11 Ouput and Diagnostics (IOM, DIA, TRD, FLO)

Contents

11.1 Old Model Output (default or key_dimgout)

The model outputs are of three types : the restart file, the output listing, and the diagnostic output file(s). The restart file is used internally by the code when the user wants to start the model with initial conditions defined by a previous simulation. It contains all the information that is necessary in order for there to be no changes in the model results (even at the computer precision) between a run performed with several restarts and the same run performed in one step. It should be noted that this requires that the restart file contain two consecutive time steps for all the prognostic variables, and that it is saved in the same binary format as the one used by the computer that is to read it (in particular, 32 bits binary IEEE format must not be used for this file).

The output listing and file(s) are predefined but should be checked and eventually adapted to the user's needs. The output listing is stored in the $ocean.output$ file. The information is printed from within the code on the logical unit $numout$. To locate these prints, use the UNIX command "*grep -i numout*" in the source code directory.

By default, diagnostic output files are written in NetCDF format but an IEEE binary output format, called DIMG, can be choosen by defining **key_dimgout**.

Since version 3.2, when defining **key_iomput**, an I/O server has been added which provides more flexibility in the choice of the fields to be written as well as how the writing work is distributed over the processors in massively parallel computing. The complete description of the use of this I/O server is presented in next section.

By default, if neither **key_iomput** nor **key_dimgout** are defined, NEMO produces NetCDF with the old IOIPSL library which has been kept for compatibility and its easy installation. However, the IOIPSL library is quite inefficient on parallel machines and, since version 3.2, many diagnostic options have been added presuming the use of **key_iomput**. The usefulness of the default IOIPSL-based option is expected to reduce with each new release. If **key_iomput** is not defined, output files and content are defined in the *diawri.F90* module and contain mean (or instantaneous if **key_diainstant** is defined) values over a period of nn_write time-step (namelist parameter).

11.2 Standard model Output (IOM)

Since version 3.2, iomput is the NEMO output interface of choice. It has been designed to be simple to use, flexible and efficient. The two main purposes of iomput are :

1. The complete and flexible control of the output files through external XML files adapted by the user from standard templates.
2. To achieve high performance and scalable output through the optional distribution of all diagnostic output related tasks to dedicated processes.

The first functionality allows the user to specify, without code changes or recompilation, aspects of the diagnostic output stream, such as :

– The choice of output frequencies that can be different for each file (including real months and years).
– The choice of file contents ; includes complete flexibility over which data are written in which files (the same data can be written in different files).
– The possibility to split output files at a choosen frequency.
– The possibility to extract a vertical or an horizontal subdomain.
– The choice of the temporal operation to perform, e.g. : average, accumulate, instantaneous, min, max and once.
– Control over metadata via a large XML ”database” of possible output fields.

In addition, iomput allows the user to add the output of any new variable (scalar, 2D or 3D) in the code in a very easy way. All details of iomput functionalities are listed in the following subsections. Examples of the XML files that control the outputs can be found in :

```
NEMOGCM/CONFIG/ORCA2_LIM/EXP00/iodef.xml
NEMOGCM/CONFIG/SHARED/field_def.xml
and
NEMOGCM/CONFIG/SHARED/domain_def.xml.
```

The second functionality targets output performance when running in parallel (**key_mpp_mpi**). Iomput provides the possibility to specify N dedicated I/O processes (in addition to the NEMO processes) to collect and write the outputs. With an appropriate choice of N by the user the bottleneck associated with the writing of the output files can be greatly reduced.

Since version 3.5, the iom_put interface depends on an external code called XIOS. This new IO server can take advantage of the parallel I/O functionality of NetCDF4 to create a single output file and therefore to bypass the rebuilding phase. Note that writing in parallel into the same NetCDF files requires that your NetCDF4 library is linked to an HDF5 library that has been correctly compiled (i.e. with the configure option −−enable-parallel). Note that the files created by iomput through XIOS are incompatible with NetCDF3. All post-processsing and visualization tools must therefore be compatible with NetCDF4 and not only NetCDF3.

Even if not using the parallel I/O functionality of NetCDF4, using N dedicated I/O servers, where N is typically much less than the number of NEMO processors, will reduce the number of output files created. This can greatly reduce the post-processing burden usually associated with using large numbers of NEMO processors. Note that for smaller configurations, the rebuilding phase can be avoided, even without a parallel-enabled NetCDF4 library, simply by employing only one dedicated I/O server.

11.2.1 XIOS : the IO_SERVER

Attached or detached mode ?

Iomput is based on XIOS, the io_server developed by Yann Meurdesoif from IPSL. The behaviour of the io subsystem is controlled by settings in the external XML files listed above. Key settings in the iodef.xml file are `using_server` and the `type` tag associated with each defined file. The `using_server` setting determines whether or not the server will be used in ”attached mode” (as a library) [`false`] or in ”detached mode” (as an external executable on N additional, dedicated cpus) [`true`]. The ”attached mode” is simpler to use but much less efficient for massively parallel applications. The type of each file can be either ”multiple_file” or ”one_file”.

In attached mode and if the type of file is ”multiple_file”, then each NEMO process will also act as an IO server and produce its own set of output files. Superficially, this emulates the standard behaviour in previous versions, However, the subdomain written out by each process does not correspond to the `jpi x jpj x jpk` domain actually computed by the process (although it may if `jpni=1`). Instead each process will have collected and written out a number of complete longitudinal strips. If the ”one_file” option is chosen then all processes will collect their longitudinal strips and write (in parallel) to a single output file.

In detached mode and if the type of file is ”multiple_file”, then each stand-alone XIOS process will collect data for a range of complete longitudinal strips and write to its own set of output files. If the ”one_file” option is chosen then all XIOS processes will collect their longitudinal strips and write (in parallel) to a single output file. Note running in detached mode requires launching a Multiple Process Multiple Data (MPMD) parallel job. The following subsection provides a typical example but the syntax will vary in different MPP environments.

Number of cpu used by XIOS in detached mode

The number of cores used by the XIOS is specified when launching the model. The number of cores dedicated to XIOS should be from 1/10 to 1/50 of the number or cores dedicated to NEMO. Some manufacturers suggest using O($\sqrt{N}$) dedicated IO processors for N processors but this is a general recommendation and not specific to NEMO. It is difficult to provide precise recommendations because the optimal choice will depend on the particular hardware properties of the target system (parallel filesystem performance, available memory, memory bandwidth etc.) and the volume and frequency of data to be created. Here is an example of 2 cpus for the io_server and 62 cpu for nemo using mpirun :

```
mpirun -np 62 ./nemo.exe : -np 2 ./xios_server.exe
```

Control of XIOS : the XIOS context in iodef.xml

As well as the `using_server` flag, other controls on the use of XIOS are set in the XIOS context in iodef.xml. See the XML basics section below for more details on XML syntax and rules.

variable name	description	example
buffer_size	buffer size used by XIOS to send data from NEMO to XIOS. Larger is more efficient. Note that needed/used buffer sizes are summarized at the end of the job	25000000
buffer_server_factor_size	ratio between NEMO and XIOS buffer size. Should be 2.	2
info_level	verbosity level (0 to 100)	0
using_server	activate attached(false) or detached(true) mode	true
using_oasis	XIOS is used with OASIS(true) or not (false)	false
oasis_codes_id	when using oasis, define the identifier of NEMO in the namcouple. Note that the identifier of XIOS is xios.x	oceanx

11.2.2 Practical issues

Installation

As mentioned, XIOS is supported separately and must be downloaded and compiled before it can be used with NEMO. See the installation guide on the XIOS wiki for help and guidance. NEMO will need to link to the compiled XIOS library. The XIOS with NEMO guide provides an example illustration of how this can be achieved.

Add your own outputs

It is very easy to add your own outputs with iomput. Many standard fields and diagnostics are already prepared (i.e., steps 1 to 3 below have been done) and simply need to be activated by including the required output in a file definition in iodef.xml (step 4). To add new output variables, all 4 of the following steps must be taken.

1. in NEMO code, add a
```
CALL iom_put( 'identifier', array )
```
where you want to output a 2D or 3D array.

2. If necessary, add
```
USE iom              ! I/O manager library
```
to the list of used modules in the upper part of your module.

3. in the field_def.xml file, add the definition of your variable using the same identifier you used in the f90 code (see subsequent sections for a details of the XML syntax and rules). For example :

```
<field_definition>
   <!-- T grid -->

  <field_group id="grid_T" grid_ref="grid_T_3D">
   ...
   <field id="identifier" long_name="blabla" ... />
   ...
</field_definition>
```

Note your definition must be added to the field_group whose reference grid is consistent with the size of the array passed to iomput. The grid_ref attribute refers to definitions set in iodef.xml which, in turn, reference grids and axes either defined in the code (iom_set_domain_attr and iom_set_axis_attr in iom.F90) or defined in the domain_def.xml file. E.g. :

```
  <grid id="grid_T_3D" domain_ref="grid_T" axis_ref="deptht"/>
```

Note, if your array is computed within the surface module each nn_fsbc time_step, add the field definition within the field_group defined with the id ”SBC” : <field_group id=”SBC”...> which has been defined with the correct frequency of operations (iom_set_field_attr in iom.F90)

4. add your field in one of the output files defined in iodef.xml (again see subsequent sections for syntax and rules)

```
<file id="file1" .../>
   ...
   <field field_ref="identifier" />
   ...
</file>
```

11.2.3 XML fundamentals

XML basic rules

XML tags begin with the less-than character (”<”) and end with the greater-than character (”>”). You use tags to mark the start and end of elements, which are the logical units of information in an XML document. In addition to marking the beginning of an element, XML start tags also provide a place to specify attributes. An attribute specifies a single property for an element, using a name/value pair, for example : <a b=”x” c=”y” b=”z”> ... </a>. See here for more details.

Structure of the xml file used in NEMO

The XML file used in XIOS is structured by 7 families of tags : context, axis, domain, grid, field, file and variable. Each tag family has hierarchy of three flavors (except for context) :

flavor	description	example
root	declaration of the root element that can contain element groups or elements	`< file_definition ... >`
group	declaration of a group element that can contain element groups or elements	`< file_group ... >`
element	declaration of an element that can contain elements	`< file ... >`

Each element may have several attributes. Some attributes are mandatory, other are optional but have a default value and other are are completely optional. Id is a special attribute used to identify an element or a group of elements. It must be unique for a kind of element. It is optional, but no reference to the corresponding element can be done if it is not defined.

The XML file is split into context tags that are used to isolate IO definition from different codes or different parts of a code. No interference is possible between 2 different contexts. Each context has its own calendar and an associated timestep. In NEMO, we used the following contexts (that can be defined in any order) :

context	description	example
context xios	context containing information for XIOS	`<context id="xios" ...`
context nemo	context containing IO information for NEMO (mother grid when using AGRIF)	`<context id="nemo" ...`
context 1_nemo	context containing IO information for NEMO child grid 1 (when using AGRIF)	`<context id="1_nemo" ...`
context n_nemo	context containing IO information for NEMO child grid n (when using AGRIF)	`<context id="n_nemo" ...`

The xios context contains only 1 tag :

context tag	description	example
variable_definition	define variables needed by XIOS. This can be seen as a kind of namelist for XIOS.	`<variable_definition ...`

Each context tag related to NEMO (mother or child grids) is divided into 5 parts (that can be defined in any order) :

context tag	description	example
field_definition	define all variables that can potentially be outputted	`<field_definition ...`
file_definition	define the netcdf files to be created and the variables they will contain	`<file_definition ...`
axis_definition	define vertical axis	`<axis_definition ...`
domain_definition	define the horizontal grids	`<domain_definition ...`
grid_definition	define the 2D and 3D grids (association of an axis and a domain)	`<grid_definition ...`

Nesting XML files

The XML file can be split in different parts to improve its readability and facilitate its use. The inclusion of XML files into the main XML file can be done through the attribute src :

```
<context src="./nemo_def.xml" />
```

In NEMO, by default, the field and domain definition is done in 2 separate files :

```
NEMOGCM/CONFIG/SHARED/field_def.xml
and
NEMOGCM/CONFIG/SHARED/domain_def.xml
```

that are included in the main iodef.xml file through the following commands :

```
<field_definition src="./field_def.xml" />
<domain_definition src="./domain_def.xml" />
```

Use of inheritance

XML extensively uses the concept of inheritance. XML has a tree based structure with a parent-child oriented relation : all children inherit attributes from parent, but an attribute defined in a child replace the inherited attribute value. Note that the special attribute ”id” is never inherited.

example 1 : Direct inheritance.

```
<field_definition operation="average" >
  <field id="sst"                    />   <!-- averaged      sst -->
  <field id="sss" operation="instant"/>   <!-- instantaneous sss -->
</field_definition>
```

The field ”sst” which is part (or a child) of the field_definition will inherit the value ”average” of the attribute ”operation” from its parent. Note that a child can overwrite the attribute definition inherited from its parents. In the example above, the field ”sss” will for example output instantaneous values instead of average values.

example 2 : Inheritance by reference.

```
<field_definition>
  <field id="sst" long_name="sea surface temperature" />
  <field id="sss" long_name="sea surface salinity"    />
</field_definition>

<file_definition>
  <file id="myfile" output_freq="1d" />
    <field field_ref="sst"                            />  <!-- default def -->
    <field field_ref="sss" long_name="my description" />  <!-- overwrite   -->
  </file>
</file_definition>
```

Inherit (and overwrite, if needed) the attributes of a tag you are refering to.

Use of Groups

Groups can be used for 2 purposes. Firstly, the group can be used to define common attributes to be shared by the elements of the group through the inheritance. In the following example, we define a group of field that will share a common grid ”grid_T_2D”. Note that for the field ”toce”, we overwrite the grid definition inherited from the group by ”grid_T_3D”.

```
<field_group id="grid_T" grid_ref="grid_T_2D">
 <field id="toce" long_name="temperature"             unit="degC" grid_ref="grid_T_3D"/>
 <field id="sst"  long_name="sea surface temperature" unit="degC"                     />
 <field id="sss"  long_name="sea surface salinity"    unit="psu"                      />
 <field id="ssh"  long_name="sea surface height"      unit="m"                        />
      ...
```

Secondly, the group can be used to replace a list of elements. Several examples of groups of fields are proposed at the end of the file `CONFIG/SHARED/field_def.xml`. For example, a short list of the usual variables related to the U grid :

```
<field_group id="groupU" >
 <field field_ref="uoce"  />
 <field field_ref="suoce" />
 <field field_ref="utau"  />
</field_group>
```

that can be directly include in a file through the following syntax :

```
<file id="myfile_U" output_freq="1d" />
 <field_group group_ref="groupU"/>
 <field field_ref="uocetr_eff"  />  <!-- add another field -->
</file>
```

11.2.4 Detailed functionalities

The file `NEMOGCM/CONFIG/ORCA2_LIM/iodef_demo.xml` provides several examples of the use of the new functionalities offered by the XML interface of XIOS.

Define horizontal subdomains

Horizontal subdomains are defined through the attributs zoom_ibegin, zoom_jbegin, zoom_ni, zoom_nj of the tag family domain. It must therefore be done in the domain part of the XML file. For example, in `CONFIG/SHARED/domain_def.xml`, we provide the following example of a definition of a 5 by 5 box with the bottom left corner at point (10,10).

```
<domain_group id="grid_T">
 <domain id="myzoom" zoom_ibegin="10" zoom_jbegin="10" zoom_ni="5" zoom_nj="5" />
```

The use of this subdomain is done through the redefinition of the attribute domain_ref of the tag family field. For example :

```
<file id="myfile_vzoom" output_freq="1d" >
   <field field_ref="toce" domain_ref="myzoom"/>
</file>
```

Moorings are seen as an extrem case corresponding to a 1 by 1 subdomain. The Equatorial section, the TAO, RAMA and PIRATA moorings are alredy registered in the code and can therefore be outputted without taking care of their (i,j) position in the grid. These predefined domains can be activated by the use of specific domain_ref : ”EqT”, ”EqU” or ”EqW” for the equatorial sections and the mooring position for TAO, RAMA and PIRATA followed by ”T” (for example : ”8s137eT”, ”1.5s80.5eT” ...)

```
<file id="myfile_vzoom" output_freq="1d" >
   <field field_ref="toce" domain_ref="0n180wT"/>
</file>
```

Note that if the domain decomposition used in XIOS cuts the subdomain in several parts and if you use the ”multiple_file” type for your output files, you will endup with several files you will need to rebuild using unprovided tools (like ncpdq and ncrcat, see nco manual). We are therefore advising to use the ”one_file” type in this case.

Define vertical zooms

Vertical zooms are defined through the attributs zoom_begin and zoom_end of the tag family axis. It must therefore be done in the axis part of the XML file. For example, in NEMOGCM/CONFIG/ORCA2_LIM/iodef_demo.xml, we provide the following example :

```
<axis_group id="deptht" long_name="Vertical T levels" unit="m" positive="down" >
   <axis id="deptht" />
   <axis id="deptht_myzoom" zoom_begin="1" zoom_end="10" />
```

The use of this vertical zoom is done through the redefinition of the attribute axis_ref of the tag family field. For example :

```
<file id="myfile_hzoom" output_freq="1d" >
   <field field_ref="toce" axis_ref="deptht_myzoom"/>
</file>
```

Control of the output file names

The output file names are defined by the attributs ”name” and ”name_suffix” of the tag family file. for example :

```
<file_group id="1d" output_freq="1d" name="myfile_1d" >
   <file id="myfileA" name_suffix="_AAA" > <!-- will create file "myfile_1d_AAA"  -->
      ...
   </file>
   <file id="myfileB" name_suffix="_BBB" > <!-- will create file "myfile_1d_BBB" -->
      ...
   </file>
</file_group>
```

However it is often very convienent to define the file name with the name of the experience, the output file frequency and the date of the beginning and the end of the simulation (which are informations stored either in the namelist or in the XML file). To do so, we added the following rule : if the id of the tag file is ”fileN”(where N = 1 to 99) or one of the predefined section or mooring (see next subsection), the following part of the name and the name_suffix (that can be inherited) will be automatically replaced by :

placeholder string	automatically replaced by
@expname@	the experience name (from cn_exp in the namelist)
@freq@	output frequency (from attribute output_freq)
@startdate@	starting date of the simulation (from nn_date0 in the restart or the namelist). `yyyymmdd` format
@startdatefull@	starting date of the simulation (from nn_date0 in the restart or the namelist). `yyyymmdd_hh:mm:ss` format
@enddate@	ending date of the simulation (from nn_date0 and nn_itend in the namelist). `yyyymmdd` format
@enddatefull@	ending date of the simulation (from nn_date0 and nn_itend in the namelist). `yyyymmdd_hh:mm:ss` format

For example,

```
<file id="myfile_hzoom" name="myfile_@expname@_@startdate@_freq@freq@" output_freq="1d" >
```

with the namelist :

```
cn_exp      =  "ORCA2"
nn_date0    =  19891231
ln_rstart   = .false.
```

will give the following file name radical :

```
myfile_ORCA2_19891231_freq1d
```

Other controls of the xml attributes from NEMO

The values of some attributes are defined by subroutine calls within NEMO (calls to iom_set_domain_attr, iom_set_axis_attr and iom_set_field_attr in iom.F90). Any definition given in the xml file will be overwritten. By convention, these attributes are defined to "auto" (for string) or "0000" (for integer) in the xml file (but this is not necessary).

Here is the list of these attributes :

tag ids affected by automatic definition of some of their attributes	name attribute	attribute value
field_definition	freq_op	*rn_rdt*
SBC	freq_op	*rn_rdt* $\times$ *nn_fsbc*
ptrc_T	freq_op	*rn_rdt* $\times$ *nn_dttrc*
diad_T	freq_op	*rn_rdt* $\times$ *nn_dttrc*
EqT, EqU, EqW	jbegin, ni, name_suffix	according to the grid
TAO, RAMA and PIRATA moorings	zoom_ibegin, zoom_jbegin, name_suffix	according to the grid

11.2.5 XML reference tables

Tag list

tag name	description	accepted attribute	child of	parent of
simulation	this tag is the root tag which encapsulates all the content of the xml file	none	none	context
context	encapsulates parts of the xml file dedicated to different codes or different parts of a code	id ("xios", "nemo" or "n_nemo" for the nth AGRIF zoom), src, time_origin	simulation	all root tags : ... _definition
field_definition	encapsulates the definition of all the fields that can potentially be outputted	axis_ref, default_value, domain_ref, enabled, grid_ref, level, operation, prec, src	context	field or field_group

tag name	description	accepted attribute	child of	parent of
field_group	encapsulates a group of fields	axis_ref, default_value, domain_ref, enabled, group_ref, grid_ref, id, level, operation, prec, src	field_definition, field_group, file	field or field_group
field	define a specific field	axis_ref, default_value, domain_ref, enabled, field_ref, grid_ref, id, level, long_name, name, operation, prec, standard_name, unit	field_definition, field_group, file	none
file_definition	encapsulates the definition of all the files that will be outputted	enabled, min_digits, name, name_suffix, output_level, split_format, split_freq, sync_freq, type, src	context	file or file_group
file_group	encapsulates a group of files that will be outputted	enabled, description, id, min_digits, name, name_suffix, output_freq, output_level, split_format, split_freq, sync_freq, type, src	file_definition, file_group	file or file_group
file	define the contents of a file to be outputted	enabled, description, id, min_digits, name, name_suffix, output_freq, output_level, split_format, split_freq, sync_freq, type, src	file_definition, file_group	field
axis_definition	define all the vertical axis potentially used by the variables	src	context	axis_group, axis
axis_group	encapsulates a group of vertical axis	id, lon_name, positive, src, standard_name, unit, zoom_begin, zoom_end, zoom_size	axis_definition, axis_group	axis_group, axis

tag name	description	accepted attribute	child of	parent of
axis	define a vertical axis	id, lon_name, positive, src, standard_name, unit, zoom_begin, zoom_end, zoom_size	axis_definition, axis_group	none
domain_-definition	define all the horizontal domains potentially used by the variables	src	context	domain_-group, domain
domain_group	encapsulates a group of horizontal domains	id, lon_name, src, zoom_ibegin, zoom_jbegin, zoom_ni, zoom_nj	domain_-definition, domain_group	domain_-group, domain
domain	define an horizontal domain	id, lon_name, src, zoom_ibegin, zoom_jbegin, zoom_ni, zoom_nj	domain_-definition, domain_group	none
grid_definition	define all the grid (association of a domain and/or an axis) potentially used by the variables	src	context	grid_group, grid
grid_group	encapsulates a group of grids	id, domain_ref, axis_ref	grid_definition, grid_group	grid_group, grid
grid	define a grid	id, domain_ref, axis_ref	grid_definition, grid_group	none

Attributes list

attribute name	description	example	accepted by
axis_ref	refers to the id of a vertical axis	axis_ref="deptht"	field, grid families
enabled	switch on/off the output of a field or a file	enabled=".TRUE."	field, file families
default_value	missing_value definition	default_value="1.e20"	field family
description	just for information, not used	description="ocean T grid variables"	all tags
domain_ref	refers to the id of a domain	domain_ref="grid_T"	field or grid families
field_ref	id of the field we want to add in a file	field_ref="toce"	field

attribute name	description	example	accepted by
grid_ref	refers to the id of a grid	grid_ref=”grid_T_2D”	field family
group_ref	refer to a group of variables	group_ref=”mooring”	field_group
id	allow to identify a tag	id=”nemo”	accepted by all tags except simulation
level	output priority of a field : 0 (high) to 10 (low)	level=”1”	field family
long_name	define the long_name attribute in the NetCDF file	long_name=”Vertical T levels”	field
min_digits	specify the minimum of digits used in the core number in the name of the NetCDF file	min_digits=”4”	file family
name	name of a variable or a file. If the name of a file is undefined, its id is used as a name	name=”tos”	field or file families
name_suffix	suffix to be inserted after the name and before the cpu number and the ”.nc” termination of a file	name_suffix=”_myzoom”	file family
operation	type of temporal operation : average, accumulate, instantaneous, min, max and once	operation=”average”	field family
output_freq	operation frequency. units can be ts (timestep), y, mo, d, h, mi, s.	output_freq=”1d12h”	field family
output_level	output priority of variables in a file : 0 (high) to 10 (low). All variables listed in the file with a level smaller or equal to output_level will be output. Other variables won’t be output even if they are listed in the file.	output_level=”10”	file family

attribute name	description	example	accepted by
positive	convention used for the orientation of vertival axis (positive downward in *NEMO*).	positive=”down”	axis family
prec	output precision : real 4 or real 8	prec=”4”	field family
split_format	date format used in the name of splitted output files. can be spcified using the following syntaxe : %y, %mo, %d, %h %mi and %s	split_format= ”%yy%mom%dd”	file family
split_freq	split output files frequency. units can be ts (timestep), y, mo, d, h, mi, s.	split_freq=”1mo”	file family
src	allow to include a file	src=”./field_def.xml”	accepted by all tags except simulation
standard_name	define the standard_name attribute in the NetCDF file	standard_name= ”Eastward_Sea_Ice_Transport”	field
sync_freq	NetCDF file synchronization frequency (update of the time_counter). units can be ts (timestep), y, mo, d, h, mi, s.	sync_freq=”10d”	file family
attribute name	description	example	accepted by
time_origin	specify the origin of the time counter	time_origin=”1900-01-01 00 :00 :00”	context
type (1)	specify if the output files must be split (multiple_file) or not (one_file)	type=”multiple_file”	file familly
type (2)	define the type of a variable tag	type=”boolean”	variable
unit	unit of a variable or the vertical axis	unit=”m”	field and axis families

attribute name	description	example	accepted by
zoom_ibegin	starting point along x direction of the zoom. Automatically defined for TAO/RAMA/PIRATA moorings	zoom_ibegin=”1”	domain family
zoom_jbegin	starting point along y direction of the zoom. Automatically defined for TAO/RAMA/PIRATA moorings	zoom_jbegin=”1”	domain family
zoom_ni	zoom extent along x direction	zoom_ni=”1”	domain family
zoom_nj	zoom extent along y direction	zoom_nj=”1”	domain family

11.3 NetCDF4 Support (key_netcdf4)

Since version 3.3, support for NetCDF4 chunking and (loss-less) compression has been included. These options build on the standard NetCDF output and allow the user control over the size of the chunks via namelist settings. Chunking and compression can lead to significant reductions in file sizes for a small runtime overhead. For a fuller discussion on chunking and other performance issues the reader is referred to the NetCDF4 documentation found here.

The new features are only available when the code has been linked with a NetCDF4 library (version 4.1 onwards, recommended) which has been built with HDF5 support (version 1.8.4 onwards, recommended). Datasets created with chunking and compression are not backwards compatible with NetCDF3 ”classic” format but most analysis codes can be relinked simply with the new libraries and will then read both NetCDF3 and NetCDF4 files. NEMO executables linked with NetCDF4 libraries can be made to produce NetCDF3 files by setting the *ln_nc4zip* logical to false in the *namnc4* namelist :

```
!-----------------------------------------------------------------------
&namnc4        !   netcdf4 chunking and compression settings            ("key_netcdf4")
!-----------------------------------------------------------------------
   nn_nchunks_i=   4       !  number of chunks in i-dimension
   nn_nchunks_j=   4       !  number of chunks in j-dimension
   nn_nchunks_k=   31      !  number of chunks in k-dimension
                           !  setting nn_nchunks_k = jpk will give a chunk size of 1 in the vertical which
                           !  is optimal for postprocessing which works exclusively with horizontal slabs
   ln_nc4zip   = .true.    !  (T) use netcdf4 chunking and compression
                           !  (F) ignore chunking information and produce netcdf3-compatible files
/
```

If **key_netcdf4** has not been defined, these namelist parameters are not read. In this case, *ln_nc4zip* is set false and dummy routines for a few NetCDF4-specific functions are defined. These functions will not be used but need to be included so that compilation is possible with NetCDF3 libraries.