ZioLib: a Parallel I/O Library

Woo-Sun Yang and Chris Ding
Lawrence Berkeley National Laboratory

September 25, 2003

Contents

1 Copyright

2 Introduction

In a distributed memory parallel environment, many applications rely on a serial I/O strategy, where the global array is gathered on a single MPI process and then written out to a file. I/O performance with this approach is largely limited by single process' I/O bandwidth. Even when parallel I/O is used, satisfactory parallel scaling is not always observed. It is because in many applications fields are not necessarily in a most favorable parallel decomposition for I/O. The best I/O rates are obtained when a field is decomposed with respect to the array's last dimension (referred to here as "Z").
Another situation often encountered in many applications is that a field in CPU resident memory is in one index order but must be stored in a disk file in another order. Changing index orders can complicate a parallel I/O implementation and slow down I/O.
ZioLib facilitates efficient parallel I/O for arrays in such situations. In case of a write, ZioLib remaps a distributed field into a Z-decomposition on a subset of processes (which will be called the *I/O staging processes*) and from there launches a write to a disk file in parallel (see Figure 1). In this Z-decomposition, the data layout of the remapped array on the staging processes' memory is the same as on disk, thus only block data transfer occurs during parallel I/O, achieving maximum efficiency. In case of a read the steps are reversed to build the required distributed arrays on the computational processes.

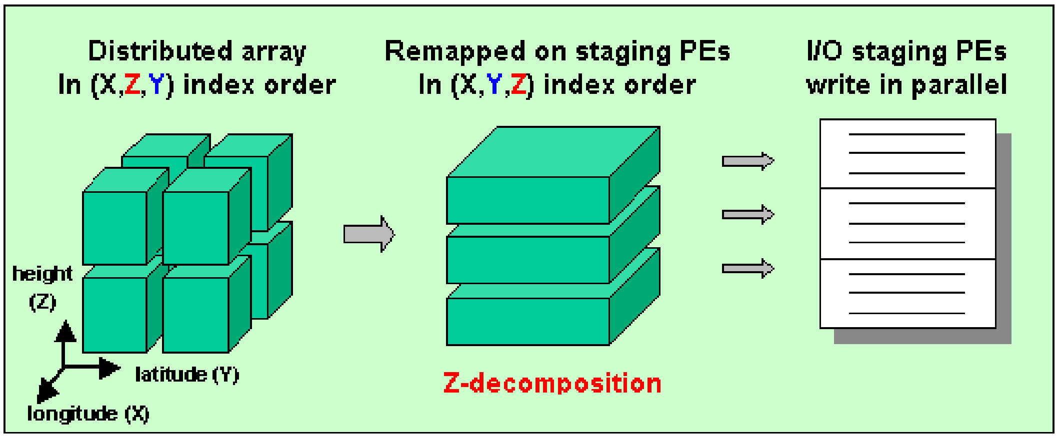

Figure 1: Writing the global field of distributed array `a(X,Z,Y)` to a disk file in `(X,Y,Z)` index order using three I/O staging processes.

Please see more information at http://www.nersc.gov/research/SCG/acpi/ZioLib/.

3 ZioLib Features

- A set of Fortran 90 modules supporting
  - netCDF I/O (wrappers for serial and parallel library routines)
  - direct-access unformatted I/O (serial and parallel)

  - sequential-access unformatted I/O (serial)

- Read or write arrays of an arbitrary number of dimensions
- Support `integer*4`, `real*4` and `real*8` data types
- Read or write in any array index order
- Work with any block parallel decompositions
- A process' subdomain can be made of multiple disjoint rectangular blocks
- Can handle ghost nodes
- Use only MPI-1 routines for greater portability

ZioLib works for regular rectangular block-decompositions. It also works for more general data decomposition cases where each process' local subdomain consists of a number of separate rectangular blocks. Such a more complex decomposition is not uncommon in many application codes. See the test codes `bintest4.F90` and `nftest4.F90` included in the source distribution to see how to use ZioLib functions in such a situation.
The parallel netCDF library for which ZioLib provides wrappers is the one that is being developed at LBNL/NERSC (http://hpcf.nersc.gov/software/libs/io/netcdf/sp/pusage.html).
Since staging processes can be set flexibly and the required remapping of data is all performed by ZioLib, we notice a few distinct advantages with ZioLib:

- Writes/reads are done in large blocks (no seeks) in parallel
- Robust I/O performance is achieved regardless of parallel decomposition types
- Compared to serial I/O, memory limitations on a process can be relieved because global field is now constructed on multiple staging processes instead of single root process
- Congestion on I/O server nodes can be relieved (because the number of disk-accessing staging processes can be controlled)
- Gather/scatter/transpose is eliminated from user codes

4 Source Files

Source files, `zio.h`, `zio_data.F90`, `zio_remap.F90`, `zio_binary.F90`, `zio_netcdf77.F90`, and `zio.F90`, are available at http://www.nersc.gov/research/SCG/acpi/ZioLib/. These codes can be compiled in either fixed or free source form.
The source distribution also contains a number of test codes: `bintest1.F90`, ..., `bintest4.F90` for Fortran unformatted I/O; `nftest1.F90`, ..., `nftest4.F90` for netCDF I/O. A short description about the tests can be found in the `Readme` file contained in the distribution. The test codes are in free source form.
As of now, ZioLib has been successfully tested on the following platforms:

- IBM SP
- SGI Origin
- Linux cluster with pgf90 (Portland Group) and lf95 (Lahey) Fortran compilers

To use the parallel netCDF library for ZioLib on an IBM SP (if it is installed on the machine), define `ZIO_NETCDF_PAR_AIX` (i.e., '`#define ZIO_NETCDF_PAR_AIX`') in zio.h. Otherwise, the serial netCDF library will be used.

5 How to Use: Quick Guide

Users can access ZioLib procedures by including '`use zio`' in their codes.
Most of the ZioLib procedures return an integer error code. When no error is found during invocation of a procedure, `zio_noerr` (0) is returned. When an error is encounted, an error message can be obtained by calling `zio_strerror(errcode[,errtype])`. It returns a character string of length 80.

5.1 Initializing and ending ZioLib

To be able to use ZioLib, MPI must be initialized first. Then ZioLib must be initialized with a call to `zio_init` with the MPI communicator used for computation as the argument. For example,

```
ret = zio_init(mpi_comm_world)
```

To end ZioLib, `zio_end()` is called.

5.2 Creating ZioLib objects

5.2.1 I/O staging communicators

This is to designate a subset of the compute communicator processes as the ones that actually stage I/O. For serial I/O (e.g., sequential-access Fortran unformatted I/O) one can select the process 0 as the staging process. For parallel I/O, you can select a subset of processes from all the compute processes to achieve an optimal I/O bandwidth.
There are a few ways to create I/O staging communicators out of the compute communicator (the handle for the created I/O staging communicator object is returned in `iocomm`):

`zio_new_iocomm(iocomm,n)` staging communicator made of first `n` processes

`zio_new_iocomm_incl(iocomm,n,ranks)` staging communicator made of `ranks(1:n)` processes

`zio_new_iocomm_member(iocomm,is_iope)` staging communicator made of the processes with `is_iops=.true`.

A single staging communicator object can be used for all subsequent I/O. But a flexible arrangement is possible. For example, if the maximum read and write rates can be obtained with different numbers of processes on a machine, you may want to define two I/O staging communicator objects, one for reads and the other for writes.

5.2.2 Distributed array descriptors

ZioLib needs to know how an array is distributed among processes. If distributed array's domain is made of `nblks` separate rectangular blocks, one way is to describe the distributed array is to specify each rectangular block's local grid size along each array dimension (`lsize(1)`, `lsize(2)`, `lsize(3)`, ...) and its grid offsets (`offset(1)`, `offset(2)`, `offset(3)`, ...) in the global array space:

```
ret = zio_new_distarray(distarray,nblks,ndims,offset,lsize)
```

It returns a distributed array descriptor in `distarray`.
In some case, not all processes have valid data assigned to them. For example, for 2-D surface data in a 3-D decomposition, some processes do not contain valid surface data. *In such a case, make sure to set either* `nblks` *or* `lsize(:)` *to 0 if the current process does not contain valid data.*
Another way to create a descriptor is to specify it via a parallel decomposition:

```
ret = zio_new_decomp(decomp,px,py,pz)   ! Parallel decomposition
ret = zio_define_distarray(distarray,decomp,nx,ny,nz)
```

Here we are dealing with an array of global grid sizes `nx`, `ny`, and `nz`, block-distributed over `px`, `py`, and `pz` processes. Distributed arrays for up to 5-D parallel decompositions can be described in this way. Note, however, that there are some restrictions with this approach. First, it is assumed that process ranks change fastest along the X-axis, then Y-axis, and then Z-axis (for MPI's Cartesian process topology, rank ordering is the opposite). When a grid size is not evenly divisible by the respective process count, it is assumed that the excess grid points (that is, the remainder) are evenly distributed among the lowest-rank processs. For example, if there are 26 grid points distributed among four processs, it is assumed that the local grid points for processes 0, 1, 2, and 3 are 7, 7, 6, and 6, respectively.
If offsets, local sizes, ghost nodes, and index order are set in some routine (presumably provided by a code developer) and a user doesn't want to figure them out himself/herself, then the user can create a distributed array descriptor by simply providing the subroutine name:

```
ret = zio_import_distarray(distarray,user_func)
```

where `user_func` must be provided in the user code. See Section 6 for details.
With a distributed array discriptor ZioLib can figure out how to read or write a global array. ZioLib can also read or write non-distributed array replicated on all processes. To read or write such an array, the flag `zio_replicated` should be used as a descriptor.

5.2.3 List of distributed array descriptors

Distribued array descriptors can be conveniently stored in the descriptor list, and they can be retrieved at a later time with variable names (character strings):

```
ret = zio_add_to_list('t',distarray1) ! Store distarray1 for variable 't'
...
ret = zio_get_distarray('t',darray)   ! Descriptor returned in darray
...                                   ! Use darray to read variable 't'
```

It is not necessary to use this feature of storing to and retrieving from the distributed array descriptor list, if a user remembers the descriptor for each field to be read or written. If this feature is used in conjuction with netCDF I/O, the variable name used for storing in the descriptor list *must* be the same as the netCDF variable name defined with `zio_nf_def_var`. See Sections 5.3 and 5.4.

5.2.4 Others

Ghost layers for distributed arrays can be specified with `zio_set_ghostlayers(distarray, ghstl, ghstr)`. It sets the numbers of ghost nodes of distributed array `distarray`. The ghost nodes preceding the real ones (LHS ghost nodes) must be provided as `(ghstl(1), ghstl(2), ...)`, and the ghost nodes after the real ones (RHS ghost nodes) as `(ghstr(1), ghstr(2), ...)` for all array dimensions. Files contain the real node values only.
If the output index order is different from the distributed array's local index order, one can set the output order with `zio_set_index_order(distarray,order)`. If `order(:)=(3,1,2)`, it means that the local array `t(i,j,k)` will be stored in disk in `(j,k,i)` order; or that the data in disk stored in `(j,k,i)` order will be read into the local array as `t(i,j,k)`.
Actually ghost layers and output index order can be specified at the same time when a new distributed array descriptor is created with `zio_new_distarray`. They can be also reset with calls to the above functions after a descriptor was created.
For arrays of a small number of dimensions, you may not want to go with parallel I/O. You can control serial vs. parallel I/O for a certain file by specifying the minimum number of array dimensions for parallel I/O. This is set with a call to `zio_set_ndims_par(filedesc, ndims_par)`. For example, if you issue `zio_set_ndims_par(filedesc, 3)`, 1- and 2-D arrays will be written or read using a single staging process, no matter how many I/O staging processes are currently assigned.

5.3 Unformatted direct- or sequential-access I/O

A file is opened with

```
ret = zio_uf_open(filedesc,iocomm,unit,file,access,recl,...)
```

Variables will be written to or read from the file `file` using unit number `unit` in access mode `access`, using the I/O staging communicator, `iocomm`. Note that `unit`, `file`, `access`, `recl`,

etc. are the usual Fortran `open` statement arguments. Such an I/O channel is established and the handle for that is returned in filedesc.
When sequential-access is specified, the process 0 of the staging communicator will be the staging process for subsequent serial I/O.
Variables are written to or read from the file associated with the file descriptor `filedesc` using

```
ret = zio_uf_read_double(filedesc,distarray,arr,rec)
ret = zio_uf_write_double(filedesc,distarray,arr,rec)
```

where `rec` is the global record number in the file (direct-access I/O; can be omitted for sequential-access I/O), `distarray` is the distributed array descriptor for the variable, and `arr` is the buffer containing the local distributed data. If the current process's parallel subdomain is made of multiple separate blocks, `arr` contains all the local data contiguously one block after another, following the block order used to define `distarray`.
When only an array section of a variable is to be written or read, the section must be specified with `istart` (beginning indices; one-based) and `icount` (numbers of array elements) arrays.

```
ret = zio_uf_read_double(filedesc,distarray,arr,rec,istart,icount)
ret = zio_uf_write_double(filedesc,distarray,arr,rec,istart,icount)
```

Note that `istart` and `icount` must be in the local array's index order (not the output index order, if those index orders are different).
If a distributed array descriptor is stored in the descriptor list, one can use instead of the descriptor the variable's name (that is, character string) that was used to store it in the list. See Section 5.2.3.

```
ret = zio_uf_read_double_name(filedesc,varname,arr,rec,istart,icount)
ret = zio_uf_write_double_name(filedesc,varname,arr,rec,istart,icount)
```

I/O for four-byte real and four-byte integer arrays can be similarly done with `zio_uf_read_real`, `zio_uf_read_real_name`, `zio_uf_read_int`, `zio_uf_read_int_name`, `zio_uf_write_real`, `zio_uf_write_real_name`, `zio_uf_write_int`, and `zio_uf_read_int_name`.
A file is closed with

```
ret = zio_uf_close(filedesc)
```

5.4 NetCDF I/O

To create a netCDF file, one can use

```
ret = zio_nf_create(filename,cmode,iocomm,filedesc)
```

which indicates that data will be written in the netCDF creation mode `cmode` (`nf_clobber`, `nf_share`, or `nf_write`) using the I/O staging communicator `iocomm`. This I/O channel is referred to as `filedesc`. Similarly a file can be open in the open mode `omode` with

```
ret = zio_nf_open(filename,omode,iocomm,filedesc)
```

Many netCDF functions can be invoked in a similar way as the regular netCDF functions would be – only with a netCDF file ID replaced with a file descriptor:

```
ret = zio_nf_set_fill(filedesc,nf_fill,old_fillmode)
ret = zio_nf_def_dim(filedesc,'longitude',nx,lonid)
ret = zio_nf_def_var(filedesc,'temperature',NF_DOUBLE,3,dimids,varid)
ret = zio_nf_put_att_double(filedesc,varid,attname,xtype,length,arr)
ret = zio_nf_enddef(filedesc)
ret = zio_nf_inq_varid(filedesc,'temperature',varid)
```

Variables are written or read using

```
ret = zio_nf_put_var_double(filedesc,varid,t)
ret = zio_nf_get_var_double(filedesc,varid,t)
```

As in the case of the unformatted I/O, if the current process's parallel subdomain is made of multiple separate blocks, `t` contains all the local data contiguously one block after another, following the block order used to define `distarray`.
Writing or reading an array section of a variable can be done with:

```
ret = zio_nf_get_vara_double(filedesc,varid,istart,icount,t)
ret = zio_nf_put_vara_double(filedesc,varid,istart,icount,t)
```

In the examples above, the distributed array descriptor for variable `t` is assumed to be already added to the distributed array descriptor list using `zio_add_to_list` with the same variable name used for defining the netCDF variable (i.e., '`temperature`'). (See Section 5.2.3.) If it is not on the list or a user does not want to use the feature of storing to and retrieving from the list, the argument `distarray` must be provided at the end of the argument list.
NOTE: In netCDF `istart` and `icount` used to refer only to the array section in the netCDF data space in disk, but not array's memory space. However, currently we let `istart` and `icount` refer to an array section in *both* the netCDF data and memory space. That is, an array section in netCDF data space is written from or read into the corresponding section of the global array.

`istart` and `icount` must be in the output index order. That is, they must be in the order specified when the netCDF variable is defined.
One thing to remember when using ZioLib netCDF functions is that a Fortran 90 module provides an explicit interface for module procedures. One consequence is that, if a dummy argument for a procedure is an array, the actual argument must be an array, too. (But array ranks of the actual and dummy arguments do not have to match if the dummy argument is an assumed-size array.) For example, if a single scalar value, say '0', is to be written with `zio_nf_put_vara_double`, the actual argument must be in an array form:

```
ret = zio_nf_put_vara_double(filedesc,varid,istart,icount,(/0/))
```

This restriction must be observed in other netCDF routines, too. For example, one must write

```
ret = zio_nf_def_var(filedesc,'lat',NF_DOUBLE,1,(/latdim/),latvar)
ret = zio_nf_def_var(filedesc,'P0', NF_DOUBLE,0,(/0/),ps0var) ! scalar var
```

instead of `zio_nf_def_var(filedesc,'lat',NF_DOUBLE,1,latdim,latvar)` and `zio_nf_def_var (filedesc,'P0',NF_DOUBLE,0,0,ps0var)`, respectively.
Currently the functions of the `varm` and `vars` families are not supported.
A file is closed with

```
ret = zio_nf_close(filedesc)
```

6 Routine/Function Prologues

6.1 Module zio (Source File: zio.F90)

Module `zio` provides a single interface for all ZioLib procedures.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INTERFACE:

```
module zio
```

USES:

```
use zio_data
use zio_remap
use zio_netcdf77
use zio_binary
implicit none
```

6.1.1 zio_strerror

INTERFACE:

```
function zio_strerror(errcode,errtype)
```

DESCRIPTION:

Returns an error message string of len=80 for error code `errcode` and error type `errtype`. If `errtype` is omitted, the error type set during execution of ZioLib procedures (`zio_errtype`) is used.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Mar-21 -- added ZIO_INCLUDE_NETCDF cpp flag
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
character(len=80) :: zio_strerror
```

INPUT PARAMETERS:

```
integer, intent(in) :: errcode
integer, optional, intent(in) :: errtype
```

6.2 Module zio_data (Source File: zio_data.F90)

Module for ZioLib data and utility functions

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task
2003-Aug-14 -- added fill component to file descriptor object
               for partial support of prefilling netCDF dataset
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INTERFACE:

```
module zio_data
```

USES:

```
  implicit none
```

DEFINED PARAMETERS:

```
  integer, parameter :: zio_i4  = selected_int_kind ( 6) ! 4 byte integer
  integer, parameter :: zio_r4  = selected_real_kind( 6) ! 4 byte real
  integer, parameter :: zio_r8  = selected_real_kind(12) ! 8 byte real

...   Maximum sizes

  integer, parameter :: zio_maxdims   = 100   ! Max dimensions
  integer, parameter :: zio_maxcomms  = 100   ! Max I/O staging comms
  integer, parameter :: zio_maxdata   = 1000  ! Max distributed array objs
  integer, parameter :: zio_maxblks   = 500   ! Max rectangular blocks
  integer, parameter :: zio_maxfiles  = 100   ! Max file descriptors
  integer, parameter :: zio_maxdecomp = 100   ! Max parallel decomps
  integer, parameter :: zio_maxlist   = 1000  ! Max list entries
  integer, parameter :: zio_maxchars  = 128   ! Max character length

...   I/O methods

  integer, parameter :: zio_maxmethods    = 3 ! Supported I/O methods
  integer, parameter :: zio_netcdf_io     = 1 ! For netCDF I/O
  integer, parameter :: zio_direct_io     = 2 ! For direct-access I/O
  integer, parameter :: zio_sequential_io = 3 ! For sequential-access I/O

...   Flag for I/O by root process only in case of replicated global array
```

```
      integer, parameter :: zio_replicated = -1

...   Flag for undefined entity

      integer, parameter :: zio_undefined = -99

...   Error codes
...     no error: zio_noerr = nf_noerr = mpi_success = 0, iostat = 0

      integer, parameter :: zio_noerr              = 0

      integer, parameter :: zio_emax               = -101 ! Code range max
      integer, parameter :: zio_emin               = -150 ! Code range min

      integer, parameter :: zio_error_allocate      = zio_emin + 0
      integer, parameter :: zio_error_argument      = zio_emin + 1
      integer, parameter :: zio_error_arraysection  = zio_emin + 2
      integer, parameter :: zio_error_deallocate    = zio_emin + 3
      integer, parameter :: zio_error_decomp        = zio_emin + 4
      integer, parameter :: zio_error_distarray     = zio_emin + 5
      integer, parameter :: zio_error_filedesc      = zio_emin + 6
      integer, parameter :: zio_error_global_values = zio_emin + 7
      integer, parameter :: zio_error_imap          = zio_emin + 8
      integer, parameter :: zio_error_index_order   = zio_emin + 9
      integer, parameter :: zio_error_init          = zio_emin + 10
      integer, parameter :: zio_error_initialized   = zio_emin + 11
      integer, parameter :: zio_error_iocomm        = zio_emin + 12
      integer, parameter :: zio_error_listentry     = zio_emin + 13
      integer, parameter :: zio_error_mpi_uninit    = zio_emin + 14
      integer, parameter :: zio_error_ndims         = zio_emin + 15
      integer, parameter :: zio_error_new_decomp    = zio_emin + 16
      integer, parameter :: zio_error_new_distarray = zio_emin + 17
      integer, parameter :: zio_error_new_filedesc  = zio_emin + 18
      integer, parameter :: zio_error_new_iocomm    = zio_emin + 19
      integer, parameter :: zio_error_new_listentry = zio_emin + 20
      integer, parameter :: zio_error_record        = zio_emin + 21
      integer, parameter :: zio_error_unlisted      = zio_emin + 22

      integer, parameter :: zio_errtype_zio    = 1  ! ZioLib objs error
      integer, parameter :: zio_errtype_mpi    = 2  ! MPI error (disabled)
      integer, parameter :: zio_errtype_binary = 3  ! Binary I/O error
      integer, parameter :: zio_errtype_netcdf = 4  ! Netcdf I/O error

...   Parallel decomposition descriptor
```

```
      type zio_decomp_obj
        integer :: ndims                    ! Number of dimensions
        integer, pointer :: npes(:)         ! Number of PEs for axes
        integer, pointer :: coords(:)       ! PE coords
      end type zio_decomp_obj

...   I/O staging communicator object

      type zio_iocomm_obj
        integer :: comm                     ! I/O staging communicator
        integer :: niopes                   ! Number of staging processes
        integer :: me                       ! Rank in comm
        integer, pointer :: ranks_cmp(:)    ! Rank in comm to rank in zio_comm
        logical, pointer :: is_iope(:)      ! Array showing whether a process
                                            ! is an I/O staging process
      end type zio_iocomm_obj

...   Distributed array descriptor

      type zio_distarray_obj
        integer :: ndims                    ! Number of dimensions
        integer, pointer :: nblks(:)        ! Number of rectangular blocks
        integer, pointer :: loff(:,:,:)     ! Offset of local array
        integer, pointer :: lsize(:,:,:)    ! Size of local array
        integer, pointer :: gsize(:)        ! Global size
        integer, pointer :: ghstl(:)        ! Number of LHS ghost nodes
        integer, pointer :: ghstr(:)        ! Number of RHS ghost nodes
        integer, pointer :: index_order(:) ! Local array index order
      end type zio_distarray_obj

...   File descriptor

      type zio_filedesc_obj
        integer :: fileid                   ! File ID
        integer :: method                   ! I/O method
        integer :: iocomm                   ! I/O comm obj ID
        integer :: ndims_par                ! Min # of dims for parallel I/O
        integer :: reclen                   ! Record length (for unformatted
                                            ! I/O)
        logical :: fill                     ! netCDF fill mode (default: true)
      end type zio_filedesc_obj

...   Distributed array list
```

```
type zio_list_entry
  character(len=zio_maxchars) :: name    ! Variable name
  integer :: distarray                   ! Array distribution ID
end type zio_list_entry
```

6.2.1 zio_init

INTERFACE:

```
integer function zio_init(comm)
```

DESCRIPTION:

Begin ZioLib.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task
2003-Aug-14 -- added fill component to file descriptor object
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: comm        ! Compute communicator;
                                   ! ignored for non-SPMD
```

6.2.2 zio_end

INTERFACE:

```
integer function zio_end()
```

DESCRIPTION:

End ZioLib

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task
2003-Aug-14 -- added fill component to file descriptor object
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

6.2.3 zio_add_to_list

INTERFACE:

```
integer function zio_add_to_list(name,distarray)
```

DESCRIPTION:

Add a distributed array entry to the distributed array list

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: name
integer, intent(in) :: distarray
```

6.2.4 zio_check_decomp

INTERFACE:

```
integer function zio_check_decomp(decomp)
```

DESCRIPTION:

Checks whether `decomp` is a valid parallel decomposition descriptor.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: decomp
```

6.2.5 zio_check_distarray

INTERFACE:

```
integer function zio_check_distarray(distarray)
```

DESCRIPTION:

Checks whether `distarray` is a valid distributed array descriptor.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

6.2.6 zio_check_filedesc

INTERFACE:

```
integer function zio_check_filedesc(filedesc)
```

DESCRIPTION:

Checks whether `filedesc` is a valid file descriptor.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: filedesc
```

6.2.7 zio_check_iocomm

INTERFACE:

```
integer function zio_check_iocomm(iocomm)
```

DESCRIPTION:

Checks whether `iocomm` is a valid I/O staging communicator object.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
```

6.2.8 zio_data_strerror

INTERFACE:

```
function zio_data_strerror(errcode)
```

DESCRIPTION:

Returns an error message of length 80 for error code `errcode` that is in general related to managing ZioLib objects.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
character(len=80) :: zio_data_strerror
```

INPUT PARAMETERS:

```
integer, intent(in) :: errcode
```

6.2.9 zio_destroy_decomp

INTERFACE:

```
integer function zio_destroy_decomp(decomp)
```

DESCRIPTION:

Destroy parallel decomposition descriptor `decomp`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: decomp
```

6.2.10 zio_destroy_distarray

INTERFACE:

```
integer function zio_destroy_distarray(distarray)
```

DESCRIPTION:

Destroy distributed array descriptor `distarray`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

6.2.11 zio_destroy_iocomm

INTERFACE:

```
integer function zio_destroy_iocomm(iocomm)
```

DESCRIPTION:

Destroy I/O staging communicator object `iocomm`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
```

6.2.12 zio_destroy_listentry

INTERFACE:

```
integer function zio_destroy_listentry(name)
```

DESCRIPTION:

Remove the distributed array descriptor entry with variable name `name` from the descriptor list.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: name
```

6.2.13 zio_find_locnx

INTERFACE:

```
integer function zio_find_locnx(nx,mype,totpes,locnx,begin)
```

DESCRIPTION:

Find local size `locnx` and zero-based offset `begin` for process `mype` when the global dimension size is `nx` and the total process count is `totpes`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in)  :: nx            ! global size
integer, intent(in)  :: mype          ! PE rank
integer, intent(in)  :: totpes        ! total number of PEs
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: locnx         ! local size for PE mype
integer, intent(out) :: begin         ! beginning index (zero-based)
```

6.2.14 zio_get_broadcast_query

INTERFACE:

```
integer function zio_get_broadcast_query(bcast)
```

DESCRIPTION:

Determine whether query values are broadcasted.

REVISION HISTORY:

```
2003-Sep-17 -- created
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
logical, intent(out) :: bcast
```

6.2.15 zio_get_comm

INTERFACE:

```
integer function zio_get_comm(iocomm,comm)
```

DESCRIPTION:

Return the MPI communicator of I/O staging communicator object `iocomm`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: comm
```

6.2.16 zio_get_coords

INTERFACE:

```
integer function zio_get_coords(decomp,coords)
```

DESCRIPTION:

Returns the current process' coordinates in parallel decomposition object `decomp`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: decomp
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: coords(:)
```

6.2.17 zio_get_distarray

INTERFACE:

```
integer function zio_get_distarray(name,distarray)
```

DESCRIPTION:

Retrieves the distributed array descriptor for the variable with name `name` from the distributed array descriptor list.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: name
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: distarray
```

6.2.18 zio_get_errtype

INTERFACE:

```
integer function zio_get_errtype(errtype)
```

DESCRIPTION:

Returns ZioLib error type.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: errtype
```

6.2.19 zio_get_fileid

INTERFACE:

```
integer function zio_get_fileid(filedesc,fileid)
```

DESCRIPTION:

Returns the file ID of file descriptor `filedesc`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: fileid
```

6.2.20 zio_get_fill

INTERFACE:

```
integer function zio_get_fill(filedesc,fill)
```

DESCRIPTION:

Returns the file fill mode of file descriptor `filedesc`.

REVISION HISTORY:

```
2003-Aug-14 -- created
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: filedesc
```

OUTPUT PARAMETERS:

```
logical, intent(out) :: fill
```

6.2.21 zio_get_ghstl

INTERFACE:

```
integer function zio_get_ghstl(distarray,ghstl)
```

DESCRIPTION:

Returns the number of LHS ghost nodes (that is, ghost nodes preceding the real nodes) of distributed array descriptor `distarray`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: ghstl(*)
```

6.2.22 zio_get_ghstr

INTERFACE:

```
integer function zio_get_ghstr(distarray,ghstr)
```

DESCRIPTION:

Returns the number of RHS ghost nodes (that is, ghost nodes after the real nodes) of distributed array descriptor `distarray`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: ghstr(*)
```

6.2.23 zio_get_gsize

INTERFACE:

```
integer function zio_get_gsize(distarray,gsize)
```

DESCRIPTION:

Returns the global grid sizes of distributed array descriptor `distarray`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: gsize(*)
```

6.2.24 zio_get_index_order

INTERFACE:

```
integer function zio_get_index_order(distarray,index_order)
```

DESCRIPTION:

Returns the array output index order of distributed array descriptor `distarray`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: index_order(*)
```

6.2.25 zio_get_io_rank

INTERFACE:

```
integer function zio_get_io_rank(iocomm,rank,io_rank)
```

DESCRIPTION:

Returns the process rank in I/O staging communicator object `iocomm` of process `rank`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
integer, intent(in) :: rank
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: io_rank
```

6.2.26 zio_get_io_root

INTERFACE:

```
integer function zio_get_io_root(iocomm,io_root)
```

DESCRIPTION:

Returns the process rank in zio_comm for process 0 in I/O staging communicator object `iocomm`.

REVISION HISTORY:

```
2003-Aug-13 -- created
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: io_root
```

6.2.27 zio_get_iocomm

INTERFACE:

```
integer function zio_get_iocomm(filedesc,iocomm)
```

DESCRIPTION:

Returns I/O staging communicator object `iocomm` associated with file descriptor `filedesc`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: iocomm
```

6.2.28 zio_get_is_iope

INTERFACE:

```
integer function zio_get_is_iope(iocomm,is_iope)
```

DESCRIPTION:

Returns a logical array indicating whether processes are I/O staging processes in I/O staging communicator object `iocomm`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
```

OUTPUT PARAMETERS:

```
logical, intent(out) :: is_iope(0:*)
```

6.2.29 zio_get_loff

INTERFACE:

```
integer function zio_get_loff(distarray,ndims,nblks,loff)
```

DESCRIPTION:

Returns the global grid offsets for all array dimensions for distributed array descriptor `distarray` on all processes.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- modified for multiple rectangular blocks per task
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
integer, intent(in) :: ndims
integer, intent(in) :: nblks
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: loff(ndims,nblks,0:*)
```

6.2.30 zio_get_loff_me

INTERFACE:

```
integer function zio_get_loff_me(distarray,ndims,loff)
```

DESCRIPTION:

Returns the global grid offsets for all array dimensions for distributed array descriptor `distarray` on the current process.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- modified for multiple rectangular blocks per task
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
integer, intent(in) :: ndims
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: loff(ndims,*)
```

6.2.31 zio_get_lsize

INTERFACE:

```
integer function zio_get_lsize(distarray,ndims,nblks,lsize)
```

DESCRIPTION:

Returns the local grid sizes for all array dimensions for distributed array descriptor `distarray` on all processes.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- modified for multiple rectangular blocks per task
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
integer, intent(in) :: ndims
integer, intent(in) :: nblks
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: lsize(ndims,nblks,0:*)
```

6.2.32 zio_get_lsize_me

INTERFACE:

```
integer function zio_get_lsize_me(distarray,ndims,lsize)
```

DESCRIPTION:

Returns the local grid sizes for all array dimensions for distributed array descriptor `distarray` on the current process.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- modified for multiple rectangular blocks per task
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
integer, intent(in) :: ndims
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: lsize(ndims,*)
```

6.2.33 zio_get_me

INTERFACE:

```
integer function zio_get_me(iocomm,me)
```

DESCRIPTION:

Returns the process rank in the I/O staging communicator associated with the object `iocomm`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: me
```

6.2.34 zio_get_method

INTERFACE:

```
integer function zio_get_method(filedesc,method)
```

DESCRIPTION:

Returns the I/O method (netCDF, direct-access or sequential-access?) for file descriptor `filedesc`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: method
```

6.2.35 zio_get_nblks

INTERFACE:

```
integer function zio_get_nblks(distarray,nblks)
```

DESCRIPTION:

Returns the number of disjoint rectangular data blocks of distributed array descriptor `distarray`.

REVISION HISTORY:

```
2003-Jul-14 -- created
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: nblks(0:*)
```

6.2.36 zio_get_nblks_max

INTERFACE:

```
integer function zio_get_nblks_max(distarray,nblks)
```

DESCRIPTION:

Returns the maximim number of disjoint rectangular data blocks of distributed array descriptor `distarray` over PEs. Size of the 2nd dimension of `loff(:,:,:)` and `lsize(:,:,:)` of distributed array desc objects.

REVISION HISTORY:

```
2003-Jul-14 -- created
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: nblks
```

6.2.37 zio_get_ndims_decomp

INTERFACE:

```
integer function zio_get_ndims_decomp(decomp,ndims)
```

DESCRIPTION:

Returns the number of dimensions of parallel decomposition object `decomp`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- changed from \texttt{zio\_get\_decomp\_ndims}
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: decomp
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: ndims
```

6.2.38 zio_get_ndims_distarray

INTERFACE:

```
integer function zio_get_ndims_distarray(distarray,ndims)
```

DESCRIPTION:

Returns the number of dimensions for distributed arrays associated with descriptor `distarray`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- changed from \texttt{zio\_get\_distarray\_ndims}
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: ndims
```

6.2.39 zio_get_ndims_par

INTERFACE:

```
integer function zio_get_ndims_par(filedesc,ndims_par)
```

DESCRIPTION:

Returns the minimum number of array dimensions for parallel I/O for the file associated with descriptor `filedesc`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: ndims_par
```

6.2.40 zio_get_npes

INTERFACE:

```
integer function zio_get_npes(decomp,npes)
```

DESCRIPTION:

Returns the numbers of processes for parallel decomposition descriptor `decomp`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype (was \texttt{zio\_get\_npes\_decomp})
2003-Jul-14 -- name changed from \texttt{zio\_get\_npes\_decomp}
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: decomp
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: npes(*)
```

6.2.41 zio_get_niopes

INTERFACE:

```
integer function zio_get_niopes(iocomm,niopes)
```

DESCRIPTION:

Returns the number of I/O staging processes for the staging communicator associated with `iocomm`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype (was \texttt{zio\_get\_npes\_iocomm})
2003-Jul-14 -- name changed from \texttt{zio\_get\_npes\_iocomm}
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: niopes
```

6.2.42 zio_get_ranks_cmp

INTERFACE:

```
integer function zio_get_ranks_cmp(iocomm,ranks_cmp)
```

DESCRIPTION:

Returns the process ranks in the compute communicator, of the I/O staging processes in staging communicator object `iocomm`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: ranks_cmp(0:*)
```

6.2.43 zio_get_reclen

INTERFACE:

```
integer function zio_get_reclen(filedesc,reclen)
```

DESCRIPTION:

Returns the record length of the file associated with file descriptor `filedesc`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: reclen
```

6.2.44 zio_get_zdim_size

INTERFACE:

```
integer function zio_get_zdim_size(iocomm,gsize,zstart,zcount,  &
&                                  niopes)
```

DESCRIPTION:

Returns "Z" dimension's offset and size on an I/O staging process. A zero is returned for `zcount` if the current process is not an I/O staging process.

REVISION HISTORY:

```
2003-Jul-14 -- created
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: iocomm, gsize
integer, intent(in), optional :: niopes
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: zstart, zcount
```

6.2.45 zio_get_zio_comm

INTERFACE:

```
integer function zio_get_zio_comm(comm)
```

DESCRIPTION:

Returns the compute communicator.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: comm
```

6.2.46 zio_get_zio_initialized

INTERFACE:

```
integer function zio_get_zio_initialized(initialized)
```

DESCRIPTION:

Checks whether ZioLib is initialized.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
logical, intent(out) :: initialized
```

6.2.47 zio_get_zio_me

INTERFACE:

```
integer function zio_get_zio_me(me)
```

DESCRIPTION:

Returns the process rank in the compute communicator.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: me
```

6.2.48 zio_get_zio_npes

INTERFACE:

```
integer function zio_get_zio_npes(npes)
```

DESCRIPTION:

Returns the number of processes in the compute communicator.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: npes
```

6.2.49 zio_new_decomp

INTERFACE:

```
integer function zio_new_decomp(decomp,p1,p2,p3,p4,p5)
```

DESCRIPTION:

Create a descriptor for parallel decomposition of up to 5-D which has `p1,p2,...` processes along axes. Here, process ranks change fastest along the first axis, then second axis,... (In MPI's Cartesian process topology, rank ordering is the opposite.)

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: decomp
```

INPUT PARAMETERS:

```
integer, intent(in)  :: p1             ! ignored for non-SPMD
integer, optional, intent(in)  :: p2    ! ignored for non-SPMD
integer, optional, intent(in)  :: p3    ! ignored for non-SPMD
integer, optional, intent(in)  :: p4    ! ignored for non-SPMD
integer, optional, intent(in)  :: p5    ! ignored for non-SPMD
```

6.2.50 zio_new_distarray

INTERFACE:

```
        integer function zio_new_distarray(distarray,nblks,ndims,loff,  &
     &                                   lsize,ghstl,ghstr,index_order)
```

DESCRIPTION:

Create a descriptor for a distributed array made of `nblks` disjoint rectangular blocks of `ndims` dimensions whose offsets, local sizes, LHS ghost nodes, RHS ghost nodes and the output order are given respectively by `loff(1:ndims,1:nblks)`, `lsize(1:ndims,1:nblks)`, `ghstl(1:ndims)`, `ghstr(1:ndims)`, and `index_order(1:ndims)`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
        implicit none
```

OUTPUT PARAMETERS:

```
        integer, intent(out) :: distarray            ! dist array obj
```

INPUT PARAMETERS:

```
        integer, intent(in)  :: nblks                ! number of rect blocks
        integer, intent(in)  :: ndims                ! number of dimensions
        integer, intent(in)  :: loff(*)              ! local offsets
        integer, intent(in)  :: lsize(*)             ! local sizes
        integer, optional, intent(in) :: ghstl(ndims)       ! LHS ghost nodes
        integer, optional, intent(in) :: ghstr(ndims)       ! RHS ghost nodes
        integer, optional, intent(in) :: index_order(ndims) ! index order
        integer, optional, intent(in) :: ghstl(*)       ! LHS ghost nodes
        integer, optional, intent(in) :: ghstr(*)       ! RHS ghost nodes
        integer, optional, intent(in) :: index_order(*) ! index order
```

6.2.51 zio_define_distarray

INTERFACE:

```
        integer function zio_define_distarray(distarray,decomp,n1,n2,n3,&
     &                                        n4,n5)
```

DESCRIPTION:

Create a descriptor for the distributed array of grid sizes `n1`,`n2`,... for the parallel decomposition (up to 5-D) denoted by `decomp`. If a grid size is not evenly divisible by the respective process count, the excess grid points (that is, remainder) are evenly distributed among the lowest-rank processes along the dimension.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- modified for multiple rectangular blocks per task
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: distarray
```

INPUT PARAMETERS:

```
integer, intent(in)  :: decomp
integer, intent(in)  :: n1
integer, optional, intent(in)  :: n2
integer, optional, intent(in)  :: n3
integer, optional, intent(in)  :: n4
integer, optional, intent(in)  :: n5
```

6.2.52 zio_import_distarray

INTERFACE:

```
integer function zio_import_distarray(distarray, user_func)
```

DESCRIPTION:

Create a distributed array descriptor whose distributed grid is defined in user-provided function, `user_func`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- modified for multiple rectangular blocks per task
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: distarray
```

INPUT PARAMETERS:

```
    interface
      subroutine user_func(ndims,nblks,loff,lsize,ghstl,ghstr,      &
 &                           index_order)
        integer :: ndims, nblks                          ! intent(out)
        integer :: loff(*), lsize(*)                     ! intent(out)
        integer :: ghstl(*), ghstr(*), index_order(*) ! intent(out)
      end subroutine user_func
    end interface
```

6.2.53 zio_new_iocomm

INTERFACE:

```
integer function zio_new_iocomm(iocomm,niopes)
```

DESCRIPTION:

Create an I/O staging communicator object with a staging communicator made of the first `niopes` processes.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: iocomm    ! communicator obj ID
```

INPUT PARAMETERS:

```
integer, intent(in) :: niopes     ! # of I/O staging processes;
                                  ! overridden by 1 for non-SPMD
```

6.2.54 zio_new_iocomm_incl

INTERFACE:

```
integer function zio_new_iocomm_incl(iocomm,n,ranks)
```

DESCRIPTION:

Create an I/O staging communicator object with a staging communicator made of the processes whose ranks are specified by `ranks(1:n)`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: iocomm    ! communicator obj ID
```

INPUT PARAMETERS:

```
integer, intent(in) :: n          ! # of I/O staging processes in comm
integer, intent(in) :: ranks(n)   ! ranks
```

6.2.55 zio_new_iocomm_member

INTERFACE:

```
integer function zio_new_iocomm_member(iocomm,is_iope)
```

DESCRIPTION:

Create an I/O staging communicator object with a staging communicator made of the processes with `is_iope` = `.true.`

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: iocomm    ! communicator obj ID
```

INPUT PARAMETERS:

```
logical, intent(in) :: is_iope    ! belongs to staging comm?
```

6.2.56 zio_set_broadcast_query

INTERFACE:

```
integer function zio_set_broadcast_query(bcast)
```

DESCRIPTION:

Set whether query values are broadcasted.

REVISION HISTORY:

```
2003-Sep-17 -- created
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
logical, intent(in) :: bcast
```

6.2.57 zio_set_ghostlayers

INTERFACE:

```
integer function zio_set_ghostlayers(distarray,ghstl,ghstr)
```

DESCRIPTION:

Sets ghost nodes for distributed array descriptor `distarray`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray              ! array distribution
integer, intent(in) :: ghstl(:), ghstr(:)     ! LHS and RHS ghost nodes
```

6.2.58 zio_set_index_order

INTERFACE:

```
integer function zio_set_index_order(distarray,index_order)
```

DESCRIPTION:

Set output index order for distributed array descriptor `distarray`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- modified for multiple rectangular blocks per task
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: distarray               ! array distribution
integer, intent(in) :: index_order(:)          ! index order
```

6.2.59 zio_set_ndims_par

INTERFACE:

```
integer function zio_set_ndims_par(filedesc,ndims_par)
```

DESCRIPTION:

Set the parameter for the minumum number of array dimensions for parallel I/O.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: filedesc    ! file object index
integer, intent(in) :: ndims_par   ! min number of dimensions eligible
                                   ! for parallel I/O
```

6.3 Module zio_remap (Source File: zio_remap.F90)

Module `zio_remap` provides procedures for remapping from and to I/O staging processes.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INTERFACE:

```
module zio_remap
```

USES:

```
use zio_data
implicit none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: zio_from_iopes
public :: zio_from_iopes_double
public :: zio_from_iopes_int
public :: zio_from_iopes_real
public :: zio_to_iopes
public :: zio_to_iopes_double
public :: zio_to_iopes_int
public :: zio_to_iopes_real

interface zio_from_iopes
  module procedure zio_from_iopes_int
  module procedure zio_from_iopes_real
  module procedure zio_from_iopes_double
end interface

interface zio_to_iopes
  module procedure zio_to_iopes_int
  module procedure zio_to_iopes_real
  module procedure zio_to_iopes_double
end interface
```

6.3.1 zio_from_iopes_double

INTERFACE:

```
    integer function zio_from_iopes_double(fin,fout,distarray,      &
&                                          iocomm,niopes,istart,    &
&                                          icount,index_order)
```

DESCRIPTION:

Construct distributed arrays from Z-decomposed arrays on I/O staging PEs

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task;
               made niopes, istart, icount, and index_order
               optional arguments
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
    implicit none
```

INPUT PARAMETERS:

```
    real(zio_r8), intent(in)  :: fin(0:*)      ! input field
    integer, intent(in) :: distarray           ! array dist descriptor
    integer, intent(in) :: iocomm              ! I/O communicator object ID;
                                               ! ignored for non-SPMD
    integer, intent(in), optional :: niopes    ! taget # of I/O PEs to use;
                                               ! ignored for non-SPMD
    integer, intent(in), optional :: istart(:), icount(:) ! start and count
    integer, intent(in), optional :: index_order(:) ! output index order
```

OUTPUT PARAMETERS:

```
    real(zio_r8), intent(out) :: fout(0:*)     ! remapped array
```

6.3.2 zio_from_iopes_int

INTERFACE:

```
    integer function zio_from_iopes_int(fin,fout,distarray,         &
&                                       iocomm,niopes,istart,       &
&                                       icount,index_order)
```

DESCRIPTION:

Construct distributed arrays from Z-decomposed arrays on I/O staging PEs

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task;
               made niopes, istart, icount, and index_order
               optional arguments
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer(zio_i4), intent(in)  :: fin(0:*)   ! input field
integer, intent(in) :: distarray           ! array dist descriptor
integer, intent(in) :: iocomm              ! I/O communicator object ID;
                                           ! ignored for non-SPMD
integer, intent(in), optional :: niopes    ! taget # of I/O PEs to use;
                                           ! ignored for non-SPMD
integer, intent(in), optional :: istart(:), icount(:) ! start and count
integer, intent(in), optional :: index_order(:) ! output index order
```

OUTPUT PARAMETERS:

```
integer(zio_i4), intent(out) :: fout(0:*)  ! remapped array
```

6.3.3 zio_from_iopes_real

INTERFACE:

```
  integer function zio_from_iopes_real(fin,fout,distarray,       &
&                                       iocomm,niopes,istart,     &
&                                       icount,index_order)
```

DESCRIPTION:

Construct distributed arrays from Z-decomposed arrays on I/O staging PEs

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task;
               made niopes, istart, icount, and index_order
               optional arguments
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
real(zio_r4), intent(in)  :: fin(0:*)      ! input field
integer, intent(in) :: distarray           ! array dist descriptor
integer, intent(in) :: iocomm              ! I/O communicator object ID;
                                           ! ignored for non-SPMD
integer, intent(in), optional :: niopes    ! taget # of I/O PEs to use;
                                           ! ignored for non-SPMD
integer, intent(in), optional :: istart(:), icount(:) ! start and count
integer, intent(in), optional :: index_order(:) ! output index order
```

OUTPUT PARAMETERS:

```
real(zio_r4), intent(out) :: fout(0:*)     ! remapped array
```

6.3.4 zio_to_iopes_double

INTERFACE:

```
    integer function zio_to_iopes_double(fin,fout,distarray,iocomm, &
  &                                      niopes,istart,icount,     &
  &                                      index_order)
```

DESCRIPTION:

Remaps distributed local arrays to Z-decomposed arrays on I/O staging PEs

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Jul-14 -- allowed multiple rectangular blocks per task;
               made niopes, istart, icount, and index_order
               optional arguments
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
real(zio_r8), intent(in)  :: fin(0:*)      ! input field
integer, intent(in) :: distarray           ! array dist descriptor
integer, intent(in) :: iocomm              ! I/O communicator object ID;
```

```
                                                ! ignored for non-SPMD
    integer, intent(in), optional :: niopes     ! taget # of I/O PEs to use;
                                                ! ignored for non-SPMD
    integer, intent(in), optional :: istart(:), icount(:) ! start and count
    integer, intent(in), optional :: index_order(:) ! output index order
```

OUTPUT PARAMETERS:

```
    real(zio_r8), intent(out) :: fout(0:*)      ! remapped array
```

6.3.5 zio_to_iopes_int

INTERFACE:

```
    integer function zio_to_iopes_int(fin,fout,distarray,iocomm,    &
  &                                   niopes,istart,icount,         &
  &                                   index_order)
```

DESCRIPTION:

Remaps distributed local arrays to Z-decomposed arrays on I/O staging PEs

REVISION HISTORY:

```
 2003-Jan-31 -- first prototype
 2003-Jul-14 -- allowed multiple rectangular blocks per task;
                made niopes, istart, icount, and index_order
                optional arguments
 2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
    implicit none
```

INPUT PARAMETERS:

```
    integer(zio_i4), intent(in)  :: fin(0:*)    ! input field
    integer, intent(in) :: distarray            ! array dist descriptor
    integer, intent(in) :: iocomm               ! I/O communicator object ID;
                                                ! ignored for non-SPMD
    integer, intent(in), optional :: niopes     ! taget # of I/O PEs to use;
                                                ! ignored for non-SPMD
    integer, intent(in), optional :: istart(:), icount(:) ! start and count
    integer, intent(in), optional :: index_order(:) ! output index order
```

OUTPUT PARAMETERS:

```
    integer(zio_i4), intent(out) :: fout(0:*)   ! remapped array
```

6.3.6 zio_to_iopes_real

INTERFACE:

```
      integer function zio_to_iopes_real(fin,fout,distarray,iocomm,   &
   &                                     niopes,istart,icount,        &
   &                                     index_order)
```

DESCRIPTION:

Remaps distributed local arrays to Z-decomposed arrays on I/O staging PEs

REVISION HISTORY:

```
    2003-Jan-31 -- first prototype
    2003-Jul-14 -- allowed multiple rectangular blocks per task;
                   made niopes, istart, icount, and index_order
                   optional arguments
    2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
      implicit none
```

INPUT PARAMETERS:

```
      real(zio_r4), intent(in)  :: fin(0:*)      ! input field
      integer, intent(in) :: distarray           ! array dist descriptor
      integer, intent(in) :: iocomm              ! I/O communicator object ID;
                                                 ! ignored for non-SPMD
      integer, intent(in), optional :: niopes    ! taget # of I/O PEs to use;
                                                 ! ignored for non-SPMD
      integer, intent(in), optional :: istart(:), icount(:) ! start and count
      integer, intent(in), optional :: index_order(:) ! output index order
```

OUTPUT PARAMETERS:

```
      real(zio_r4), intent(out) :: fout(0:*)     ! remapped array
```

6.4 Module zio_binary (Source File: zio_binary.F90)

Module `zio_binary` provides procedures for Fortran direct- and sequential-access unformatted I/O.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Apr-24 -- array expressions used in dist versions of reads/writes
2003-Aug-14 -- added partial support of filling dataset
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INTERFACE:

```
module zio_binary
```

USES:

```
use zio_data
use zio_remap
implicit none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: zio_uf_open, zio_uf_close, zio_uf_strerror

public :: zio_uf_read_double,  zio_uf_read_double_name
public :: zio_uf_read_real,    zio_uf_read_real_name
public :: zio_uf_read_int,     zio_uf_read_int_name

public :: zio_uf_write_double, zio_uf_write_double_name
public :: zio_uf_write_real,   zio_uf_write_real_name
public :: zio_uf_write_int,    zio_uf_write_int_name
```

6.4.1 zio_uf_close

INTERFACE:

```
integer function zio_uf_close(filedesc,status)
```

DESCRIPTION:

Close a file

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc                        ! file desc index
character(len=*), optional, intent(in):: status   ! keep or delete
```

6.4.2 zio_uf_open

INTERFACE:

```
  integer function zio_uf_open(filedesc,iocomm,unit,file,        &
&                               status,access,recl,position,action)
```

DESCRIPTION:

Open a file for reading and/or writing

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

USES:

```
implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out):: filedesc            ! file desc index
```

INPUT PARAMETERS:

```
integer, intent(in):: iocomm               ! I/O communicator ID;
                                           ! ignored for non-SPMD
integer, intent(in):: unit                       ! file unit number
character(len=*), intent(in):: file              ! file name
character(len=*), optional, intent(in):: status   ! old, new, replace,
                                            ! scratch, or unknown
character(len=*), intent(in):: access            ! direct or sequential
integer, optional, intent(in):: recl             ! record length
character(len=*), optional, intent(in):: position ! asis, rewind or
                                                 ! append
character(len=*), optional, intent(in):: action   ! read, write or
                                                 ! readwrite
```

6.4.3 zio_uf_read_double

INTERFACE:

```
    integer function zio_uf_read_double(filedesc,distarray,arr,rec, &
 &                                      istart,icount,n)
```

DESCRIPTION:

Unformatted read

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
    implicit none
```

INPUT PARAMETERS:

```
    integer, intent(in):: filedesc, distarray
    integer, optional, intent(in) :: rec       ! global beg record number
                                               ! (1-based)
    integer, optional, intent(in) :: istart(:), icount(:)
                                               ! in local index order
    integer, optional, intent(in) :: n         ! # of elements in case of
                                               !  replicated array
```

OUTPUT PARAMETERS:

```
    real(zio_r8), intent(out) :: arr(*)
```

6.4.4 zio_uf_read_double_name

INTERFACE:

```
    integer function zio_uf_read_double_name(filedesc,varname,arr,  &
 &                                           rec,istart,icount,n)
```

DESCRIPTION:

Unformatted read for variable with variable name of '`varname`'. The distributed array descriptor for the variable must have been added to the descriptor list with the name of '`varname`'. See `zio_add_to_list`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc
character(len=*), intent(in) :: varname
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
                                           ! in local index order
integer, optional, intent(in) :: n         ! # of elements in case of
                                           !  replicated array
```

OUTPUT PARAMETERS:

```
real(zio_r8), intent(out) :: arr(*)
```

6.4.5 zio_uf_read_int

INTERFACE:

```
    integer function zio_uf_read_int(filedesc,distarray,arr,rec,    &
 &                                   istart,icount,n)
```

DESCRIPTION:

Unformatted read

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc, distarray
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
                                           ! in local index order
integer, optional, intent(in) :: n         ! # of elements in case of
                                           !  replicated array
```

OUTPUT PARAMETERS:

```
integer(zio_i4), intent(out) :: arr(*)
```

6.4.6 zio_uf_read_int_name

INTERFACE:

```
    integer function zio_uf_read_int_name(filedesc,varname,arr,     &
 &                                        rec,istart,icount,n)
```

DESCRIPTION:

Unformatted read for variable with variable name of '`varname`'. The distributed array descriptor for the variable must have been added to the descriptor list with the name of '`varname`'. See `zio_add_to_list`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
    implicit none
```

INPUT PARAMETERS:

```
    integer, intent(in):: filedesc
    character(len=*), intent(in) :: varname
    integer, optional, intent(in) :: rec       ! global beg record number
                                               ! (1-based)
    integer, optional, intent(in) :: istart(:), icount(:)
                                               ! in local index order
    integer, optional, intent(in) :: n         ! # of elements in case of
                                               !  replicated array
```

OUTPUT PARAMETERS:

```
    integer(zio_i4), intent(out) :: arr(*)
```

6.4.7 zio_uf_read_real

INTERFACE:

```
    integer function zio_uf_read_real(filedesc,distarray,arr,rec,   &
 &                                    istart,icount,n)
```

DESCRIPTION:

Unformatted read

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc, distarray
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
                                           ! in local index order
integer, optional, intent(in) :: n         ! # of elements in case of
                                           !  replicated array
```

OUTPUT PARAMETERS:

```
real(zio_r4), intent(out) :: arr(*)
```

6.4.8 zio_uf_read_real_name

INTERFACE:

```
   integer function zio_uf_read_real_name(filedesc,varname,arr,    &
&                                          rec,istart,icount,n)
```

DESCRIPTION:

Unformatted read for variable with variable name of '`varname`'. The distributed array descriptor for the variable must have been added to the descriptor list with the name of '`varname`'. See `zio_add_to_list`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc
character(len=*), intent(in) :: varname
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
                                           ! in local index order
integer, optional, intent(in) :: n         ! # of elements in case of
                                           !  replicated array
```

OUTPUT PARAMETERS:

```
real(zio_r4), intent(out) :: arr(*)
```

6.4.9 zio_uf_strerror

INTERFACE:

```
function zio_uf_strerror(errcode)
```

DESCRIPTION:

Returns an error message string of len=80 for a unformatted I/O error given by errcode.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: errcode
```

OUTPUT PARAMETERS:

```
character(len=80) :: zio_uf_strerror
```

6.4.10 zio_uf_write_double

INTERFACE:

```
    integer function zio_uf_write_double(filedesc,distarray,arr,    &
  &                                       rec,istart,icount,n,      &
  &                                       fillvalue)
```

DESCRIPTION:

Unformatted write

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-14 -- added partial support of filling dataset
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc, distarray
real(zio_r8), intent(in) :: arr(*)
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
                                           ! in local index order
integer, optional, intent(in) :: n         ! # of elements in case of
                                           !  replicated array
real(zio_r8), optional, intent(in) :: fillvalue
```

6.4.11 zio_uf_write_double_name

INTERFACE:

```
    integer function zio_uf_write_double_name(filedesc,varname,arr, &
  &                                          rec,istart,icount,n,  &
  &                                          fillvalue)
```

DESCRIPTION:

Unformatted write for variable with variable name of '`varname`'. The distributed array descriptor for the variable must have been added to the descriptor list with the name of '`varname`'. See `zio_add_to_list`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-14 -- added partial support of filling dataset
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc
character(len=*), intent(in) :: varname
real(zio_r8), intent(in) :: arr(*)
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
```

```
                                              ! in local index order
      integer, optional, intent(in) :: n         ! # of elements in case of
                                              !  replicated array
      real(zio_r8), optional, intent(in) :: fillvalue
```

6.4.12 zio_uf_write_int

INTERFACE:

```
      integer function zio_uf_write_int(filedesc,distarray,arr,      &
    &                                   rec,istart,icount,n,fillvalue)
```

DESCRIPTION:

Unformatted write

REVISION HISTORY:

```
    2003-Jan-31 -- first prototype
```

USES:

```
      implicit none
```

INPUT PARAMETERS:

```
      integer, intent(in):: filedesc, distarray
      integer(zio_i4), intent(in) :: arr(*)
      integer, optional, intent(in) :: rec       ! global beg record number
                                              ! (1-based)
      integer, optional, intent(in) :: istart(:), icount(:)
                                              ! in local index order
      integer, optional, intent(in) :: n         ! # of elements in case of
                                              !  replicated array
      integer(zio_i4), optional, intent(in) :: fillvalue
```

6.4.13 zio_uf_write_int_name

INTERFACE:

```
      integer function zio_uf_write_int_name(filedesc,varname,arr,    &
    &                                        rec,istart,icount,n,     &
    &                                        fillvalue)
```

DESCRIPTION:

Unformatted write for variable with variable name of '`varname`'. The distributed array descriptor for the variable must have been added to the descriptor list with the name of '`varname`'. See `zio_add_to_list`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-14 -- added partial support of filling dataset
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc
character(len=*), intent(in) :: varname
integer(zio_i4), intent(in) :: arr(*)
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
                                           ! in local index order
integer, optional, intent(in) :: n         ! # of elements in case of
                                           !  replicated array
integer(zio_i4), optional, intent(in) :: fillvalue
```

6.4.14 zio_uf_write_real

INTERFACE:

```
     integer function zio_uf_write_real(filedesc,distarray,arr,      &
&                                      rec,istart,icount,n,        &
&                                      fillvalue)
```

DESCRIPTION:

Unformatted write

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc, distarray
real(zio_r4), intent(in) :: arr(*)
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
                                           ! in local index order
integer, optional, intent(in) :: n         ! # of elements in case of
                                           !  replicated array
real(zio_r4), optional, intent(in) :: fillvalue
```

6.4.15 zio_uf_write_real_name

INTERFACE:

```
   integer function zio_uf_write_real_name(filedesc,varname,arr,   &
&                                          rec,istart,icount,n,    &
&                                          fillvalue)
```

DESCRIPTION:

Unformatted write for variable with variable name of '`varname`'. The distributed array descriptor for the variable must have been added to the descriptor list with the name of '`varname`'. See `zio_add_to_list`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-14 -- added partial support of filling dataset
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in):: filedesc
character(len=*), intent(in) :: varname
real(zio_r4), intent(in) :: arr(*)
integer, optional, intent(in) :: rec       ! global beg record number
                                           ! (1-based)
integer, optional, intent(in) :: istart(:), icount(:)
                                           ! in local index order
integer, optional, intent(in) :: n         ! # of elements in case of
                                           !  replicated array
real(zio_r4), optional, intent(in) :: fillvalue
```

6.5 Module zio_netcdf77 (Source File: zio_netcdf77.F90)

Module `zio_netcdf77` is a collection of wrappers for routines of the netCDF Fortran 77 interface.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Mar-21 -- added ZIO_INCLUDE_NETCDF cpp flag
2003-Aug-14 -- added partial support of prefilling netCDF dataset
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INTERFACE:

```
      module zio_netcdf77
#if ( defined ZIO_INCLUDE_NETCDF)
```

USES:

```
        use zio_data
        use zio_remap
        implicit none
#include <netcdf.inc>
```

6.5.1 zio_nf__comm

INTERFACE:

```
        integer function zio_nf__comm(iocomm)
```

DESCRIPTION:

Sets the communicator for netCDF I/O to the communicator associated with I/O staging communicator object `iocomm`. Specific to the parallel netCDF library being developed at NERSC.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in) :: iocomm       ! I/O staging comm object
```

6.5.2 zio_nf__create

INTERFACE:

```
      integer function zio_nf__create(path,cmode,initialsize,        &
   &                                chunksize,iocomm,filedesc)
```

DESCRIPTION:

Creates a netCDF file using I/O staging communicator object `iocomm`. Returns the created file descriptor in `filedesc`. Specific to the parallel netCDF library being developed at NERSC.

REVISION HISTORY:

```
  2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
      implicit none
      character(len=*), intent(in):: path
      integer, intent(in) :: cmode        ! cmode in netCDF format
      integer, intent(in) :: initialsize  ! initial size
      integer, intent(in) :: chunksize    ! chunk size
      integer, intent(in) :: iocomm       ! I/O staging comm object
```

OUTPUT PARAMETERS:

```
      integer, intent(out):: filedesc
```

6.5.3 zio_nf__create_mp

INTERFACE:

```
      integer function zio_nf__create_mp(path,cmode,initialsize,     &
   &                                   basepe,chunksize,iocomm,     &
   &                                   filedesc)
```

DESCRIPTION:

Creates a netCDF file using I/O staging communicator object `iocomm`. Returns the created file descriptor in `filedesc`. Specific to the parallel netCDF library being developed at NERSC.

REVISION HISTORY:

```
  2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
character(len=*), intent(in):: path
integer, intent(in) :: cmode          ! cmode in netCDF format
integer, intent(in) :: initialsize    ! initial size
integer, intent(in) :: basepe         ! base pe
integer, intent(in) :: chunksize      ! chunk size
integer, intent(in) :: iocomm         ! I/O staging comm object
```

OUTPUT PARAMETERS:

```
integer, intent(out):: filedesc
```

6.5.4 zio_nf__open

INTERFACE:

```
     integer function zio_nf__open(path,omode,chunksize,iocomm,      &
  &                                 filedesc)
```

DESCRIPTION:

Opens a netCDF file using I/O staging communicator object `iocomm`. Returns the created file descriptor in `filedesc`. Specific to the parallel netCDF library being developed at NERSC.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
character(len=*), intent(in):: path
integer, intent(in) :: omode          ! omode in netCDF format
integer, intent(in) :: chunksize      ! chunk size
integer, intent(in) :: iocomm         ! I/O staging comm object
```

OUTPUT PARAMETERS:

```
integer, intent(out):: filedesc
```

6.5.5 zio_nf__open_mp

INTERFACE:

```
        integer function zio_nf__open_mp(path,omode,basepe,chunksize,   &
     &                                   iocomm,filedesc)
```

DESCRIPTION:

Opens a netCDF file using I/O staging communicator object `iocomm`. Returns the created file descriptor in `filedesc`. Specific to the parallel netCDF library being developed at NERSC.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
        implicit none
        character(len=*), intent(in):: path
        integer, intent(in) :: omode          ! omode in netCDF format
        integer, intent(in) :: basepe         ! base pe
        integer, intent(in) :: chunksize      ! chunk size
        integer, intent(in) :: iocomm         ! I/O staging comm object
```

OUTPUT PARAMETERS:

```
        integer, intent(out):: filedesc
```

6.5.6 zio_nf_abort

INTERFACE:

```
        integer function zio_nf_abort(filedesc)
```

DESCRIPTION:

Backs out of recent definitions.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in):: filedesc
```

6.5.7 zio_nf_close

INTERFACE:

```
integer function zio_nf_close(filedesc)
```

DESCRIPTION:

Closes a netCDF file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```

6.5.8 zio_nf_copy_att

INTERFACE:

```
    integer function zio_nf_copy_att(filedesc_in,varid_in,name,     &
&                                     filedesc_out,varid_out)
```

DESCRIPTION:

Copys attribute

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc_in
integer, intent(in):: varid_in
character(len=*), intent(in) :: name
integer, intent(in):: filedesc_out
integer, intent(in):: varid_out
```

6.5.9 zio_nf_create

INTERFACE:

```
integer function zio_nf_create(path,cmode,iocomm,filedesc)
```

DESCRIPTION:

Creates a netCDF file using I/O staging communicator object `iocomm`. Returns the created file descriptor in `filedesc`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
character(len=*), intent(in):: path
integer, intent(in) :: cmode         ! cmode in netCDF format
integer, intent(in) :: iocomm        ! I/O comm obj
```

OUTPUT PARAMETERS:

```
integer, intent(out):: filedesc      ! created filedesc
```

6.5.10 zio_nf_def_dim

INTERFACE:

```
integer function zio_nf_def_dim(filedesc,name,length,dimid)
```

DESCRIPTION:

Defines a netCDF dimension.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: length
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer, intent(out):: dimid
```

6.5.11 zio_nf_def_var

INTERFACE:

```
        integer function zio_nf_def_var(filedesc,name,xtype,nvdims,     &
     &                                  vdims,varid)
```

DESCRIPTION:

Defines a netCDF variable.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in):: filedesc
        integer, intent(in):: xtype
        integer, intent(in):: nvdims
        integer, intent(in):: vdims(*)
        character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
        integer, intent(out)::varid
```

6.5.12 zio_nf_del_att

INTERFACE:

```
        integer function zio_nf_del_att(filedesc,varid,name)
```

DESCRIPTION:

Deletes an attribute for a netCDF variable.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in):: filedesc
        integer, intent(in):: varid
        character(len=*), intent(in):: name
```

6.5.13 zio_nf_enddef

INTERFACE:

```
integer function zio_nf_enddef(filedesc)
```

DESCRIPTION:

Ends the define mode.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```

6.5.14 zio_nf_get_att_double

INTERFACE:

```
    integer function zio_nf_get_att_double(filedesc,varid,name,     &
&                                             dvals)
```

DESCRIPTION:

Returns the `real*8` attribute values for a netCDF variable.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-16 -- attribute value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
real(zio_r8), intent(out) :: dvals(*)
```

6.5.15 zio_nf_get_att_int

INTERFACE:

```
integer function zio_nf_get_att_int(filedesc,varid,name,ivals)
```

DESCRIPTION:

Returns the `integer*4` attribute values for a netCDF variable.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-16 -- attribute value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer(zio_i4), intent(out) :: ivals(*)
```

6.5.16 zio_nf_get_att_real

INTERFACE:

```
integer function zio_nf_get_att_real(filedesc,varid,name,rvals)
```

DESCRIPTION:

Returns the `real*4` attribute values for a netCDF variable.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-16 -- attribute value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
         implicit none
         integer, intent(in):: filedesc
         integer, intent(in):: varid
         character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
#ifdef ZIO_CRAY
         real(zio_r8), intent(out) :: rvals(*)
#else
         real(zio_r4), intent(out) :: rvals(*)
#endif
```

6.5.17 zio_nf_get_att_text

INTERFACE:

```
         integer function zio_nf_get_att_text(filedesc,varid,name,text)
```

DESCRIPTION:

Returns the attribute text from the given variable ID and attribute name.

REVISION HISTORY:

```
       2003-Jan-31 -- first prototype
       2003-Aug-13 -- minor change regarding broadcasting strategy
       2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
       2003-Sep-16 -- attribute value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
         implicit none
         integer, intent(in):: filedesc
         integer, intent(in):: varid
         character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
         character(len=*), intent(out):: text
```

6.5.18 zio_nf_get_var_double

INTERFACE:

```
      integer function zio_nf_get_var_double(filedesc,varid,dvals,    &
     &                                          distarray)
```

DESCRIPTION:

Gets the given `real*8` variable from an input file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: varid
integer, optional, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
real(zio_r8), intent(out) :: dvals(*)
```

6.5.19 zio_nf_get_var_int

INTERFACE:

```
      integer function zio_nf_get_var_int(filedesc,varid,ivals,       &
     &                                       distarray)
```

DESCRIPTION:

Gets the given `integer*4` variable from an input file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: varid
integer, optional, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer(zio_i4), intent(out) :: ivals(*)
```

6.5.20 zio_nf_get_var_real

INTERFACE:

```
        integer function zio_nf_get_var_real(filedesc,varid,rvals,      &
    &                                       distarray)
```

DESCRIPTION:

Gets the given `real*4` variable from an input file.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
      2003-Aug-13 -- minor change regarding broadcasting strategy
      2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in) :: filedesc
        integer, intent(in) :: varid
        integer, optional, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
#ifdef ZIO_CRAY
        real(zio_r8), intent(out) :: rvals(*)
#else
        real(zio_r4), intent(out) :: rvals(*)
#endif
```

6.5.21 zio_nf_get_var_text

INTERFACE:

```
        integer function zio_nf_get_var_text(filedesc,varid,text)
```

DESCRIPTION:

Gets the given text variable from an input file. Note that `text` is not distributed.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
      2003-Aug-13 -- minor change regarding broadcasting strategy
      2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: varid
```

OUTPUT PARAMETERS:

```
character(len=*), intent(out) :: text
```

6.5.22 zio_nf_get_var1_double

INTERFACE:

```
    integer function zio_nf_get_var1_double(filedesc,varid,indx,    &
 &                                           dval)
```

DESCRIPTION:

Gets the given `real*8` variable from the specified location of an input file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: indx(*)
integer, intent(in) :: varid
```

OUTPUT PARAMETERS:

```
real(zio_r8), intent(out) :: dval
```

6.5.23 zio_nf_get_var1_int

INTERFACE:

```
integer function zio_nf_get_var1_int(filedesc,varid,indx,ival)
```

DESCRIPTION:

Gets the given `integer*4` variable from the specified location of an input file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: indx(*)
integer, intent(in) :: varid
```

OUTPUT PARAMETERS:

```
integer(zio_i4), intent(out) :: ival
```

6.5.24 zio_nf_get_var1_real

INTERFACE:

```
integer function zio_nf_get_var1_real(filedesc,varid,indx,rval)
```

DESCRIPTION:

Gets the given `real*4` variable from the specified location in of an input file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: indx(*)
integer, intent(in) :: varid
```

OUTPUT PARAMETERS:

```
#ifdef ZIO_CRAY
        real(zio_r8), intent(out) :: rval
#else
        real(zio_r4), intent(out) :: rval
#endif
```

6.5.25 zio_nf_get_var1_text

INTERFACE:

```
integer function zio_nf_get_var1_text(filedesc,varid,indx,chval)
```

DESCRIPTION:

Gets the given text variable from the specified location of an input file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: indx(*)
integer, intent(in) :: varid
```

OUTPUT PARAMETERS:

```
character, intent(out) :: chval
```

6.5.26 zio_nf_get_vara_double

INTERFACE:

```
    integer function zio_nf_get_vara_double(filedesc,varid,istart,  &
 &                                          icount,dvals,distarray)
```

DESCRIPTION:

Gets a range of the given `real*8` variable from an input file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Feb-20 -- istart and icount changed to assumed-size arrays
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: varid
integer, intent(in) :: istart(*), icount(*)  ! in output index order
integer, optional, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
real(zio_r8), intent(out) :: dvals(*)
```

REMARKS:

```
istart and icount used to refer only to the array section in the
netCDF data space in disk, but not array's memory space.  However,
currently we let istart and icount refer to an array section in BOTH
the netCDF data and memory space.  That is, an array section in
netCDF data space is written from or read into the corresponding
section of the global array.
```

6.5.27 zio_nf_get_vara_int

INTERFACE:

```
      integer function zio_nf_get_vara_int(filedesc,varid,istart,     &
    &                                          icount,ivals,distarray)
```

DESCRIPTION:

Gets a range of the given `integer*4` variable from an input file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Feb-20 -- istart and icount changed to assumed-size arrays
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: varid
integer, intent(in) :: istart(*), icount(*)  ! in output index order
integer, optional, intent(in) :: distarray
```

OUTPUT PARAMETERS:

```
integer(zio_i4), intent(out) :: ivals(*)
```

REMARKS:

```
istart and icount used to refer only to the array section in the
netCDF data space in disk, but not array's memory space.  However,
currently we let istart and icount refer to an array section in BOTH
the netCDF data and memory space.  That is, an array section in
netCDF data space is written from or read into the corresponding
section of the global array.
```

6.5.28 zio_nf_get_vara_real

INTERFACE:

```
        integer function zio_nf_get_vara_real(filedesc,varid,istart,    &
     &                                       icount,rvals,distarray)
```

DESCRIPTION:

Gets a range of the given `real*4` variable from an input file.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
      2003-Feb-20 -- istart and icount changed to assumed-size arrays
      2003-Aug-13 -- minor change regarding broadcasting strategy
      2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in) :: filedesc
        integer, intent(in) :: varid
        integer, intent(in) :: istart(*), icount(*)  ! in output index order
        integer, optional, intent(in) :: distarray
```

INPUT PARAMETERS:

```
#ifdef ZIO_CRAY
        real(zio_r8), intent(out) :: rvals(*)
#else
        real(zio_r4), intent(out) :: rvals(*)
#endif
```

REMARKS:

```
istart and icount used to refer only to the array section in the
netCDF data space in disk, but not array's memory space.  However,
currently we let istart and icount refer to an array section in BOTH
the netCDF data and memory space.  That is, an array section in
netCDF data space is written from or read into the corresponding
section of the global array.
```

6.5.29 zio_nf_get_vara_text

INTERFACE:

```
      integer function zio_nf_get_vara_text(filedesc,varid,istart,    &
   &                                        icount,text)
```

DESCRIPTION:

Gets a range of the given text variable from an input file. Note that `text` is not distributed.

REVISION HISTORY:

```
    2003-Jan-31 -- first prototype
    2003-Aug-13 -- minor change regarding broadcasting strategy
    2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
      implicit none
      integer, intent(in) :: filedesc
      integer, intent(in) :: varid
      integer, intent(in) :: istart(*), icount(*)
```

OUTPUT PARAMETERS:

```
      character(len=*), intent(out) :: text
```

6.5.30 zio_nf_inq

INTERFACE:

```
      integer function zio_nf_inq(filedesc,ndims,nvars,ngatts,        &
   &                              unlimdimid)
```

DESCRIPTION:

Gets number of dimensions, number of variables, number of attributes and unlimited dimension ID.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out):: ndims
integer, intent(out):: nvars
integer, intent(out):: ngatts
integer, intent(out):: unlimdimid
```

6.5.31 zio_nf_inq_att

INTERFACE:

```
    integer function zio_nf_inq_att(filedesc,varid,name,xtype,      &
  &                                  length)
```

DESCRIPTION:

Gets information about an attribute.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer, intent(out):: xtype
integer, intent(out):: length
```

6.5.32 zio_nf_inq_attid

INTERFACE:

```
integer function zio_nf_inq_attid(filedesc,varid,name,attnum)
```

DESCRIPTION:

Gets the attribute number.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer, intent(out):: attnum
```

6.5.33 zio_nf_inq_attlen

INTERFACE:

```
integer function zio_nf_inq_attlen(filedesc,varid,name,length)
```

DESCRIPTION:

Gets the number of values (or characters) stored for an attribute.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer, intent(out):: length
```

6.5.34 zio_nf_inq_attname

INTERFACE:

```
integer function zio_nf_inq_attname(filedesc,varid,attnum,name)
```

DESCRIPTION:

Gets attribute name.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
integer, intent(in):: attnum
```

OUTPUT PARAMETERS:

```
character(len=*), intent(out):: name
```

6.5.35 zio_nf_inq_atttype

INTERFACE:

```
integer function zio_nf_inq_atttype(filedesc,varid,name,xtype)
```

DESCRIPTION:

Gets the attribute type.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer, intent(out):: xtype
```

6.5.36 zio_nf_inq_base_pe

INTERFACE:

```
integer function zio_nf_inq_base_pe(filedesc,basepe)
```

DESCRIPTION:

Gets the base PE. Specific to the parallel netCDF library being developed at NERSC.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out):: basepe
```

6.5.37 zio_nf_inq_dim

INTERFACE:

```
integer function zio_nf_inq_dim(filedesc,dimid,name,length)
```

DESCRIPTION:

Gets dimension name and length for a given dimension ID.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: dimid
```

OUTPUT PARAMETERS:

```
integer, intent(out):: length
character(len=*), intent(out):: name
```

6.5.38 zio_nf_inq_dimid

INTERFACE:

```
integer function zio_nf_inq_dimid(filedesc,name,dimid)
```

DESCRIPTION:

Gets the dimension ID for a given dimension name.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer, intent(out):: dimid
```

6.5.39 zio_nf_inq_dimlen

INTERFACE:

```
integer function zio_nf_inq_dimlen(filedesc,dimid,length)
```

DESCRIPTION:

Gets the dimension length for a given dimension.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in)::  dimid
```

OUTPUT PARAMETERS:

```
integer, intent(out):: length
```

6.5.40 zio_nf_inq_dimname

INTERFACE:

```
integer function zio_nf_inq_dimname(filedesc,dimid,name)
```

DESCRIPTION:

Gets dimension name for a given dimension ID.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: dimid
```

OUTPUT PARAMETERS:

```
character(len=*), intent(out):: name
```

6.5.41 zio_nf_inq_libvers

INTERFACE:

```
function zio_nf_inq_libvers()
```

DESCRIPTION:

Get netCDF library version.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

OUTPUT PARAMETERS:

```
implicit none
character(len=80) :: zio_nf_inq_libvers
```

6.5.42 zio_nf_inq_natts

INTERFACE:

```
integer function zio_nf_inq_natts(filedesc,ngatts)
```

DESCRIPTION:

Returns the number of global attributes.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out):: ngatts
```

6.5.43 zio_nf_inq_ndims

INTERFACE:

```
integer function zio_nf_inq_ndims(filedesc,ndims)
```

DESCRIPTION:

Returns the number of dimensions.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out):: ndims
```

6.5.44 zio_nf_inq_nvars

INTERFACE:

```
integer function zio_nf_inq_nvars(filedesc,nvars)
```

DESCRIPTION:

Returns the number of variables.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out):: nvars
```

6.5.45 zio_nf_inq_unlimdim

INTERFACE:

```
integer function zio_nf_inq_unlimdim(filedesc,unlimdimid)
```

DESCRIPTION:

Returns the unlimited dimension ID.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```

OUTPUT PARAMETERS:

```
integer, intent(out):: unlimdimid
```

6.5.46 zio_nf_inq_var

INTERFACE:

```
    integer function zio_nf_inq_var(filedesc,varid,name,xtype,      &
&                                   ndims,dimids,natts)
```

DESCRIPTION:

Returns the variable name, type, number of dimensions, dimension ID's, and number of attributes.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
```

OUTPUT PARAMETERS:

```
integer, intent(out):: xtype
integer, intent(out):: ndims
integer, intent(out):: dimids(*)
integer, intent(out):: natts
character(len=*), intent(out):: name
```

6.5.47 zio_nf_inq_vardimid

INTERFACE:

```
integer function zio_nf_inq_vardimid(filedesc,varid,dimids)
```

DESCRIPTION:

Returns the dimension ID's from a variable.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
```

OUTPUT PARAMETERS:

```
integer, intent(out)::  dimids(*)
```

6.5.48 zio_nf_inq_varid

INTERFACE:

```
integer function zio_nf_inq_varid(filedesc,name,varid)
```

DESCRIPTION:

Returns the variable ID for a given variable name.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer, intent(out):: varid
```

6.5.49 zio_nf_inq_varname

INTERFACE:

```
integer function zio_nf_inq_varname(filedesc,varid,name)
```

DESCRIPTION:

Returns the variable name from the variable ID.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
```

OUTPUT PARAMETERS:

```
character(len=*), intent(out):: name
```

6.5.50 zio_nf_inq_varnatts

INTERFACE:

```
integer function zio_nf_inq_varnatts(filedesc,varid,natts)
```

DESCRIPTION:

Returns the number of attributes from the variable ID.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
```

OUTPUT PARAMETERS:

```
integer, intent(out):: natts
```

6.5.51 zio_nf_inq_varndims

INTERFACE:

```
integer function zio_nf_inq_varndims(filedesc,varid,ndims)
```

DESCRIPTION:

Returns the number of variable dimensions from the variable ID.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
```

OUTPUT PARAMETERS:

```
integer, intent(out):: ndims
```

6.5.52 zio_nf_inq_vartype

INTERFACE:

```
integer function zio_nf_inq_vartype(filedesc,varid,xtype)
```

DESCRIPTION:

Returns the variable type from the variable ID.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
2003-Sep-03 -- return value for inquiry broadcasted to all PEs
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
```

OUTPUT PARAMETERS:

```
integer, intent(out):: xtype
```

6.5.53 zio_nf_open

INTERFACE:

```
integer function zio_nf_open(path,omode,iocomm,filedesc)
```

DESCRIPTION:

Opens a netCDF file using I/O staging communicator object `iocomm`. Returns the created file descriptor in `filedesc`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
character(len=*), intent(in):: path
integer, intent(in) :: omode          ! omode in netCDF format
integer, intent(in) :: iocomm         ! I/O staging comm object
```

OUTPUT PARAMETERS:

```
integer, intent(out):: filedesc
```

6.5.54 zio_nf_put_att_double

INTERFACE:

```
        integer function zio_nf_put_att_double(filedesc,varid,name,     &
     &                                         xtype,length,dvals)
```

DESCRIPTION:

Puts the given `real*8` attribute values to variable ID.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in):: filedesc
        integer, intent(in):: varid
        character(len=*), intent(in):: name
        integer, intent(in):: xtype
        integer, intent(in):: length
        real(zio_r8), intent(in):: dvals(*)
```

6.5.55 zio_nf_put_att_int

INTERFACE:

```
        integer function zio_nf_put_att_int(filedesc,varid,name,        &
     &                                      xtype,length,ivals)
```

DESCRIPTION:

Puts the given `integer*4` attribute value to variable ID.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in):: filedesc
        integer, intent(in):: varid
        character(len=*), intent(in):: name
        integer, intent(in):: xtype
        integer, intent(in):: length
        integer(zio_i4), intent(in):: ivals(*)
```

6.5.56 zio_nf_put_att_real

INTERFACE:

```
        integer function zio_nf_put_att_real(filedesc,varid,name,       &
     &                                     xtype,length,rvals)
```

DESCRIPTION:

Puts the given `real*4` attribute value to variable ID.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in):: filedesc
        integer, intent(in):: varid
        character(len=*), intent(in):: name
        integer, intent(in):: xtype
        integer, intent(in):: length
#ifdef ZIO_CRAY
        real(zio_r8), intent(in):: rvals(*)
#else
        real(zio_r4), intent(in):: rvals(*)
#endif
```

6.5.57 zio_nf_put_att_text

INTERFACE:

```
        integer function zio_nf_put_att_text(filedesc,varid,name,       &
     &                                     length,text)
```

DESCRIPTION:

Puts the given attribute text to variable ID.

REVISION HISTORY:

```
      2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
character(len=*), intent(in):: name
integer, intent(in):: length
character(len=*), intent(in):: text
```

6.5.58 zio_nf_put_var_double

INTERFACE:

```
    integer function zio_nf_put_var_double(filedesc,varid,dvals,    &
&                                               distarray)
```

DESCRIPTION:

Puts the given `real*8` variable to an output file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: varid
real(zio_r8), intent(in) :: dvals(*)
integer, optional, intent(in) :: distarray
```

6.5.59 zio_nf_put_var_int

INTERFACE:

```
    integer function zio_nf_put_var_int(filedesc,varid,ivals,       &
&                                            distarray)
```

DESCRIPTION:

Puts the given `integer*4` variable to an output file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
    implicit none
    integer, intent(in) :: filedesc
    integer, intent(in) :: varid
    integer(zio_i4), intent(in) :: ivals(*)
    integer, optional, intent(in) :: distarray
```

6.5.60 zio_nf_put_var_real

INTERFACE:

```
    integer function zio_nf_put_var_real(filedesc,varid,rvals,      &
  &                                         distarray)
```

DESCRIPTION:

Puts the given `real*4` variable to an output file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-13 -- minor change regarding broadcasting strategy
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
    implicit none
    integer, intent(in) :: filedesc
    integer, intent(in) :: varid
#ifdef ZIO_CRAY
    real(zio_r8), intent(in) :: rvals(*)
#else
    real(zio_r4), intent(in) :: rvals(*)
#endif
```

6.5.61 zio_nf_put_var_text

INTERFACE:

```
    integer function zio_nf_put_var_text(filedesc,varid,text)
```

DESCRIPTION:

Puts the given text variable to an output file. Note that `text` is not distributed.

REVISION HISTORY:

```
  2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
    implicit none
    integer, intent(in) :: filedesc
    integer, intent(in) :: varid
    character(len=*), intent(in) :: text
```

6.5.62 zio_nf_put_var1_double

INTERFACE:

```
    integer function zio_nf_put_var1_double(filedesc,varid,indx,    &
 &                                          dval)
```

DESCRIPTION:

Puts the given `real*8` variable at the specified location in an output file.

REVISION HISTORY:

```
  2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
    implicit none
    integer, intent(in) :: filedesc
    integer, intent(in) :: indx(*)
    integer, intent(in) :: varid
    real(zio_r8), intent(in) :: dval
```

6.5.63 zio_nf_put_var1_int

INTERFACE:

```
    integer function zio_nf_put_var1_int(filedesc,varid,indx,ival)
```

DESCRIPTION:

Puts the given `integer*4` variable at the specified location in an output file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
    implicit none
    integer, intent(in) :: filedesc
    integer, intent(in) :: indx(*)
    integer, intent(in) :: varid
    integer(zio_i4), intent(in) :: ival
```

6.5.64 zio_nf_put_var1_real

INTERFACE:

```
    integer function zio_nf_put_var1_real(filedesc,varid,indx,rval)
```

DESCRIPTION:

Puts the given `real*4` variable at the specified location in an output file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
    implicit none
    integer, intent(in) :: filedesc
    integer, intent(in) :: indx(*)
    integer, intent(in) :: varid
#ifdef ZIO_CRAY
    real(zio_r8), intent(in) :: rval
#else
    real(zio_r4), intent(in) :: rval
#endif
```

6.5.65 zio_nf_put_var1_text

INTERFACE:

```
integer function zio_nf_put_var1_text(filedesc,varid,indx,chval)
```

DESCRIPTION:

Puts the given character at the specified location in an output file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: indx(*)
integer, intent(in) :: varid
character, intent(in) :: chval
```

6.5.66 zio_nf_put_vara_double

INTERFACE:

```
    integer function zio_nf_put_vara_double(filedesc,varid,istart,  &
 &                                          icount,dvals,distarray, &
 &                                          fillvalue)
```

DESCRIPTION:

Puts a range of the given `real*8` variable to an output file.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Feb-20 -- istart and icount changed to assumed-size arrays
2003-Aug-14 -- added partial support of prefilling netCDF dataset
2003-Aug-16 -- support single task mode: ZIO_SPMD undefined
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: varid
integer, intent(in) :: istart(*), icount(*)  ! in output index order
real(zio_r8), intent(in) :: dvals(*)
integer, optional, intent(in) :: distarray
real(zio_r8), optional, intent(in) :: fillvalue
```

REMARKS:

```
istart and icount used to refer only to the array section in the
netCDF data space in disk, but not array's memory space.  However,
currently we let istart and icount refer to an array section in BOTH
the netCDF data and memory space.  That is, an array section in
netCDF data space is written from or read into the corresponding
section of the global array.
```

6.5.67 zio_nf_put_vara_int

INTERFACE:

```
      integer function zio_nf_put_vara_int(filedesc,varid,istart,     &
   &                                       icount,ivals,distarray,    &
   &                                       fillvalue)
```

DESCRIPTION:

Puts a range of the given `integer*4` variable to an output file. Closes a netCDF file

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Feb-20 -- istart and icount changed to assumed-size arrays
2003-Aug-14 -- added partial support of prefilling netCDF dataset
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: filedesc
integer, intent(in) :: varid
integer, intent(in) :: istart(*), icount(*)  ! in output index order
integer(zio_i4), intent(in) :: ivals(*)
integer, optional, intent(in) :: distarray
integer(zio_i4), optional, intent(in) :: fillvalue
```

REMARKS:

```
istart and icount used to refer only to the array section in the
netCDF data space in disk, but not array's memory space.  However,
currently we let istart and icount refer to an array section in BOTH
the netCDF data and memory space.  That is, an array section in
netCDF data space is written from or read into the corresponding
section of the global array.
```

6.5.68 zio_nf_put_vara_real

INTERFACE:

```
        integer function zio_nf_put_vara_real(filedesc,varid,istart,    &
     &                                        icount,rvals,distarray,   &
     &                                        fillvalue)
```

DESCRIPTION:

Puts a range of the given `real*4` variable to an output file.

REVISION HISTORY:

```
       2003-Jan-31 -- first prototype
       2003-Feb-20 -- istart and icount changed to assumed-size arrays
       2003-Aug-14 -- added partial support of prefilling netCDF dataset
```

INPUT PARAMETERS:

```
        implicit none
        integer, intent(in) :: filedesc
        integer, intent(in) :: varid
        integer, intent(in) :: istart(*), icount(*)  ! in output index order
#ifdef ZIO_CRAY
        real(zio_r8), intent(in) :: rvals(*)
        real(zio_r8), optional, intent(in) :: fillvalue
#else
        real(zio_r4), intent(in) :: rvals(*)
        real(zio_r4), optional, intent(in) :: fillvalue
#endif
        integer, optional, intent(in) :: distarray
```

REMARKS:

```
  istart and icount used to refer only to the array section in the
  netCDF data space in disk, but not array's memory space.  However,
  currently we let istart and icount refer to an array section in BOTH
  the netCDF data and memory space.  That is, an array section in
  netCDF data space is written from or read into the corresponding
  section of the global array.
```

6.5.69 zio_nf_put_vara_text

INTERFACE:

```
          integer function zio_nf_put_vara_text(filedesc,varid,istart,    &
      &                                         icount,text)
```

DESCRIPTION:

Puts a range of the given text variable to an output file. Note that `text` is not distributed.

REVISION HISTORY:

```
        2003-Jan-31 -- first prototype
        2003-Feb-20 -- istart and icount changed to assumed-size arrays
```

INPUT PARAMETERS:

```
          implicit none
          integer, intent(in) :: filedesc
          integer, intent(in) :: varid
          integer, intent(in) :: istart(*), icount(*)
          character(len=*), intent(in) :: text
```

6.5.70 zio_nf_redef

INTERFACE:

```
          integer function zio_nf_redef(filedesc)
```

DESCRIPTION:

Puts in the define mode.

REVISION HISTORY:

```
        2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
          implicit none
          integer, intent(in):: filedesc
```

6.5.71 zio_nf_rename_att

INTERFACE:

```
          integer function zio_nf_rename_att(filedesc,varid,name,newname)
```

DESCRIPTION:

Renames an attribute.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: varid
character(len=*), intent(in):: name
character(len=*), intent(in):: newname
```

6.5.72 zio_nf_rename_dim

INTERFACE:

```
integer function zio_nf_rename_dim(filedesc,dimid,name)
```

DESCRIPTION:

Renames a dimension.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
character(len=*), intent(in):: name
```

OUTPUT PARAMETERS:

```
integer, intent(out):: dimid
```

6.5.73 zio_nf_rename_var

INTERFACE:

```
integer function zio_nf_rename_var(filedesc,varid,newname)
```

DESCRIPTION:

Renames variable's name.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
character(len=*), intent(in):: newname
```

OUTPUT PARAMETERS:

```
integer, intent(out):: varid
```

6.5.74 zio_nf_set_base_pe

INTERFACE:

```
integer function zio_nf_set_base_pe(filedesc,basepe)
```

DESCRIPTION:

Resets the base PE. Specific to the parallel netCDF library being developed at NERSC.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: basepe
```

6.5.75 zio_nf_set_fill

INTERFACE:

```
integer function zio_nf_set_fill(filedesc,fillmode,old_mode)
```

DESCRIPTION:

Sets the fill mode.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
2003-Aug-14 -- added fill component to file descriptor object
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
integer, intent(in):: fillmode
```

OUTPUT PARAMETERS:

```
integer, intent(out):: old_mode
```

6.5.76 zio_nf_strerror

INTERFACE:

```
function zio_nf_strerror(ncerr)
```

DESCRIPTION:

Returns an error message of len=80 for netCDF I/O error `ncerr`.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

OUTPUT PARAMETERS:

```
implicit none
character(len=80) :: zio_nf_strerror
```

INPUT PARAMETERS:

```
integer, intent(in) :: ncerr
```

6.5.77 zio_nf_sync

INTERFACE:

```
integer function zio_nf_sync(filedesc)
```

DESCRIPTION:

Synchronizes netCDF dataset to disk.

REVISION HISTORY:

```
2003-Jan-31 -- first prototype
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in):: filedesc
```