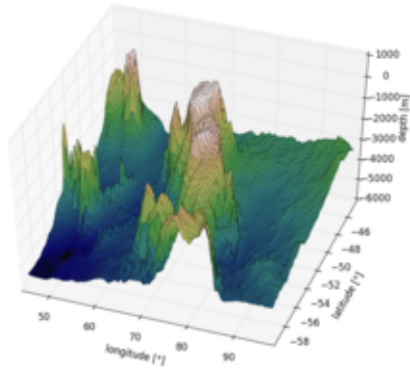
This program creates fine grid bathymetry file.

12.2.2 method

This bathymetry could be :

- extracted from a wider fine grid bathymetry file

- interpolated from a wider coarse grid bathymetry file
- handwritten



12.2.3 how to

USAGE: create_bathy create_bathy.nam [-v] [-h]

- positional arguments:
 - create_bathy.nam
namelist of create_bathy

Note

a template of the namelist could be created running (in templates directory):

```
python create_templates.py create_bathy
```

- optional arguments:
 - -h, --help
show this help message (and exit)
 - -v, --version
show Siren's version (and exit)

12.2.4 create_bathy.nam

create_bathy.nam contains 7 sub-namelists:

- **namlog** to set logger parameters
- **namcfg** to set configuration file parameters
- **namsrc** to set source/coarse grid parameters
- **namtgt** to set target/fine grid parameters
- **namvar** to set variable parameters
- **namnst** to set sub domain and nesting parameters
- **namout** to set output parameters

here after, each sub-namelist parameters is detailed.

Note

default values are specified between brackets

12.2.4.1 namlog

the logger sub-namelist parameters are :

- **cn_logfile** [*create_bathy.log*]
logger filename
- **cn_verbosity** [*warning*]
verbosity level, choose between :
 - trace
 - debug
 - info
 - warning
 - error
 - fatal
 - none
- **in_maxerror** [*5*]
maximum number of error allowed

12.2.4.2 namcfg

the configuration sub-namelist parameters are :

- **cn_varcfg** [*./cfg/variable.cfg*]
path to the variable configuration file.
the variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.
- **cn_dimcfg** [*./cfg/dimension.cfg*]
path to the dimension configuration file.
the dimension configuration file defines dimensions allowed.
- **cn_dumcfg** [*./cfg/dummy.cfg*]
path to the useless (dummy) configuration file.
the dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

12.2.4.3 namsrc

the source/coarse grid sub-namelist parameters are :

- **cn_coord0** []
path to the coordinate file
- **in_perio0** []
NEMO periodicity index
the NEMO periodicity could be choose between 0 to 6:

- in_perio=0** standard regional model
- in_perio=1** east-west cyclic model
- in_perio=2** model with symmetric boundary condition across the equator
- in_perio=3** regional model with North fold boundary and T-point pivot
- in_perio=4** global model with a T-point pivot.
example: ORCA2, ORCA025, ORCA12
- in_perio=5** regional model with North fold boundary and F-point pivot
- in_perio=6** global model with a F-point pivot
example: ORCA05

See Also

For more information see [NEMO periodicity](#) and Model Boundary Condition paragraph in the [NEMO documentation](#)

12.2.4.4 namtgt

the target/fine grid sub-namelist parameters are :

- **cn_coord1** []
path to coordinate file
- **in_perio1** []
NEMO periodicity index (see above)

Note

if the fine/target coordinates file (*cn_coord1*) was created by SIREN, you do not need to fill this parameter. SIREN will read it on the global attributes of the coordinates file.

- **In_fillclosed** [.TRUE.]
logical to fill closed sea or not

12.2.4.5 namvar

the variable sub-namelist parameters are :

- **cn_varfile** []
list of variable, and associated file

Warning

variable name must be **Bathymetry** here.

cn_varfile is the path and filename of the file where find variable.

Note

cn_varfile could be a matrix of value, if you want to handwrite variable value.
the variable array of value is split into equal subdomain.
each subdomain is filled with the corresponding value of the matrix.
separators used to defined matrix are:

- ';' for line
- '/' for row Example:

$$3,2,3/1,4,5 \Rightarrow \begin{pmatrix} 3 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}$$

Examples:

- 'Bathymetry:gridT.nc'
- 'Bathymetry:5000,5000,5000/5000,3000,5000/5000,5000,5000'

Note

Optionnaly, NEMO periodicity could be added following the filename. the periodicity must be separated by ';'.

Example:

- 'Bathymetry:gridT.nc ; perio=4'

- **cn_varinfo** []

list of variable and extra information about request(s) to be used

each elements of *cn_varinfo* is a string character (separated by ',').

it is composed of the variable name follow by ':', then request(s) to be used on this variable.

request could be:

- int = interpolation method
 - ext = extrapolation method
 - flt = filter method
 - min = minimum value
 - max = maximum value
 - unt = new units
 - unf = unit scale factor (linked to new units)
- requests must be separated by ';'.
order of requests does not matter.

informations about available method could be find in [interp](#), [extrap](#) and [filter](#) modules.

Example:

- 'Bathymetry: flt=2*hamming(2,3); min=0'

Note

If you do not specify a method which is required, default one is apply.

- **In_rand** [.False.]

logical to add random value to Bathymetry

Only for handmade Bathymetry. A random value (+/- 0.1% of the maximum depth) will will be added to avoid flat Bathymetry (which may cause issue).

12.2.4.6 namnst

the nesting sub-namelist parameters are :

- **in_rhoi** [1]
refinement factor in i-direction
- **in_rhoj** [1]
refinement factor in j-direction

Note

coarse grid indices will be deduced from fine grid coordinate file.

12.2.4.7 namout

the output sub-namelist parameter is :

- **cn_fileout** [*bathy_fine.nc*]
output bathymetry filename

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header for user
- Bug fix, compute offset depending of grid point

June, 2015

- extrapolate all land points.
- allow to change unit.

September, 2015

- manage useless (dummy) variable, attributes, and dimension

January, 2016

- add create_bathy_check_depth as in create_boundary
- add create_bathy_check_time as in create_boundary

February, 2016

- do not closed sea for east-west cyclic domain

October, 2016

- dimension to be used select from configuration file

July, 2017

- add random value to avoid flat bathymetry

January, 2019

- add option to add random value to a flat Bathymetry
- create and clean file structure to avoid memory leaks
- check dimension of matrix for 'handmade' bathymetry
- add url path to global attributes of output [file\(s\)](#)

February, 2019

- rename sub namelist namcrs to namsrc
- rename sub namelist namfin to namtgt

August, 2019

- use periodicity read from namelist, and store in multi structure

October, 2019

- add help and version optional arguments

Todo

- check tl_multi is not empty

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

12.2.5 Function/Subroutine Documentation

12.2.5.1 subroutine `create_bathy::create_bathy_check_depth` (`type(tmpp)`, `intent(in) td_mpp`, `type(tvar)`, `intent(inout) td_depth`)

This subroutine get depth variable value in an open mpp structure and check if agree with already input depth variable.

Parameters

in	<i>td_mpp</i>
----	---------------

Author

J.Paul

Date

January, 2016 - Initial Version

Parameters

in	<i>td_mpp</i>	mpp structure
in, out	<i>td_depth</i>	depth variable structure

12.2.5.2 subroutine `create_bathy::create_bathy_check_time` (type(tmpp), intent(in) *td_mpp*, type(tvar), intent(inout) *td_time*)

This subroutine get date and time in an open mpp structure and check if agree with date and time already read.

Parameters

in	<i>td_mpp</i>
----	---------------

Author

J.Paul

Date

January, 2016 - Initial Version

Parameters

in	<i>td_mpp</i>	mpp structure
in, out	<i>td_time</i>	time variable structure

12.2.5.3 type(tvar) function `create_bathy::create_bathy_extract` (type(tvar), intent(in) *td_var*, type(tmpp), intent(in) *td_mpp*, type(tmpp), intent(in) *td_coord*)

This function extract variable from file over coordinate domain and return variable structure.

Parameters

in	<i>td_var</i>
----	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_var</i>	variable structure
in	<i>td_mpp</i>	mpp file structure
in	<i>td_coord</i>	coordinate file structure

Returns

variable structure

12.2.5.4 `type(tvar) function create_bathy::create_bathy_get_var (type(tvar), intent(in) td_var, type(tmpp), intent(in) td_mpp, integer(i4), intent(in) id_imin, integer(i4), intent(in) id_jmin, integer(i4), intent(in) id_imax, integer(i4), intent(in) id_jmax, integer(i4), dimension(:,:), intent(in) id_offset, integer(i4), dimension(:,:), intent(in) id_rho)`

This function get coarse grid variable, interpolate variable, and return variable structure over fine grid.

Parameters

in	<i>td_var</i>
----	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_var</i>	variable structure
in	<i>td_mpp</i>	mpp file structure
in	<i>id_imin</i>	i-direction lower left corner indice
in	<i>id_imax</i>	i-direction upper right corner indice
in	<i>id_jmin</i>	j-direction lower left corner indice
in	<i>id_jmax</i>	j-direction upper right corner indice
in	<i>id_offset</i>	offset between fine grid and coarse grid
in	<i>id_rho</i>	array of refinement factor

Returns

variable structure

12.2.5.5 `subroutine create_bathy::create_bathy_interp (type(tvar), intent(inout) td_var, integer(i4), dimension(:,:), intent(in) id_rho, integer(i4), dimension(:,:), intent(in) id_offset, integer(i4), intent(in), optional id_iext, integer(i4), intent(in), optional id_jext)`

This subroutine interpolate variable.

Parameters

in, out	<i>td_var</i>
---------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>id_rho</i>	array of refinement factor
in	<i>id_offset</i>	array of offset between fine and coarse grid
in	<i>id_iext</i>	i-direction size of extra bands (default=im_minext)
in	<i>id_jext</i>	j-direction size of extra bands (default=im_minext)

12.2.5.6 `type(tvar) function create_bathy::create_bathy_matrix (type(tvar), intent(in) td_var, type(tmpp), intent(in) td_coord, logical, intent(in) ld_rand)`

This function create variable, filled with matrix value.

Parameters

in	<i>td_var</i>
----	---------------

A variable is create with the same name that the input variable, and with dimension of the coordinate file.

Then the variable array of value is split into equal subdomain. Each subdomain is filled with the corresponding value of the matrix.

Optionally, you could add a random value of 0.1% of maximum depth to each points of the bathymetry

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_var</i>	variable structure
in	<i>td_coord</i>	coordinate file structure
in	<i>ld_rand</i>	add random value to bathymetry

Returns

variable structure

12.3 src/create_boundary.F90 File Reference

This program creates boundary files.

Functions/Subroutines

- program **create_boundary**
- subroutine **create_boundary__mono** (cd_namelist)
This subroutine create boundary files.
- type(tdom) function, dimension(ip_npoint) **create_boundary_get_dom** (td_bathy1, td_bdy, id_seg)
This subroutine compute boundary domain for each grid point (T,U,V,F)
- subroutine **create_boundary_get_coord** (td_coord1, td_dom1, cd_point, td_lon1, td_lat1)
This subroutine get coordinates over boundary domain.
- subroutine **create_boundary_interp** (td_var, id_rho, id_offset, id_iext, id_jext)
This subroutine interpolate variable on boundary.

- type(tvar) function `create_boundary_matrix` (td_var, td_dom, id_nlevel)
This function create variable, filled with matrix value.
- subroutine `create_boundary_use_mask` (td_var, td_mask)
This subroutine use mask to filled land point with `_FillValue`.
- type(tvar) function, dimension(ip_npoint) `create_boundary_get_level` (td_level, td_dom)
This function extract level over domain on each grid point, and return array of variable structure.
- subroutine `create_boundary_check_depth` (td_var, td_mpp, id_nlevel, td_depth)
This subroutine check if variable need depth dimension, get depth variable value in an open mpp structure and check if agree with already input depth variable.
- subroutine `create_boundary_check_time` (td_var, td_mpp, td_time)
This subroutine check if variable need time dimension, get date and time in an open mpp structure and check if agree with date and time already read.

12.3.1 Detailed Description

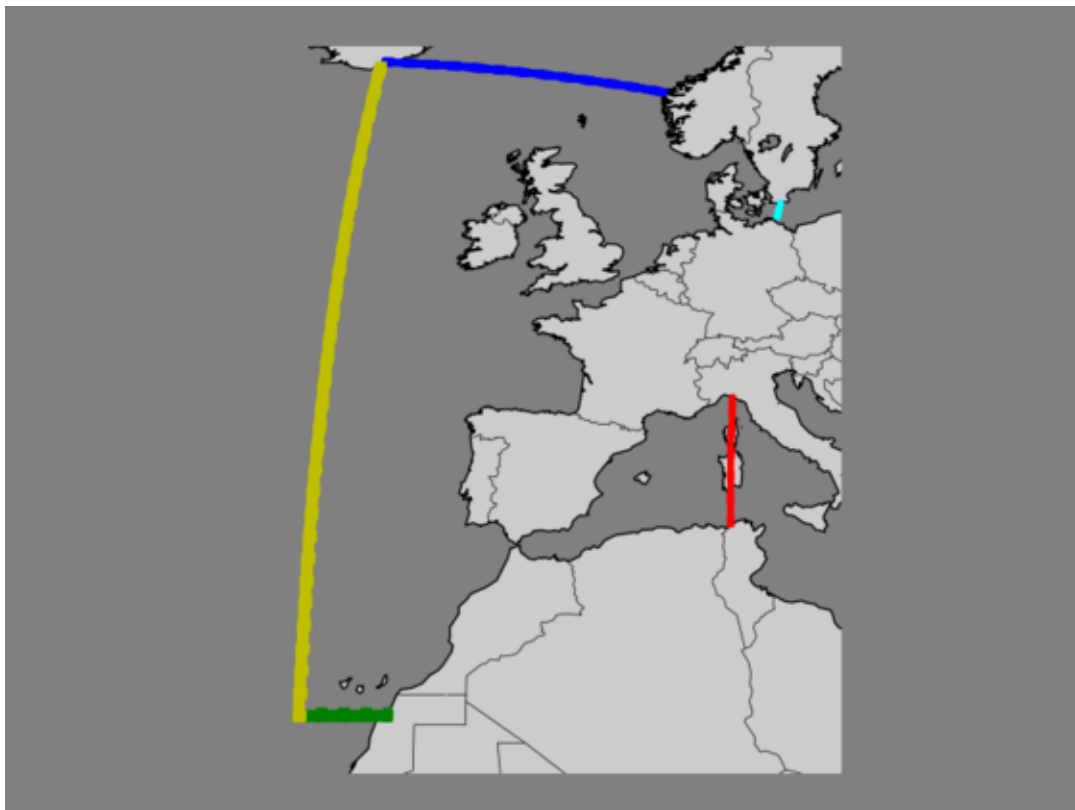
This program creates boundary files.

12.3.2 method

Variables are read from coarse grid standard output, extracted or interpolated on fine grid. Variables could also be manually written.

Note

method could be different for each variable.



12.3.3 how to

USAGE: `create_boundary create_bounary.nam [-v] [-h]`

- positional arguments:

- `create_boundary.nam`
namelist of `create_boundary`

Note

a template of the namelist could be created running (in templates directory):

```
python create_templates.py create_boundary
```

- optional arguments:

- `-h, --help`
show this help message (and exit)
- `-v, --version`
show Siren's version (and exit)

Note

compiled with *key_mpp_mpi*, could be run on multi processor :

```
USAGE: create_boundary create_boundary.nam create_bounary2.nam ... [-v] [-h]
```

12.3.4 `create_boundary.nam`

`create_boundary.nam` contains 9 namelists:

- **namlog** to set logger parameters
- **namcfg** to set configuration file parameters
- **namsrc** to set source/coarse grid parameters
- **namtgt** to set target/fine grid parameters
- **namvar** to set variable parameters
- **namnst** to set sub domain and nesting paramters
- **nambdy** to set boundary parameters
- **namzgr** to set vertical grid parameters
- **namout** to set output parameters

here after, each sub-namelist parameters is detailed.

Note

default values are specified between brackets

12.3.4.1 `namlog`

the logger sub-namelist parameters are :

- **cn_logfile** [*create_boundary.log*]
logger filename
- **cn_verbosity** [*warning*]
verbosity level, choose between :
 - trace
 - debug

- info
 - warning
 - error
 - fatal
 - none
- **in_maxerror** [5]
maximum number of error allowed

12.3.4.2 namcfg

the configuration sub-namelist parameters are :

- **cn_varcfg** [./cfg/variable.cfg]
path to the variable configuration file.
the variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.
- **cn_dimcfg** [./cfg/dimension.cfg]
path to the dimension configuration file.
the dimension configuration file defines dimensions allowed.
- **cn_dumcfg** [./cfg/dummy.cfg]
path to the useless (dummy) configuration file.
the dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

12.3.4.3 namcrs

the coarse grid sub-namelist parameters are :

- **cn_coord0** []
path to the coordinate file
- **in_perio0** []
NEMO periodicity index
the NEMO periodicity could be choose between 0 to 6:

in_perio=0 standard regional model
in_perio=1 east-west cyclic model
in_perio=2 model with symmetric boundary condition across the equator
in_perio=3 regional model with North fold boundary and T-point pivot
in_perio=4 global model with a T-point pivot.
example: ORCA2, ORCA025, ORCA12
in_perio=5 regional model with North fold boundary and F-point pivot
in_perio=6 global model with a F-point pivot
example: ORCA05

See Also

For more information see [NEMO periodicity](#) and Model Boundary Condition paragraph in the [NEMO documentation](#)

12.3.4.4 namfin

the fine grid sub-namelist parameters are :

- **cn_coord1** []
path to coordinate file
- **cn_bathy1** []
path to bathymetry file

Warning

- **in_perio1** []
NEMO periodicity index (see above)

Note

if the fine/target coordinates file (cn_coord1) was created by SIREN, you do not need to fill this parameter. SIREN will read it on the global attributes of the coordinates file.

12.3.4.5 namzgr

the vertical grid sub-namelist parameters are :

- **dn_pp_to_be_computed** [0]
coefficient to compute vertical grid
- **dn_ppsur** [-3958.951371276829]
coefficient to compute vertical grid
- **dn_ppa0** [103.953009600000]
coefficient to compute vertical grid
- **dn_ppa1** [2.415951269000]
coefficient to compute vertical grid
- **dn_ppa2** [100.760928500000]
double tanh function parameter
- **dn_ppkth** [15.351013700000]
coefficient to compute vertical grid
- **dn_ppkth2** [48.029893720000]
double tanh function parameter
- **dn_ppacr** [7.000000000000]
coefficient to compute vertical grid
- **dn_ppacr2** [13.000000000000]
double tanh function parameter
- **dn_ppdzmin** [6.]
minimum vertical spacing
- **dn_pphmax** [5750.]
maximum depth
- **in_nlevel** [75]
number of vertical level

Note

If *dn_ppa1*, *dn_ppa0* and *dn_ppsur* are undefined, NEMO will compute them from *dn_ppdzmin*, *dn_pphmax*, *dn_ppkth*, *dn_ppacr*

12.3.4.6 namzps

the partial step sub-namelist parameters are :

- **dn_e3zps_min** [25.]
minimum thickness of partial step level (meters)
- **dn_e3zps_rat** [0.2]
minimum thickness ratio of partial step level

12.3.4.7 namvar

the variable sub-namelist parameters are :

- **cn_varfile** []
list of variable, and associated file
cn_varfile is the path and filename of the file where find variable.

Note

cn_varfile could be a matrix of value, if you want to handwrite variable value.
the variable array of value is split into equal subdomain.
each subdomain is filled with the corresponding value of the matrix.
separators used to defined matrix are:

- ';' for line
- '/' for row
- '\' for level

Example:

$$3,2,3/1,4,5 \Rightarrow \begin{pmatrix} 3 & 2 & 3 & 1 \\ 4 & 5 & & \end{pmatrix}$$

Warning

the same matrix is used for all boundaries.

Examples:

- 'votemper:gridT.nc', 'vozocrtx:gridU.nc'
- 'votemper:10\25', 'vozocrtx:gridU.nc'

Note

Optionnaly, NEMO periodicity could be added following the filename. the periodicity must be separated by ','

Example:

- 'votemper:gridT.nc ; perio=4'

- **cn_varinfo** []

list of variable and extra information about request(s) to be used
each elements of *cn_varinfo* is a string character (separated by ',').

it is composed of the variable name follow by ':', then request(s) to be used on this variable.

request could be:

- int = interpolation method
 - ext = extrapolation method
 - flt = filter method
 - min = minimum value
 - max = maximum value
 - unt = new units
 - unf = unit scale factor (linked to new units)
- requests must be separated by ';'.
order of requests does not matter.

informations about available method could be find in [interp](#), [extrap](#) and [filter](#) modules.

Example:

- 'votemper: int=linear; flt=hann; ext=dist_weight', 'vosaline: int=cubic'

Note

If you do not specify a method which is required, default one is apply.

12.3.4.8 namnst

the nesting sub-namelist parameters are :

- **in_rhoi** [*1*]
refinement factor in i-direction
- **in_rhoj** [*1*]
refinement factor in j-direction

Note

coarse grid indices will be deduced from fine grid coordinate file.

12.3.4.9 nambdy

the boundary sub-namelist parameters are :

- **In_north** [.TRUE.]
logical to use north boundary or not
- **In_south** [.TRUE.]
logical to use south boundary or not
- **In_east** [.TRUE.]
logical to use east boundary or not
- **In_west** [.TRUE.]
logical to use west boundary or not
- **cn_north** []
north boundary indices on fine grid
- **cn_south** []
south boundary indices on fine grid

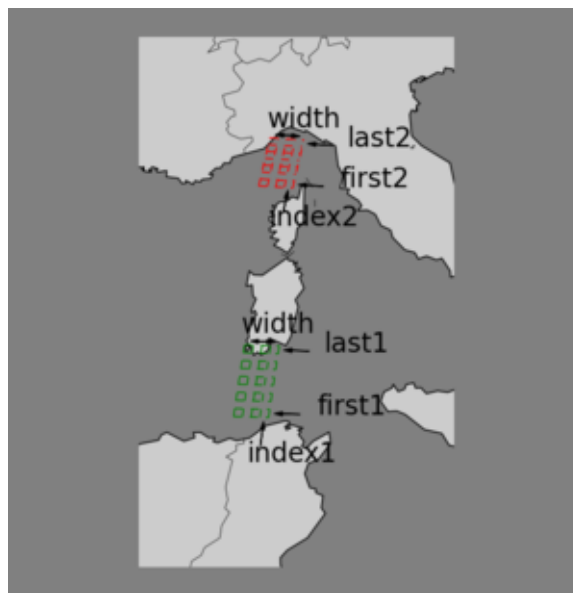
- **cn_east** []
east boundary indices on fine grid
- **cn_west** []
west boundary indices on fine grid
cn_north is a string character defining boundary segmentation.
segments are separated by '|'.
each segments of the boundary is composed of:
 - indice of velocity (orthogonal to boundary .ie. for north boundary, J-indice).
 - indice of segment start (I-indice for north boundary)
 - indice of segment end (I-indice for north boundary)
indices must be separated by ':' .
 - optionally, boundary size could be added between '(' and ')' in the first segment defined.

Note

boundary size is the same for all segments of one boundary.

Examples:

- `cn_north='index1,first1:last1(width)'`
- `cn_north='index1(width),first1:last1|index2,first2:last2'`



- **In_oneseg** [.TRUE.]
logical to use only one segment for each boundary or not

Note

the number of point(s) with coarse value save at boundaries is defined with the *weight* variable (see merge_bathy)

12.3.4.10 namout

the output sub-namelist parameter is :

- **cn_fileout** [*boundary.nc*]
output bathymetry filename

Note

cardinal point and segment number will be automatically added

- **In_extrap** [.FALSE.]
extrapolate on land point
- **dn_dayofs** [0]
date offset in day (change only output file name)

Examples:

- `cn_fileout='boundary.nc'`
if `time_counter` (16/07/2015 00h) is read on input file (see `varfile`), west boundary will be named `boundary_west_y2015m07d16`
- `dn_dayofs=-2.`
if you use day offset you get `boundary_west_y2015m07d14`

12.3.4.11 How to fill Lateral Boundary Condition in NEMO namelist

To use boundary condition within NEMO, you need to fill the NEMO namelist.

As this is a little bit messy for lateral boundary condition, here after is an explanation of how to do it.

This will be done in 3 steps.

12.3.4.11.1 nambdy

The *nambdy* NEMO sub-namelist defines open boundaries.

Here we indicate the number of open boundary (**nb_bdy**).

Note

we have to fill most of the parameters with as many elements as there are open boundaries

Regarding the width of the relaxation zone **nn_rimwidth**, this information is available as a global attribute (**bdy_width**) in the metadata of boundary files created with SIREN

```
ncdump -h boundary_east.nc
```

Warning

The order of the boundaries must stay unchanged, in parameters list as well as in the next sub-namelsits

Example:

here is an example for a domain with two boundaries East and North

```
!-----
&nambdy      ! unstructured open boundaries                                ("key_bdy")
!-----
  nb_bdy      = 2                                ! number of open boundary sets
  ln_coords_file = .false.,.false.              ! =T : read bdy coordinates from file
  cn_coords_file = ','                          ! bdy coordinates file
  ln_mask_file  = .false.                        ! =T : read mask from file
  cn_mask_file  = ''                            ! name of mask file (if ln_mask_file=.TRUE.)
  cn_dyn2d      = 'flather','flather'          !
  nn_dyn2d_dta  = 1,1                          ! = 0, bdy data are equal to the initial state
                                          ! = 1, bdy data are read in 'bdydata .nc' files
                                          ! = 2, use tidal harmonic forcing data from files
                                          ! = 3, use external data AND tidal harmonic forcing
  cn_dyn3d      = 'specified','specified'      !
  nn_dyn3d_dta  = 1,1                          ! = 0, bdy data are equal to the initial state
                                          ! = 1, bdy data are read in 'bdydata .nc' files
  cn_tra        = 'specified','specified'      !
```

```

nn_tra_dta      = 1,1           ! = 0, bdy data are equal to the initial state
                                   ! = 1, bdy data are read in 'bdydata .nc' files
                                   !
ln_tra_dmp      =.true.,.true.  ! open boudaries conditions for tracers
ln_dyn3d_dmp    =.true.,.true.  ! open boundary condition for baroclinic velocities
rn_time_dmp     = 1.,1.         ! Damping time scale in days
rn_time_dmp_out = 1.,1.         ! Outflow damping time scale
nn_rimwidth    = 10,10         ! width of the relaxation zone
ln_vol         = .false.        ! total volume correction (see nn_volctl parameter)
nn_volctl      = 1             ! = 0, the total water flux across open boundaries is zero
/

```

12.3.4.11.2 nambdy_index

The *nambdy_index* NEMO sub-namelist describes the boundaries we will use.

Warning

We have to add as many as sub namelist *nambdy_index* than open boundaries (*nb_bdy*), and keep them in the same order as above

Here we indicate if the open boundary is North, South, East, or West (**ctypebdy**).

We also indicate indice of segment start and end (respectively **nbdybeg** and **nbdyend**) as well as indice of velocity row or column (**nbdyind**).

Those informations are available as global attributes (respectively **bdy_deb**, **bdy_end**, **bdy_ind**) in the metadata of our boundary files created with SIREN.

Example:

here is an example for a domain with two boundaries East and North

```

!-----
&nambdy_index ! structured open boundaries definition ("key_bdy")
!-----
  ctypebdy = 'E'           ! Open boundary type (W,E,S or N)
  nbdyind  = 407           ! indice of velocity row or column
                                   ! if ==-1, set obc at the domain boundary
                                   !           , discard start and end indices
  nbdybeg  = 32           ! indice of segment start
  nbdyend  = 300          ! indice of segment end
/
!-----
&nambdy_index ! structured open boundaries definition ("key_bdy")
!-----
  ctypebdy = 'N'           ! Open boundary type (W,E,S or N)
  nbdyind  = 299           ! indice of velocity row or column
                                   ! if ==-1, set obc at the domain boundary
                                   !           , discard start and end indices
  nbdybeg  = 200          ! indice of segment start
  nbdyend  = 408          ! indice of segment end
/

```

12.3.4.11.3 nambdy_dta

The *nambdy_dta* NEMO sub-namelists describes the boundary data and files to be used.

Warning

We have to add as many as sub namelist *nambdy_dta* than open boundaries (*nb_bdy*), and keep them in the same order as above

Example:

here is an example for a domain with two boundaries East and North

```

!-----
&nambdy_dta   ! open boundaries - external data ("key_bdy")
!-----
!           ! file name ! freq (hours) ! variable ! time interp. ! clim ! 'yearly' / ! weights.
..

```

```

!           !           ! (if < 0 months) !   name   !   (logical) ! (T/F) ! 'monthly' ! filename
bn_ssh = 'boundary_east' , -12 , 'sossheig' , .false. , .true. , 'yearly' , '' , '' , ''
bn_u2d = 'boundary_east' , -12 , 'vobtcrtx' , .false. , .true. , 'yearly' , '' , '' , ''
bn_v2d = 'boundary_east' , -12 , 'vobtcrtx' , .false. , .true. , 'yearly' , '' , '' , ''
bn_u3d = 'boundary_east' , -12 , 'vozocrtx' , .false. , .true. , 'yearly' , '' , '' , ''
bn_v3d = 'boundary_east' , -12 , 'vomecrty' , .false. , .true. , 'yearly' , '' , '' , ''
bn_tem = 'boundary_east' , -12 , 'votemper' , .false. , .true. , 'yearly' , '' , '' , ''
bn_sal = 'boundary_east' , -12 , 'vosaline' , .false. , .true. , 'yearly' , '' , '' , ''
cn_dir = './'
ln_full_vel = .true.
/
!-----
&nambdy_dta      ! open boundaries - external data          ("key_bdy")
!-----
!           ! file name !   freq (hours)   ! variable ! time interp. ! clim ! 'yearly' / ! weights.
..
!           !           ! (if < 0 months) !   name   !   (logical) ! (T/F) ! 'monthly' ! filename
bn_ssh = 'boundary_north' , -12 , 'sossheig' , .false. , .true. , 'yearly' , '' , '' , ''
bn_u2d = 'boundary_north' , -12 , 'vobtcrtx' , .false. , .true. , 'yearly' , '' , '' , ''
bn_v2d = 'boundary_north' , -12 , 'vobtcrtx' , .false. , .true. , 'yearly' , '' , '' , ''
bn_u3d = 'boundary_north' , -12 , 'vozocrtx' , .false. , .true. , 'yearly' , '' , '' , ''
bn_v3d = 'boundary_north' , -12 , 'vomecrty' , .false. , .true. , 'yearly' , '' , '' , ''
bn_tem = 'boundary_north' , -12 , 'votemper' , .false. , .true. , 'yearly' , '' , '' , ''
bn_sal = 'boundary_north' , -12 , 'vosaline' , .false. , .true. , 'yearly' , '' , '' , ''
cn_dir = './'
ln_full_vel = .true.
/

```

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header for user
- take into account grid point to compute boundaries
- reorder output dimension for north and south boundaries

June, 2015

- extrapolate all land points, and add ln_extrap in namelist.
- allow to change unit.

July, 2015

- add namelist parameter to shift date of output file name.

September, 2015

- manage useless (dummy) variable, attributes, and dimension
- allow to run on multi processors with key_mpp_mpi

January, 2016

- same process use for variable extracted or interpolated from input file.

October, 2016

- dimension to be used select from configuration file

January, 2019

- add url path to global attributes of output [file\(s\)](#)
- create and clean file structure to avoid memory leaks
- explain how to fill Lateral Boundary Condition in NEMO namelist

February, 2019

- rename sub namelist namcrs to namsrc
- rename sub namelist namfin to namtgt

August, 2019

- use periodicity read from namelist, and store in multi structure

October, 2019

- add help and version optional arguments

Todo • rewitre using meshmask instead of bathymetry and coordinates files.

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

12.3.5 Function/Subroutine Documentation

12.3.5.1 subroutine create_boundary::create_boundary__mono (character(len=lc), intent(in) *cd_namelist*)

This subroutine create boundary files.

Parameters

<i>in</i>	<i>cd_namelist</i>
-----------	--------------------

Author

J.Paul

Date

January, 2016 - Initial Version

Parameters

<i>in</i>	<i>cd_namelist</i>	namelist file
-----------	--------------------	---------------

12.3.5.2 subroutine create_boundary::create_boundary_check_depth (type(tvar), intent(in) *td_var*, type(tmpp), intent(in) *td_mpp*, integer(i4), intent(in) *id_nlevel*, type(tvar), intent(inout) *td_depth*)

This subroutine check if variable need depth dimension, get depth variable value in an open mpp structure and check if agree with already input depth variable.

Parameters

<i>in</i>	<i>td_var</i>
-----------	---------------

Author

J.Paul

Date

November, 2014 - Initial Version

January, 2016

- check if variable need/use depth dimension

Parameters

in	<i>td_var</i>	variable structure
in	<i>td_mpp</i>	mpp structure
in	<i>id_nlevel</i>	mpp structure
in, out	<i>td_depth</i>	depth variable structure

12.3.5.3 subroutine `create_boundary::create_boundary_check_time` (type(tvar), intent(in) *td_var*, type(tmpp), intent(in) *td_mpp*, type(tvar), intent(inout) *td_time*)

This subroutine check if variable need time dimension, get date and time in an open mpp structure and check if agree with date and time already read.

Parameters

in	<i>td_var</i>	
----	---------------	--

Author

J.Paul

Date

November, 2014 - Initial Version

January, 2016

- check if variable need/use time dimension

Parameters

in	<i>td_var</i>	variable structure
in	<i>td_mpp</i>	mpp structure
in, out	<i>td_time</i>	time variable structure

12.3.5.4 subroutine `create_boundary::create_boundary_get_coord` (type(tmpp), intent(in) *td_coord1*, type(tdom), intent(in) *td_dom1*, character(len=*) , intent(in) *cd_point*, type(tvar), intent(out) *td_lon1*, type(tvar), intent(out) *td_lat1*)

This subroutine get coordinates over boundary domain.

Parameters

in	<i>td_coord1</i>	
----	------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- take into account grid point

Parameters

in	<i>td_coord1</i>	coordinates file structure
in	<i>td_dom1</i>	boundary domain structure
in	<i>cd_point</i>	grid point
out	<i>td_lon1</i>	longitude variable structure
out	<i>td_lat1</i>	latitude variable structure

12.3.5.5 `type(tdom)` function, `dimension(ip_npoint)` `create_boundary::create_boundary_get_dom (type(tmpp), intent(in) td_bathy1, type(tbdy), intent(in) td_bdy, integer(i4), intent(in) id_seg)`

This subroutine compute boundary domain for each grid point (T,U,V,F)

Parameters

in	<i>td_bathy1</i>	
----	------------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- take into account grid point to compute boundary indices

Parameters

in	<i>td_bathy1</i>	file structure
in	<i>td_bdy</i>	boundary structure
in	<i>id_seg</i>	segment indice

Returns

array of domain structure

12.3.5.6 `type(tvar)` function, `dimension(ip_npoint)` `create_boundary::create_boundary_get_level (type(tvar), dimension(:), intent(in) td_level, type(tdom), dimension(:), intent(in) td_dom)`

This function extract level over domain on each grid point, and return array of variable structure.

Parameters

in	<i>td_level</i>	
----	-----------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_level</i>	array of level variable structure
in	<i>td_dom</i>	array of domain structure

Returns

array of variable structure

12.3.5.7 subroutine `create_boundary::create_boundary_interp` (`type(tvar)`, `intent(inout) td_var`, `integer(i4)`, `dimension(:)`, `intent(in) id_rho`, `integer(i4)`, `dimension(:,:)`, `intent(in) id_offset`, `integer(i4)`, `intent(in)`, optional `id_iext`, `integer(i4)`, `intent(in)`, optional `id_jext`)

This subroutine interpolate variable on boundary.

Parameters

in, out	<i>td_var</i>	
---------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>id_rho</i>	array of refinement factor
in	<i>id_offset</i>	array of offset between fine and coarse grid
in	<i>id_iext</i>	i-direction size of extra bands (default=im_minext)
in	<i>id_jext</i>	j-direction size of extra bands (default=im_minext)

12.3.5.8 `type(tvar)` function `create_boundary::create_boundary_matrix` (`type(tvar)`, `intent(in) td_var`, `type(tdom)`, `intent(in) td_dom`, `integer(i4)`, `intent(in) id_nlevel`)

This function create variable, filled with matrix value.

Parameters

in	<i>td_var</i>	
----	---------------	--

A variable is create with the same name that the input variable, and with dimension of the coordinate file. Then the variable array of value is split into equal subdomain. Each subdomain is fill with the associated value of the matrix.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in	<i>td_var</i>	variable structure
in	<i>td_dom</i>	domain structure
in	<i>id_nlevel</i>	number of levels

Returns

variable structure

12.3.5.9 subroutine create_boundary::create_boundary_use_mask (type(tvar), intent(inout) *td_var*, type(tvar), intent(in) *td_mask*)

This subroutine use mask to filled land point with _FillValue.

Parameters

in, out	<i>td_var</i>
---------	---------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>td_mask</i>	mask variable structure

12.4 src/create_coord.f90 File Reference

this program creates fine grid coordinate file.

Functions/Subroutines

- program **create_coord**
- integer(i4) function,
dimension(2, 2, ip_npoint) [create_coord_get_offset](#) (id_rho)
This function compute offset over Arakawa grid points, given refinement factor.
- subroutine [create_coord_interp](#) (td_var, id_rho, id_offset, id_iext, id_jext)
This subroutine interpolate variable, given refinement factor.

12.4.1 Detailed Description

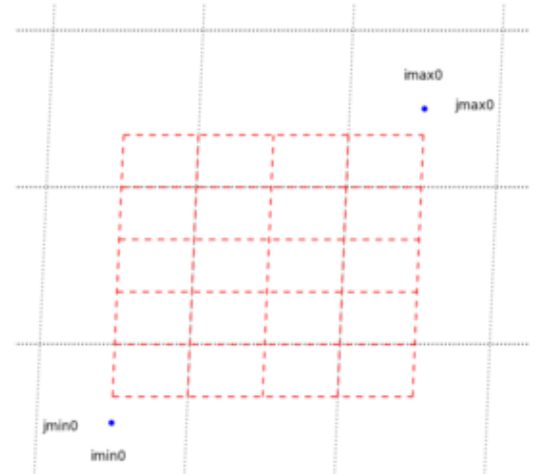
this program creates fine grid coordinate file.

12.4.2 method

variables from the input coordinates coarse/source grid file, are extracted and interpolated to create a fine/target grid coordinates file.

Note

interpolation method could be different for each variable.



12.4.3 how to

USAGE: create_coord create_coord.nam [-v] [-h]

- positional arguments:

- create_coord.nam
namelist of create_coord

Note

a template of the namelist could be created running (in templates directory):

```
python create_templates.py create_coord
```

- optional arguments:

- -h, –help
show this help message (and exit)
- -v, –version
show Siren's version (and exit)

12.4.4 create_coord.nam

create_coord.nam contains 6 sub-namelists:

- **namlog** to set logger parameters
- **namcfg** to set configuration file parameters
- **namsrc** to set source/coarse grid parameters
- **namvar** to set variable parameters

- **namnst** to set sub domain and nesting paramters
- **namout** to set output parameters

here after, each sub-namelist parameters is detailed.

Note

default values are specified between brackets

12.4.4.1 namlog

the logger sub-namelist parameters are :

- **cn_logfile** [*create_coord.log*]
logger filename
- **cn_verbosity** [*warning*]
verbosity level, choose between :
 - trace
 - debug
 - info
 - warning
 - error
 - fatal
 - none
- **in_maxerror** [*5*]
maximum number of error allowed

12.4.4.2 namcfg

the configuration sub-namelist parameters are :

- **cn_varcfg** [*./cfg/variable.cfg*]
path to the variable configuration file.
the variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.
- **cn_dimcfg** [*./cfg/dimension.cfg*]
path to the dimension configuration file.
the dimension configuration file defines dimensions allowed.
- **cn_dumcfg** [*./cfg/dummy.cfg*]
path to the useless (dummy) configuration file.
the dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

12.4.4.3 namsrc

the source/coarse grid sub-namelist parameters are :

- **cn_coord0** []
path to the coordinate file
- **in_perio0** []
NEMO periodicity index
the NEMO periodicity could be choose between 0 to 6:
 - in_perio=0** standard regional model
 - in_perio=1** east-west cyclic model
 - in_perio=2** model with symmetric boundary condition across the equator
 - in_perio=3** regional model with North fold boundary and T-point pivot
 - in_perio=4** global model with a T-point pivot.
example: ORCA2, ORCA025, ORCA12
 - in_perio=5** regional model with North fold boundary and F-point pivot
 - in_perio=6** global model with a F-point pivot
example: ORCA05

See Also

For more information see [NEMO periodicity](#) and Model Boundary Condition paragraph in the [NEMO documentation](#)

12.4.4.4 namvar

the variable sub-namelist parameters are :

- **cn_varinfo** []
list of variable and extra information about request(s) to be used
each elements of *cn_varinfo* is a string character (separated by ',').
it is composed of the variable name follow by ':', then request(s) to be used on this variable.
request could be:
 - int = interpolation method
 - ext = extrapolation method
requests must be separated by ','.
order of requests does not matter.

informations about available method could be find in [interp](#), [extrap](#) and [filter](#) modules.

Example:

- 'glamt: int=linear; ext=dist_weight', 'e1t: int=cubic/rho'

Note

If you do not specify a method which is required, default one is apply.

12.4.4.5 namnst

the nesting sub-namelist parameters are :

- **in_imin0** []
i-direction lower left point indice of coarse grid subdomain to be used
- **in_imax0** []
i-direction upper right point indice of coarse grid subdomain to be used
- **in_jmin0** []
j-direction lower left point indice of coarse grid subdomain to be used
- **in_jmax0** []
j-direction upper right point indice of coarse grid subdomain to be used
or
- **rn_lonmin0** []
lower left longitude of coarse grid subdomain to be used
- **rn_lonmax0** []
upper right longitude of coarse grid subdomain to be used
- **rn_latmin0** []
lower left latitude of coarse grid subdomain to be used
- **rn_latmax0** []
upper right latitude of coarse grid subdomain to be used

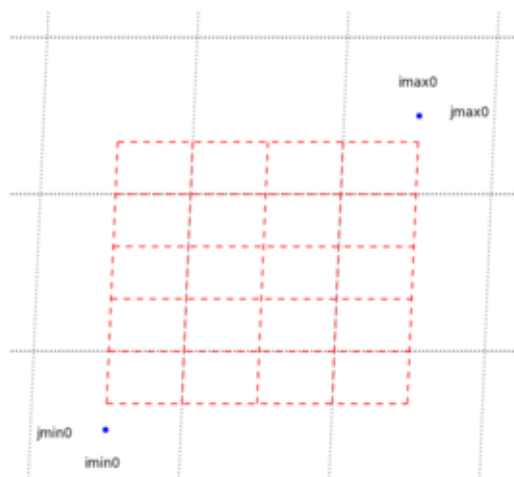
Note

you could define sub domain with

- coarse/source grid indices
or
- coordinates.

if coordinates are defined ($-180 < \text{lon} < 360$ and $-90 < \text{lat} < 90$), SIREN does not take into account indices.

- **in_rhoi** [1]
refinement factor in i-direction
- **in_rhoj** [1]
refinement factor in j-direction



12.4.4.6 namout

the output sub-namelist parameter is :

- **cn_fileout** [*coord_fine.nc*]
output bathymetry filename

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header for user
- compute offset considering grid point
- add global attributes in output file

September, 2015

- manage useless (dummy) variable, attributes, and dimension

September, 2016

- allow to use coordinate to define subdomain

October, 2016

- dimension to be used select from configuration file

January, 2019

- add url path to global attributes of output [file\(s\)](#)

February, 2019

- rename sub namelist namcrs to namsrc
- create and clean file structure to avoid memory leaks

October, 2019

- add help and version optional arguments

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

12.4.5 Function/Subroutine Documentation

12.4.5.1 `integer(i4) function, dimension(2,2,ip_npoint) create_coord::create_coord_get_offset (integer(i4), dimension(:), intent(in) id_rho)`

This function compute offset over Arakawa grid points, given refinement factor.

Parameters

<code>in</code>	<code>id_rho</code>
-----------------	---------------------

Author

J.Paul

Date

August, 2014 - Initial Version

Parameters

<code>in</code>	<code>id_rho</code>	array of refinement factor
-----------------	---------------------	----------------------------

Returns

array of offset

12.4.5.2 subroutine `create_coord::create_coord_interp` (`type(tvar)`, `intent(inout) td_var`, `integer(i4)`, `dimension(:)`, `intent(in) id_rho`, `integer(i4)`, `dimension(:,:)`, `intent(in) id_offset`, `integer(i4)`, `intent(in)`, optional `id_iext`, `integer(i4)`, `intent(in)`, optional `id_jext`)

This subroutine interpolate variable, given refinement factor.

Parameters

<code>in, out</code>	<code>td_var</code>	
----------------------	---------------------	--

Optionaly, you could specify number of points to be extrapolated in i- and j-direction.

variable mask is first computed (using `_FillValue`) and interpolated.

variable is then extrapolated, and interpolated.

Finally interpolated mask is applied on refined variable.

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_var</code>	variable strcuture
<code>in</code>	<code>id_rho</code>	array of refinement factor
<code>in</code>	<code>id_offset</code>	offset between fine grid and coarse grid
<code>in</code>	<code>id_iext</code>	number of points to be extrapolated in i-direction
<code>in</code>	<code>id_jext</code>	number of points to be extrapolated in j-direction

Todo check if mask is really needed

12.5 src/create_layout.f90 File Reference

This program creates/computes the domain layout for you configuration.

Functions/Subroutines

- program `create_layout`

12.5.1 Detailed Description

This program creates/computes the domain layout for you configuration.

12.5.2 method

Domain layout is computed, with domain dimension, overlap between subdomain, and the number of processors available or following i and j-direction. Then the number of sea/land processors is compute with mask.

The optimized domain layout is assumed to be the domain layout, with the the most land processors removed. If no land processor could be removed, it assumed to be the domain layout with the most sea processors.

12.5.3 how to

USAGE: create_layout create_layout.nam [-v] [-h]

- positional arguments:

- create_layout.nam
namelist of create_layout

Note

a template of the namelist could be created running (in templates directory):

```
python create_templates.py create_layout
```

- optional arguments:

- -h, –help
show this help message (and exit)
- -v, –version
show Siren's version (and exit)

12.5.4 create_layout.nam

create_layout.nam contains 4 namelists:

- **namlog** to set logger parameters
- **namcfg** to set configuration file parameters
- **namvar** to set variable parameters
- **namout** to set output parameters

here after, each sub-namelist parameters is detailed.

Note

default values are specified between brackets

12.5.4.1 namlog

the logger sub-namelist parameters are :

- **cn_logfile** [*create_layout.log*]
logger filename
- **cn_verbosity** [*warning*]
verbosity level, choose between :
 - trace

- debug
- info
- warning
- error
- fatal
- none

- **in_maxerror** [5]

maximum number of error allowed

12.5.4.2 namcfg

the configuration sub-namelist parameters are :

- **cn_varcfg** [./cfg/variable.cfg]

path to the variable configuration file.

the variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.

- **cn_dimcfg** [./cfg/dimension.cfg]

path to the dimension configuration file.

the dimension configuration file defines dimensions allowed.

- **cn_dumcfg** [./cfg/dummy.cfg]

path to the useless (dummy) configuration file.

the dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

12.5.4.3 namvar

the variable sub-namelist parameters are :

- **cn_varfile** []

list of variable, and associated file

Warning

variable name must be **Bathymetry** here.

- **cn_varfile** []

: list of variable, and associated file.

cn_varfile is the path and filename of the file where find variable to be used as mask grid.

Examples:

- 'Bathymetry:bathy_meter.nc'

12.5.4.4 namout

the output sub-namelist parameters are :

- **in_niproc** [1]
: number of processor in i-direction
- **in_njproc** [1]
: number of processor in j-direction
- **in_nproc** [1]
: total number of processor to be used

Note

- if *in_niproc*, and *in_njproc* are provided : the program only look for land processor to be removed
- if *in_nproc* is provided : the program compute each possible domain layout, and save the one with the most land processor to be removed
- with no information about number of processors, the program assume to use only one processor

Author

J.Paul

Date

January, 2019 - Initial Version

Ocober, 2019

- add help and version optional arguments

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

12.6 src/create_meshmask.f90 File Reference

This program creates the NetCDF [file\(s\)](#) which contain(s) all the ocean domain informations. it also permits to create the `domain_cfg.nc` file (needed to run NEMO v4.0 and upper), or the `mesh_mask` [file\(s\)](#).

Functions/Subroutines

- program **create_meshmask**
- subroutine [create_meshmask__mask](#) (td_nam, jpi, jpj, jpk, ld_domcfg)
This subroutine compute land/ocean mask arrays at tracer points, horizontal velocity points (u & v), vorticity points (f) and barotropic stream function points (b).
- type(tatt) function, dimension(ip_maxatt) [create_meshmask__gloatt](#) (cd_bathy, cd_coord, cd_isfdep, td_namh, td_namz)
this function create array of global attributes.

12.6.1 Detailed Description

This program creates the NetCDF [file\(s\)](#) which contain(s) all the ocean domain informations. it also permits to create the `domain_cfg.nc` file (needed to run NEMO v4.0 and upper), or the `mesh_mask` [file\(s\)](#).

12.6.2 method

bathymetry (and optionally ice shelf draft) is read on input file.

horizontal grid-point position, scale factors, and the coriolis factor are read in coordinates file or computed.

vertical coordinate is defined, and the bathymetry recomputed to fit the vertical grid.

finally the masks from the bathymetry are computed.

all the variables read and or computed, are written in one to three [file\(s\)](#) depending on output option.

Note

the file contain depends on the vertical coordinate used (z-coord, partial steps, s-coord)

12.6.3 how to

USAGE: create_meshmask create_meshmask.nam [-v] [-h]

- positional arguments:

- create_meshmask.nam
namelist of create_meshmask

Note

a template of the namelist could be created running (in templates directory):

```
python create_templates.py create_meshmask
```

- optional arguments:

- -h, –help
show this help message (and exit)
- -v, –version
show Siren's version (and exit)

12.6.4 create_meshmask.nam

create_meshmask.nam contains 13 sub-namelists:

- **namlog** to set logger parameters
- **namcfg** to set configuration file parameters
- **namsrc** to set source files parameters
- **namhgr** to set horizontal grid parameters
- **namzgr** to set vertical grid parameters
- **namdmin** to set minimum depth parameters
- **namzco** to set vertical coordinate parameters
- **namzps** to set partial step parameters
- **namsco** to set sigma or hybrid parameters
- **namlbc** to set lateral boundary condition parameters
- **namwd** to set wetting and drying parameters
- **namgrd** to set grid parameters

- **namout** to set output parameters

here after, each sub-namelist parameters is detailed.

Note

default values are specified between brackets

12.6.4.1 namlog

the logger sub-namelist parameters are :

- **cn_logfile** [*create_meshmask.log*]

logger filename

- **cn_verbosity** [*warning*]

verbosity level, choose between :

- trace
- debug
- info
- warning
- error
- fatal
- none

- **in_maxerror** [*5*]

maximum number of error allowed

12.6.4.2 namcfg

the configuration sub-namelist parameters are :

- **cn_varcfg** [*./cfg/variable.cfg*]

path to the variable configuration file.

the variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.

- **cn_dimcfg** [*./cfg/dimension.cfg*]

path to the dimension configuration file.

the dimension configuration file defines dimensions allowed.

- **cn_dumcfg** [*./cfg/dummy.cfg*]

path to the useless (dummy) configuration file.

the dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

12.6.4.3 namsrc

the source grid sub-namelist parameters are :

- **cn_bathy** []
path to the bathymetry file
- **cn_varbathy** []
bathymetry variable name
- **cn_coord** []
path to the coordinate file (in_mshhgr=0)
- **cn_isfdep** []
iceshelf draft (ln_isfcav=true, see namzgr)
- **cn_varisfdep** [*isfdraft*]
iceshelf draft variable name (ln_isfcav=true, see namzgr)
- **in_perio** []
NEMO periodicity
- **ln_closea** [TRUE.]
logical to fill closed sea or not

12.6.4.4 namhgr

the grid sub-namelist parameters are :

- **in_mshhgr** [0]
type of horizontal mesh
 - 0: curvilinear coordinate on the sphere read in coordinate.nc
 - 1: geographical mesh on the sphere with regular grid-spacing
 - 2: f-plane with regular grid-spacing
 - 3: beta-plane with regular grid-spacing
 - 4: Mercator grid with T/U point at the equator
 - 5: beta-plane with regular grid-spacing and rotated domain (GYRE configuration)
- **dn_ppglam0** []
longitude of first row and column T-point (in_mshhgr = 1 or 4)
- **dn_ppgphi0** []
latitude of first row and column T-point (in_mshhgr = 1 or 4)
- **dn_ppe1_deg** []
zonal grid-spacing (degrees) (in_mshhgr = 1,2,3 or 4)
- **dn_ppe2_deg** []
meridional grid-spacing (degrees) (in_mshhgr = 1,2,3 or 4)

12.6.4.5 namzgr

the vertical grid sub-namelist parameters are :

- **In_zco** [.FALSE.]
z-coordinate - full steps
- **In_zps** [.FALSE.]
z-coordinate - partial steps
- **In_sco** [.FALSE.]
s- or hybrid z-s-coordinate
- **In_isfcav** [.FALSE.]
ice shelf cavities
- **In_iscpl** [.FALSE.]
coupling with ice sheet
- **In_wd** [.FALSE.]
Wetting/drying activation
- **in_nlevel** [75]
number of vertical level

12.6.4.6 namdmin

the minimum depth sub-namelist parameters are :

- **dn_hmin** []
minimum ocean depth (>0) or minimum number of ocean levels (<0)
- **dn_isfhmin** []
threshold to discriminate grounded ice to floating ice

12.6.4.7 namzco

the vertical coordinate sub-namelist parameters are :

- **dn_pp_to_be_computed** [0]
- **dn_ppsur** [-3958.951371276829]
coefficient to compute vertical grid
- **dn_ppa0** [103.953009600000]
coefficient to compute vertical grid
- **dn_ppa1** [2.415951269000]
coefficient to compute vertical grid
- **dn_ppkth** [15.351013700000]
coefficient to compute vertical grid
- **dn_ppacr** [7.000000000000]
coefficient to compute vertical grid

- **dn_ppdzmin** [6.]
minimum vertical spacing
- **dn_pphmax** [5750.]
maximum depth
- **In_dbletanh** [TRUE.]
use double tanh to compute vertical grid
- **dn_ppa2** [100.760928500000]
double tanh function parameter
- **dn_ppkth2** [48.029893720000]
double tanh function parameter
- **dn_ppacr2** [13.000000000000]
double tanh function parameter

Note

If *dn_ppa1*, *dn_ppa0* and *dn_ppsur* are undefined, NEMO will compute them from *dn_ppdzmin*, *dn_pphmax*, *dn_ppkth*, *dn_ppacr*

Warning

this namelist is also needed to define partial steps, sigma or hybrid coordinate.

12.6.4.8 namzps

the partial step sub-namelist parameters are :

- **dn_e3zps_min** [25.]
minimum thickness of partial step level (meters)
- **dn_e3zps_rat** [0.2]
minimum thickness ratio of partial step level

12.6.4.9 namsko

the sigma or hybrid sub-namelist parameters are :

- **In_s_sh94** [FALSE.]
use hybrid s-sig Song and Haidvogel 1994 stretching function fssig1
- **In_s_sf12** [FALSE.]
use hybrid s-z-sig Siddorn and Furner 2012 stretching function fgamma
- **dn_sbot_min** []
minimum depth of s-bottom surface (>0) (m)
- **dn_sbot_max** []
maximum depth of s-bottom surface (= ocean depth) (>0) (m)
- **dn_hc** []
Critical depth for transition from sigma to stretched coordinates
Song and Haidvogel 1994 stretching additional parameters

- **dn_rmax** []
maximum cut-off r-value allowed ($0 < \text{dn_rmax} < 1$)
- **dn_theta** []
surface control parameter ($0 \leq \text{dn_theta} \leq 20$)
- **dn_thetb** []
bottom control parameter ($0 \leq \text{dn_thetb} \leq 1$)
- **dn_bb** []
stretching parameter ($\text{dn_bb}=0$; top only, $\text{dn_bb}=1$; top and bottom)
Siddorn and Furner stretching additional parameters
- **In_sigcrit** [.FALSE.]
switching to sigma (T) or Z (F) at $H < H_c$
- **dn_alpha** []
stretchin parameter (>1 surface; <1 bottom)
- ****dn_efold**** []
e-fold length scale for transition region
- **dn_zs** []
Surface cell depth (Zs) (m)
Bottom cell (Zb) (m) = $H * \text{rn_zb_a} + \text{rn_zb_b}$
 - **dn_zb_a** []
Bathymetry multiplier for Zb
 - **dn_zb_b** []
Offset for Zb

12.6.4.10 namlbc

the lateral boundary condition sub-namelist parameters are :

- **rn_shlat** [2.]
lateral boundary conditions at the coast (modify fmask)
 - shlat = 0 : free slip
 - $0 < \text{shlat} < 2$: partial slip
 - shlat = 2 : no slip
 - shlat > 2 : strong slip

for more information see Boundary Condition at the Coast in [NEMO documentation](#)

12.6.4.11 namwd

the wetting and drying sub-namelist parameters are :

- **dn_wdmin1** []
minimum water depth on dried cells
- **dn_wdmin2** []
tolerrance of minimum water depth on dried cells
- **dn_wdld** []
land elevation below which wetting/drying

12.6.4.12 namgrd

the grid sub-namelist parameters are :

- **in_cfg** [0]
inverse resolution of the configuration ($1/4^\circ \Rightarrow 4$)
- **In_bench** [.FALSE.]
GYRE (in_mshhgr = 5) used as Benchmark.
 \Rightarrow forced the resolution to be about 100 km
- **In_c1d** [.FALSE.]
use configuration 1D
- **In_e3_dep** [.FALSE.]
vertical scale factors =T: e3.=dk[depth] =F: old definition

12.6.4.13 namout

the output sub-namelist parameters are :

- **cn_domcfg** [*domain_cfg.nc*]
output file name
- **in_msh** [0]
number of output file and contain (0-9)
- **in_nproc** [1]
number of processor to be used
- **in_niproc** [1]
i-direction number of processor
- **in_njproc** [1]
j-direction numebr of processor
 - if niproc, and njproc are provided : the program only look for land processor to be removed
 - if nproc is provided : the program compute each possible domain layout, and save the one with the most land processor to be removed
 - with no information about number of processors, the program assume to use only one processor

Note

- if in_msh = 0 : write '**domain_cfg.nc**' file.
- if MOD(in_msh, 3) = 1 : write '**mesh_mask.nc**' file.
- if MOD(in_msh, 3) = 2 : write '**mesh.nc**' and '**mask.nc**' files.
- if MOD(in_msh, 3) = 0 : write '**mesh_hgr.nc**', '**mesh_zgr.nc**' and '**mask.nc**' files.
For huge size domain, use option 2 or 3 depending on your vertical coordinate.
- if $0 < \text{in_msh} \leq 3$: write full 3D arrays for e3[tuvw] and gdep[tuvw]
- if $3 < \text{in_msh} \leq 6$: write full 3D arrays for e3[tuvw] and 2D arrays corresponding to the depth of the bottom t- and w-points
- if $6 < \text{in_msh} \leq 9$: write 2D arrays corresponding to the depth and the thickness (e3[tw]_ps) of the bottom points

Author

J.Paul

Date

September, 2015 - Initial Version (based on domhgr.F90, domzgr.F90, domwri.F90)

October, 2016

- update from trunk (revision 6961): add wetting and drying, ice sheet coupling..

October, 2016

- dimension to be used select from configuration file
- do not use anymore special case for ORCA grid
- allow to write domain_cfg file

November, 2016

- choose vertical scale factors (e3.=dk[depth] or old definition)

January, 2019

- add url path to global attributes of output [file\(s\)](#)

February, 2019

- rename sub namelist namin to namsrc

October, 2019

- add help and version optional arguments

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

12.6.5 Function/Subroutine Documentation

12.6.5.1 `type(tatt) function, dimension(ip_maxatt) create_meshmask::create_meshmask_gloatt (character(len=*), intent(in) cd_bathy, character(len=*), intent(in) cd_coord, character(len=*), intent(in) cd_isfdep, type(tnamh), intent(in) td_namh, type(tnamz), intent(in) td_namz)`

this function create array of global attributes.

Parameters

in	<i>cd_bathy</i>
----	-----------------

Author

J.Paul

Date

October, 2016 - initial release

Parameters

in	<i>cd_bathy</i>
----	-----------------

in	<i>cd_coord</i>	
in	<i>cd_isfdep</i>	
in	<i>td_namh</i>	
in	<i>td_namz</i>	

12.6.5.2 subroutine create_meshmask::create_meshmask__mask (type(tnamh), intent(in) *td_nam*, integer(i4), intent(in) *jpi*, integer(i4), intent(in) *npj*, integer(i4), intent(in) *jpj*, logical, intent(in) *ld_domcfg*)

This subroutine compute land/ocean mask arrays at tracer points, horizontal velocity points (u & v), vorticity points (f) and barotropic stream function points (b).

Parameters

in	<i>td_nam</i>	
----	---------------	--

** Method : The ocean/land mask is computed from the basin bathymetry in level (mbathy) which is defined or read in dommba. mbathy equals 0 over continental T-point and the number of ocean level over the ocean.

At a given position (ji,jj,jk) the ocean/land mask is given by:

- t-point :
 - 0. IF mbathy(ji ,jj) =< 0
 - 1. IF mbathy(ji ,jj) >= jk
- u-point :
 - 0. IF mbathy(ji ,jj) or mbathy(ji+1, jj) =< 0
 - 1. IF mbathy(ji ,jj) and mbathy(ji+1, jj) >= jk.
- v-point :
 - 0. IF mbathy(ji ,jj) or mbathy(ji ,jj+1) =< 0
 - 1. IF mbathy(ji ,jj) and mbathy(ji ,jj+1) >= jk.
- f-point :
 - 0. IF mbathy(ji ,jj) or mbathy(ji ,jj+1) or mbathy(ji+1,jj) or mbathy(ji+1,jj+1) =< 0
 - 1. IF mbathy(ji ,jj) and mbathy(ji ,jj+1) and mbathy(ji+1,jj) and mbathy(ji+1,jj+1) >= jk.
- b-point : the same definition as for f-point of the first ocean level (surface level) but with 0 along coastlines.
- tmask_i : interior ocean mask at t-point, i.e. excluding duplicated rows/lines due to cyclic or North Fold boundaries as well as MPP halos.

Warning

do not set the lateral friction through the value of fmask along the coast and topography.

Note

If nperio not equal to 0, the land/ocean mask arrays are defined with the proper value at lateral domain boundaries, but bmask. indeed, bmask defined the domain over which the barotropic stream function is computed. this domain cannot contain identical columns because the matrix associated with the barotropic stream function equation is then no more invertible. therefore bmask is set to 0 along lateral domain boundaries even IF nperio is not zero.

In case of open boundaries (lk_bdy=T):

- tmask is set to 1 on the points to be computed bay the open boundaries routines.

- bmask is set to 0 on the open boundaries.

** Action :

- tmask : land/ocean mask at t-point (=0. or 1.)
- umask : land/ocean mask at u-point (=0. or 1.)
- vmask : land/ocean mask at v-point (=0. or 1.)
- fmask : land/ocean mask at f-point (=0. or 1.)
- bmask : land/ocean mask at barotropic stream function point (=0. or 1.) and set to 0 along lateral boundaries
- tmask_i : interior ocean mask

Author

J.Paul

Date

September, 2015 - rewrite from dom_msk

October, 2016

- do not use anymore special case for ORCA grid

Parameters

in	<i>td_nam</i>	
in	<i>jpi</i>	
in	<i>npj</i>	
in	<i>npk</i>	

12.7 src/create_restart.f90 File Reference

this program creates restart file or initial state.

Functions/Subroutines

- program **create_restart**
- type(tvar) function [create_restart_matrix](#) (td_var, td_coord, id_nlevel, id_xghost)
This function create variable, filled with matrix value.
- subroutine [create_restart_mask](#) (td_var, td_mask)
This subroutine use mask to filled land point with _FillValue.
- subroutine [create_restart_interp](#) (td_var, id_rho, id_offset, id_iext, id_jext)
This subroutine interpolate variable.
- subroutine [create_restart_check_depth](#) (td_mpp, td_depth)
This subroutine get depth variable value in an open mpp structure and check if agree with already input depth variable.
- subroutine [create_restart_check_time](#) (td_mpp, td_time)
This subroutine get date and time in an open mpp structure and check if agree with date and time already read.

12.7.1 Detailed Description

this program creates restart file or initial state.

12.7.2 method

variables could be

- extracted from fine grid file
- interpolated from coarse grid file
- interpolated from restart file
- handwritten

then they are split over new layout.

Note

interpolation and/or extrapolation method could be different for each variable.

12.7.3 how to

USAGE: create_restart create_restart.nam [-v] [-h]

- positional arguments:

- create_restart.nam
namelist of create_restart

Note

a template of the namelist could be created running (in templates directory):

```
python create_templates.py create_restart
```

- optional arguments:

- -h, -help
show this help message (and exit)
- -v, -version
show Siren's version (and exit)

12.7.4 create_restart.nam

create_restart.nam contains 9 namelists:

- **namlog** to set logger parameters
- **namcfg** to set configuration file parameters
- **namsrc** to set source/coarse grid parameters
- **namtgt** to set target/fine grid parameters
- **namzgr** to set vertical grid parameters
- **namzps** to set partial step parameters
- **namvar** to set variable parameters
- **namnst** to set sub domain and nesting parameters
- **namout** to set output parameters

here after, each sub-namelist parameters is detailed.

Note

default values are specified between brackets

12.7.4.1 namlog

the logger sub-namelist parameters are :

- **cn_logfile** [*create_restart.log*]
logger filename
- **cn_verbosity** [*warning*]
verbosity level, choose between :
 - trace
 - debug
 - info
 - warning
 - error
 - fatal
 - none
- **in_maxerror** [*5*]
maximum number of error allowed

12.7.4.2 namcfg

the configuration sub-namelist parameters are :

- **cn_varcfg** [*./cfg/variable.cfg*]
path to the variable configuration file.
the variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.
- **cn_dimcfg** [*./cfg/dimension.cfg*]
path to the dimension configuration file.
the dimension configuration file defines dimensions allowed.
- **cn_dumcfg** [*./cfg/dummy.cfg*]
path to the useless (dummy) configuration file.
the dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

12.7.4.3 namsrc

the coarse grid sub-namelist parameters are :

- **cn_coord0** []
path to the coordinate file
- **in_perio0** []
NEMO periodicity index
the NEMO periodicity could be choose between 0 to 6:
in_perio=0 standard regional model
in_perio=1 east-west cyclic model

- in_perio=2** model with symmetric boundary condition across the equator
- in_perio=3** regional model with North fold boundary and T-point pivot
- in_perio=4** global model with a T-point pivot.
example: ORCA2, ORCA025, ORCA12
- in_perio=5** regional model with North fold boundary and F-point pivot
- in_perio=6** global model with a F-point pivot
example: ORCA05

See Also

For more information see [NEMO periodicity](#) and Model Boundary Condition paragraph in the [NEMO documentation](#)

12.7.4.4 namtgt

the fine grid sub-namelist parameters are :

- **cn_coord1** []
path to coordinate file
- **cn_bathy1** []
path to bathymetry file
- **in_perio1** []
NEMO periodicity index (see above)

Note

if the fine/target coordinates file (cn_coord1) was created by SIREN, you do not need to fill this parameter. SIREN will read it on the global attributes of the coordinates file.

12.7.4.5 namzgr

the vertical grid sub-namelist parameters are :

- **dn_pp_to_be_computed** [0]
- **dn_ppsur** [-3958.951371276829]
coefficient to compute vertical grid
- **dn_ppa0** [103.953009600000]
coefficient to compute vertical grid
- **dn_ppa1** [2.415951269000]
coefficient to compute vertical grid
- **dn_ppa2** [100.760928500000]
double tanh function parameter
- **dn_ppkth** [15.351013700000]
coefficient to compute vertical grid
- **dn_ppkth2** [48.029893720000]
double tanh function parameter
- **dn_ppacr** [7.000000000000]
coefficient to compute vertical grid

- **dn_ppacr2** [13.000000000000]
 - double tanh function parameter
- **dn_ppdzmin** [6.]
 - minimum vertical spacing
- **dn_pphmax** [5750.]
 - maximum depth
- **in_nlevel** [75]
 - number of vertical level

Note

If *dn_ppa1*, *dn_ppa0* and *dn_ppsur* are undefined, NEMO will compute them from *dn_ppdzmin*, *dn_pphmax*, *dn_ppkth*, *dn_ppacr*

12.7.4.6 namzps

the partial step sub-namelist parameters are :

- **dn_e3zps_min** [25.]
 - minimum thickness of partial step level (meters)
- **dn_e3zps_rat** [0.2]
 - minimum thickness ratio of partial step level

12.7.4.7 namvar

the variable sub-namelist parameters are :

- **cn_varfile** []
 - list of variable, and associated file
 - cn_varfile* is the path and filename of the file where find variable.

Note

cn_varfile could be a matrix of value, if you want to handwrite variable value.
 the variable array of value is split into equal subdomain.
 each subdomain is filled with the corresponding value of the matrix.
 separators used to defined matrix are:

- ‘,’ for line
 - ‘/’ for row Example:
- $$3,2,3/1,4,5 \Rightarrow \begin{pmatrix} 3 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}$$

Examples:

- ‘votemper:gridT.nc’, ‘vozocrtx:gridU.nc’
- ‘votemper:10\25’, ‘vozocrtx:gridU.nc’

Note

to get all variables from one file:

Example:

- ‘all:restart.dimg’

Note

Optionnaly, NEMO periodicity could be added following the filename. the periodicity must be separated by ';'

Example:

```
– 'votemper:gridT.nc ; perio=4'
```

- **cn_varinfo []**

list of variable and extra information about request(s) to be used

each elements of *cn_varinfo* is a string character (separated by ';').

it is composed of the variable name follow by ':', then request(s) to be used on this variable.

request could be:

- int = interpolation method
 - ext = extrapolation method
 - flt = filter method
 - min = minimum value
 - max = maximum value
 - unt = new units
 - unf = unit scale factor (linked to new units)
- requests must be separated by ';'.
order of requests does not matter.

informations about available method could be find in [interp](#), [extrap](#) and [filter](#) modules.

Example:

```
– 'votemper: int=linear; flt=hann; ext=dist_weight', 'vosaline: int=cubic'
```

Note

If you do not specify a method which is required, default one is apply.

12.7.4.8 namnst

the nesting sub-namelist parameters are :

- **in_rhoi [1]**
refinement factor in i-direction
- **in_rhoj [1]**
refinement factor in j-direction

Note

coarse grid indices will be deduced from fine grid coordinate file.

12.7.4.9 namout

the output sub-namelist parameter is :

- **cn_fileout** [*restart.nc*]
output bathymetry filename
- **In_extrap** [FALSE.]
logical to extrapolate land point or not
- **in_niproc** [*1*]
number of processor in i-direction
- **in_njproc** [*1*]
number of processor in j-direction
- **in_nproc** [*1*]
total number of processor to be used
- **cn_type** []
output format ('dimg', 'cdf')

Note

- if *in_niproc*, and *in_njproc* are provided : the program only look for land processor to be removed
- if *in_nproc* is provided : the program compute each possible domain layout, and save the one with the most land processor to be removed
- with no information about number of processors, the program assume to use only one processor

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header for user
- offset computed considering grid point
- add attributes in output variable

June, 2015

- extrapolate all land points, and add *In_extrap* in namelist.
- allow to change unit.

September, 2015

- manage useless (dummy) variable, attributes, and dimension

October, 2016

- dimension to be used select from configuration file

January, 2019

- add url path to global attributes of output [file\(s\)](#)
- check name and standard name for longitude and latitude

February, 2019

- rename sub namelist *namcrs* to *namsrc*

- rename sub namelist namfin to namtgt

May, 2019

- create and clean file structure to avoid memory leaks

August, 2019

- use periodicity read from namelist, and store in multi structure

October, 2019

- add help and version optional arguments

Todo • rewrite using meshmask instead of bathymetry and coordinates files

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

12.7.5 Function/Subroutine Documentation

12.7.5.1 subroutine `create_restart::create_restart_check_depth` (type(tmpp), intent(in) *td_mpp*, type(tvar), intent(inout) *td_depth*)

This subroutine get depth variable value in an open mpp structure and check if agree with already input depth variable.

Parameters

<i>in</i>	<i>td_mpp</i>
-----------	---------------

Author

J.Paul

Date

November, 2014 - Initial Version

Parameters

<i>in</i>	<i>td_mpp</i>	mpp structure
<i>in, out</i>	<i>td_depth</i>	depth variable structure

12.7.5.2 subroutine `create_restart::create_restart_check_time` (type(tmpp), intent(in) *td_mpp*, type(tvar), intent(inout) *td_time*)

This subroutine get date and time in an open mpp structure and check if agree with date and time already read.

Parameters

<i>in</i>	<i>td_mpp</i>
-----------	---------------

Author

J.Paul

Date

November, 2014 - Initial Version

Parameters

in	<i>td_mpp</i>	mpp structure
in, out	<i>td_time</i>	time variable structure

12.7.5.3 subroutine `create_restart::create_restart_interp` (type(tvar), intent(inout) *td_var*, integer(i4), dimension(:), intent(in) *id_rho*, integer(i4), dimension(:,:), intent(in) *id_offset*, integer(i4), intent(in), optional *id_jext*, integer(i4), intent(in), optional *id_jext*)

This subroutine interpolate variable.

Parameters

in, out	<i>td_var</i>	
---------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- do not use level anymore (for extrapolation)

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>id_rho</i>	array of refinement factor
in	<i>id_offset</i>	array of offset between fine and coarse grid
in	<i>id_jext</i>	i-direction size of extra bands (default=im_minext)
in	<i>id_jext</i>	j-direction size of extra bands (default=im_minext)

12.7.5.4 subroutine `create_restart::create_restart_mask` (type(tvar), intent(inout) *td_var*, type(tvar), dimension(:), intent(in) *td_mask*)

This subroutine use mask to filled land point with `_FillValue`.

Parameters

in, out	<i>td_var</i>	
---------	---------------	--

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

in, out	<i>td_var</i>	variable structure
in	<i>td_mask</i>	mask variable structure

12.7.5.5 `type(tvar) function create_restart::create_restart_matrix (type(tvar), intent(in) td_var, type(tmpp), intent(in) td_coord, integer(i4), intent(in) id_nlevel, integer(i4), dimension(:,,:), intent(in) id_xghost)`

This function create variable, filled with matrix value.

Parameters

in	<i>td_var</i>
----	---------------

A variable is create with the same name that the input variable, and with dimension of the coordinate file.

Then the variable array of value is split into equal subdomain. Each subdomain is filled with the associated value of the matrix.

Author

J.Paul

Date

November, 2013 - Initial Version

June, 2015

- do not use level anymore

Parameters

in	<i>td_var</i>	variable structure
in	<i>td_coord</i>	coordinate file structure
in	<i>id_nlevel</i>	number of vertical level
in	<i>id_xghost</i>	ghost cell array

Returns

variable structure

12.8 src/merge_bathy.f90 File Reference

this program merges bathymetry file at boundaries.

Functions/Subroutines

- program **merge_bathy**
- subroutine [merge_bathy_get_boundary](#) (*td_bathy0*, *td_bathy1*, *td_bdy*, *id_rho*, *id_ncrs*, *dd_refined*, *dd_weight*, *dd_fill*)
This subroutine compute refined bathymetry on boundary from coarse grid.
- subroutine [merge_bathy_interp](#) (*td_var*, *id_rho*, *id_offset*, *id_iext*, *id_jext*)
This subroutine interpolate variable.

12.8.1 Detailed Description

this program merges bathymetry file at boundaries.

12.8.2 method

coarse grid bathymetry is interpolated on fine grid (nearest interpolation method is used).
then fine bathymetry and refined coarse bathymetry are merged at boundaries.

$$BathyFine = Weight * BathyCoarse + (1 - Weight) * BathyFine$$

the weight function used is :

$$Weight = 0.5 + 0.5 * COS\left(\frac{\pi * dist}{width}\right)$$

with

- dist : number of point to border
- width : boundary size

12.8.3 how to

USAGE: merge_bathy merge_bathy.nam [-v] [-h]

- positional arguments:
 - merge_bathy.nam
namelist of merge_bathy

Note

a template of the namelist could be created running (in templates directory):
`python create_templates.py merge_bathy`

- optional arguments:
 - -h, -help
show this help message (and exit)
 - -v, -version
show Siren's version (and exit)

12.8.4 merge_bathy.nam

merge_bathy.nam contains 7 namelists:

- **namlog** to set logger parameters
- **namcfg** to set configuration file parameters
- **namsrc** to set source/coarse grid parameters
- **namtgt** to set target/fine grid parameters
- **namnst** to set sub domain and nesting paramters
- **nambdy** to set boundary parameters
- **namout** to set output parameters

here after, each sub-namelist parameters is detailed.

Note

default values are specified between brackets

12.8.4.1 namlog

the logger sub-namelist parameters are :

- **cn_logfile** [*merge_bathy.log*]
logger filename
- **cn_verbosity** [*warning*]
verbosity level, choose between :
 - trace
 - debug
 - info
 - warning
 - error
 - fatal
 - none
- **in_maxerror** [*5*]
maximum number of error allowed

12.8.4.2 namcfg

the configuration sub-namelist parameters are :

- **cn_varcfg** [*./cfg/variable.cfg*]
path to the variable configuration file.
the variable configuration file defines standard name, default interpolation method, axis,... to be used for some known variables.
- **cn_dimcfg** [*./cfg/dimension.cfg*]
path to the dimension configuration file.
the dimension configuration file defines dimensions allowed.
- **cn_dumcfg** [*./cfg/dummy.cfg*]
path to the useless (dummy) configuration file.
the dummy configuration file defines useless dimension or variable. these dimension(s) or variable(s) will not be processed.

12.8.4.3 namsrc

the source/coarse grid sub-namelist parameters are :

- **cn_bathy0** []
path to the bathymetry file

Warning

variable name must be **Bathymetry** here.

- **in_perio0** []

NEMO periodicity index

the NEMO periodicity could be choose between 0 to 6:

in_perio=0 standard regional model

in_perio=1 east-west cyclic model

in_perio=2 model with symmetric boundary condition across the equator

in_perio=3 regional model with North fold boundary and T-point pivot

in_perio=4 global model with a T-point pivot.

example: ORCA2, ORCA025, ORCA12

in_perio=5 regional model with North fold boundary and F-point pivot

in_perio=6 global model with a F-point pivot

example: ORCA05

See Also

For more information see [NEMO periodicity](#) and Model Boundary Condition paragraph in the [NEMO documentation](#)

12.8.4.4 namtgt

the target/fine grid sub-namelist parameters are :

- **cn_bathy1** []

path to bathymetry file

Warning

variable name must be **Bathymetry** here.

- **in_perio1** []

NEMO periodicity index (see above)

Note

if the fine/target coordinates file (cn_coord1) was created by SIREN, you do not need to fill this parameter. SIREN will read it on the global attributes of the coordinates file.

12.8.4.5 namnst

the nesting sub-namelist parameters are (default value are specified between brackets):

- **in_rhoi** [1]

refinement factor in i-direction

- **in_rhoj** [1]

refinement factor in j-direction

Note

coarse grid indices will be deduced from fine grid coordinate file.

12.8.4.6 nambdy

the boundary sub-namelist parameters are :

- **In_north** [.TRUE.]

logical to use north boundary or not

- **In_south** [.TRUE.]

logical to use south boundary or not

- **In_east** [.TRUE.]

logical to use east boundary or not

- **In_west** [.TRUE.]

logical to use west boundary or not

- **cn_north** []

north boundary indices on fine grid

- **cn_south** []

south boundary indices on fine grid

- **cn_east** []

east boundary indices on fine grid

- **cn_west** []

west boundary indices on fine grid

cn_north is a string character defining boundary segmentation.

segments are separated by '|'.

each segments of the boundary is composed of:

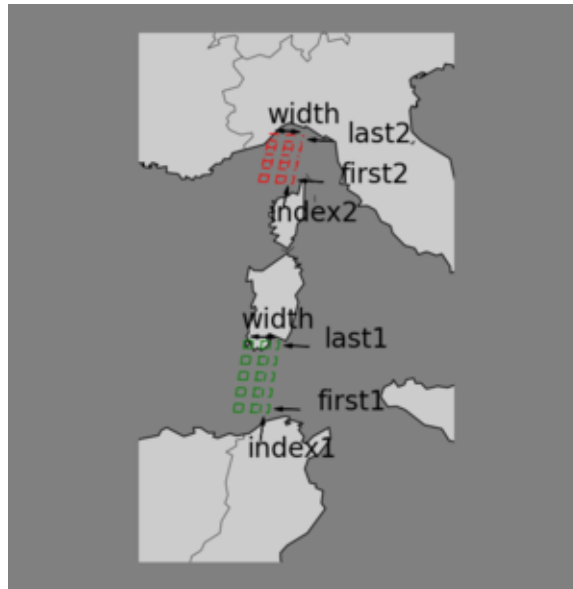
- indice of velocity (orthogonal to boundary .ie. for north boundary, J-indice).
- indice of segment start (I-indice for north boundary)
- indice of segment end (I-indice for north boundary)
indices must be separated by ':' .
- optionally, boundary size could be added between '(' and ')' in the first segment defined.

Note

boundary size is the same for all segments of one boundary.

Examples:

- `cn_north='index1,first1:last1(width)'`
- `cn_north='index1(width),first1:last1|index2,first2:last2'`



- **in_ncrs** [2]
number of point(s) with coarse value save at boundaries
- **In_oneseq** [.TRUE.]
logical to use only one segment for each boundary or not

12.8.4.7 namout

the output sub-namelist parameter is :

- **cn_fileout** [*bathy_merged.nc*]
output bathymetry filename

Author

J.Paul

Date

November, 2013 - Initial Version

September, 2014

- add header for user

July, 2015

- extrapolate all land points
- add attributes with boundary string character (as in namelist)

September, 2015

- manage useless (dummy) variable, attributes, and dimension

October, 2016

- allow to choose the number of boundary point with coarse grid value.
- dimension to be used select from configuration file

January, 2019

- add url path to global attributes of output [file\(s\)](#)

February, 2019

- rename sub namelist namcrs to namsrc
- rename sub namelist namfin to namtgt

May, 2019

- create and clean file structure to avoid memory leaks

October, 2019

- add help and version optional arguments

Note

Software governed by the CeCILL licence (NEMOGCM/NEMO_CeCILL.txt)

12.8.5 Function/Subroutine Documentation

12.8.5.1 subroutine `merge_bathy::merge_bathy_get_boundary` (`type(tmp)`, `intent(in) td_bathy0`, `type(tmp)`, `intent(in) td_bathy1`, `type(tbdy)`, `intent(in) td_bdy`, `integer(i4)`, `dimension(:)`, `intent(in) id_rho`, `integer(i4)`, `intent(in) id_ncrs`, `real(dp)`, `dimension(:, :, :)`, `intent(inout) dd_refined`, `real(dp)`, `dimension(:, :, :)`, `intent(inout) dd_weight`, `real(dp)`, `intent(in) dd_fill`)

This subroutine compute refined bathymetry on boundary from coarse grid.

Parameters

<code>in</code>	<code>td_bathy0</code>
-----------------	------------------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in</code>	<code>td_bathy0</code>	coarse grid bathymetry file structure
<code>in</code>	<code>td_bathy1</code>	fine grid bathymetry file structure
<code>in</code>	<code>td_bdy</code>	boundary structure
<code>in</code>	<code>id_rho</code>	array of refinement factor
<code>in</code>	<code>id_ncrs</code>	number of point with coarse value save at boundaries
<code>in, out</code>	<code>dd_refined</code>	array of refined bathymetry
<code>in, out</code>	<code>dd_weight</code>	array of weight
<code>in</code>	<code>dd_fill</code>	fillValue

12.8.5.2 subroutine `merge_bathy::merge_bathy_interp` (`type(tvar)`, `intent(inout) td_var`, `integer(i4)`, `dimension(:)`, `intent(in) id_rho`, `integer(i4)`, `dimension(:,:)`, `intent(in) id_offset`, `integer(i4)`, `intent(in)`, `optional id_jext`, `integer(i4)`, `intent(in)`, `optional id_jext`)

This subroutine interpolate variable.

Parameters

<code>in, out</code>	<code>td_var</code>
----------------------	---------------------

Author

J.Paul

Date

November, 2013 - Initial Version

Parameters

<code>in, out</code>	<code>td_var</code>	variable structure
<code>in</code>	<code>id_rho</code>	array of refinement factor
<code>in</code>	<code>id_offset</code>	array of offset between fine and coarse grid
<code>in</code>	<code>id_jext</code>	i-direction size of extra bands (default= <code>im_minext</code>)
<code>in</code>	<code>id_jext</code>	j-direction size of extra bands (default= <code>im_minext</code>)