NEMSIO PACKAGE USER GUIDE

Jun Wang

I) INTRODUCTION

The NEMSIO package is developed to support the input/output for NEMS project. The basic functions it provides are to read and write data sets for all the NEMS applications. The current version of NEMSIO could handle binary and GRIB1 data. More data formats such as GRIB2 or NETCDF will be added in the future. Now NEMSIO has both serial version and parallel version which uses MPI-II parallel I/O.

NEMSIO file consists of meta data and data fields. Meta data precedes all the data fields. NEMSIO meta data has two parts, standard meta data and user defined meta data. The standard meta data was designed to contain all the information needed by GSI and NEMS systems. For general NEMSIO files only the first two meta data records are required, therefore the standard meta can be minimized by only outputting first several meta data records. The total number of standard meta data records is controlled by nmeta. Users can defined their own meta data, which, if exist, are present after the standard meta data.

II) META DATA STRUCTURE

1. First two standard meta data records

- meta1 is fixed to 48 bytes in length and it contains variables listed in the table below:

variable name	***kind***	***meaning***
gtype	character(8)	NEMSIO file indicator
gdatatype	character(8)	data format
modelname	character(8)	model name
version	integer	version number
nmeta	integer	number of standard meta data records
lmeta	integer	length of second standard meta records
reserve(3)	integer	reserved

- meta2 contains some integers, real numbers and a logic variable. All the variables in meta2 have a default value -9999 for integer, -9999. for real number or false for logical variables before the meta data is initialized.

variable name	*kind*	*meaning*
nrec	integer	total data fields in data part
idate(7)	integer	starting date contains: year,month,day,hour,minute,second (numerator part), second(dominator part)
nfday	integer	forecast days
nfhour	integer	forecast hours
nfminute	integer	forecast minutes
nfsecondn	ingeger	forecast seconds(numerator)
nfseconds	integer	forecast seconds(dominator)
dimx	integer	dimension in x-direction(not including halo)
dimy	integer	dimension in y-direction(not including halo)
dimz	integer	dimension in z-direction
nframe	integer	dimension of halo, not zero if output data has halo
nsoil	integer	number of soil layers
ntrac	integer	number of tracers
jcap	integer	spectral truncation wave number
ncldt	integer	number of cloud types
idvc	integer	vertical coordinate id
idsl	integer	semi-Lagrangian id
idvm	integer	mass variable id
idrt	integer	grid identifier
rlon_min	real (4)	minimal longitude of regional domain
rlon_max	real (4)	maximal longitude of regional domain
rlat_min	real (4)	minimal latitude of regional domain
rlat_max	real (4)	maximal latitude of regional domain

variable name	*kind*	*meaning*
extrameta	logical	extra user defined meta data flag

2.Other standard meta data records include:

name	*kind*	*dimension*	*meaning*
recname	character(16)	nrec	name for each data field record
reclevtyp	character(16)	nrec	level type for each data field record
reclev	integer	nrec	level for each data field record
lat	real (4)	fieldsize	latitude
lon	real (4)	fieldsize	longitude
dx	real(4)	fieldsize	grid length in longitude
dy	real(4)	fieldsize	grid length in latitude
Cpi	real(4)	ntrac	constant-pressure specific heat capacity for tracers
Ri	real(4)	ntrac	specific gas constant for tracers

fieldsize=(dimx+2*nframe)*(dimy+2*nframe)

Users could omit other meta records by setting nmeta. to 2. Setting nmeta to 3 will keep only recname in the meta data. Setting nmeta to 4 will keep recname and reclevtyp in the meta data, etc.

3. User defined meta data
   This part of meta data contains all the extra meta data users would like to output in the file. The logical variable extrameta in the second standard meta record indicates if there exists any user defined meta data.

If extrameta is true, a metadata record containing the numbers of different kind extra meta data exits after the standard meta data. The possible extra meta data is listed below,

variable	*kind*	*dimension*	*meaning*
variname	character(16)	nmetavari	names of user defined integer variables
varival	integer	nmetavari	values of user defined integer variables

variable	*kind*	*dimension*	*meaning*
varrname	character(16)	nmetavarr	names of user defined real variables
varrval	real(4)	nmetavarr	values of user defined real variables
varlname	character(16)	nmetavarl	names of user defined logical variables
varlval	logical	nmetavarl	values of user defined logical variables
varcname	character(16)	nmetavarc	names of user defined character variables
varcval	character(16)	nmetavarc	values of user defined character variables
aryiname	character(16)	nmetaaryi	names of user defined integer arrays
aryilen	integer	nmetaaryi	lengths of user defined integer arrays
aryival	integer	(maxval(aryilen), nmetaaryi)	values of user defined integer arrays
aryrname	character(16)	nmetaaryr	names of user defined real arrays
aryrlen	integer	nmetaaryr	lengths of user defined real arrays
aryrval	real(4)	(maxval(aryrlen), nmetaaryr)	values of user defined real arrays
arylname	character(16)	nmetaaryl	names of user defined logical arrays
aryllen	integer	nmetaaryl	lengths of user defined logical arrays
arylval	logical	(maxval(aryllen), nmetaaryl)	values of user defined logical arrays
arycname	character(16)	nmetaaryc	names of user defined character arrays
aryclen	integer	nmetaaryc	lengths of user defined character arrays
arycval	character(16)	(maxval(aryclen), nmetaaryc)	values of user defined character arrays

where nmetavari,nmetavarr,nmetavarl and nmetavarc specify the number of integer, real, logical and character variables in extra meta data.
nmetaaryi,nmetaaryr,nmetaaryl, and nmetaaryc specify the number of 1-D integer, real, logical and character arrays in extra mete data.

If any of these numbers is greater than 0, corresponding meta data record including variables/array names, length(for array only) and values will be present in the file.
For example, if nmetavari>0, then two meta data records variname and varival will be

shown in the file. Similarly, if nmetaaryi>0, then aryiname, aryilen and aryival will be in the extra meta data part. All the values of 1D arrays are hold in the 2D array according to their kind. For example, aryival holds all 1D integer arrays, the first dimension of varival is the maximal length of all the 1D integer arrays, the second dimension is the total number of integer arrays. However, in the data file, for each 1D array, it will be written in the file in its own length, the value holder array such as aryival will not be written out.

III) DATA FIELDS

NEMSIO supports binary and GRIB1 data format, the default is GRIB1 data. For binary data, the gdatatype has to be set to either 'bin4' for 4 bytes binary data files or 'bin8' for 8 bytes binary data files. The standard meta data records: recname,reclevtyp and lev, if exists, record the name, level type and level for each data field. The data fields can be read/written/updated by using the data field record number or by given the name, level type and level of a data field. For binary data files, users could get/write/update data by only using the variables name, for GRIB1 data, level type and level of a data field are required along with the name. Details on this will be shown in part III application interfaces.

IV) NEMSIO INTERFACES

Data type:

NEMSIO defined a data type called nemsio_gfile, which contains all the meta data information. All the variables in meta1 and meta2, other standard meta data, and user defined meta data are the components of nemsio_gfile, besides that, nesmio_gfile holds some file information such as file name and the action to the file (READ/WRITE). For paralle version, MPI information such as MPI communicator and lead _task is in nemsio_gfile too. All the components of nemsio_gfile has private attribute, users have to call subroutines nemsio_getfilehead or nemsio_getheadvar to get the values of the components in nemsio_gfile.

1. Serial version

NEMSIO provides various interfaces for users to access the NEMSIO data files. The most commonly used interfaces and their functions are described as follows.

1.1 Initialization and finalization

Users are required to call nemsio_init before they call any NEMSIO functions and to call nemsio_finalize to end the use of NEMSIO functions. Here are examples on how to use them.

```
call nemsio_init(iret)
     iret output integer return code
call nemsio_finalize(iret)
```

```
      iret output integer return code
```

1.2 Open nemsio file

nemsio_open must be called when users are ready to read/write/update data. For READ or RDWR (RDWR should be used when users need to update data fields), no optional arguments are needed since all information will be obtained from the file, NEMSIO data type nemsio_gfile will contain all the meta data information after nemsio_open. For WRITE, users need to specify some optional arguments such as dimx,dimy,dimz,idate,and nmeta, nrec, otherwise default initialization will be called to set up the values for thoese meta data, and for default the standard meta data will be set up, which contains 12 standard meta data records including meta1, meta2, recname,reclevtyp,reclev,vcoord, lat,lon,dx,dy,cpi, and ri. Users can set optional argument nmeta to output fewer standard meta data, however as specified before, the first two meta data must be written out in any cases. Users also need to set extrameta in nemsio_open if they want to output their own meta data.

```
call nemsio_open(gfile,gfname,gaction,optargs,iret)
     gfile input/output, nemsio_gfile file meta
     gfname input, character, file name
     gaction input operations to the file:'READ', 'WRITE',
'RDWR'
     optargs input optional arguments
     iret output integer return code

Optargs includes:
      gdatatype,version,
      nmeta,lmeta,modelname,nrec,idate,nfday,nfhour,
      nfminute,nfsecondn,nfsecondd,
      dimx,dimy,dimz,nframe,nsoil,ntrac,jcap,ncldt,idvc,ids
      l,idvm,idrt,
      rlon_min,rlon_max,rlat_min,rlat_max,extrameta,
      recname,reclevtyp,reclev,vcoord,lat,lon,dx,dy,cpi,ri,
      nmetavari,nmetavarr,nmetavarl,nmetavarc,
      nmetaaryi,nmetaaryr,nmetaaryl,nmetaaryc,
      variname,varival,varrname,varrval,varlname,varlval,va
      rcname,varcval,
      aryiname,aryilen,aryival,aryrname,aryrlen,aryrval,
      arylname,aryllen,arylval,arycname,aryclen,arycval
```

1.3 Close nemsio file

nemsio_close closes the data file, de-allocates all the meta data arrays and sets meta data variables into default values in the file meta data holder.

```
call nemsio_close(gfile,iret)
     gfile input nemsio_gfile file meta
     iret output integer return code
```

After nemsio_close, all the variables in nemsio data type gfile have a default value -9999,

or false for logical, and arrays are all de-allocated.

1.4 Read data fields

NEMSIO has two read subroutines to read out a data field. One is nemsio_readrec, which will read data field from file by the order of the data fields. The other is nemsio_recrecv, which allows users to read out the data field by giving the data field's name, level type and level; levtyp and level are optional for binary data. Only when the recname, possiblely reclevtyp and reclev, are set in the standard meta data, can the second function be used. level type and level are required to exist in the meta data when levtyp and lev are present at the nemsio_recrecv argument list. The output data can either be 4 byte real number or 8 byte double precision real number no matter whether the data file is 4 byte binary, 8 byte binary file or GRIB 1 data file.

```
  call nemsio_readrec(gfile,jrec,data,iret)
     gfile     input/output gfsio_gfile file metadata
     jrec      input integer the j-th record in the data
fields
     data      output real or double precision data(:)
     iret      output integer return code

 call nemsio_readrecv(gfile,name,levtyp,lev,data,iret)
     gfile     input/output gfsio_gfile file metadata
     name      input, character(8) name of data field
     levtyp    input, character(16) level type of data
field
     lev       input, integer level that the data field is
at
     data      output real or double precision data(:)
     iret      output integer return code
```

1.5 Write data fields

Similar to READ, NEMSIO has two write functions. One is nemsio_writerec, which will write data field by the order of data field, the other is nemsio_writerecv, which allows users to write out data by giving the data field's name, levtyp and lev; like READ, the recname,reclevtyp and reclev have to be written out in the standard meta data for this function to be called. Also the input data can be either 4 byte real or 8 byte double precision real number regardless of the data type of the file, some inner transform will write out the data sets according to the data type of the file.

```
 call nemsio_writerec(gfile,jrec,data,iret)
     gfile     input/output nemsio_gfile file metadata
     jrec      input integer the j-th record in the data
fields
     data      input real or double precision data(:)
     iret      output integer return code

 call nemsio_writerecv(gfile,name,levtyp,lev,data,iret)
     gfile     input/output nemsio_gfile file metadata
```

```
     name      input character(8) name of data field
     levtyp    input character(16) level type of data field
     lev       input integer level that the data field is at
     data      input real or double precision data(:)
     iret      output integer return code
```

When reading or writing GRIB1 data using NEMSIO, users who use w3_4 rather than w3_d when compiling their code will need to use different interfaces. This is because nemsio has no knowledge which w3 library (w3_4 or w3_d) is used. Below is a list of interfaces to be used instead.

```
 call nemsio_readrecw34(gfile,jrec,data,iret)
 call nemsio_writerecw34(gfile,jrec,data,iret)
 call nemsio_readrecvw34(gfile,name,levtyp,lev,data,iret)
 call nemsio_writerecvw34(gfile,name,levtyp,lev,data,iret)
```

The argument lists are same as in these subroutines using w3_d in compile.

1.6. Get meta data information

Since the nemsio_gfile has private attribute for its meta data, all meta data has to be obtained through nemsio_getfilehead or nemsio_getheadvar. Users can use nemsio_getfilehead to get all the meta data as variables listed in above tables; or they can get any single meta data variables including those in the user defined meta data arrays.

```
  call nemsio_getfilehead(gfile,optargs,iret)
     gfile     input gfsio_gfile file metadata
     optargs   output optional arguments
     iret      output integer return code

Optargs includes:
      gdatatype,version,
      nmeta,lmeta,modelname,nrec,idate,nfday,nfhour,
nfminute,nfsecondn,nfsecondd,
dimx,dimy,dimz,nframe,nsoil,ntrac,jcap,ncldt,idvc,idsl,idvm
,idrt,
rlon_min,rlon_max,rlat_min,rlat_max,extrameta,
      recname,reclevtyp,reclev,vcoord,lat,lon,dx,dy,cpi,ri,
      nmetavari,nmetavarr,nmetavarl,nmetavarc,
      nmetaaryi,nmetaaryr,nmetaaryl,nmetaaryc,
variname,varival,varrname,varrval,varlname,varlval,varcname
,varcval,
      aryiname,aryilen,aryival,aryrname,aryrlen,aryrval,
      arylname,aryllen,arylval,arycname,aryclen,arycval


  call nemsio_getheadvar(gfile,argname,argval,iret)
     gfile     input gfsio_gfile file meta
```

```
     argname   input argument name
     argval    output argument value
     iret      output integer return code
```

where argname can be any meta data variable name, including names in variname, varrname, varlname, varcname, aryiname, aryrname, arylname, and arycname. Users can call nemsio_getfilehead to get variname (or other array names) then call nemsio_getheadvar to get the value of a specific variable or array.

1.7. Get data field information

User can get the name, level type and level of each data field by calling nemsio_getrechead if they need this information. For example, users can obtain the values of data fields by calling nemsio_readrecv or nemsio_writerecv with those information. As specified before, meta data records recname,reclevtyp and reclev must be present in meta data in order to use this interface.

```
  call nemsio_getrechead(gfile,jrec,name,levtyp,lev,iret)
     gfile     input/output nemsio_gfile file metadata
     jrec      input integer the j-th record in the data
fields
     name      input character(8) name of data field
     levtyp    input character(16) level type of data field
     lev       input integer level that the data field is at
     iret      output integer return code
```

1.8. Set meta data information

In some special occasions when the meta data information such as lat, lon dx and dy can't be obtained at the time nemsio file is opened, nemsio allows users to set these four meta data later. Users need to be careful with the dimension of those four variables since it may destroy the data in the file.

```
  call nemsio_setfilehead(gfile,optarg,iret)
     gfile     input/output nemsio_gfile file metadata
     optargs   input optional arguments
     iret      output integer return code
```

optargs can only be lat, lon, dx and dy.

2. Parallel version

In parallel version of NEMSIO, MPI-II parallel I/O is used for MPI I/O. For meta data, the read-broadcast method is used, so one processor will read the header information and broadcast it to all the other processors. For data fields, parallel I/O is used so all the processors will read/write data simultaneously. Users will decided which portion of data will be accessed by each processor. Therefore the argument lists for interfaces of opening files, reading/writing/updating data fields are different from the serial version.

2.1 Following are the interfaces that are same as serial version.

```
call nemsio_init(iret)
```

```
call nemsio_finalize(iret)
call nemsio_close(gfile,iret)
call nemsio_getfilehead(gfile,optargs,iret)
call nemsio_getheadvar (gfile, argname, argval,iret)
call nemsio_getrechead(gfile, jrec, vname, vlevtyp, vlev,iret)
call nemsio_setfilehead(gfile,optarg,iret)
```

2.2 Open nemsio file

To open a nemsio file, MPI information such as MPI communicator is needed. Also a lead task who will read/write meta data is needed. The interface for opening files is:

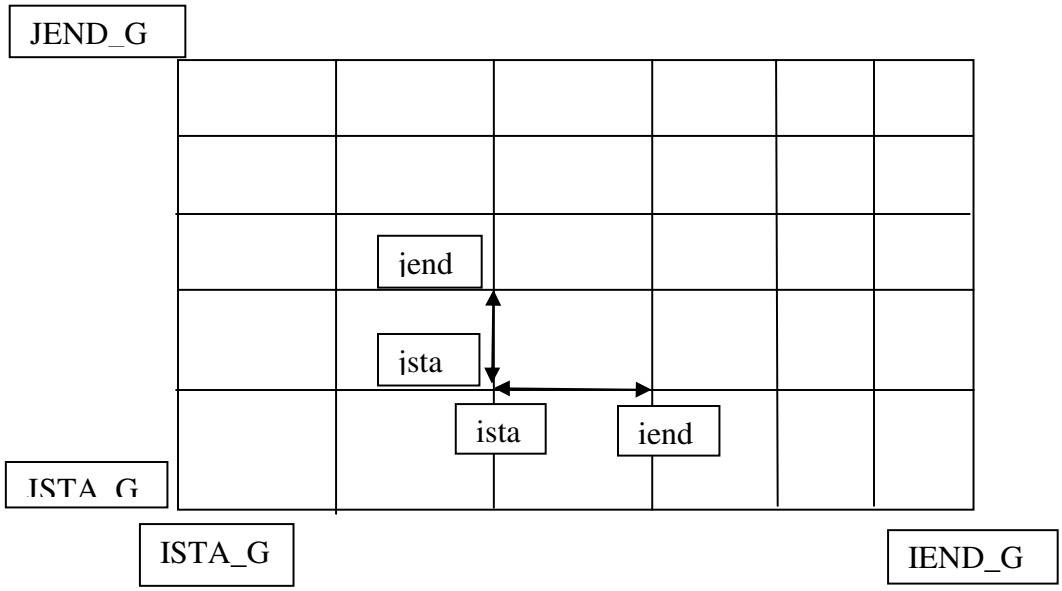
```
call nemsio_open(gfile,gfname,gaction,mpi_comm,
optargs,iret)
     gfile input/output nemsio_gfile nemsio_gfile file meta
     gfname input character file name
     gaction input character, allowed
operations:'READ','WRITE','RDWR'
     mpi_comm input integer MPI communicator

     optargs input optional arguments
     iret output integer return code
```

optargs are the same as serial version.

2.3 Read data field

To read data, users need to provide information on what portion of data each processor deals with. A map array describing the location of each element of the data array for each processor is all that needed. In NEMSIO, a simplified implementation is to specify four arguments ista,iend,jsta,jend. Those four variables will specify the subdomain of data array for each processor located on the whole data field domain. The figure 1 shows an example. A 2D data field (global array)is shown in dimension(ISTA_G:IEND_G, (JSTA_G:JEND_G), each small rectangle box specifies the portion of data (local array) (ISTA:IEND,JSTA:JEND) for each processor. Ista and iend are the staring and ending points in X-direction, jsta and jend are the staring and ending points in Y-direction.



The interface for reading by using the data field record number is:

```
call
nemsio_readrec(gfile,ista,iend,jsta,jend,jrec,data,iret)
ista,iend,jsta,jend: input dimension of local array
```

If calling by data field name, level type and level, the interface would be:

```
call
nemsio_readrecv(gfile,ista,iend,jsta,jend,jrec,name,levtyp,
lev,data,iret)
ista,iend,jsta,jend: input dimension of local array
```

Because the parallel I/O works more efficiently for large size data size, getting out all the data fields by one reading call will save time compared to reading one data field at a time. Therefore the interface to reading all data fields in a file is:

```
call
nemsio_denseread(gfile,ista,iend,jsta,jend,jrec,data,iret)
ista,iend,jsta,jend: input dimension of local array
data(:): data((iend-ista+1)*(jend-jsta+1)*nrec)
```

2.4 Write data field

The interfaces for writing data fields are similar to those for reading data field.

```
call
nemsio_writerec(gfile,ista,iend,jsta,jend,jrec,data,iret)
ista,iend,jsta,jend: input dimension of local array


call
```

```
nemsio_writerecv(gfile,ista,iend,jsta,jend,jrec,name,levtyp
,lev,data,iret)
ista,iend,jsta,jend: input dimension of local array

call
nemsio_densewrite(gfile,ista,iend,jsta,jend,jrec,name,levty
p,lev,data,iret)
ista,iend,jsta,jend: input dimension of local array
data(:): data((iend-ista+1)*(jend-jsta+1)*nrec)
```

V) EXAMPLES

1. Eg 1. read a nemsio file (serial version):

```
program main

 use nemsio_module
 implicit none
!
  type(nemsio_gfile) :: gfile
  integer im,jm,jrec,nframe,nrec,fieldsize,vlev,iret
  character(8) vname
  character(16) vlevtyp
  character(255) cin
  real(4),allocatable :: fis(:)
!
!--- Initialize
call nemsio_init(iret=iret)
!
!--- open a nemsio file
 call getarg(1,cin)
 call nemsio_open(gfile,trim(cin),'READ',iret=iret)
!
!---get dimension
call nemsio_getfilehead(gfile,iret=iret,dimx=im,dimy=jm, nframe=nframe)
!
!---allocate array
  fieldsize=(im+2*nframe)*(jm+2*nframe)
  allocate(fis(fieldsize))
!
!--you could get the name , levtyp, and level of j-th recrod data field by calling:
  Jrec=1
   call nemsio_getrechead(gfile,jrec,vname,vlevtyp,vlev,iret=iret)
!
!--- get one data field out by data field record number
   call nemsio_readrec(gfile,jrec,fis(:),iret=iret)
!
```

```
!--- get one data field out by giving field name, levtyp and level
  call nemsio_readrecv(gfile,'hgt','sfc',1,fis(:),iret=iret)
!
!--- close the nemsio file
   call nemsio_close(gfile,iret=iret)
!
!--- finalize
  call nemsio_finalize()
!
   end
```

2. Eg 2. write a nemsio file (1) (serial version)

```
!-------------set up nemsio --------------------------
   call nemsio_init(iret=iret)
   print *,'nemsio_init, iret=',iret
!
!--- open gfile for writing, the default meta data setting
for model nmmb contains all the meta data information in
NMMB run history plain binary file.

 call
nemsio_open(gfilew,trim(cout),'write',iret,modelname="NMMB",
gdatatype="bin4", &

idate=idate,nfhour=nfhour,nfminute=nfminute,nfsecondn=nfseco
ndn, &

nfsecondd=nfsecondd,dimx=im,dimy=jm,dimz=lm,nframe=nframe,ns
oil=nsoil,&

ntrac=3,ncldt=1,rlon_min=minval(glon1d),rlon_max=maxval(glon
1d), rlat_max=maxval(glat1d), &

rlat_min=minval(glat1d),extrameta=.true.,nmetavari=10,nmetav
arr=12,nmetavarl=2,&
    nmetaaryi=1,
nmetaaryr=7,variname=variname,varival=varival,varrname=varrn
ame, &

varrval=varrval,varlname=varlname,varlval=varlval,aryiname=a
ryiname,aryilen=aryilen, &

aryival=aryival,aryrname=aryrname,aryrlen=aryrlen,aryrval=ar
yrval, &
    lat=glat1d,lon=glon1d,dx=dx,dy=dy)

!
!--- write out data field
   call
nemsio_writerec(gfilew,jrec,datatmp(:,jrec),iret=iret)
!
```

```
!--- write data field by names
  call nemsio_writerecv(gfilew2,'tmp','mid
layer',L,tmp(:,L), iret=iret)
!
!--- close nemsio file
  call nemsio_close(gfile,iret=iret)
!
!--- finilize
  call nemsio_finalize()
```

3. Eg 3. write a nemsio file (2),
This nemsio file just has first two meta data records. User can't call nemsio_readrecv, or nemsio_writerecv since not enough information is provide in meta data.

```
!--- prepare meta data
 cin='nemsio_2meta'
 nmeta=2
 nrec=60
 idate(1:7)=(/2007,05,18,0,0,0,100/)
 im=231;jm=141;lm=60;nframe=0
!
!--- open nemsio file for write
 call nemsio_open(gfilem2,trim(cin),'write',modelname='NMMB',gdatatype='bin4', &
      idate=idate,dimx=im,dimy=jm,dimz=lm, &
      nmeta=nmeta,nrec=nrec,iret=iret)
   print *,'after open write, iret=',iret
!
!--- allocate array and set data field for write
 fieldsize=(im+2*nframe)*(jm+2*nframe)
 allocate(tmp(fieldsize,nrec),fis(fieldsize))
 fis(1:fieldsize)=1000.
!
!--- write 2 fields: tmp and hgt
 do jrec=1,lm
   tmp(1:fieldsize,jrec)=jrec
   call nemsio_writerec(gfilem2,jrec,tmp(:,jrec),iret=iret)
   print *,'after write,jrec=',jrec,'iret=',iret
  enddo
!
!--- close nemsio file
  call nemsio_close(gfilem2,iret=iret)
```

4. Eg 4. get user defined meta data
If users don't know which variables are in the meta data part of the file, they can call nemsio_getfilehead

```
  call
nemsio_getfilehead(gfile,iret=iret,nmetavari=nmetavari,nmeta
varr=nmetavarr, &
```

```
nmetavarl=nmetavarl,nmetaaryi=nmetaaryi,nmetaaryr=nmetaaryr)


allocate(variname(nmetavari),varrname(nmetavarr),varlname(nm
etavarl) )
      allocate(aryiname(nmetaaryi),aryrname(nmetaaryr))
      allocate(aryilen(nmetaaryi),aryrlen(nmetaaryr))

  call
neMsio_getfilehead(gfile,iret=iret,variname=variname,varrnam
e=varrname, &

varlname=varlname,aryiname=aryiname,aryrname=aryrname,aryile
n=aryilen,aryrlen=aryrlen)
```

Then they can call:

```
  call nemsio_getheadvar(gfile,'run',run,iret)

  call nemsio_getheadvar(gfile,'DSG1',dsg1,iret)
```

5.Read data field (MPI version)
This example gives an example on how to read a NMMB nemsio data sets. It also shows how to set up the dimensions of local arrays.

```
call nemsio_init(ierr)
!
cin='nemsiowrt'
  root=0
  print *,'before call nemsio_open'
  call nemsio_open(gfile,trim(cin),'READ',mpi_comm_world,
iret=iret)
!
  call nemsio_getfilehead(gfile,iret=iret,nrec=nrec,dimx=im,
nframe=nframe,dimy=jm,dimz=lm,nsoil=nsoil, nrec=nrec)
!
!-get fieldsize
    im2=im+2*nframe
    jm2=jm+2*nframe
    fieldsize=im2*jm2
!
!-set up the subdomain
    inum_base=im2/inpes
    in_remain=im2-inpes*inum_base
    imype=mod(mype,inpes)
    iadd=1
    if(imype>=in_remain) iadd=0
    if(imype<=in_remain) then
      ista=(inum_base+1)*imype+1
    else
      ista=inum_base*imype+in_remain+1
```

```
    endif
    iend=ista+inum_base+iadd-1
!
    jnum_base=jm2/jnpes
    jn_remain=jm2-jnpes*jnum_base
    jmype=mype/inpes
    jadd=1
    if(jmype>=jn_remain) jadd=0
    if(jmype<=jn_remain) then
      jsta=(jnum_base+1)*jmype+1
    else
      jsta=jnum_base*jmype+jn_remain+1
    endif
    jend=jsta+jnum_base+jadd-1
!
!--- allocate array
   subdmfldsize=(iend-ista+1)*(jend-jsta+1)
  allocate(data(subdmfldsize,nrec),data1(subdmfldsize,nrec))
   allocate(data2(subdmfldsize*nrec))
   allocate(tmp(subdmfldsize,lm),tmp1(subdmfldsize,lm))
   allocate(fis(subdmfldsize),fis1(subdmfldsize))
!
!--- read by data filed record number
   do jrec=1,nrec
     call nemsio_getrechead(gfile,jrec,vname,vlevtyp,vlev,
iret=iret)
    call nemsio_readrec(gfile,ista,iend,jsta,jend,jrec,
data1(:,jrec),iret=iret)
   print
*,'read,iret=',iret,jrec,vname,vlevtyp,vlev,'data=',
maxval(data1(:,jrec)),minval(data1(:,jrec)),data1(1,jrec),
data1(subdmfldsize,jrec)
   enddo
!
!--- dense read
  call
nemsio_denseread(gfile,ista,iend,jsta,jend,data2,iret)
!
!--- read by data field name, level type and level
  do L=1,lm
     call nemsio_readrecv(gfile,ista,iend,jsta,jend,'tmp',
'mid layer',L,tmp(:,L), iret=iret)

   enddo
!
!--- close nemsio file
  call nemsio_close(gfile,iret=iret)
!
!--- finalize
  call nemsio_finalize()
```

6.Write data field (MPI version)
This example shows how to write out all the model output from GFS.

```
!
  cout2='nemsio2rec_gfs'
  lonb=1152
  latb=576
  lm=64
  root=0
  nrec=lm+1
  allocate(recname(nrec),reclevtyp(nrec), reclev(nrec))
  recname(1:lm)='tmp '
  recname(1+lm)='hgt '
  reclevtyp(1:lm)='mid layer'
  reclevtyp(1+lm)='sfc'
  do l=1,lm
  reclev(l)=l
  enddo
  reclev(lm+1)=1
!
!--- open nemsio file
    call nemsio_open(gfile,trim(cout2),'write',
mpi_comm_world, modelname="GFS", &
     gdatatype="bin4",idate=idate,nrec=nrec,dimx=lonb, &
     dimy=latb,dimz=levs,iret=iret,recname=recname, &
     reclevtyp=reclevtyp,reclev=reclev,nmeta=5)
!

!--- call dense write
   call
nemsio_densewrite(gfile,ista,iend,jsta,jend,data2,iret=iret)
!
!--- close nemsio file
  call nemsio_close(gfile,iret=iret)
!
!--- finalize
  call nemsio_finalize()
```