```
#ifndef key_noIO
#ifndef __PARNETCDF
!-----------------------------------------------------------------------
!                 Parallel I/O for message-passing codes
!
! AUTHOR: V. Balaji (vbalaji@noaa.gov)
!         Princeton University/GFDL
!
! MODIFICATIONS: Reiner Vogelsang (reiner@sgi.com)
!                Reiner Vogelsang, Rene Redler: added pnetcdf
!                Rene Redler: Included Modification from MPI Met
!                (Luis Kornblueh, Stephanie Legutke)
!                - Replaced -huge(1.0_4) by -huge(1.0_ip_single_p)
!                - Initialisation of pointer
!                - changed x to 1x in format descriptor
!                Sophie Valcke: replaced ip_single_p with ip_single_mpp
!
!
! This program is free software; The author agrees that you can
! redistribute and/or modify this version of the program under the
! terms of the Lesser GNU General Public License as published
! by the Free Software Foundation.
!
! This program is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
! Lesser GNU General Public License for more details
! (http://www.gnu.org/copyleft/lesser.html).
!-----------------------------------------------------------------------
!#include <os.h>

module mpp_io_mod_oa
  use mod_kinds_mpp
  use mpp_mod_oa
  use mpp_domains_mod_oa
  implicit none
#include <os.h>
  private

  character(len=128), private :: version= &
       '$Id$'
  character(len=128), private :: tagname= &
       '$Name$'

  integer, private :: pe, npes

  type, public :: axistype
     private
     character(len=128) :: name
     character(len=128) :: units
     character(len=256) :: longname
     character(len=8) :: cartesian
     integer :: sense, len           !+/-1, depth or height?
     type(domain1D) :: domain !if pointer is associated, it is a distributed data axis
     real, pointer :: data(:)   !axis values (not used if time axis)
     character(len=64), pointer :: cdata(:)  !RV,bundles
     integer :: clenid                       !RV,bundles
     integer :: id, did, type, natt         !id is the "variable ID", did is the "dimensic
     type(atttype), pointer :: Att(:)
  end type axistype

  type, public :: atttype
     integer :: type, len
     character(len=128) :: name
     character(len=256) :: catt
! just use type conversion for integers
     real, pointer :: fatt(:)
  end type atttype

  type, public :: fieldtype
     private
     character(len=128) :: name
     character(len=128) :: units
     character(len=256) :: longname
     real :: min, max, missing, fill, scale, add
     integer :: pack
     type(axistype), pointer :: axes(:) !axes associated with field
!size, time_axis_index redundantly hold info already contained in axes
!it's clunky and inelegant, but required so that axes can be shared among multiple files
     integer, pointer :: size(:)
     integer :: time_axis_index
     integer :: id, type, natt, ndim
     type(atttype), pointer :: Att(:)
  end type fieldtype

  type, private :: filetype
     character(len=256) :: name
     integer :: action, format, access, threading, fileset, record, ncid
     logical :: opened, initialized, nohdrs
     integer :: time_level
     real(DOUBLE_KIND) :: time
     integer :: id             !variable ID of time axis associated with file (only one ti
     integer :: recdimid              !dim ID of time axis associated with file (only one t
!
! time axis values are stored here instead of axis%data since mpp_write
! assumes these values are not time values. Not used in mpp_write
!
     real(DOUBLE_KIND), pointer :: time_values(:)

! additional elements of filetype for mpp_read (ignored for mpp_write)
     integer :: ndim, nvar, natt  ! number of dimensions, non-dimension variables and glob
! redundant axis types stored here and in associated fieldtype
! some axes are not used by any fields, i.e. "edges"
     type(axistype), pointer  :: axis(:)
     type(fieldtype), pointer :: var(:)
     type(atttype), pointer   :: att(:)
  end type filetype

  type(axistype), public  :: default_axis !provided to users with default components
  type(fieldtype), public :: default_field !provided to users with default components
  type(atttype), public   :: default_att !provided to users with default components
!action on open
  integer, parameter, public :: MPP_WRONLY=100, MPP_RDONLY=101, MPP_APPEND=102, MPP_OVERWR
!format
  integer, parameter, public :: MPP_ASCII=200,  MPP_IEEE32=201, MPP_NATIVE=202, MPP_NETCDF
!access
  integer, parameter, public :: MPP_SEQUENTIAL=300, MPP_DIRECT=301
!threading, fileset
  integer, parameter, public :: MPP_SINGLE=400, MPP_MULTI=401, MPP_PARALLEL=401
!action on close
  integer, parameter, public :: MPP_DELETE=501, MPP_COLLECT=502

  type(filetype), private, allocatable :: mpp_file(:)
  integer, private :: records_per_pe
  integer, private :: maxunits, unit_begin, unit_end
  integer, private :: varnum=0
  integer, private :: error
  character(len=256) :: text
!null unit: returned by PEs not participating in IO after a collective call
  integer, parameter, private :: NULLUNIT=-1
  real(DOUBLE_KIND), parameter, private :: NULLTIME=-1.
#ifdef DEBUG
  logical, private :: verbose=.FALSE., debug=.TRUE., module_is_initialized=.FALSE.
#else
  logical, private :: verbose=.FALSE., debug=.FALSE., module_is_initialized=.FALSE.
#endif

  real(DOUBLE_KIND), private, allocatable :: mpp_io_stack(:)
  integer, private :: mpp_io_stack_size=0, mpp_io_stack_hwm=0

  interface mpp_write_meta
     module procedure mpp_write_meta_var
     module procedure mpp_write_meta_scalar_r
     module procedure mpp_write_meta_scalar_i
     module procedure mpp_write_meta_axis
     module procedure mpp_write_meta_field
     module procedure mpp_write_meta_global
     module procedure mpp_write_meta_global_scalar_r
     module procedure mpp_write_meta_global_scalar_i
  end interface

  interface mpp_copy_meta
     module procedure mpp_copy_meta_axis
     module procedure mpp_copy_meta_field
     module procedure mpp_copy_meta_global
  end interface

  interface mpp_write
     module procedure mpp_write_2ddecomp_r1d
     module procedure mpp_write_2ddecomp_r2d
     module procedure mpp_write_2ddecomp_r3d
     module procedure mpp_write_2ddecomp_r4d
     module procedure mpp_write_r0D
     module procedure mpp_write_r1D
     module procedure mpp_write_r2D
     module procedure mpp_write_r3D
     module procedure mpp_write_r4D
     module procedure mpp_write_axis
  end interface

  interface mpp_read
     module procedure mpp_read_2ddecomp_r1d
     module procedure mpp_read_2ddecomp_r2d
     module procedure mpp_read_2ddecomp_r3d
     module procedure mpp_read_2ddecomp_r4d
     module procedure mpp_read_r0D
     module procedure mpp_read_r1D
     module procedure mpp_read_r2D
     module procedure mpp_read_r3D
     module procedure mpp_read_r4D
  end interface

  interface mpp_get_id
     module procedure mpp_get_axis_id
     module procedure mpp_get_field_id
  end interface

  interface mpp_get_atts
     module procedure mpp_get_global_atts
     module procedure mpp_get_field_atts
     module procedure mpp_get_axis_atts
  end interface

  interface mpp_modify_meta
!     module procedure mpp_modify_att_meta
     module procedure mpp_modify_field_meta
     module procedure mpp_modify_axis_meta
  end interface

  public :: mpp_close, mpp_flush, mpp_get_iospec, mpp_get_id, mpp_get_ncid, mpp_get_unit_r
            mpp_open, mpp_set_unit_range, mpp_write, mpp_write_meta, mpp_read, mpp_get_inf
            mpp_get_fields, mpp_get_times, mpp_get_axes, mpp_copy_meta, mpp_get_recdimid,
            mpp_io_set_stack_size, mpp_get_field_index, mpp_nullify_axistype, mpp_nullify_

  private :: read_record, mpp_read_meta, lowercase

#ifdef use_netCDF
#include <netcdf.inc>
#endif

  contains

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!               mpp_io_init: initialize parallel I/O                   !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine mpp_io_init( flags, maxunit,maxresunit )
      integer, intent(in), optional :: flags, maxunit ,maxresunit
!rv
!rv I introduced the variable to indentify that the top max_reserved_units
!rv of maxunits are reserved for OASIS coupler specific things like the trace
!rv files. This variable is active only if one specifies explicitely  the
!rv argument maxunit.
      integer::max_reserved_units
!rv
!initialize IO package: initialize mpp_file array, set valid range of units for fortran IC

      if( module_is_initialized )return
      call mpp_init(flags)           !if mpp_init has been called, this call will merely r
      pe = mpp_pe()
      npes = mpp_npes()
      call mpp_domains_init(flags)

      maxunits = 64
      if( PRESENT(maxunit) )maxunits = maxunit

      max_reserved_units=5
      if( PRESENT(maxresunit) )max_reserved_units = maxresunit

      if( PRESENT(flags) )then
          debug   = flags.EQ.MPP_DEBUG
          verbose = flags.EQ.MPP_VERBOSE .OR. debug
      end if
!initialize default_field
      default_field%name = 'noname'
      default_field%units = 'nounits'
      default_field%longname = 'noname'
      default_field%id = -1
      default_field%type = -1
      default_field%natt = -1
      default_field%ndim = -1
```

```
!largest possible 4-byte reals
      default_field%min = -huge(1._ip_single_mpp)
      default_field%max =  huge(1._ip_single_mpp)
      default_field%missing = -1e36
      default_field%fill = -1e36
      default_field%scale = 0.
      default_field%add = huge(1._ip_single_mpp)
      default_field%pack = 1
      default_field%time_axis_index = -1 !this value will never match any index
      Nullify(default_field%axes)
      Nullify(default_field%size)
      Nullify(default_field%att)
! Initialize default axis
      default_axis%name = 'noname'
      default_axis%units = 'nounits'
      default_axis%longname = 'noname'
      default_axis%cartesian = 'none'
      default_axis%sense = 0
      default_axis%len = -1
      default_axis%id = -1
      default_axis%did = -1
      default_axis%type = -1
      default_axis%natt = -1
      Nullify(default_axis%data)
! Initialize default attribute
      default_att%name = 'noname'
      default_att%type = -1
      default_att%len = -1
      default_att%catt = 'none'
      Nullify(default_att%fatt)

!up to MAXUNITS fortran units and MAXUNITS netCDF units are supported
!file attributes (opened, format, access, threading, fileset) are saved against the unit r
!external handles to netCDF units are saved from maxunits+1:2*maxunits
      allocate( mpp_file(NULLUNIT:2*maxunits) ) !starts at NULLUNIT=-1, used by non-partic
      mpp_file(:)%name   = ' '
      mpp_file(:)%action    = -1
      mpp_file(:)%format    = -1
      mpp_file(:)%threading = -1
      mpp_file(:)%fileset   = -1
      mpp_file(:)%record    = -1
      mpp_file(:)%ncid      = -1
      mpp_file(:)%opened = .FALSE.
      mpp_file(:)%initialized = .FALSE.
      mpp_file(:)%time_level = 0
      mpp_file(:)%time = NULLTIME
      mpp_file(:)%id = -1
!
      mpp_file(:)%ndim = -1
      mpp_file(:)%nvar = -1
!NULLUNIT "file" is always single-threaded, open and initialized (to pass checks in mpp_wr
      mpp_file(NULLUNIT)%threading = MPP_SINGLE
      mpp_file(NULLUNIT)%opened = .TRUE.
      mpp_file(NULLUNIT)%initialized = .TRUE.
!declare the stdunits to be open
      mpp_file(stdin ())%opened = .TRUE.
      mpp_file(stdout())%opened = .TRUE.
      mpp_file(stderr())%opened = .TRUE.
      mpp_file(stdout())%opened = .TRUE.
!set range of allowed fortran unit numbers: could be compiler-dependent (should not overla
!
!rv For OASIS 3 I consider the top max_reserved_units to be excluded from
!rv the list of files ito closed during mpp_io_exit.
```

```
!rv      call mpp_set_unit_range( 7, maxunits )
      if(present(maxunit)) then
        call mpp_set_unit_range( 7, maxunits-max_reserved_units )
      else
        call mpp_set_unit_range( 7, maxunits )
      endif
!rv

      if( pe.EQ.mpp_root_pe() )then
          write( stdout(),'(/a)' )'MPP_IO module '//trim(version)
#ifdef use_netCDF
          text = NF_INQ_LIBVERS()
          write( stdout(),'(a)' )'Using netCDF library version '//trim(text)
#endif
      endif

#ifdef CRAYPVP
!we require every file to be assigned threadwise: PVPs default to global, and are reset he
      call ASSIGN( 'assign -P thread p:%', error )
#endif

      call mpp_io_set_stack_size(131072) ! default initial value
      call mpp_sync()
      module_is_initialized = .TRUE.
      return
    end subroutine mpp_io_init

    subroutine mpp_io_exit()
      integer :: unit

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_IO_EXIT: must first call
!close all open fortran units
      do unit = unit_begin,unit_end
         if( mpp_file(unit)%opened )call mpp_flushstd(unit)
      end do
      call mpp_sync()
      do unit = unit_begin,unit_end
         if( mpp_file(unit)%opened )close(unit)
      end do
#ifdef use_netCDF
!close all open netCDF units
      do unit = maxunits+1,2*maxunits
         if( mpp_file(unit)%opened )error = NF_CLOSE(mpp_file(unit)%ncid)
      end do
#endif

      call mpp_max(mpp_io_stack_hwm)

      if( pe.EQ.mpp_root_pe() )then
!          write( stdout,'(/a)' )'Exiting MPP_IO module...'
!          write( stdout,* )'MPP_IO_STACK high water mark=', mpp_io_stack_hwm
      end if
      deallocate(mpp_file)
      module_is_initialized = .FALSE.
      return
    end subroutine mpp_io_exit

    subroutine mpp_io_set_stack_size(n)
!set the mpp_io_stack variable to be at least n LONG words long
      integer, intent(in) :: n
      character(len=8) :: text

      if( n.GT.mpp_io_stack_size .AND. allocated(mpp_io_stack) )deallocate(mpp_io_stack)
      if( .NOT.allocated(mpp_io_stack) )then
          allocate( mpp_io_stack(n) )
          mpp_io_stack_size = n
          write( text,'(i8)' )n
          if( pe.EQ.mpp_root_pe() ) &
          call mpp_error( NOTE, 'MPP_IO_SET_STACK_SIZE: stack size set to '//text//'.' )
      end if

      return
    end subroutine mpp_io_set_stack_size

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                            !
!           OPENING AND CLOSING FILES: mpp_open() and mpp_close()            !
!                                                                            !
! mpp_open( unit, file, action, form, access, threading, &                   !
!           fileset, iospec, nohdrs, recl, pelist )                          !
!      integer, intent(out) :: unit                                          !
!      character(len=*), intent(in) :: file                                  !
!      integer, intent(in), optional :: action, form, access, threading,     !
!                                       fileset, recl                        !
!      character(len=*), intent(in), optional :: iospec                      !
!      logical, intent(in), optional :: nohdrs                               !
!      integer, optional, intent(in) :: pelist(:) !default ALL               !
!                                                                            !
!  unit is intent(OUT): always _returned_by_ mpp_open()                      !
!  file is the filename: REQUIRED                                            !
!    we append .nc to filename if it is a netCDF file                        !
!    we append .<pppp> to filename if fileset is private (pppp is PE number) !
!  iospec is a system hint for I/O organization                              !
!         e.g assign(1) on SGI/Cray systems.                                 !
!  if nohdrs is .TRUE. headers are not written on non-netCDF writes.         !
!  nohdrs has no effect when action=MPP_RDONLY|MPP_APPEND                    !
!                    or when form=MPP_NETCDF                                 !
! FLAGS:                                                                     !
!    action is one of MPP_RDONLY, MPP_APPEND or MPP_WRONLY                   !
!    form is one of MPP_ASCII:  formatted read/write                         !
!                   MPP_NATIVE: unformatted read/write, no conversion        !
!                   MPP_IEEE32: unformatted read/write, conversion to IEEE32 !
!                   MPP_NETCDF: unformatted read/write, conversion to netCDF !
!    access is one of MPP_SEQUENTIAL or MPP_DIRECT (ignored for netCDF)      !
!      RECL argument is REQUIRED for direct access IO                        !
!    threading is one of MPP_SINGLE or MPP_MULTI                             !
!      single-threaded IO in a multi-PE run is done by PE0                   !
!    fileset is one of MPP_MULTI and MPP_SINGLE                              !
!      fileset is only used for multi-threaded I/O                           !
!      if all I/O PEs in <pelist> use a single fileset,                      !
!              they write to the same file                                   !
!      if all I/O PEs in <pelist> use a multi  fileset,                      !
!              they each write an independent file                           !
!  recl is the record length in bytes                                        !
!  pelist is the list of I/O PEs (currently ALL)                             !
!                                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine mpp_open( unit, file, action, form, access, threading, &
                                     fileset, iospec, nohdrs, recl, pelist )
      integer, intent(out) :: unit
      character(len=*), intent(in) :: file
      integer, intent(in), optional :: action, form, access, threading, &
           fileset, recl
      character(len=*), intent(in), optional :: iospec
      logical, intent(in), optional :: nohdrs
      integer, intent(in), optional :: pelist(:) !default ALL
      character(len=16) :: act, acc, for, pos
      integer :: action_flag, form_flag, access_flag, threading_flag, fileset_flag, length
      logical :: exists
      character(len=64) :: filespec
      type(axistype) :: unlim    !used by netCDF with mpp_append

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_OPEN: must first call mp
!set flags
      action_flag = MPP_WRONLY        !default
      if( PRESENT(action) )action_flag = action
      form_flag = MPP_ASCII
      if( PRESENT(form) )form_flag = form
#ifndef use_netCDF
      if( form_flag.EQ.MPP_NETCDF ) &
           call mpp_error( FATAL, 'MPP_OPEN: To open a file with form=MPP_NETCDF, you must
#endif
      access_flag = MPP_SEQUENTIAL
      if( PRESENT(access) )access_flag = access
      threading_flag = MPP_SINGLE
      if( npes.GT.1 .AND. PRESENT(threading) )threading_flag = threading
      fileset_flag = MPP_MULTI
      if( PRESENT(fileset) )fileset_flag = fileset
      if( threading_flag.EQ.MPP_SINGLE )fileset_flag = MPP_SINGLE

!get a unit number
      if( threading_flag.EQ.MPP_SINGLE )then
          if( pe.NE.mpp_root_pe() .AND. action_flag.NE.MPP_RDONLY )then
              unit = NULLUNIT           !PEs not participating in IO from this mpp_open()
              return
          end if
      end if
      if( form_flag.EQ.MPP_NETCDF )then
          do unit = maxunits+1,2*maxunits
             if( .NOT.mpp_file(unit)%opened )exit
          end do
          if( unit.GT.2*maxunits )call mpp_error( FATAL, 'MPP_OPEN: too many open netCDF f
      else
          do unit = unit_begin, unit_end
             inquire( unit,OPENED=mpp_file(unit)%opened )
             if( .NOT.mpp_file(unit)%opened )exit
          end do
          if( unit.GT.unit_end )call mpp_error( FATAL, 'MPP_OPEN: no available units.' )
      end if

!get a filename
      text = file
      length = len(file)

!RV   I dropped the automatic file name extension. PSMILE will always
!RV   provide netcdf file names with an extension .nc or names containing .nc.
!RV      if( form_flag.EQ.MPP_NETCDF.AND. file(length-2:length) /= '.nc' ) &
!RV         text = trim(file)//'.nc'

      if( fileset_flag.EQ.MPP_MULTI )write( text,'(a,i4.4)' )trim(text)//'.', pe
      mpp_file(unit)%name = text
      if( verbose ) write (stdout(), '(a,2i3,1x,a,5i5)') &
           'MPP_OPEN: PE, unit, filename, action, format, access, threading, fileset=', &
           pe, unit, trim(mpp_file(unit)%name), action_flag, form_flag, access_flag, threa

!action: read, write, overwrite, append: act and pos are ignored by netCDF
      if( action_flag.EQ.MPP_RDONLY )then
          act = 'READ'
          pos = 'REWIND'
!          if( form_flag.EQ.MPP_NETCDF )call mpp_error( FATAL, 'MPP_OPEN: only writes are
      else if( action_flag.EQ.MPP_WRONLY .OR. action_flag.EQ.MPP_OVERWR )then
          act = 'WRITE'
          pos = 'REWIND'
      else if( action_flag.EQ.MPP_APPEND )then
          act = 'WRITE'
          pos = 'APPEND'
      else
          call mpp_error( FATAL, 'MPP_OPEN: action must be one of MPP_WRONLY, MPP_APPEND c
      end if

!access: sequential or direct: ignored by netCDF
      if( form_flag.NE.MPP_NETCDF )then
          if( access_flag.EQ.MPP_SEQUENTIAL )then
              acc = 'SEQUENTIAL'
          else if( access_flag.EQ.MPP_DIRECT )then
              acc = 'DIRECT'
              if( form_flag.EQ.MPP_ASCII )call mpp_error( FATAL, 'MPP_OPEN: formatted dire
              if( .NOT.PRESENT(recl) ) &
                   call mpp_error( FATAL, 'MPP_OPEN: recl (record length in bytes) must be
              mpp_file(unit)%record = 1
              records_per_pe = 1 !each PE writes 1 record per mpp_write
          else
              call mpp_error( FATAL, 'MPP_OPEN: access must be one of MPP_SEQUENTIAL or MP
          end if
      end if

!threading: SINGLE or MULTI
      if( threading_flag.EQ.MPP_MULTI )then
!fileset: MULTI or SINGLE (only for multi-threaded I/O
          if( fileset_flag.EQ.MPP_SINGLE )then
              if( form_flag.EQ.MPP_NETCDF .AND. act.EQ.'WRITE' ) &
                   call mpp_error( FATAL, 'MPP_OPEN: netCDF currently does not support sir

#ifdef _CRAYT3E
              call ASSIGN( 'assign -I -F global.privpos f:'//trim(mpp_file(unit)%name), er
#endif
          else if( fileset_flag.NE.MPP_MULTI )then
              call mpp_error( FATAL, 'MPP_OPEN: fileset must be one of MPP_MULTI or MPP_SI
          end if
      else if( threading_flag.NE.MPP_SINGLE )then
          call mpp_error( FATAL, 'MPP_OPEN: threading must be one of MPP_SINGLE or MPP_MUL
      end if

!apply I/O specs before opening the file
!note that -P refers to the scope of a fortran unit, which is always thread-private even i
#ifdef CRAYPVP
      call ASSIGN( 'assign -I -P thread  f:'//trim(mpp_file(unit)%name), error )
#endif
#ifdef _CRAYT3E
      call ASSIGN( 'assign -I -P private f:'//trim(mpp_file(unit)%name), error )
#endif
      if( PRESENT(iospec) )then
!iospec provides hints to the system on how to organize I/O
!on Cray systems this is done through 'assign', see assign(1) and assign(3F)
!on other systems this will be expanded as needed
!no error checks here on whether the supplied iospec is valid
#ifdef SGICRAY
          call ASSIGN( 'assign -I '//trim(iospec)//' f:'//trim(mpp_file(unit)%name), error
          if( form_flag.EQ.MPP_NETCDF )then
!for netCDF on SGI/Cray systems we pass it to the environment variable NETCDF_XFFIOSPEC
!ideally we should parse iospec, pass the argument of -F to NETCDF_FFIOSPEC, and the rest
```

```
!maybe I'll get around to it someday
!PXFSETENV is a POSIX-standard routine for setting environment variables from fortran
              call PXFSETENV( 'NETCDF_XFFIOSPEC', 0, trim(iospec), 0, 1, error )
          end if
#endif
      end if

!open the file as specified above for various formats
      if( form_flag.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          if( action_flag.EQ.MPP_WRONLY )then
              error = NF_CREATE( trim(mpp_file(unit)%name), NF_NOCLOBBER, mpp_file(unit)%n
              if( verbose ) write (stdout(), '(a,i3,i16)') 'MPP_OPEN: new netCDF file: pe,
          else if( action_flag.EQ.MPP_OVERWR )then
              error = NF_CREATE( trim(mpp_file(unit)%name), NF_CLOBBER,   mpp_file(unit)%n
              action_flag = MPP_WRONLY !after setting clobber, there is no further distinc
              if( verbose ) write (stdout(), '(a,i3,i16)') 'MPP_OPEN: overwrite netCDF fil
          else if( action_flag.EQ.MPP_APPEND )then
              error = NF_OPEN( trim(mpp_file(unit)%name), NF_WRITE, mpp_file(unit)%ncid );
!get the current time level of the file: writes to this file will be at next time level
              error = NF_INQ_UNLIMDIM( mpp_file(unit)%ncid, unlim%did )
              if( error.EQ.NF_NOERR )then
                  error = NF_INQ_DIM( mpp_file(unit)%ncid, unlim%did, unlim%name, mpp_file
                  call netcdf_err(error)
                  error = NF_INQ_VARID( mpp_file(unit)%ncid, unlim%name, mpp_file(unit)%i
              end if
              if( verbose ) write (stdout(), '(a,i3,i16,i4)') 'MPP_OPEN: append to existin
                   pe, mpp_file(unit)%ncid, mpp_file(unit)%id
          else if( action_flag.EQ.MPP_RDONLY )then
              error = NF_OPEN( trim(mpp_file(unit)%name), NF_NOWRITE, mpp_file(unit)%ncid
              if( verbose ) write (stdout(), '(a,i3,i16,i4)') 'MPP_OPEN: opening existing
                   pe, mpp_file(unit)%ncid, mpp_file(unit)%id
              mpp_file(unit)%format=form_flag ! need this for mpp_read
              call mpp_read_meta(unit)
          end if
          mpp_file(unit)%opened = .TRUE.
#endif
      else
!format: ascii, native, or IEEE 32 bit
          if( form_flag.EQ.MPP_ASCII )then
              for = 'FORMATTED'
          else if( form_flag.EQ.MPP_IEEE32 )then
              for = 'UNFORMATTED'
!assign -N is currently unsupported on SGI
#ifdef _CRAY
              call ASSIGN( 'assign -I -N ieee_32 f:'//trim(mpp_file(unit)%name), error )
#endif
          else if( form_flag.EQ.MPP_NATIVE )then
              for = 'UNFORMATTED'
          else
              call mpp_error( FATAL, 'MPP_OPEN: form must be one of MPP_ASCII, MPP_NATIVE,
          end if
          inquire( file=trim(mpp_file(unit)%name), EXIST=exists )
          if( exists .AND. action_flag.EQ.MPP_WRONLY ) &
               call mpp_error( WARNING, 'MPP_OPEN: File '//trim(mpp_file(unit)%name)//' op
          if( action_flag.EQ.MPP_OVERWR )action_flag = MPP_WRONLY
!perform the OPEN here
          if( PRESENT(recl) )then
              if( verbose ) write (stdout(), '(2(1x,a,i3),5(1x,a),a,i8)') 'MPP_OPEN: PE=',
                   'unit=', unit, trim(mpp_file(unit)%name), 'attributes=', trim(acc), tri
              open( unit, file=trim(mpp_file(unit)%name), access=acc, form=for, action=act
          else
              if( verbose ) write (stdout(), '(2(1x,a,i3),6(1x,a))')      'MPP_OPEN: PE=',
                   'unit=', unit, trim(mpp_file(unit)%name), 'attributes=', trim(acc), tri
              open( unit, file=trim(mpp_file(unit)%name), access=acc, form=for, action=act
          end if
!check if OPEN worked
          inquire( unit,OPENED=mpp_file(unit)%opened )
          if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_OPEN: error in OPEN(
      end if
      mpp_file(unit)%action = action_flag
      mpp_file(unit)%format = form_flag
      mpp_file(unit)%access = access_flag
      mpp_file(unit)%threading = threading_flag
      mpp_file(unit)%fileset = fileset_flag
      if( PRESENT(nohdrs) )mpp_file(unit)%nohdrs = nohdrs

      if( action_flag.EQ.MPP_WRONLY )then
          if( form_flag.NE.MPP_NETCDF .AND. access_flag.EQ.MPP_DIRECT )call mpp_write_meta
!actual file name
          call mpp_write_meta( unit, 'filename', cval=mpp_file(unit)%name )
!MPP_IO package version
          call mpp_write_meta( unit, 'MPP_IO_VERSION', cval=trim(version) )
!filecount for multifileset
          if( threading_flag.EQ.MPP_MULTI .AND. fileset_flag.EQ.MPP_MULTI ) &
               call mpp_write_meta( unit, 'NumFilesInSet', ival=npes )
      end if

      return
    end subroutine mpp_open

    subroutine mpp_close( unit, action )
      integer, intent(in) :: unit
      integer, intent(in), optional :: action
      character(len=8) :: status
      logical :: collect

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_CLOSE: must first call m
      if( unit.EQ.NULLUNIT )return !nothing was actually opened on this unit

!action on close
      status = 'KEEP'
!collect is supposed to launch the post-processing collector tool for multi-fileset
      collect = .FALSE.
      if( PRESENT(action) )then
          if( action.EQ.MPP_DELETE )then
              status = 'DELETE'
          else if( action.EQ.MPP_COLLECT )then
              collect = .FALSE.         !should be TRUE but this is not yet ready
              call mpp_error( WARNING, 'MPP_CLOSE: the COLLECT operation is not yet implem
          else
              call mpp_error( FATAL, 'MPP_CLOSE: action must be one of MPP_DELETE or MPP_C
          end if
      end if
      if( mpp_file(unit)%fileset.NE.MPP_MULTI )collect = .FALSE.
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          error = NF_CLOSE(mpp_file(unit)%ncid); call netcdf_err(error)
#endif
      else
          close(unit,status=status)
      end if
#ifdef SGICRAY
!this line deleted: since the FILENV is a shared file, this might cause a problem in
! multi-threaded I/O if one PE does assign -R before another one has opened it.
!      call ASSIGN( 'assign -R f:'//trim(mpp_file(unit)%name), error )
```

```
#endif
      mpp_file(unit)%name = ' '
      mpp_file(unit)%action    = -1
      mpp_file(unit)%format    = -1
      mpp_file(unit)%access    = -1
      mpp_file(unit)%threading = -1
      mpp_file(unit)%fileset   = -1
      mpp_file(unit)%record    = -1
      mpp_file(unit)%ncid      = -1
      mpp_file(unit)%opened = .FALSE.
      mpp_file(unit)%initialized = .FALSE.
      mpp_file(unit)%id = -1
      mpp_file(unit)%time_level = 0
      mpp_file(unit)%time = NULLTIME
      return
    end subroutine mpp_close

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                             !
!                             MPP_WRITE_META                                  !
!                                                                             !
!This series of routines is used to describe the contents of the file         !
!being written on <unit>. Each file can contain any number of fields,         !
!which can be functions of 0-3 spatial axes and 0-1 time axes. Axis           !
!descriptors are stored in the <axistype> structure and field                 !
!descriptors in the <fieldtype> structure.                                    !
!                                                                             !
!  type, public :: axistype                                                   !
!     sequence                                                                !
!     character(len=128) :: name                                              !
!     character(len=128) :: units                                             !
!     character(len=256) :: longname                                          !
!     integer :: sense           !+/-1, depth or height?                      !
!     type(domain1D) :: domain                                      !
!     real, pointer :: data(:) !axis values (not used if time axis)           !
!     integer :: id                                                           !
!  end type axistype                                                          !
!                                                                             !
!  type, public :: fieldtype                                                  !
!     sequence                                                                !
!     character(len=128) :: name                                              !
!     character(len=128) :: units                                             !
!     character(len=256) :: longname                                          !
!     real :: min, max, missing, fill, scale, add                             !
!     type(axistype), pointer :: axis(:)                                      !
!     integer :: id                                                           !
!  end type fieldtype                                                         !
!                                                                             !
!The metadata contained in the type is always written for each axis and       !
!field. Any other metadata one wishes to attach to an axis or field           !
!can subsequently be passed to mpp_write_meta using the ID, as shown below.   !
!                                                                             !
!mpp_write_meta can take several forms:                                       !
!                                                                             !
!  mpp_write_meta( unit, name, rval=rval, pack=pack )                         !
!  mpp_write_meta( unit, name, ival=ival )                                    !
!  mpp_write_meta( unit, name, cval=cval )                                    !
!      integer, intent(in) :: unit                                            !
!      character(len=*), intent(in) :: name                                   !
!      real, intent(in), optional :: rval(:)                                  !
!      integer, intent(in), optional :: ival(:)                               !
!      character(len=*), intent(in), optional :: cval                         !
!                                                                             !
!    This form defines global metadata associated with the file as a         !
!    whole. The attribute is named <name> and can take on a real, integer    !
!    or character value. <rval> and <ival> can be scalar or 1D arrays.       !
!                                                                            !
!  mpp_write_meta( unit, id, name, rval=rval, pack=pack )                    !
!  mpp_write_meta( unit, id, name, ival=ival )                               !
!  mpp_write_meta( unit, id, name, cval=cval )                               !
!      integer, intent(in) :: unit, id                                       !
!      character(len=*), intent(in) :: name                                  !
!      real, intent(in), optional :: rval(:)                                 !
!      integer, intent(in), optional :: ival(:)                              !
!      character(len=*), intent(in), optional :: cval                        !
!                                                                            !
!    This form defines metadata associated with a previously defined         !
!    axis or field, identified to mpp_write_meta by its unique ID <id>.      !
!    The attribute is named <name> and can take on a real, integer           !
!    or character value. <rval> and <ival> can be scalar or 1D arrays.       !
!    This need not be called for attributes already contained in             !
!    the type.                                                               !
!                                                                            !
!    PACK can take values 1,2,4,8. This only has meaning when writing        !
!    floating point numbers. The value of PACK defines the number of words   !
!    written into 8 bytes. For pack=4 and pack=8, an integer value is        !
!    written: rval is assumed to have been scaled to the appropriate dynamic !
!    range.                                                                  !
!    PACK currently only works for netCDF files, and is ignored otherwise.   !
!                                                                            !
!   subroutine mpp_write_meta_axis( unit, axis, name, units, longname, &     !
!        cartesian, sense, domain, data )                                    !
!     integer, intent(in) :: unit                                            !
!     type(axistype), intent(inout) :: axis                                  !
!     character(len=*), intent(in) :: name, units, longname                  !
!     character(len=*), intent(in), optional :: cartesian                    !
!     integer, intent(in), optional :: sense                                 !
!     type(domain1D), intent(in), optional :: domain                 !
!     real, intent(in), optional :: data(:)                                  !
!                                                                            !
!    This form defines a time or space axis. Metadata corresponding to the   !
!    type above are written to the file on <unit>. A unique ID for subsequent!
!    references to this axis is returned in axis%id. If the <domain>         !
!    element is present, this is recognized as a distributed data axis       !
!    and domain decomposition information is also written if required (the   !
!    domain decomposition info is required for multi-fileset multi-threaded  !
!    I/O). If the <data> element is allocated, it is considered to be a space!
!    axis, otherwise it is a time axis with an unlimited dimension. Only one !
!    time axis is allowed per file.                                          !
!                                                                            !
!   subroutine mpp_write_meta_field( unit, field, axes, name, units, longname!
!        min, max, missing, fill, scale, add, pack )                         !
!     integer, intent(in) :: unit                                            !
!     type(fieldtype), intent(out) :: field                                  !
!     type(axistype), intent(in) :: axes(:)                                  !
!     character(len=*), intent(in) :: name, units, longname                  !
!     real, intent(in), optional :: min, max, missing, fill, scale, add      !
!     integer, intent(in), optional :: pack                                  !
!                                                                            !
!    This form defines a field. Metadata corresponding to the type           !
!    above are written to the file on <unit>. A unique ID for subsequent     !
!    references to this field is returned in field%id. At least one axis     !
!    must be associated, 0D variables are not considered. mpp_write_meta     !
!    must previously have been called on all axes associated with this       !
!    field.                                                                  !
!                                                                            !
! The mpp_write_meta package also includes subroutines write_attribute and    !
! write_attribute_netcdf, that are private to this module.                    !
!                                                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine mpp_write_meta_global( unit, name, rval, ival, cval, pack )
!writes a global metadata attribute to unit <unit>
!attribute <name> can be an real, integer or character
!one and only one of rval, ival, and cval should be present
!the first found will be used
!for a non-netCDF file, it is encoded into a string "GLOBAL <name> <val>"
      integer, intent(in) :: unit
      character(len=*), intent(in) :: name
      real,             intent(in), optional :: rval(:)
      integer,          intent(in), optional :: ival(:)
      character(len=*), intent(in), optional :: cval
      integer, intent(in), optional :: pack

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          call write_attribute_netcdf( unit, NF_GLOBAL, name, rval, ival, cval, pack )
#endif
      else
          call write_attribute( unit, 'GLOBAL '//trim(name), rval, ival, cval, pack )
      end if

      return
    end subroutine mpp_write_meta_global

!versions of above to support <rval> and <ival> as scalars (because of f90 strict rank mat
    subroutine mpp_write_meta_global_scalar_r( unit, name, rval, pack )
      integer, intent(in) :: unit
      character(len=*), intent(in) :: name
      real, intent(in) :: rval
      integer, intent(in), optional :: pack

      call mpp_write_meta_global( unit, name, rval=(/rval/), pack=pack )
      return
    end subroutine mpp_write_meta_global_scalar_r

    subroutine mpp_write_meta_global_scalar_i( unit, name, ival )
      integer, intent(in) :: unit
      character(len=*), intent(in) :: name
      integer, intent(in) :: ival

      call mpp_write_meta_global( unit, name, ival=(/ival/) )
      return
    end subroutine mpp_write_meta_global_scalar_i

    subroutine mpp_write_meta_var( unit, id, name, rval, ival, cval, pack )
!writes a metadata attribute for variable <id> to unit <unit>
!attribute <name> can be an real, integer or character
!one and only one of rval, ival, and cval should be present
!the first found will be used
!for a non-netCDF file, it is encoded into a string "<id> <name> <val>"
      integer, intent(in) :: unit, id
      character(len=*), intent(in) :: name
      real,             intent(in), optional :: rval(:)
      integer,          intent(in), optional :: ival(:)
      character(len=*), intent(in), optional :: cval
      integer, intent(in), optional :: pack

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
          call write_attribute_netcdf( unit, id, name, rval, ival, cval, pack )
      else
          write( text, '(a,i4,a)' )'VARIABLE ', id, ' '//name
          call write_attribute( unit, trim(text), rval, ival, cval, pack )
      end if

      return
    end subroutine mpp_write_meta_var

!versions of above to support <rval> and <ival> as scalar (because of f90 strict rank matc
    subroutine mpp_write_meta_scalar_r( unit, id, name, rval, pack )
      integer, intent(in) :: unit, id
      character(len=*), intent(in) :: name
      real, intent(in) :: rval
      integer, intent(in), optional :: pack

      call mpp_write_meta( unit, id, name, rval=(/rval/), pack=pack )
      return
    end subroutine mpp_write_meta_scalar_r

    subroutine mpp_write_meta_scalar_i( unit, id, name, ival )
      integer, intent(in) :: unit, id
      character(len=*), intent(in) :: name
      integer, intent(in) :: ival

      call mpp_write_meta( unit, id, name, ival=(/ival/) )
      return
    end subroutine mpp_write_meta_scalar_i

    subroutine mpp_write_meta_axis( unit, axis, name, units, longname, cartesian, sense, c
!load the values in an axistype (still need to call mpp_write)
!write metadata attributes for axis
!it is declared intent(inout) so you can nullify pointers in the incoming object if neede
!the f90 standard doesn't guarantee that intent(out) on a type guarantees that its pointer
      integer, intent(in) :: unit
      type(axistype), intent(inout) :: axis
      character(len=*), intent(in) :: name, units, longname
      character(len=*), intent(in), optional :: cartesian
      integer, intent(in), optional :: sense
      type(domain1D), intent(in), optional :: domain
      real, intent(in), optional :: data(:)
      character(len=*), intent(in), optional :: cdata(:) !RV,bundles
      integer :: is, ie, isg, ieg

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

!pre-existing pointers need to be nullified
      if( ASSOCIATED(axis%data) )NULLIFY(axis%data)
      if( ASSOCIATED(axis%cdata) )NULLIFY(axis%cdata) !RV,bundles
!load axistype
      axis%name     = name
      axis%units    = units
      axis%longname = longname
      if( PRESENT(cartesian) )axis%cartesian = cartesian
      if( PRESENT(sense)     )axis%sense     = sense
      if( PRESENT(domain)    )then
          axis%domain = domain
          call mpp_get_global_domain( domain, isg, ieg )
          call mpp_get_compute_domain( domain, is, ie )
      else
          axis%domain = NULL_DOMAIN1D
          if( PRESENT(data) )then
             isg=1; ieg=size(data); is=isg; ie=ieg
          endif
          if( PRESENT(cdata) )then !!RV,bundles
             isg=1; ieg=size(cdata); is=isg; ie=ieg !!RV,bundles
          endif !!RV,bundles
      end if
      if( PRESENT(data) )then
          if( PRESENT(domain) )then
              if( size(data).NE.ieg-isg+1 ) &
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: size(data).NE.domain%globa
              allocate( axis%data(isg:ieg) )
          else
              allocate( axis%data(size(data)) )
          end if
          axis%data = data
      end if
      if( PRESENT(cdata) )then !RV,bundles
          if( PRESENT(domain) )then !RV,bundles
              if( size(cdata).NE.ieg-isg+1 ) & !RV,bundles
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: size(cdata).NE.domain%glob
              allocate( axis%cdata(isg:ieg) ) !RV,bundles
              allocate( axis%data(isg:ieg) ) !RV,bundles
          else !RV,bundles
              allocate( axis%cdata(size(cdata)) ) !RV,bundles
              allocate( axis%data(size(cdata)) ) !RV,bundles
          end if !RV,bundles
          axis%cdata = cdata !RV,bundles
      end if !RV,bundles


!write metadata
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
!write axis def
!space axes are always floats, time axis is always double
          if( ASSOCIATED(axis%data).or. ASSOCIATED(axis%cdata) )then !space axisRV,bundles
              if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )
                  error = NF_DEF_DIM( mpp_file(unit)%ncid, axis%name, ie-is+1,         axi
              else
                  if(  ASSOCIATED(axis%data).and.(.not.present(cdata)))then !!RV,bundles
                  error = NF_DEF_DIM( mpp_file(unit)%ncid, axis%name, size(axis%data), axi
                  else !!RV,bundles
                  error = NF_DEF_DIM( mpp_file(unit)%ncid, 'MAX_STRLEN', len(axis%cdata),
                  error = NF_DEF_DIM( mpp_file(unit)%ncid, axis%name, size(axis%cdata), ax
                  endif !!RV,bundles
              end if
              call netcdf_err(error)
              if(present(cdata)) then !!RV, bundles
              error = NF_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_CHAR, 2,(/axis%clenic
              call netcdf_err(error) !!Bundles
              else !!Bundles
              error = NF_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_FLOAT, 1, axis%did, a
              endif !!Bundles

          else                            !time axis
              if( mpp_file(unit)%id.NE.-1 ) &
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: There is already a time ax
              error = NF_DEF_DIM( mpp_file(unit)%ncid, axis%name, NF_UNLIMITED, axis%did )
              error = NF_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_DOUBLE, 1, axis%did,
              mpp_file(unit)%id = axis%id !file ID is the same as time axis varID
          end if
#endif
      else
          varnum = varnum + 1
          axis%id = varnum
          axis%did = varnum
!write axis def
          write( text, '(a,i4,a)' )'AXIS ', axis%id, ' name'
          call write_attribute( unit, trim(text), cval=axis%name )
          write( text, '(a,i4,a)' )'AXIS ', axis%id, ' size'
          if( ASSOCIATED(axis%data) )then !space axis
              if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )
                  call write_attribute( unit, trim(text), ival=(/ie-is+1/) )
              else
                  if(ASSOCIATED(axis%data).and.(.not.present(cdata))) then !!RV,bundles

                  call write_attribute( unit, trim(text), ival=(/size(axis%data)/) )
                  else !!RV,bundles
                  call write_attribute( unit, trim(text), ival=(/size(axis%cdata)/) ) !!RV
                  endif !!RV, bundles
              end if
          else                            !time axis
              if( mpp_file(unit)%id.NE.-1 ) &
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: There is already a time ax
              call write_attribute( unit, trim(text), ival=(/0/) ) !a size of 0 indicates
              mpp_file(unit)%id = axis%id
          end if
      end if
!write axis attributes
      call mpp_write_meta( unit, axis%id, 'long_name', cval=axis%longname )
      call mpp_write_meta( unit, axis%id, 'units',     cval=axis%units    )
      if( PRESENT(cartesian) )call mpp_write_meta( unit, axis%id, 'cartesian_axis', cval=a
      if( PRESENT(sense) )then
          if( sense.EQ.-1 )then
              call mpp_write_meta( unit, axis%id, 'positive', cval='down' )
          else if( sense.EQ.1 )then
              call mpp_write_meta( unit, axis%id, 'positive', cval='up' )
          end if
!silently ignore values of sense other than +/-1.
      end if
      if( mpp_file(unit)%threading.EQ.MPP_MULTI .AND. mpp_file(unit)%fileset.EQ.MPP_MULTI
          call mpp_write_meta( unit, axis%id, 'domain_decomposition', ival=(/isg,ieg,is,ie
      end if
      if( verbose ) write (stdout(), '(a,2i3,1x,a,2i3)') &
           'MPP_WRITE_META: Wrote axis metadata, pe, unit, axis%name, axis%id, axis%did=',
           pe, unit, trim(axis%name), axis%id, axis%did
      return
    end subroutine mpp_write_meta_axis

    subroutine mpp_write_meta_field( unit, field, axes, name, units, longname, min, max, m
!define field: must have already called mpp_write_meta(axis) for each axis
      integer, intent(in) :: unit
      type(fieldtype), intent(out) :: field
      type(axistype), intent(in) :: axes(:)
      character(len=*), intent(in) :: name, units, longname
      real, intent(in), optional :: min, max, missing, fill, scale, add
      integer, intent(in), optional :: pack
!this array is required because of f77 binding on netCDF interface
      integer, allocatable :: axis_id(:)
      real :: a, b
      integer :: i

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an
!
! 2005-07-20, Luis Kornblueh, MPIMet
!     if( ASSOCIATED(field%axes) )NULLIFY(field%axes)
!fill in field metadata
      field%name = name
      field%units = units
      field%longname = longname
      allocate( field%axes(size(axes)) )
      field%axes = axes
      field%time_axis_index = -1 !this value will never match any axis index
!size is buffer area for the corresponding axis info: it is required to buffer this info i
!because axis might be reused in different files
      allocate( field%size(size(axes)) )
      do i = 1,size(axes)
         if( ASSOCIATED(axes(i)%data) )then !space axis
             field%size(i) = size(axes(i)%data)
         else              !time
             field%size(i) = 1
             field%time_axis_index = i
         end if
      end do
!attributes
      if( PRESENT(min) )field%min = min
      if( PRESENT(max) )field%max = max
      if( PRESENT(missing) )field%missing = missing
      if( PRESENT(fill) )field%fill = fill
      if( PRESENT(scale) )field%scale = scale
      if( PRESENT(add) )field%add = add

!pack is currently used only for netCDF
      field%pack = 2       !default write 32-bit floats
      if( PRESENT(pack) )field%pack = pack
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          allocate( axis_id(size(field%axes)) )
          do i = 1,size(field%axes)
             axis_id(i) = field%axes(i)%did
          end do
!write field def
```

```
          select case (field%pack)
              case(1)
                  error = NF_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_DOUBLE, size(fie
              case(2)
                  error = NF_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_FLOAT,  size(fie
              case(4)
                  if( .NOT.PRESENT(scale) .OR. .NOT.PRESENT(add) ) &
                       call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: scale and add must be
                  error = NF_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_SHORT,  size(fie
              case(8)
                  if( .NOT.PRESENT(scale) .OR. .NOT.PRESENT(add) ) &
                       call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: scale and add must be
                  error = NF_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_BYTE,   size(fie
              case default
                  call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: only legal packing values
          end select
          call netcdf_err(error)
#endif
      else
          varnum = varnum + 1
          field%id = varnum
          if( PRESENT(pack) )call mpp_error( WARNING, 'MPP_WRITE_META: Packing is currentl
!write field def
          write( text, '(a,i4,a)' )'FIELD ', field%id, ' name'
          call write_attribute( unit, trim(text), cval=field%name )
          write( text, '(a,i4,a)' )'FIELD ', field%id, ' axes'
          call write_attribute( unit, trim(text), ival=field%axes(:)%did )
      end if
!write field attributes: these names follow netCDF conventions
      call mpp_write_meta( unit, field%id, 'long_name', cval=field%longname )
      call mpp_write_meta( unit, field%id, 'units',     cval=field%units    )
!all real attributes must be written as packed
      if( PRESENT(min) .AND. PRESENT(max) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_range', rval=(/min,max/), pack=p
          else
              a = nint((min-add)/scale)
              b = nint((max-add)/scale)
              call mpp_write_meta( unit, field%id, 'valid_range', rval=(/a,  b  /), pack=p
          end if
      else if( PRESENT(min) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_min', rval=field%min, pack=pack
          else
              a = nint((min-add)/scale)
              call mpp_write_meta( unit, field%id, 'valid_min', rval=a, pack=pack )
          end if
      else if( PRESENT(max) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_max', rval=field%max, pack=pack
          else
              a = nint((max-add)/scale)
              call mpp_write_meta( unit, field%id, 'valid_max', rval=a, pack=pack )
          end if
      end if
      if( PRESENT(missing) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'missing_value', rval=field%missing, pa
          else
              a = nint((missing-add)/scale)
              call mpp_write_meta( unit, field%id, 'missing_value', rval=a, pack=pack )
          end if
      end if
```

```
   if( PRESENT(fill) )then
     if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
       call mpp_write_meta( unit, field%id, '_FillValue', rval=field%missing, pack=
     else
       a = nint((fill-add)/scale)
       call mpp_write_meta( unit, field%id, '_FillValue', rval=a, pack=pack )
     end if
   end if
   if( field%pack.NE.1 .AND. field%pack.NE.2 )then
     call mpp_write_meta( unit, field%id, 'packing', ival=field%pack )
     if( PRESENT(scale) )call mpp_write_meta( unit, field%id, 'scale_factor',  rval=f
     if( PRESENT(add)   )call mpp_write_meta( unit, field%id, 'add_offset',    rval=f
   end if
   if( verbose ) write (stdout(), '(a,2i3,1x,a,i3)') 'MPP_WRITE_META: Wrote field metac
      pe, unit, trim(field%name), field%id

   return
  end subroutine mpp_write_meta_field

  subroutine write_attribute( unit, name, rval, ival, cval, pack )
!called to write metadata for non-netCDF I/O
   integer, intent(in) :: unit
   character(len=*), intent(in) :: name
   real, intent(in), optional :: rval(:)
   integer, intent(in), optional :: ival(:)
   character(len=*), intent(in), optional :: cval
!pack is currently ignored in this routine: only used by netCDF I/O
   integer, intent(in), optional :: pack

   if( mpp_file(unit)%nohdrs )return
!encode text string
   if( PRESENT(rval) )then
     write( text,* )trim(name)//'=', rval
   else if( PRESENT(ival) )then
     write( text,* )trim(name)//'=', ival
   else if( PRESENT(cval) )then
     text = ' '//trim(name)//'='//trim(cval)
   else
     call mpp_error( FATAL, 'WRITE_ATTRIBUTE: one of rval, ival, cval must be present
   end if
   if( mpp_file(unit)%format.EQ.MPP_ASCII )then
!implies sequential access
     write( unit,fmt='(a)' )trim(text)//char(10)
   else                        !MPP_IEEE32 or MPP_NATIVE
     if( mpp_file(unit)%access.EQ.MPP_SEQUENTIAL )then
       write(unit)trim(text)//char(10)
     else                      !MPP_DIRECT
       write( unit,rec=mpp_file(unit)%record )trim(text)//char(10)
       if( verbose ) write (stdout(), '(a,i3,a,i3)') 'WRITE_ATTRIBUTE: PE=', pe, '
       mpp_file(unit)%record = mpp_file(unit)%record + 1
     end if
   end if
   return
  end subroutine write_attribute

  subroutine write_attribute_netcdf( unit, id, name, rval, ival, cval, pack )
!called to write metadata for netCDF I/O
   integer, intent(in) :: unit
   integer, intent(in) :: id
   character(len=*), intent(in) :: name
   real,             intent(in), optional :: rval(:)
   integer,          intent(in), optional :: ival(:)
   character(len=*), intent(in), optional :: cval
```

```
      integer, intent(in), optional :: pack
      integer :: lenc
      integer, allocatable :: rval_i(:)
#ifdef use_netCDF
      integer :: ii, il_bytesize, il_iosize
      integer :: il_int_iosize, il_rbyt
!
      if( PRESENT(rval) )then
!
      il_bytesize = BIT_SIZE(ii)/8
      INQUIRE (iolength=il_iosize) ii
      il_int_iosize = il_iosize
      INQUIRE (iolength=il_iosize) rval(1)
      il_rbyt = il_iosize/il_int_iosize*il_bytesize
!pack is only meaningful for FP numbers
          if( PRESENT(pack) )then
              if( pack.EQ.1 )then
                  if( il_rbyt.EQ.DOUBLE_KIND )then
                      error = NF_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_DOUBLE,
                  else if( il_rbyt.EQ.FLOAT_KIND )then
                      call mpp_error( WARNING, &
                           'WRITE_ATTRIBUTE_NETCDF: attempting to write internal 32-bit re
                      error = NF_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_DOUBLE,
                  end if
                  call netcdf_err(error)
              else if( pack.EQ.2 )then
                  if( il_rbyt.EQ.DOUBLE_KIND )then
                      error = NF_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_FLOAT,
                  else if( il_rbyt.EQ.FLOAT_KIND )then
                      error = NF_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_FLOAT,
                  end if
                  call netcdf_err(error)
              else if( pack.EQ.4 )then
                  allocate( rval_i(size(rval)) )
                  rval_i = rval
                  call mpp_flushstd(6)
                  if( il_rbyt.EQ.DOUBLE_KIND )then
                      error = NF_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_SHORT,
                  else if( il_rbyt.EQ.FLOAT_KIND )then
                      error = NF_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_SHORT,
                  end if
                  call netcdf_err(error)
                  deallocate(rval_i)
              else if( pack.EQ.8 )then
                  allocate( rval_i(size(rval)) )
                  rval_i = rval
                  if( il_rbyt.EQ.DOUBLE_KIND )then
                      error = NF_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_BYTE,
                  else if( il_rbyt.EQ.FLOAT_KIND )then
                      error = NF_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_BYTE,
                  end if
                  call netcdf_err(error)
                  deallocate(rval_i)
              else
                  call mpp_error( FATAL, 'WRITE_ATTRIBUTE_NETCDF: only legal packing value
              end if
          else
!default is to write FLOATs (32-bit)
              if( il_rbyt.EQ.DOUBLE_KIND )then
                  error = NF_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_FLOAT,  siz
              else if( il_rbyt.EQ.FLOAT_KIND )then
                  error = NF_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_FLOAT,  siz
              end if
              call netcdf_err(error)
          end if
      else if( PRESENT(ival) )then
          error = NF_PUT_ATT_INT ( mpp_file(unit)%ncid, id, name, NF_INT, size(ival), ival
      else if( present(cval) )then
          error = NF_PUT_ATT_TEXT( mpp_file(unit)%ncid, id, name, len_trim(cval), cval );
      else
          call mpp_error( FATAL, 'WRITE_ATTRIBUTE_NETCDF: one of rval, ival, cval must be
      end if
#endif /* use_netCDF */
      return
    end subroutine write_attribute_netcdf

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!                             MPP_WRITE                                !
!                                                                      !
! mpp_write is used to write data to the file on <unit> using the      !
! file parameters supplied by mpp_open(). Axis and field definitions   !
! must have previously been written to the file using mpp_write_meta.  !
!                                                                      !
! mpp_write can take 2 forms, one for distributed data and one for     !
! non-distributed data. Distributed data refer to arrays whose two     !
! fastest-varying indices are domain-decomposed. Distributed data      !
! must be 2D or 3D (in space). Non-distributed data can be 0-3D.       !
!                                                                      !
! In all calls to mpp_write, tstamp is an optional argument. It is to  !
! be omitted if the field was defined not to be a function of time.    !
! Results are unpredictable if the argument is supplied for a time-    !
! independent field, or omitted for a time-dependent field. Repeated   !
! writes of a time-independent field are also not recommended. One     !
! time level of one field is written per call.                         !
!                                                                      !
!                                                                      !
! For non-distributed data, use                                        !
!                                                                      !
!  mpp_write( unit, field, data, tstamp )                              !
!     integer, intent(in) :: unit                                      !
!     type(fieldtype), intent(in) :: field                             !
!     real, optional :: tstamp                                         !
!     data is real and can be scalar or of rank 1-3.                   !
!                                                                      !
! For distributed data, use                                            !
!                                                                      !
!  mpp_write( unit, field, domain, data, tstamp )                      !
!     integer, intent(in) :: unit                                      !
!     type(fieldtype), intent(in) :: field                             !
!     type(domain2D), intent(in) :: domain                             !
!     real, optional :: tstamp                                         !
!     data is real and can be of rank 2 or 3.                          !
!                                                                      !
!  mpp_write( unit, axis )                                             !
!     integer, intent(in) :: unit                                      !
!     type(axistype), intent(in) :: axis                               !
!                                                                      !
! This call writes the actual co-ordinate values along each space      !
! axis. It must be called once for each space axis after all other     !
! metadata has been written.                                           !
!                                                                      !
! The mpp_write package also includes the routine write_record which   !
! performs the actual write. This routine is private to this module.   !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
#define MPP_WRITE_2DDECOMP_1D_ mpp_write_2ddecomp_r1d
#define MPP_WRITE_2DDECOMP_2D_ mpp_write_2ddecomp_r2d
#define MPP_WRITE_2DDECOMP_3D_ mpp_write_2ddecomp_r3d
#define MPP_WRITE_2DDECOMP_4D_ mpp_write_2ddecomp_r4d
#define MPP_TYPE_ real
#include <mpp_write_2Ddecomp.h>

#define MPP_WRITE_ mpp_write_r0D
#define MPP_TYPE_ real
#define MPP_RANK_ !
#define MPP_WRITE_RECORD_ call write_record( unit, field, 1, (/data/), tstamp )
#include <mpp_write.h>

#define MPP_WRITE_ mpp_write_r1D
#define MPP_TYPE_ real
#define MPP_WRITE_RECORD_ call write_record( unit, field, size(data), data, tstamp )
#define MPP_RANK_ (:)
#include <mpp_write.h>

#define MPP_WRITE_ mpp_write_r2D
#define MPP_TYPE_ real
#define MPP_WRITE_RECORD_ call write_record( unit, field, size(data), data, tstamp )
#define MPP_RANK_ (:,:)
#include <mpp_write.h>

#define MPP_WRITE_ mpp_write_r3D
#define MPP_TYPE_ real
#define MPP_WRITE_RECORD_ call write_record( unit, field, size(data), data, tstamp )
#define MPP_RANK_ (:,:,:)
#include <mpp_write.h>

#define MPP_WRITE_ mpp_write_r4D
#define MPP_TYPE_ real
#define MPP_WRITE_RECORD_ call write_record( unit, field, size(data), data, tstamp )
#define MPP_RANK_ (:,:,:,:)
#include <mpp_write.h>

    subroutine mpp_write_axis( unit, axis )
      integer, intent(in) :: unit
      type(axistype), intent(in) :: axis
      type(fieldtype) :: field
      integer :: is, ie


      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_WRITE: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE: invalid unit numb
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset  .EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
!we convert axis to type(fieldtype) in order to call write_record
      field = default_field
      allocate( field%axes(1) )
      field%axes(1) = axis
      allocate( field%size(1) )
      field%id = axis%id
      if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )then
          call mpp_get_compute_domain( axis%domain, is, ie )
          field%size(1) = ie-is+1
!!RV,bundles
          if(associated( axis%cdata)) then
          call write_record_c( unit, field, field%size(1), axis%cdata(is:) )
          else
          call write_record( unit, field, field%size(1), axis%data(is:) )
          endif
```

```
!!RV,bundles
      else
!!RV,bundles
          if(associated( axis%cdata)) then
          field%size(1) = size(axis%cdata)
          call write_record_c(unit,field, field%size(1), axis%cdata )
          else
          field%size(1) = size(axis%data)
          call write_record( unit, field, field%size(1), axis%data )
          endif
!!RV,bundles
      end if
      return
    end subroutine mpp_write_axis

    subroutine write_record_c( unit, field, nwords, cdata, time_in, domain ) !!RV,bundles
!routine that is finally called by all mpp_write routines to perform the write
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
!RV,bundles
      character(len=64), intent(in) :: cdata(nwords)
      real(DOUBLE_KIND), intent(in), optional :: time_in
      type(domain2D), intent(in), optional :: domain
!RV      integer, dimension(size(field%axes)) :: start, axsiz

      integer,allocatable,dimension(:) :: start, axsiz
!RV
      real :: time
      integer :: time_level
      logical :: newtime
      integer :: subdomain(4)
      integer :: packed_data(nwords)
      integer :: i, is, ie, js, je, isg, ieg, jsg, jeg, isizc, jsizc, isizg, jsizg
#ifdef use_netCDF
      integer :: ii, il_bytesize, il_iosize
      integer :: il_int_iosize, il_rbyt
#endif

#ifdef use_CRI_pointers
      real(FLOAT_KIND) :: data_r4(nwords)
      pointer( ptr1, data_r4)
      pointer( ptr2, packed_data)

      if (mpp_io_stack_size < 2*nwords) call mpp_io_set_stack_size(2*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
#endif
```

```
      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_WRITE: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE: invalid unit numb
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset  .EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return

!RV
      allocate(start(size(field%axes)))
      allocate(axsiz(size(field%axes)))
!RV
      if( .NOT.mpp_file(unit)%initialized )then
!this is the first call to mpp_write
!we now declare the file to be initialized
!if this is netCDF we switch file from DEFINE mode to DATA mode
          if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
!NOFILL is probably required for parallel: any circumstances in which not advisable?
              error = NF_SET_FILL( mpp_file(unit)%ncid, NF_NOFILL, i ); call netcdf_err(er
              if( mpp_file(unit)%action.EQ.MPP_WRONLY )error = NF_ENDDEF(mpp_file(unit)%nc
#endif
          else
              call mpp_write_meta( unit, 'END', cval='metadata' )
          end if
          mpp_file(unit)%initialized = .TRUE.
          if( verbose ) write (stdout(), '(a,i3,a)') 'MPP_WRITE: PE=', pe, ' initialized f
      end if

!initialize time: by default assume NULLTIME
      time = NULLTIME
      time_level = -1
      newtime = .FALSE.
      if( PRESENT(time_in) )time = time_in
!increment time level if new time
      if( time.GT.mpp_file(unit)%time+EPSILON(time) )then !new time
          mpp_file(unit)%time_level = mpp_file(unit)%time_level + 1
          mpp_file(unit)%time = time
          newtime = .TRUE.
      end if
      if( verbose ) write (stdout(), '(a,2i3,2i5,es13.5)') 'MPP_WRITE: PE, unit, %id, %tim
           pe, unit, mpp_file(unit)%id, mpp_file(unit)%time_level, mpp_file(unit)%time

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be written:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NF_PUT_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1! NOTE: pa
!       since that attempts to gather all data on PE 0.
```

```
          start = 1
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( i.EQ.field%time_axis_index )start(i) = mpp_file(unit)%time_level
             start(i) = max(start(i),1)
          end do
          if( PRESENT(domain) )then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je,  xsize=isizc, ysize=
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg, xsize=isizg, ysize=
              axsiz(1) = isizc
              axsiz(2) = jsizc
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( isizc.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( jsizc.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
          end if
          if( debug ) &
           write (stdout(), '(a,2i3,12i4)') 'a WRITE_RECORD: PE, unit, start, axsiz=', pe,
#ifdef use_netCDF
!write time information if new time
          if( newtime )then
              il_bytesize = BIT_SIZE(ii)/8
              INQUIRE (iolength=il_iosize) ii
              il_int_iosize = il_iosize
              INQUIRE (iolength=il_iosize) time
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt.EQ.DOUBLE_KIND )then
                  error = NF_PUT_VAR1_DOUBLE( mpp_file(unit)%ncid, mpp_file(unit)%id, mpp_
              else if( il_rbyt.EQ.FLOAT_KIND )then
                  error = NF_PUT_VAR1_REAL  ( mpp_file(unit)%ncid, mpp_file(unit)%id, mpp_
              end if
          end if
          if( field%pack.LE.2 )then
                 write(6,*) ' Iam here 6!'
                 call mpp_flushstd(6)
               error = NF_PUT_VARA_TEXT( mpp_file(unit)%ncid, field%id, (/1,start/), (/ler
                 write(6,*) ' Iam here 7!'
                 call mpp_flushstd(6)
          else  !!RV, bundles
                 write(6,*) ' Iam here 8!'
                 call mpp_flushstd(6)
            call mpp_error( FATAL, 'MPP_WRITE_RECORD_C: pack on text !' )
          end if !!RV, bundles
                 write(6,*) ' Iam here 9!',error
                 call mpp_flushstd(6)
          call netcdf_err(error)
#endif
      else                      !non-netCDF
!subdomain contains (/is,ie,js,je/)
          if( PRESENT(domain) )then
              subdomain(:) = (/ is, ie, js, je /)
          else
              subdomain(:) = -1    ! -1 means use global value from axis metadata
          end if
          if( mpp_file(unit)%format.EQ.MPP_ASCII )then
!implies sequential access
              write( unit,* )field%id, subdomain, time_level, time, cdata
          else                       !MPP_IEEE32 or MPP_NATIVE
              if( mpp_file(unit)%access.EQ.MPP_SEQUENTIAL )then
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      write(unit)field%id, subdomain, time_level, time, cdata
                  else
                      write(unit)field%id, subdomain, time_level, time, cdata
                  end if
#else
                  write(unit)field%id, subdomain, time_level, time, cdata
#endif
              else                   !MPP_DIRECT
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  else
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  end if
#else
                  write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_level,
#endif
                  if( debug ) write (stdout(), '(a,i3,a,i3)') 'MPP_WRITE: PE=', pe, ' wrot
              end if
          end if
      end if

!recompute current record for direct access I/O
      if( mpp_file(unit)%access.EQ.MPP_DIRECT )then
          if( mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
!assumes all PEs participate in I/O: modify later
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe*npes
          else
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe
          end if
      end if
!RV
      deallocate(start)
      deallocate(axsiz)
!RV
      return
    end subroutine write_record_c

    subroutine write_record_b( unit, field, nwords, data, time_in, domain,block_id )
!routine that is finally called by all mpp_write routines to perform the write
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.
!RV,SGI:
!   The routine write_record_b is a special clone of write_record.
!   The assumption is here that the user has declared a data structure
!   like a(:,:,:,1:no_of_blocks). For whatever reason that arrray is written
!   is not written in a big chunk but on a per block basis for a certain time
!   stamp: At t_i write a(:,:,:,block_id).After all block are written the data structure c
!   should look like as if array a was written in one big chunk.
!   Moreover, I assume that the time axis is always the last one and that the block axis
!   comes befor the time axis, means the block axis is the last pseudo spatial axis.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
      real, intent(in) :: data(nwords)
      real(DOUBLE_KIND), intent(in), optional :: time_in
      integer,intent(in),optional :: block_id
      type(domain2D), intent(in), optional :: domain
!RV      integer, dimension(size(field%axes)) :: start, axsiz

      integer,allocatable,dimension(:) :: start, axsiz
!RV
      real :: time
      integer :: time_level
      logical :: newtime
      integer :: subdomain(4)
      integer :: packed_data(nwords)
      integer :: i, is, ie, js, je, isg, ieg, jsg, jeg, isizc, jsizc, isizg, jsizg
#ifdef use_netCDF
      integer :: ii, il_bytesize, il_iosize
      integer :: il_int_iosize, il_rbyt
#endif

#ifdef use_CRI_pointers
      real(FLOAT_KIND) :: data_r4(nwords)
      pointer( ptr1, data_r4)
      pointer( ptr2, packed_data)

      if (mpp_io_stack_size < 2*nwords) call mpp_io_set_stack_size(2*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
#endif

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_WRITE: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE: invalid unit numb
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset  .EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return

!RV
      allocate(start(size(field%axes)))
      allocate(axsiz(size(field%axes)))
!RV
      if( .NOT.mpp_file(unit)%initialized )then
!this is the first call to mpp_write
!we now declare the file to be initialized
!if this is netCDF we switch file from DEFINE mode to DATA mode
          if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
!NOFILL is probably required for parallel: any circumstances in which not advisable?
              error = NF_SET_FILL( mpp_file(unit)%ncid, NF_NOFILL, i ); call netcdf_err(er
              if( mpp_file(unit)%action.EQ.MPP_WRONLY )error = NF_ENDDEF(mpp_file(unit)%nc
#endif
          else
              call mpp_write_meta( unit, 'END', cval='metadata' )
          end if
          mpp_file(unit)%initialized = .TRUE.
          if( verbose ) write (stdout(), '(a,i3,a)') 'MPP_WRITE: PE=', pe, ' initialized f
      end if

!initialize time: by default assume NULLTIME
      time = NULLTIME
      time_level = -1
      newtime = .FALSE.
      if( PRESENT(time_in) )time = time_in
!increment time level if new time
      if( time.GT.mpp_file(unit)%time+EPSILON(time) )then !new time
          mpp_file(unit)%time_level = mpp_file(unit)%time_level + 1
          mpp_file(unit)%time = time
          newtime = .TRUE.
      end if
      if( verbose ) write (stdout(), '(a,2i3,2i5,es13.5)') 'MPP_WRITE: PE, unit, %id, %tim
           pe, unit, mpp_file(unit)%id, mpp_file(unit)%time_level, mpp_file(unit)%time

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be written:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NF_PUT_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1
! NOTE: passing of subdomains will fail for multi-PE single-threaded I/O,
!       since that attempts to gather all data on PE 0.
          start = 1
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( i.EQ.field%time_axis_index )start(i) = mpp_file(unit)%time_level
             start(i) = max(start(i),1)
          end do
          if( PRESENT(domain) )then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je,  xsize=isizc, ysize=
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg, xsize=isizg, ysize=
              axsiz(1) = isizc
              axsiz(2) = jsizc
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( isizc.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( jsizc.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
```

```
               end if
           end if
!RV,SGI
          if( PRESENT(block_id) )then
            if (block_id.le.0) then
               call mpp_error( FATAL, 'MPP_RECORD_B: block_id <= 0!' )
            endif
            if( PRESENT(time_in) )then

               if(block_id.gt. axsiz(size(field%axes)-1)) &
                 call mpp_error( FATAL, 'MPP_RECORD_B: block_id > axis range!' )

               start(size(field%axes)-1)=block_id

            else

               if(block_id.gt. axsiz(size(field%axes))) &
                 call mpp_error( FATAL, 'MPP_RECORD_B: block_id > axis range!' )

               start(size(field%axes))=block_id

            endif
          endif
!RV,SGI
          if( debug ) &
           write (stdout(), '(a,2i3,12i4)') 'b WRITE_RECORD: PE, unit, start, axsiz=', pe,
#ifdef use_netCDF
!write time information if new time
          il_bytesize = BIT_SIZE(ii)/8
          INQUIRE (iolength=il_iosize) ii
          il_int_iosize = il_iosize
          if( newtime )then
              INQUIRE (iolength=il_iosize) time
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt .EQ. DOUBLE_KIND )then
                  error = NF_PUT_VAR1_DOUBLE( mpp_file(unit)%ncid, mpp_file(unit)%id, mpp_
              else if( il_rbyt .EQ. FLOAT_KIND )then
                  error = NF_PUT_VAR1_REAL  ( mpp_file(unit)%ncid, mpp_file(unit)%id, mpp_
              end if
          end if
          if( field%pack.LE.2 )then
              INQUIRE (iolength=il_iosize) data(1)
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt .EQ. DOUBLE_KIND )then
!                  write(stderr,*)data
                  error = NF_PUT_VARA_DOUBLE( mpp_file(unit)%ncid, field%id, start, axsiz,
              else if( il_rbyt .EQ. FLOAT_KIND )then
                  error = NF_PUT_VARA_REAL  ( mpp_file(unit)%ncid, field%id, start, axsiz,
              end if
          else              !convert to integer using scale and add: no error check on pac
              packed_data = nint((data-field%add)/field%scale)
              error = NF_PUT_VARA_INT   ( mpp_file(unit)%ncid, field%id, start, axsiz, pac
          end if
          call netcdf_err(error)
#endif
      else                      !non-netCDF
!subdomain contains (/is,ie,js,je/)
          if( PRESENT(domain) )then
              subdomain(:) = (/ is, ie, js, je /)
          else
              subdomain(:) = -1    ! -1 means use global value from axis metadata
          end if
          if( mpp_file(unit)%format.EQ.MPP_ASCII )then
```

```
!implies sequential access
              write( unit,* )field%id, subdomain, time_level, time, data
          else                      !MPP_IEEE32 or MPP_NATIVE
              if( mpp_file(unit)%access.EQ.MPP_SEQUENTIAL )then
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      data_r4 = data !IEEE conversion layer on SGI until assign -N ieee_32
                      write(unit)field%id, subdomain, time_level, time, data_r4
                  else
                      write(unit)field%id, subdomain, time_level, time, data
                  end if
#else
                  write(unit)field%id, subdomain, time_level, time, data
#endif
              else                   !MPP_DIRECT
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      data_r4 = data !IEEE conversion layer on SGI until assign -N ieee_32
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  else
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  end if
#else
                  write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_level,
#endif
                  if( debug ) write (stdout(), '(a,i3,a,i3)') 'MPP_WRITE: PE=', pe, ' wrot
              end if
          end if
      end if

!recompute current record for direct access I/O
      if( mpp_file(unit)%access.EQ.MPP_DIRECT )then
          if( mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
!assumes all PEs participate in I/O: modify later
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe*npes
          else
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe
          end if
      end if

!RV
      deallocate(start)
      deallocate(axsiz)
!RV
      return
    end subroutine write_record_b

    subroutine write_record( unit, field, nwords, data, time_in, domain )
!routine that is finally called by all mpp_write routines to perform the write
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.
```

```
      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
      real, intent(in) :: data(nwords)
      real(DOUBLE_KIND), intent(in), optional :: time_in
      type(domain2D), intent(in), optional :: domain
!RV   Very unsafe!!!! One can not use size(field%axes) before it
!RV   is clear that every thing has been initialized.
!RV   The code crashes in a multi-PE run.
!RV      integer, dimension(size(field%axes)) :: start, axsiz
      integer,allocatable,dimension(:) :: start, axsiz
!RV
      real :: time
      integer :: time_level
      logical :: newtime
      integer :: subdomain(4)
      integer :: packed_data(nwords)
      integer :: i, is, ie, js, je, isg, ieg, jsg, jeg, isizc, jsizc, isizg, jsizg
!rv,sgi<
      integer :: icount_domains
!rv,sgi>
#ifdef use_netCDF
      integer :: ii, il_bytesize, il_iosize
      integer :: il_int_iosize, il_rbyt
#endif

#ifdef use_CRI_pointers
      real(FLOAT_KIND) :: data_r4(nwords)
      pointer( ptr1, data_r4)
      pointer( ptr2, packed_data)

      if (mpp_io_stack_size < 2*nwords) call mpp_io_set_stack_size(2*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
#endif

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_WRITE: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE: invalid unit numb
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset  .EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return

!RV
      allocate(start(size(field%axes)))
      allocate(axsiz(size(field%axes)))
!RV
      if( .NOT.mpp_file(unit)%initialized )then
!this is the first call to mpp_write
!we now declare the file to be initialized
!if this is netCDF we switch file from DEFINE mode to DATA mode
          if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
!NOFILL is probably required for parallel: any circumstances in which not advisable?
              error = NF_SET_FILL( mpp_file(unit)%ncid, NF_NOFILL, i ); call netcdf_err(er
              if( mpp_file(unit)%action.EQ.MPP_WRONLY )error = NF_ENDDEF(mpp_file(unit)%nc
#endif
          else
              call mpp_write_meta( unit, 'END', cval='metadata' )
          end if
          mpp_file(unit)%initialized = .TRUE.
          if( verbose ) write (stdout(), '(a,i3,a)') 'MPP_WRITE: PE=', pe, ' initialized f
      end if
```

```
!initialize time: by default assume NULLTIME
      time = NULLTIME
      time_level = -1
      newtime = .FALSE.
      if( PRESENT(time_in) )time = time_in
!increment time level if new time
      if( time.GT.mpp_file(unit)%time+EPSILON(time) )then !new time
          mpp_file(unit)%time_level = mpp_file(unit)%time_level + 1
          mpp_file(unit)%time = time
          newtime = .TRUE.
      end if
      if( verbose ) write (stdout(), '(a,2i3,2i5,es13.5)') 'MPP_WRITE: PE, unit, %id, %tim
           pe, unit, mpp_file(unit)%id, mpp_file(unit)%time_level, mpp_file(unit)%time

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be written:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NF_PUT_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1
! NOTE: passing of subdomains will fail for multi-PE single-threaded I/O,
!       since that attempts to gather all data on PE 0.
          start = 1
!
!rv,sgi<
!Treatment of the case x(k,i,j) where k is a common, non-decompsoed axis of
!all PEs and i,j are 2D decomposed .
!the array x(k,i,j) is collapsed allong the two first axis. It is treated 2D.
!A corresponding domain is defined as well which is used for stitching.
!However, for writing to a file the decomposition information is taken
!from the field axes rather then from the domain 'domain'.
!If icount_domains is 2 we have exactly that case.
          icount_domains=0
!rv,sgi<
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( i.EQ.field%time_axis_index )start(i) = mpp_file(unit)%time_level
             start(i) = max(start(i),1)
!rv,sgi<
             if((field%axes(i)%domain .ne.  NULL_DOMAIN1D) .and. &
                (field%axes(1)%domain .eq.  NULL_DOMAIN1D)) &
                icount_domains=icount_domains+1
!rv,sgi>
          end do
          if( PRESENT(domain) )then
            if(icount_domains .ne. 2 ) then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je &
                                         ,  xsize=isizc, ysize=jsizc )
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg &
                                         , xsize=isizg, ysize=jsizg )
              axsiz(1) = isizc
              axsiz(2) = jsizc
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( isizc.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( jsizc.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
!rv,sgi<
            else

              call mpp_get_compute_domain( field%axes(2)%domain, is,  ie &
                                         ,  size=isizc)
              call mpp_get_global_domain ( field%axes(2)%domain, isg, ieg &
                                         , size=isizg )
              call mpp_get_compute_domain( field%axes(3)%domain, js,  je &
                                         ,  size=jsizc)
              call mpp_get_global_domain ( field%axes(3)%domain, jsg, jeg &
                                         , size=jsizg )
              axsiz(2) = isizc
              axsiz(3) = jsizc
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(2) = is - isg + 1
                  start(3) = js - jsg + 1
              else
                  if( isizc.NE.ie-is+1 )then
                      start(2) = is - isg + 1
                      axsiz(2) = ie - is + 1
                  end if
                  if( jsizc.NE.je-js+1 )then
                      start(3) = js - jsg + 1
                      axsiz(3) = je - js + 1
                  end if
              end if

            endif
!rv,sgi>
          end if
          if( debug ) write (stdout(),'(a,3i3,12i4)') &
                      'c WRITE_RECORD: PE, unit, icount_domains, start, axsiz=' &
                    , pe, unit, icount_domains, start, axsiz
#ifdef use_netCDF
!write time information if new time
          il_bytesize = BIT_SIZE(ii)/8
          INQUIRE (iolength=il_iosize) ii
          il_int_iosize = il_iosize
          if( newtime )then
              INQUIRE (iolength=il_iosize) time
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt.EQ.DOUBLE_KIND )then
                  error = NF_PUT_VAR1_DOUBLE( mpp_file(unit)%ncid, mpp_file(unit)%id, mpp_
              else if( il_rbyt.EQ.FLOAT_KIND )then
                  error = NF_PUT_VAR1_REAL  ( mpp_file(unit)%ncid, mpp_file(unit)%id, mpp_
              end if
          end if
          if( field%pack.LE.2 )then
              INQUIRE (iolength=il_iosize) data(1)
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt .EQ. DOUBLE_KIND )then
                  error = NF_PUT_VARA_DOUBLE( mpp_file(unit)%ncid, field%id, start, axsiz,
              else if( il_rbyt .EQ. FLOAT_KIND )then
                  error = NF_PUT_VARA_REAL  ( mpp_file(unit)%ncid, field%id, start, axsiz,
              end if
          else              !convert to integer using scale and add: no error check on pac
              packed_data = nint((data-field%add)/field%scale)
              error = NF_PUT_VARA_INT   ( mpp_file(unit)%ncid, field%id, start, axsiz, pac
          end if
          call netcdf_err(error)
#endif
      else                     !non-netCDF
!subdomain contains (/is,ie,js,je/)
          if( PRESENT(domain) )then
              subdomain(:) = (/ is, ie, js, je /)
          else
              subdomain(:) = -1    ! -1 means use global value from axis metadata
          end if
          if( mpp_file(unit)%format.EQ.MPP_ASCII )then
!implies sequential access
              write( unit,* )field%id, subdomain, time_level, time, data
          else                      !MPP_IEEE32 or MPP_NATIVE
              if( mpp_file(unit)%access.EQ.MPP_SEQUENTIAL )then
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      data_r4 = data !IEEE conversion layer on SGI until assign -N ieee_32
                      write(unit)field%id, subdomain, time_level, time, data_r4
                  else
                      write(unit)field%id, subdomain, time_level, time, data
                  end if
#else
                  write(unit)field%id, subdomain, time_level, time, data
#endif
              else                  !MPP_DIRECT
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      data_r4 = data !IEEE conversion layer on SGI until assign -N ieee_32
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  else
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  end if
#else
                  write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_level,
#endif
                  if( debug ) write (stdout(), '(a,i3,a,i3)') 'MPP_WRITE: PE=', pe, ' wrot
              end if
          end if
      end if

!recompute current record for direct access I/O
      if( mpp_file(unit)%access.EQ.MPP_DIRECT )then
          if( mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
!assumes all PEs participate in I/O: modify later
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe*npes
          else
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe
          end if
      end if
!RV
```

```
      deallocate(start)
      deallocate(axsiz)
!RV

      return
    end subroutine write_record

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!                           MPP_COPY_META                              !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine mpp_copy_meta_global( unit, gatt )
!writes a global metadata attribute to unit <unit>
!attribute <name> can be an real, integer or character
!one and only one of rval, ival, and cval should be present
!the first found will be used
!for a non-netCDF file, it is encoded into a string "GLOBAL <name> <val>"
      integer, intent(in) :: unit
      type(atttype), intent(in) :: gatt
      integer :: len

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an
#ifdef use_netCDF
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
         if( gatt%type.EQ.NF_CHAR )then
            len = gatt%len
            call write_attribute_netcdf( unit, NF_GLOBAL, gatt%name, cval=gatt%catt(1:len)
         else
            call write_attribute_netcdf( unit, NF_GLOBAL, gatt%name, rval=gatt%fatt )
         endif
      else
         if( gatt%type.EQ.NF_CHAR )then
            len=gatt%len
            call write_attribute( unit, 'GLOBAL '//trim(gatt%name), cval=gatt%catt(1:len)
         else
            call write_attribute( unit, 'GLOBAL '//trim(gatt%name), rval=gatt%fatt )
         endif
     end if
#else
     call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine mpp_copy_meta_global

    subroutine mpp_copy_meta_axis( unit, axis, domain )
!load the values in an axistype (still need to call mpp_write)
!write metadata attributes for axis.  axis is declared inout
!because the variable and dimension ids are altered

      integer, intent(in) :: unit
      type(axistype), intent(inout) :: axis
      type(domain1D), intent(in), optional :: domain
      character(len=512) :: text
      integer :: i, len, is, ie, isg, ieg

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

! redefine domain if present
      if( PRESENT(domain) )then
          axis%domain = domain
      else
          axis%domain = NULL_DOMAIN1D
      end if

#ifdef use_netCDF
!write metadata
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then

!write axis def
          if( ASSOCIATED(axis%data) )then !space axis
              if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )
                  call mpp_get_compute_domain( axis%domain, is, ie )
                  call mpp_get_global_domain( axis%domain, isg, ieg )
                  error = NF_DEF_DIM( mpp_file(unit)%ncid, axis%name, ie-is+1, axis%did )
              else
                  error = NF_DEF_DIM( mpp_file(unit)%ncid, axis%name, size(axis%data),
              end if
              call netcdf_err(error)
              error = NF_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_FLOAT, 1, axis%did, a
          else                            !time axis
              error = NF_DEF_DIM( mpp_file(unit)%ncid, axis%name, NF_UNLIMITED, axis%did )
              error = NF_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_DOUBLE, 1, axis%did,
              mpp_file(unit)%id = axis%id !file ID is the same as time axis varID
              mpp_file(unit)%recdimid = axis%did ! record dimension id
          end if
      else
          varnum = varnum + 1
          axis%id = varnum
          axis%did = varnum
!write axis def
          write( text, '(a,i4,a)' )'AXIS ', axis%id, ' name'
          call write_attribute( unit, trim(text), cval=axis%name )
          write( text, '(a,i4,a)' )'AXIS ', axis%id, ' size'
          if( ASSOCIATED(axis%data) )then !space axis
              if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )
                  call write_attribute( unit, trim(text), ival=(/ie-is+1/) )
              else
                  call write_attribute( unit, trim(text), ival=(/size(axis%data)/) )
              end if
          else                            !time axis
              if( mpp_file(unit)%id.NE.-1 ) &
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: There is already a time ax
              call write_attribute( unit, trim(text), ival=(/0/) ) !a size of 0 indicates
              mpp_file(unit)%id = axis%id
          end if
      end if
!write axis attributes

      do i=1,axis%natt
         if( axis%Att(i)%name.NE.default_att%name )then
            if( axis%Att(i)%type.EQ.NF_CHAR )then
               len = axis%Att(i)%len
               call mpp_write_meta( unit, axis%id, axis%Att(i)%name, cval=axis%Att(i)%catt
            else
               call mpp_write_meta( unit, axis%id, axis%Att(i)%name, rval=axis%Att(i)%fatt
            endif
         endif
      enddo

      if( mpp_file(unit)%threading.EQ.MPP_MULTI .AND. mpp_file(unit)%fileset.EQ.MPP_MULTI
          call mpp_write_meta( unit, axis%id, 'domain_decomposition', ival=(/isg,ieg,is,ie
      end if
      if( verbose ) write (stdout(), '(a,2i3,1x,a,2i3)') &
           'MPP_WRITE_META: Wrote axis metadata, pe, unit, axis%name, axis%id, axis%did=',
           pe, unit, trim(axis%name), axis%id, axis%did
#else
      call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine mpp_copy_meta_axis

    subroutine mpp_copy_meta_field( unit, field, axes )
!useful for copying field metadata from a previous call to mpp_read_meta
!define field: must have already called mpp_write_meta(axis) for each axis
      integer, intent(in) :: unit
      type(fieldtype), intent(inout) :: field
      type(axistype), intent(in), optional :: axes(:)
!this array is required because of f77 binding on netCDF interface
      integer, allocatable :: axis_id(:)
      real :: a, b
      integer :: i

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

       if( field%pack.NE.1 .AND. field%pack.NE.2 )then
            if( field%pack.NE.4 .AND. field%pack.NE.8 ) &
               call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: only legal packing values are
      end if

      if (PRESENT(axes)) then
         deallocate(field%axes)
         deallocate(field%size)
         allocate(field%axes(size(axes)))
         allocate(field%size(size(axes)))
         field%axes = axes
         do i=1,size(axes)
            if (ASSOCIATED(axes(i)%data)) then
               field%size(i) = size(axes(i)%data)
            else
               field%size(i) = 1
               field%time_axis_index = i
            endif
         enddo
      endif

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          allocate( axis_id(size(field%axes)) )
          do i = 1,size(field%axes)
             axis_id(i) = field%axes(i)%did
          end do
!write field def
          select case (field%pack)
              case(1)
                  error = NF_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_DOUBLE, size(fie
              case(2)
                  error = NF_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_FLOAT,  size(fie
              case(4)
                  if( field%scale.EQ.default_field%scale .OR. field%add.EQ.default_field%a
                       call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: scale and add must be
                  error = NF_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_SHORT,  size(fie
              case(8)
                  if( field%scale.EQ.default_field%scale .OR. field%add.EQ.default_field%a
                       call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: scale and add must be
                  error = NF_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_BYTE,   size(fie
              case default
                  call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: only legal packing values
          end select
#endif
      else
          varnum = varnum + 1
          field%id = varnum
          if( field%pack.NE.default_field%pack ) &
           call mpp_error( WARNING, 'MPP_WRITE_META: Packing is currently available only c
!write field def
          write( text, '(a,i4,a)' )'FIELD ', field%id, ' name'
          call write_attribute( unit, trim(text), cval=field%name )
          write( text, '(a,i4,a)' )'FIELD ', field%id, ' axes'
          call write_attribute( unit, trim(text), ival=field%axes(:)%did )
      end if
!write field attributes: these names follow netCDF conventions
      call mpp_write_meta( unit, field%id, 'long_name', cval=field%longname )
      call mpp_write_meta( unit, field%id, 'units',     cval=field%units    )
!all real attributes must be written as packed
      if( (field%min.NE.default_field%min) .AND. (field%max.NE.default_field%max) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_range', rval=(/field%min,field%m
          else
              a = nint((field%min-field%add)/field%scale)
              b = nint((field%max-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, 'valid_range', rval=(/a,  b  /), pack=f
          end if
      else if( field%min.NE.default_field%min )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_min', rval=field%min, pack=fielc
          else
              a = nint((field%min-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, 'valid_min', rval=a, pack=field%pack )
          end if
      else if( field%max.NE.default_field%max )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_max', rval=field%max, pack=fielc
          else
              a = nint((field%max-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, 'valid_max', rval=a, pack=field%pack )
          end if
      end if
      if( field%missing.NE.default_field%missing )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'missing_value', rval=field%missing, pa
          else
              a = nint((field%missing-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, 'missing_value', rval=a, pack=field%pac
          end if
      end if
      if( field%fill.NE.default_field%fill )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, '_FillValue', rval=field%missing, pack=
          else
              a = nint((field%fill-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, '_FillValue', rval=a, pack=field%pack )
          end if
      end if
      if( field%pack.NE.1 .AND. field%pack.NE.2 )then
          call mpp_write_meta( unit, field%id, 'packing', ival=field%pack )
          if( field%scale.NE.default_field%scale )call mpp_write_meta( unit, field%id, 'sc
          if( field%add.NE.default_field%add   )call mpp_write_meta( unit, field%id, 'add_
      end if
      if( verbose ) write (stdout(), '(a,2i3,1x,a,i3)') 'MPP_WRITE_META: Wrote field metac
           pe, unit, trim(field%name), field%id

      return
    end subroutine mpp_copy_meta_field

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!                               MPP_READ                               !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

#define MPP_READ_2DDECOMP_1D_ mpp_read_2ddecomp_r1d
#define MPP_READ_2DDECOMP_2D_ mpp_read_2ddecomp_r2d
#define MPP_READ_2DDECOMP_3D_ mpp_read_2ddecomp_r3d
#define MPP_READ_2DDECOMP_4D_ mpp_read_2ddecomp_r4d
#define MPP_TYPE_ real
#include <mpp_read_2Ddecomp.h>

    subroutine read_record( unit, field, nwords, data, time_level, domain )
!routine that is finally called by all mpp_read routines to perform the read
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(nwords)
      integer, intent(in), optional  :: time_level
      type(domain2D), intent(in), optional :: domain
      integer, dimension(size(field%axes)) :: start, axsiz
      real :: time

      logical :: newtime
      integer :: subdomain(4), tlevel

      integer(SHORT_KIND) :: i2vals(nwords)
```

```
!#ifdef __sgi
      integer(INT_KIND) :: ivals(nwords)
      real(FLOAT_KIND) :: rvals(nwords)
!#else
!      integer :: ivals(nwords)
!      real :: rvals(nwords)
!#endif

      real(DOUBLE_KIND) :: r8vals(nwords)

      integer :: i, error, is, ie, js, je, isg, ieg, jsg, jeg

#ifdef use_CRI_pointers
      pointer( ptr1, i2vals )
      pointer( ptr2, ivals )
      pointer( ptr3, rvals )
      pointer( ptr4, r8vals )

      if (mpp_io_stack_size < 4*nwords) call mpp_io_set_stack_size(4*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
      ptr3 = LOC(mpp_io_stack(2*nwords+1))
      ptr4 = LOC(mpp_io_stack(3*nwords+1))
#endif
      if (.not.PRESENT(time_level)) then
          tlevel = 0
      else
          tlevel = time_level
      endif

#ifdef use_netCDF
      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'READ_RECORD: must first call
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'READ_RECORD: invalid unit nu
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
!RV
      if( mpp_file(unit)%fileset.EQ.MPP_MULTI .and. present(domain)) &
        call mpp_error( FATAL, 'READ_RECORD: multiple filesets not supported for MPP_READ'

      if( .NOT.mpp_file(unit)%initialized ) call mpp_error( FATAL, 'MPP_READ: must first c



      if( verbose ) write (stdout(), '(a,2i3,2i5)') 'MPP_READ: PE, unit, %id, %time_level
           pe, unit, mpp_file(unit)%id, tlevel

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be read:
!  time axis: START = time level
!               AXSIZ = 1
!  space axis: if there is no domain info
!                START = 1
!                AXSIZ = field%size(axis)
!           if there IS domain info:
!                start of domain is compute%start_index for multi-file I/O
!                                   global%start_index for all other cases
!                this number must be converted to 1 for NF_GET_VAR
!                    (netCDF fortran calls are with reference to 1),
!           So, START = compute%start_index - <start of domain> + 1
!                AXSIZ = usually compute%size
!           However, if compute%start_index-compute%end_index+1.NE.compute%size,
!                we assume that the call is passing a subdomain.
!                To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1
! NOTE: passing of subdomains will fail for multi-PE single-threaded I/O,
!       since that attempts to gather all data on PE 0.
          start = 1
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( field%axes(i)%did.EQ.field%time_axis_index )start(i) = tlevel
          end do
          if( PRESENT(domain) )then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je  )
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg )
              axsiz(1) = ie-is+1
              axsiz(2) = je-js+1
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( ie-is+1.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( je-js+1.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
          end if

          if( verbose ) write (stdout(), '(a,2i3,i6,12i4)') 'READ_RECORD: PE, unit, nwords
                           pe, unit, nwords, start, axsiz

          select case (field%type)
             case(NF_BYTE)
! use type conversion
                call mpp_error( FATAL, 'MPP_READ: does not support NF_BYTE packing' )
             case(NF_SHORT)
                error = NF_GET_VARA_INT2  ( mpp_file(unit)%ncid, field%id, start, axsiz, i
                 data(:)=i2vals(:)*field%scale + field%add
             case(NF_INT)
                error = NF_GET_VARA_INT   ( mpp_file(unit)%ncid, field%id, start, axsiz, i
                data(:)=ivals(:)
             case(NF_FLOAT)
                error = NF_GET_VARA_REAL  ( mpp_file(unit)%ncid, field%id, start, axsiz, r
                data(:)=rvals(:)
             case(NF_DOUBLE)
                error = NF_GET_VARA_DOUBLE( mpp_file(unit)%ncid, field%id, start, axsiz, r
                data(:)=r8vals(:)
             case default
                call mpp_error( FATAL, 'MPP_READ: invalid pack value' )
          end select
      else                        !non-netCDF
!subdomain contains (/is,ie,js,je/)
          call mpp_error( FATAL, 'Currently dont support non-NetCDF mpp read' )

      end if
#else
      call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine read_record
    subroutine read_record_b(unit,field,nwords,data,time_level,domain,block_id)
!routine that is finally called by all mpp_read routines to perform the read
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(nwords)
      integer, intent(in), optional  :: time_level
!RV
      integer, intent(in), optional  ::  block_id
!RV
      type(domain2D), intent(in), optional :: domain
      integer, dimension(size(field%axes)) :: start, axsiz
      real :: time

      logical :: newtime
      integer :: subdomain(4), tlevel

      integer(SHORT_KIND) :: i2vals(nwords)
!#ifdef __sgi
      integer(INT_KIND) :: ivals(nwords)
      real(FLOAT_KIND) :: rvals(nwords)
!#else
!      integer :: ivals(nwords)
!      real :: rvals(nwords)
!#endif

      real(DOUBLE_KIND) :: r8vals(nwords)

      integer :: i, error, is, ie, js, je, isg, ieg, jsg, jeg

#ifdef use_CRI_pointers
      pointer( ptr1, i2vals )
      pointer( ptr2, ivals )
      pointer( ptr3, rvals )
      pointer( ptr4, r8vals )

      if (mpp_io_stack_size < 4*nwords) call mpp_io_set_stack_size(4*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
      ptr3 = LOC(mpp_io_stack(2*nwords+1))
      ptr4 = LOC(mpp_io_stack(3*nwords+1))
#endif
      if (.not.PRESENT(time_level)) then
          tlevel = 0
      else
          tlevel = time_level
      endif
```

```
#ifdef use_netCDF
      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'READ_RECORD: must first call
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'READ_RECORD: invalid unit nu
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
!RV
      if( mpp_file(unit)%fileset.EQ.MPP_MULTI .and. present(domain)) &
        call mpp_error( FATAL, 'READ_RECORD: multiple filesets not supported for MPP_READ'

      if( .NOT.mpp_file(unit)%initialized ) call mpp_error( FATAL, 'MPP_READ: must first c



      if( verbose ) write (stdout(), '(a,2i3,2i5)') 'MPP_READ: PE, unit, %id, %time_level
           pe, unit, mpp_file(unit)%id, tlevel

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be read:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NF_GET_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1
! NOTE: passing of subdomains will fail for multi-PE single-threaded I/O,
!       since that attempts to gather all data on PE 0.
          start = 1
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( field%axes(i)%did.EQ.field%time_axis_index )start(i) = tlevel
          end do
          if( PRESENT(domain) )then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je  )
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg )
              axsiz(1) = ie-is+1
              axsiz(2) = je-js+1
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( ie-is+1.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( je-js+1.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
          end if
!RV,SGI
```

```
          if( PRESENT(block_id) )then
            if (block_id.le.0) then
               call mpp_error( FATAL, 'READ_RECORD_B: block_id <= 0!' )
            endif
            if( PRESENT(time_level) )then

               if(block_id.gt. axsiz(size(field%axes)-1)) &
                 call mpp_error( FATAL, 'READ_RECORD_B: block_id > axis range!' )
               start(size(field%axes)-1)=block_id

            else

               if(block_id.gt. axsiz(size(field%axes))) &
                 call mpp_error( FATAL, 'READ_RECORD_B: block_id > axis range!' )
               start(size(field%axes))=block_id

            endif
          endif

!RV,SGI

          if( verbose ) write (stdout(), '(a,2i3,i6,12i4)') 'READ_RECORD: PE, unit, nwords
                           pe, unit, nwords, start, axsiz

          select case (field%type)
             case(NF_BYTE)
! use type conversion
                call mpp_error( FATAL, 'MPP_READ: does not support NF_BYTE packing' )
             case(NF_SHORT)
                error = NF_GET_VARA_INT2  ( mpp_file(unit)%ncid, field%id, start, axsiz, i
                 data(:)=i2vals(:)*field%scale + field%add
             case(NF_INT)
                error = NF_GET_VARA_INT   ( mpp_file(unit)%ncid, field%id, start, axsiz, i
                data(:)=ivals(:)
             case(NF_FLOAT)
                error = NF_GET_VARA_REAL  ( mpp_file(unit)%ncid, field%id, start, axsiz, r
                data(:)=rvals(:)
             case(NF_DOUBLE)
                error = NF_GET_VARA_DOUBLE( mpp_file(unit)%ncid, field%id, start, axsiz, r
                data(:)=r8vals(:)
             case default
                call mpp_error( FATAL, 'MPP_READ: invalid pack value' )
          end select
      else                          !non-netCDF
!subdomain contains (/is,ie,js,je/)
          call mpp_error( FATAL, 'Currently dont support non-NetCDF mpp read' )

      end if
#else
      call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine read_record_b

    subroutine mpp_read_r4D( unit, field, data, tindex,blockid)
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(:,:,:,:)
      integer, intent(in), optional :: tindex
      integer, intent(in), optional :: blockid

      if(present(blockid)) then
        call read_record_b(unit,field,size(data),data,tindex,block_id=blockid )
      else
        call read_record( unit, field, size(data), data, tindex )
      endif
    end subroutine mpp_read_r4D

    subroutine mpp_read_r3D( unit, field, data, tindex,blockid)
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(:,:,:)
      integer, intent(in), optional :: tindex
      integer, intent(in), optional :: blockid

      if(present(blockid)) then
        call read_record_b(unit,field,size(data),data,tindex,block_id=blockid )
      else
        call read_record( unit, field, size(data), data, tindex )
      endif
    end subroutine mpp_read_r3D

    subroutine mpp_read_r2D( unit, field, data, tindex )
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(:,:)
      integer, intent(in), optional :: tindex

      call read_record( unit, field, size(data), data, tindex )
    end subroutine mpp_read_r2D

    subroutine mpp_read_r1D( unit, field, data, tindex )
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(:)
      integer, intent(in), optional :: tindex

      call read_record( unit, field, size(data), data, tindex )
    end subroutine mpp_read_r1D

    subroutine mpp_read_r0D( unit, field, data, tindex )
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data
      integer, intent(in), optional :: tindex
      real, dimension(1) :: data_tmp

      data_tmp(1)=data
      call read_record( unit, field, 1, data_tmp, tindex )
      data=data_tmp(1)
    end subroutine mpp_read_r0D

    subroutine mpp_read_meta(unit)
!
! read file attributes including dimension and variable attributes
! and store in filetype structure.  All of the file information
! with the exception of the (variable) data is stored.  Attributes
! are supplied to the user by get_info,get_atts,get_axes and get_fields
!
! every PE is eligible to call mpp_read_meta
!
      integer, parameter :: MAX_DIMVALS = 100000
      integer, intent(in) :: unit

      integer         :: ncid,ndim,nvar_total,natt,recdim,nv,nvar,len
      integer :: error,i,j
      integer         :: type,nvdims,nvatts, dimid
      integer, allocatable, dimension(:) :: dimids
      type(axistype) , allocatable, dimension(:) :: Axis
      character(len=128) :: name, attname, unlimname, attval
      logical :: isdim

      integer(SHORT_KIND) :: i2vals(MAX_DIMVALS)
!#ifdef __sgi
      integer(INT_KIND) :: ivals(MAX_DIMVALS)
      real(FLOAT_KIND)  :: rvals(MAX_DIMVALS)
!#else
!      integer :: ivals(MAX_DIMVALS)
!      real    :: rvals(MAX_DIMVALS)
!#endif
      real(DOUBLE_KIND) :: r8vals(MAX_DIMVALS)

#ifdef use_netCDF

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
        ncid = mpp_file(unit)%ncid
        error = NF_INQ(ncid,ndim, nvar_total,&
                      natt, recdim);call netcdf_err(error)


        mpp_file(unit)%ndim = ndim
        mpp_file(unit)%natt = natt
        mpp_file(unit)%recdimid = recdim
!
! if no recdim exists, recdimid = -1
! variable id of unlimdim and length
!
        if( recdim.NE.-1 )then
           error = NF_INQ_DIM( ncid, recdim, unlimname, mpp_file(unit)%time_level );call r
           error = NF_INQ_VARID( ncid, unlimname, mpp_file(unit)%id ); call netcdf_err(err
        else
           mpp_file(unit)%time_level = -1 ! set to zero so mpp_get_info returns ntime=0 if
        endif

        if ( natt .gt. 0 ) allocate(mpp_file(unit)%Att(natt))
        allocate(Axis(ndim))
        allocate(dimids(ndim))
        allocate(mpp_file(unit)%Axis(ndim))

!
! initialize fieldtype and axis type
!


        do i=1,ndim
           Axis(i) = default_axis
           mpp_file(unit)%Axis(i) = default_axis
        enddo

        do i=1,natt
           mpp_file(unit)%Att(i) = default_att
        enddo

!
! assign global attributes
!
        do i=1,natt
           error=NF_INQ_ATTNAME(ncid,NF_GLOBAL,i,name);call netcdf_err(error)
           error=NF_INQ_ATT(ncid,NF_GLOBAL,trim(name),type,len);call netcdf_err(error)
           mpp_file(unit)%Att(i)%name = name
           mpp_file(unit)%Att(i)%len = len
           mpp_file(unit)%Att(i)%type = type
!
!  allocate space for att data and assign
!
           select case (type)
              case (NF_CHAR)
                 if (len.gt.512) then
                    call mpp_error(NOTE,'GLOBAL ATT too long - not reading this metadata')
                    len=7
                    mpp_file(unit)%Att(i)%len=len
                    mpp_file(unit)%Att(i)%catt = 'unknown'
                 else
                     error=NF_GET_ATT_TEXT(ncid,NF_GLOBAL,name,mpp_file(unit)%Att(i)%catt)
                     if (verbose.and.pe == 0) write (stdout(),*)  'GLOBAL ATT ',trim(name)
                 endif
!
! store integers in float arrays
!
              case (NF_SHORT)
                 allocate(mpp_file(unit)%Att(i)%fatt(len))
                 error=NF_GET_ATT_INT2(ncid,NF_GLOBAL,name,i2vals);call netcdf_err(error)
                 if( verbose .and. pe == 0 )write (stdout(),*)  'GLOBAL ATT ',trim(name),'
                 mpp_file(unit)%Att(i)%fatt(1:len)=i2vals(1:len)
              case (NF_INT)
                 allocate(mpp_file(unit)%Att(i)%fatt(len))
                 error=NF_GET_ATT_INT(ncid,NF_GLOBAL,name,ivals);call netcdf_err(error)
                 if( verbose .and. pe == 0 )write (stdout(),*)  'GLOBAL ATT ',trim(name),'
                 mpp_file(unit)%Att(i)%fatt(1:len)=ivals(1:len)
              case (NF_FLOAT)
                 allocate(mpp_file(unit)%Att(i)%fatt(len))
                 error=NF_GET_ATT_REAL(ncid,NF_GLOBAL,name,rvals);call netcdf_err(error)
                 mpp_file(unit)%Att(i)%fatt(1:len)=rvals(1:len)
                 if( verbose .and. pe == 0)write (stdout(),*)  'GLOBAL ATT ',trim(name),'
              case (NF_DOUBLE)
                 allocate(mpp_file(unit)%Att(i)%fatt(len))
                 error=NF_GET_ATT_DOUBLE(ncid,NF_GLOBAL,name,r8vals);call netcdf_err(error
                 mpp_file(unit)%Att(i)%fatt(1:len)=r8vals(1:len)
                 if( verbose .and. pe == 0)write (stdout(),*)  'GLOBAL ATT ',trim(name),'
           end select

        enddo
!
! assign dimension name and length
!
        do i=1,ndim
           error = NF_INQ_DIM(ncid,i,name,len);call netcdf_err(error)
           Axis(i)%name = name
           Axis(i)%len = len
        enddo

        nvar=0
        do i=1, nvar_total
           error=NF_INQ_VAR(ncid,i,name,type,nvdims,dimids,nvatts);call netcdf_err(error)
           isdim=.false.
           do j=1,ndim
              if( trim(lowercase(name)).EQ.trim(lowercase(Axis(j)%name)) )isdim=.true.
           enddo
           if (.not.isdim) nvar=nvar+1
        enddo
        mpp_file(unit)%nvar = nvar
        allocate(mpp_file(unit)%Var(nvar))
        do i=1,nvar
           mpp_file(unit)%Var(i) = default_field
        enddo

!
! assign dimension info
!
        do i=1, nvar_total
           error=NF_INQ_VAR(ncid,i,name,type,nvdims,dimids,nvatts);call netcdf_err(error)
           isdim=.false.
           do j=1,ndim
              if( trim(lowercase(name)).EQ.trim(lowercase(Axis(j)%name)) )isdim=.true.
           enddo

           if( isdim )then
              error=NF_INQ_DIMID(ncid,name,dimid);call netcdf_err(error)
              Axis(dimid)%type = type
              Axis(dimid)%did = dimid
              Axis(dimid)%id = i
              Axis(dimid)%natt = nvatts
              ! get axis values
              if( i.NE.mpp_file(unit)%id )then   ! non-record dims
                 select case (type)
                 case (NF_INT)
                    len=Axis(dimid)%len
                    allocate(Axis(dimid)%data(len))
                    error = NF_GET_VAR_INT(ncid,i,ivals);call netcdf_err(error)
                    Axis(dimid)%data(1:len)=ivals(1:len)
                 case (NF_FLOAT)
                    len=Axis(dimid)%len
                    allocate(Axis(dimid)%data(len))
                    error = NF_GET_VAR_REAL(ncid,i,rvals);call netcdf_err(error)
                    Axis(dimid)%data(1:len)=rvals(1:len)
                 case (NF_DOUBLE)
                    len=Axis(dimid)%len
                    allocate(Axis(dimid)%data(len))
                    error = NF_GET_VAR_DOUBLE(ncid,i,r8vals);call netcdf_err(error)
                    Axis(dimid)%data(1:len) = r8vals(1:len)
                 case (NF_CHAR) !RV,bundle
                    len=Axis(dimid)%len !RV,bundle
                    allocate(Axis(dimid)%cdata(len)) !RV,bundle
                    error = NF_GET_VAR_TEXT(ncid,i,Axis(dimid)%cdata) !RV,bundle
                    print*,'cdata',Axis(dimid)%cdata !RV,bundle
                    call netcdf_err(error) !RV,bundle
                 case default
                    call mpp_error( FATAL, 'Invalid data type for dimension' )
                 end select
             else
                 len = mpp_file(unit)%time_level
                 allocate(mpp_file(unit)%time_values(len))
                 select case (type)
                 case (NF_FLOAT)
                    error = NF_GET_VAR_REAL(ncid,i,rvals);call netcdf_err(error)
                    mpp_file(unit)%time_values(1:len) = rvals(1:len)
                 case (NF_DOUBLE)
                    error = NF_GET_VAR_DOUBLE(ncid,i,r8vals);call netcdf_err(error)
                    mpp_file(unit)%time_values(1:len) = r8vals(1:len)
                 case default
                    call mpp_error( FATAL, 'Invalid data type for dimension' )
                 end select
              endif
              ! assign dimension atts
```

```
if( nvatts.GT.0 )allocate(Axis(dimid)%Att(nvatts))

do j=1,nvatts
   Axis(dimid)%Att(j) = default_att
enddo

do j=1,nvatts
   error=NF_INQ_ATTNAME(ncid,i,j,attname);call netcdf_err(error)
   error=NF_INQ_ATT(ncid,i,trim(attname),type,len);call netcdf_err(error)

   Axis(dimid)%Att(j)%name = trim(attname)
   Axis(dimid)%Att(j)%type = type
   Axis(dimid)%Att(j)%len = len

   select case (type)
   case (NF_CHAR)
      if (len.gt.512) call mpp_error(FATAL,'DIM ATT too long')
      error=NF_GET_ATT_TEXT(ncid,i,trim(attname),Axis(dimid)%Att(j)%catt);ca
      if( verbose .and. pe == 0 ) &
           write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
      ! store integers in float arrays
      ! assume dimension data not packed
   case (NF_SHORT)
      allocate(Axis(dimid)%Att(j)%fatt(len))
      error=NF_GET_ATT_INT2(ncid,i,trim(attname),i2vals);call netcdf_err(err
      Axis(dimid)%Att(j)%fatt(1:len)=i2vals(1:len)
      if( verbose .and. pe == 0  ) &
           write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
   case (NF_INT)
      allocate(Axis(dimid)%Att(j)%fatt(len))
      error=NF_GET_ATT_INT(ncid,i,trim(attname),ivals);call netcdf_err(error
      Axis(dimid)%Att(j)%fatt(1:len)=ivals(1:len)
      if( verbose .and. pe == 0  ) &
           write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
   case (NF_FLOAT)
      allocate(Axis(dimid)%Att(j)%fatt(len))
      error=NF_GET_ATT_REAL(ncid,i,trim(attname),rvals);call netcdf_err(erro
      Axis(dimid)%Att(j)%fatt(1:len)=rvals(1:len)
      if( verbose  .and. pe == 0 ) &
           write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
   case (NF_DOUBLE)
      allocate(Axis(dimid)%Att(j)%fatt(len))
      error=NF_GET_ATT_DOUBLE(ncid,i,trim(attname),r8vals);call netcdf_err(e
      Axis(dimid)%Att(j)%fatt(1:len)=r8vals(1:len)
      if( verbose  .and. pe == 0 ) &
           write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
   case default
      call mpp_error( FATAL, 'Invalid data type for dimension at' )
   end select
   ! assign pre-defined axis attributes
   select case(trim(attname))
   case('long_name')
      Axis(dimid)%longname=Axis(dimid)%Att(j)%catt(1:len)
   case('units')
      Axis(dimid)%units=Axis(dimid)%Att(j)%catt(1:len)
   case('cartesian_axis')
      Axis(dimid)%cartesian=Axis(dimid)%Att(j)%catt(1:len)
   case('positive')
      attval = Axis(dimid)%Att(j)%catt(1:len)
      if( attval.eq.'down' )then
         Axis(dimid)%sense=-1
      else if( attval.eq.'up' )then
         Axis(dimid)%sense=1
```

```
                    endif
                 end select

              enddo
              ! store axis info in filetype
              mpp_file(unit)%Axis(dimid) = Axis(dimid)
           endif
        enddo
! assign variable info
        nv = 0
        do i=1, nvar_total
           error=NF_INQ_VAR(ncid,i,name,type,nvdims,dimids,nvatts);call netcdf_err(error)
!
! is this a dimension variable?
!
           isdim=.false.
           do j=1,ndim
              if( trim(lowercase(name)).EQ.trim(lowercase(Axis(j)%name)) )isdim=.true.
           enddo

           if( .not.isdim )then
! for non-dimension variables
              nv=nv+1; if( nv.GT.mpp_file(unit)%nvar )call mpp_error( FATAL, 'variable inc
              mpp_file(unit)%Var(nv)%type = type
              mpp_file(unit)%Var(nv)%id = i
              mpp_file(unit)%Var(nv)%name = name
              mpp_file(unit)%Var(nv)%natt = nvatts
! determine packing attribute based on NetCDF variable type
             select case (type)
             case(NF_SHORT)
                 mpp_file(unit)%Var(nv)%pack = 4
             case(NF_FLOAT)
                 mpp_file(unit)%Var(nv)%pack = 2
             case(NF_DOUBLE)
                 mpp_file(unit)%Var(nv)%pack = 1
             case (NF_INT)
                 mpp_file(unit)%Var(nv)%pack = 2
             case default
                   call mpp_error( FATAL, 'Invalid variable type in NetCDF file' )
             end select
! assign dimension ids
              mpp_file(unit)%Var(nv)%ndim = nvdims
              allocate(mpp_file(unit)%Var(nv)%axes(nvdims))
              do j=1,nvdims
                 mpp_file(unit)%Var(nv)%axes(j) = Axis(dimids(j))
              enddo
              allocate(mpp_file(unit)%Var(nv)%size(nvdims))

              do j=1,nvdims
                 if( dimids(j).eq.mpp_file(unit)%recdimid )then
                    mpp_file(unit)%Var(nv)%time_axis_index = dimids(j)
                    mpp_file(unit)%Var(nv)%size(j)=1    ! dimid length set to 1 here for c
                 else
                    mpp_file(unit)%Var(nv)%size(j)=Axis(dimids(j))%len
                 endif
              enddo
! assign variable atts
              if( nvatts.GT.0 )allocate(mpp_file(unit)%Var(nv)%Att(nvatts))

              do j=1,nvatts
                 mpp_file(unit)%Var(nv)%Att(j) = default_att
              enddo
              do j=1,nvatts
                 error=NF_INQ_ATTNAME(ncid,i,j,attname);call netcdf_err(error)
                 error=NF_INQ_ATT(ncid,i,attname,type,len);call netcdf_err(error)
                 mpp_file(unit)%Var(nv)%Att(j)%name = trim(attname)
                 mpp_file(unit)%Var(nv)%Att(j)%type = type
                 mpp_file(unit)%Var(nv)%Att(j)%len = len

                 select case (type)
                   case (NF_CHAR)
                     if (len.gt.512) call mpp_error(FATAL,'VAR ATT too long')
                     error=NF_GET_ATT_TEXT(ncid,i,trim(attname),mpp_file(unit)%Var(nv)%Att
                     if (verbose .and. pe == 0 )&
                           write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_fil
! store integers as float internally
                   case (NF_SHORT)
                     allocate(mpp_file(unit)%Var(nv)%Att(j)%fatt(len))
                     error=NF_GET_ATT_INT2(ncid,i,trim(attname),i2vals);call netcdf_err(er
                     mpp_file(unit)%Var(nv)%Att(j)%fatt(1:len)= i2vals(1:len)
                     if( verbose  .and. pe == 0 )&
                          write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_file
                   case (NF_INT)
                     allocate(mpp_file(unit)%Var(nv)%Att(j)%fatt(len))
                     error=NF_GET_ATT_INT(ncid,i,trim(attname),ivals);call netcdf_err(erro
                     mpp_file(unit)%Var(nv)%Att(j)%fatt(1:len)=ivals(1:len)
                     if( verbose .and. pe == 0  )&
                          write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_file
                   case (NF_FLOAT)
                     allocate(mpp_file(unit)%Var(nv)%Att(j)%fatt(len))
                     error=NF_GET_ATT_REAL(ncid,i,trim(attname),rvals);call netcdf_err(err
                     mpp_file(unit)%Var(nv)%Att(j)%fatt(1:len)=rvals(1:len)
                     if( verbose  .and. pe == 0 )&
                          write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_file
                   case (NF_DOUBLE)
                     allocate(mpp_file(unit)%Var(nv)%Att(j)%fatt(len))
                     error=NF_GET_ATT_DOUBLE(ncid,i,trim(attname),r8vals);call netcdf_err(
                     mpp_file(unit)%Var(nv)%Att(j)%fatt(1:len)=r8vals(1:len)
                     if( verbose .and. pe == 0  ) &
                          write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_file
                   case default
                        call mpp_error( FATAL, 'Invalid data type for variable att' )
                 end select
! assign pre-defined field attributes
                 select case (trim(attname))
                    case ('long_name')
                      mpp_file(unit)%Var(nv)%longname=mpp_file(unit)%Var(nv)%Att(j)%catt(1
                    case('units')
                      mpp_file(unit)%Var(nv)%units=mpp_file(unit)%Var(nv)%Att(j)%catt(1:le
                    case('scale_factor')
                       mpp_file(unit)%Var(nv)%scale=mpp_file(unit)%Var(nv)%Att(j)%fatt(1)
                    case('missing')
                       mpp_file(unit)%Var(nv)%missing=mpp_file(unit)%Var(nv)%Att(j)%fatt(1
                    case('add_offset')
                       mpp_file(unit)%Var(nv)%add=mpp_file(unit)%Var(nv)%Att(j)%fatt(1)
                    case('valid_range')
                       mpp_file(unit)%Var(nv)%min=mpp_file(unit)%Var(nv)%Att(j)%fatt(1)
                       mpp_file(unit)%Var(nv)%max=mpp_file(unit)%Var(nv)%Att(j)%fatt(2)
                 end select
              enddo
           endif
        enddo   ! end variable loop
      else
        call mpp_error( FATAL,  'MPP READ CURRENTLY DOES NOT SUPPORT NON-NETCDF' )
      endif
      mpp_file(unit)%initialized = .TRUE.
#else
      call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine mpp_read_meta


    subroutine mpp_get_info( unit, ndim, nvar, natt, ntime )

      integer, intent(in) :: unit
      integer, intent(out) :: ndim, nvar, natt, ntime


      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_INFO: must first cal
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_INFO: invalid unit r

      ndim = mpp_file(unit)%ndim
      nvar = mpp_file(unit)%nvar
      natt = mpp_file(unit)%natt
      ntime = mpp_file(unit)%time_level

      return

    end subroutine mpp_get_info


    subroutine mpp_get_global_atts( unit, global_atts )
!
!  copy global file attributes for use by user
!
!  global_atts is an attribute type which is allocated from the
!  calling routine

      integer,       intent(in)    :: unit
      type(atttype), intent(inout) :: global_atts(:)
      integer :: natt,i

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_INFO: must first cal
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_INFO: invalid unit r

      if (size(global_atts).lt.mpp_file(unit)%natt) &
      call mpp_error(FATAL, 'MPP_GET_ATTS: atttype not dimensioned properly in calling rou

      natt = mpp_file(unit)%natt
      global_atts = default_att

      do i=1,natt
         global_atts(i) = mpp_file(unit)%Att(i)
      enddo

      return
   end subroutine mpp_get_global_atts

   subroutine mpp_get_field_atts( field, name, units, longname, min, max, missing, ndim, s

     type(fieldtype), intent(in) :: field
     character(len=*), intent(out) , optional :: name, units
     character(len=*), intent(out), optional :: longname
     real,intent(out), optional :: min,max,missing
     integer, intent(out), optional :: ndim
     integer, intent(out), dimension(:), optional :: siz
     type(atttype), intent(out), optional, dimension(:) :: atts
     type(axistype), intent(out), optional, dimension(:) :: axes

     integer :: n,m

     if (PRESENT(name)) name = field%name
     if (PRESENT(units)) units = field%units
     if (PRESENT(longname)) longname = field%longname
     if (PRESENT(min)) min = field%min
     if (PRESENT(max)) max = field%max
     if (PRESENT(missing)) missing = field%missing
     if (PRESENT(ndim)) ndim = field%ndim
     if (PRESENT(atts)) then
        atts = default_att
        n = size(atts);m=size(field%Att)
        if (n.LT.m) call mpp_error(FATAL,'attribute array not large enough in mpp_get_fiel
        atts(1:m) = field%Att(1:m)
     end if
     if (PRESENT(axes)) then
        axes = default_axis
        n = size(axes);m=field%ndim
        if (n.LT.m) call mpp_error(FATAL,'axis array not large enough in mpp_get_field_att
        axes(1:m) = field%axes(1:m)
     end if
     if (PRESENT(siz)) then
        siz = -1
        n = size(siz);m=field%ndim
        if (n.LT.m) call mpp_error(FATAL,'size array not large enough in mpp_get_field_att
        siz(1:m) = field%size(1:m)
     end if
     return
   end subroutine mpp_get_field_atts

   subroutine mpp_get_axis_atts( axis, name, units, longname, cartesian, sense, len, natts

     type(axistype), intent(in) :: axis
     character(len=*), intent(out) , optional :: name, units
     character(len=*), intent(out), optional :: longname, cartesian
     integer,intent(out), optional :: sense, len , natts
     type(atttype), intent(out), optional, dimension(:) :: atts

     integer :: n,m

     if (PRESENT(name)) name = axis%name
     if (PRESENT(units)) units = axis%units
     if (PRESENT(longname)) longname = axis%longname
     if (PRESENT(cartesian)) cartesian = axis%cartesian
     if (PRESENT(sense)) sense = axis%sense
     if (PRESENT(len)) len = axis%len
     if (PRESENT(atts)) then
        atts = default_att
        n = size(atts);m=size(axis%Att)
        if (n.LT.m) call mpp_error(FATAL,'attribute array not large enough in mpp_get_fiel
        atts(1:m) = axis%Att(1:m)
     end if
     if (PRESENT(natts)) natts = size(axis%Att)

     return
   end subroutine mpp_get_axis_atts


    subroutine mpp_get_fields( unit, variables )
```

```
!
!  copy variable information from file (excluding data)
!  global_atts is an attribute type which is allocated from the
!  calling routine
!
      integer,         intent(in)    :: unit
      type(fieldtype), intent(inout) :: variables(:)

      integer :: nvar,i

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_FIELDS: must first c
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_FIELDS: invalid unit

      if (size(variables).ne.mpp_file(unit)%nvar) &
      call mpp_error(FATAL, 'MPP_GET_FIELDS: fieldtype not dimensioned properly in calling

      nvar = mpp_file(unit)%nvar

      do i=1,nvar
         variables(i) = mpp_file(unit)%Var(i)
      enddo

      return
   end subroutine mpp_get_fields

    subroutine mpp_get_axes( unit, axes, time_axis )
!
!  copy variable information from file (excluding data)
!  global_atts is an attribute type which is allocated from the
!  calling routine
!
      integer, intent(in) :: unit
      type(axistype), intent(out) :: axes(:)
      type(axistype), intent(out), optional :: time_axis
      character(len=128) :: name
      logical :: save
      integer :: ndim,i, nvar, j, num_dims, k

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_AXES: must first cal
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_AXES: invalid unit r

      if (size(axes).ne.mpp_file(unit)%ndim) &
      call mpp_error(FATAL, 'MPP_GET_AXES: axistype not dimensioned properly in calling rc


      if (PRESENT(time_axis)) time_axis = default_axis
      ndim = mpp_file(unit)%ndim
      do i=1,ndim
        if (ASSOCIATED(mpp_file(unit)%Axis(i)%data)) then
           axes(i)=mpp_file(unit)%Axis(i)
       else
           axes(i)=mpp_file(unit)%Axis(i)
           if (PRESENT(time_axis)) time_axis = mpp_file(unit)%Axis(i)
        endif
      enddo

      return
   end subroutine mpp_get_axes

   subroutine mpp_get_times( unit, time_values )
!
!  copy time information from file and convert to time_type
!
```

```
   integer, intent(in) :: unit
   real(DOUBLE_KIND), intent(inout) :: time_values(:)

   integer :: ntime,i

   if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_TIMES: must first ca
   if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_TIMES: invalid unit

   if (size(time_values).ne.mpp_file(unit)%time_level) &
   call mpp_error(FATAL, 'MPP_GET_TIMES: time_values not dimensioned properly in callin

   ntime = mpp_file(unit)%time_level

   do i=1,ntime
      time_values(i) = mpp_file(unit)%time_values(i)
   enddo



   return
end subroutine mpp_get_times

function mpp_get_field_index(fields,fieldname)

  type(fieldtype), dimension(:) :: fields
  character(len=*) :: fieldname
  integer :: mpp_get_field_index

  integer :: n

  mpp_get_field_index = -1

  do n=1,size(fields)
     if (lowercase(fields(n)%name) == lowercase(fieldname)) then
        mpp_get_field_index = n
        exit
     endif
  enddo

  return
end function mpp_get_field_index

function mpp_get_field_size(field)

  type(fieldtype) :: field
  integer :: mpp_get_field_size(4)

  integer :: n

  mpp_get_field_size = -1

  mpp_get_field_size(1) = field%size(1)
  mpp_get_field_size(2) = field%size(2)
  mpp_get_field_size(3) = field%size(3)
  mpp_get_field_size(4) = field%size(4)

  return
end function mpp_get_field_size


subroutine mpp_get_axis_data( axis, data )

  type(axistype), intent(in) :: axis
```

```
     real, dimension(:), intent(out) :: data


     if (size(data).lt.axis%len) call mpp_error(FATAL,'MPP_GET_AXIS_DATA: data array not l
     if (.NOT.ASSOCIATED(axis%data)) then
        call mpp_error(NOTE,'MPP_GET_AXIS_DATA: use mpp_get_times for record dims')
        data = 0.
     else
        data(1:axis%len) = axis%data
     endif

     return
   end subroutine mpp_get_axis_data


   function mpp_get_recdimid(unit)
!
      integer, intent(in) :: unit
      integer  :: mpp_get_recdimid


      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_RECDIMID: must first
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_RECDIMID: invalid ur

      mpp_get_recdimid = mpp_file(unit)%recdimid

      return
   end function mpp_get_recdimid

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!         mpp_get_iospec, mpp_flush: OS-dependent calls                !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    subroutine mpp_flush(unit)
!flush the output on a unit, syncing with disk
      integer, intent(in) :: unit

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_FLUSH: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_FLUSH: invalid unit numb
      if( .NOT.mpp_file(unit)%initialized )call mpp_error( FATAL, 'MPP_FLUSH: cannot flush
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          error = NF_SYNC(mpp_file(unit)%ncid); call netcdf_err(error)
#endif
      else
          call mpp_flushstd(unit)
      end if
      return
    end subroutine mpp_flush

    subroutine mpp_get_iospec( unit, iospec )
      integer, intent(in) :: unit
      character(len=*), intent(out) :: iospec

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_IOSPEC: must first c
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_IOSPEC: invalid unit
#ifdef SGICRAY
!currently will write to stdout: don't know how to trap and return as string to iospec
      call ASSIGN( 'assign -V f:'//trim(mpp_file(unit)%name), error )
```

```
#endif
      return
    end subroutine mpp_get_iospec

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!         netCDF-specific routines: mpp_get_id, netcdf_error         !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    function mpp_get_ncid(unit)
      integer :: mpp_get_ncid
      integer, intent(in) :: unit

      mpp_get_ncid = mpp_file(unit)%ncid
      return
    end function mpp_get_ncid

    function mpp_get_axis_id(axis)
      integer mpp_get_axis_id
      type(axistype), intent(in) :: axis
      mpp_get_axis_id = axis%id
      return
    end function mpp_get_axis_id

    function mpp_get_field_id(field)
      integer mpp_get_field_id
      type(fieldtype), intent(in) :: field
      mpp_get_field_id = field%id
      return
    end function mpp_get_field_id

    subroutine netcdf_err(err)
      integer, intent(in) :: err
      character(len=80) :: errmsg
      integer :: unit

#ifdef use_netCDF
      if( err.EQ.NF_NOERR )return
      errmsg = NF_STRERROR(err)
      call mpp_io_exit()        !make sure you close all open files
      call mpp_error( FATAL, 'NETCDF ERROR: '//trim(errmsg) )
#endif
      return
    end subroutine netcdf_err

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!       minor routines: mpp_get_unit_range, mpp_set_unit_range        !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    subroutine mpp_get_unit_range( unit_begin_out, unit_end_out )
      integer, intent(out) ::      unit_begin_out, unit_end_out

      unit_begin_out = unit_begin; unit_end_out = unit_end
      return
    end subroutine mpp_get_unit_range

    subroutine mpp_set_unit_range( unit_begin_in, unit_end_in )
      integer, intent(in) ::       unit_begin_in, unit_end_in

      if( unit_begin_in.GT.unit_end_in )call mpp_error( FATAL, 'MPP_SET_UNIT_RANGE: unit_b
      if( unit_begin_in.LT.0            )call mpp_error( FATAL, 'MPP_SET_UNIT_RANGE: unit_b
      if( unit_end_in  .GT.maxunits     )call mpp_error( FATAL, 'MPP_SET_UNIT_RANGE: unit_e
      unit_begin = unit_begin_in; unit_end = unit_end_in
      return
    end subroutine mpp_set_unit_range

    subroutine mpp_modify_axis_meta( axis, name, units, longname, cartesian, data )

      type(axistype), intent(inout) :: axis
      character(len=*), intent(in), optional :: name, units, longname, cartesian
      real, dimension(:), intent(in), optional :: data

      if (PRESENT(name)) axis%name = trim(name)
      if (PRESENT(units)) axis%units = trim(units)
      if (PRESENT(longname)) axis%longname = trim(longname)
      if (PRESENT(cartesian)) axis%cartesian = trim(cartesian)
      if (PRESENT(data)) then
         axis%len = size(data)
         if (ASSOCIATED(axis%data)) deallocate(axis%data)
         allocate(axis%data(axis%len))
         axis%data = data
      endif

      return
    end subroutine mpp_modify_axis_meta

    subroutine mpp_modify_field_meta( field, name, units, longname, min, max, missing, axe

      type(fieldtype), intent(inout) :: field
      character(len=*), intent(in), optional :: name, units, longname
      real, intent(in), optional :: min, max, missing
      type(axistype), dimension(:), intent(inout), optional :: axes

      if (PRESENT(name)) field%name = trim(name)
      if (PRESENT(units)) field%units = trim(units)
      if (PRESENT(longname)) field%longname = trim(longname)
      if (PRESENT(min)) field%min = min
      if (PRESENT(max)) field%max = max
      if (PRESENT(missing)) field%missing = missing
!      if (PRESENT(axes)) then
!         axis%len = size(data)
!         deallocate(axis%data)
!         allocate(axis%data(axis%len))
!         axis%data = data
!      endif

      return
    end subroutine mpp_modify_field_meta

    function lowercase (cs)
      implicit none
      character(len=*), intent(in) :: cs
      character(len=len(cs))       :: lowercase

      integer, parameter :: co=iachar('a')-iachar('A') ! case offset
      integer :: i
      character :: ca

      lowercase = cs
      do i = 1, len(cs)
        ca = cs(i:i)
        if (ca >= "A" .and. ca <= "Z") then
          lowercase(i:i) =  achar(iachar(ca)+co)
        endif
      enddo

    end function lowercase

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                     !
!       minor routines: mpp_nullify_axistype,                         !
!                       mpp_nullify_axistype_array                    !
!                                                                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    subroutine mpp_nullify_axistype(axis)
      type(axistype), intent(inout) :: axis

      Nullify(axis%data)
      Nullify(axis%cdata)
      Nullify(axis%Att)
    end subroutine mpp_nullify_axistype

    subroutine mpp_nullify_axistype_array(axis)
      type(axistype), intent(inout), dimension(:) :: axis
      integer :: i

      do i=1, size(axis)
        Nullify(axis(i)%data)
        Nullify(axis(i)%cdata)
        Nullify(axis(i)%Att)
      enddo
    end subroutine mpp_nullify_axistype_array

end module mpp_io_mod_oa

#else
!
!ParNetCDF
!
module mpp_io_mod_oa
  use mod_kinds_mpp
  use mpp_mod_oa
  use mpp_domains_mod_oa
  implicit none
#include <os.h>
  private

  character(len=128), private :: version= &
       '$Id$'
  character(len=128), private :: tagname= &
       '$Name$'

  integer, private :: pe, npes

  type, public :: axistype
     private
     character(len=128) :: name
     character(len=128) :: units
     character(len=256) :: longname
     character(len=8) :: cartesian
     integer :: sense, len           !+/-1, depth or height?
     type(domain1D) :: domain !if pointer is associated, it is a distributed data axis
     real, pointer :: data(:)   !axis values (not used if time axis)
     character(len=64), pointer :: cdata(:)  !RV,bundles
     integer :: clenid                        !RV,bundles
     integer :: id, did, type, natt         !id is the "variable ID", did is the "dimensio
     type(atttype), pointer :: Att(:)
  end type axistype

  type, public :: atttype
     integer :: type, len
     character(len=128) :: name
     character(len=256) :: catt
! just use type conversion for integers
     real, pointer :: fatt(:)
  end type atttype

  type, public :: fieldtype
     private
     character(len=128) :: name
     character(len=128) :: units
     character(len=256) :: longname
     real :: min, max, missing, fill, scale, add
     integer :: pack
     type(axistype), pointer :: axes(:) !axes associated with field
!size, time_axis_index redundantly hold info already contained in axes
!it's clunky and inelegant, but required so that axes can be shared among multiple files
     integer, pointer :: size(:)
     integer :: time_axis_index
     integer :: id, type, natt, ndim
     type(atttype), pointer :: Att(:)
  end type fieldtype

  type, private :: filetype
     character(len=256) :: name
     integer :: action, format, access, threading, fileset, record, ncid
     logical :: opened, initialized, nohdrs
     integer :: time_level
     real(DOUBLE_KIND) :: time
     integer :: id             !variable ID of time axis associated with file (only one ti
     integer :: recdimid             !dim ID of time axis associated with file (only one t
!
! time axis values are stored here instead of axis%data since mpp_write
! assumes these values are not time values. Not used in mpp_write
!
     real(DOUBLE_KIND), pointer :: time_values(:)

! additional elements of filetype for mpp_read (ignored for mpp_write)
     integer :: ndim, nvar, natt  ! number of dimensions, non-dimension variables and glob
! redundant axis types stored here and in associated fieldtype
! some axes are not used by any fields, i.e. "edges"
     type(axistype), pointer  :: axis(:)
     type(fieldtype), pointer :: var(:)
     type(atttype), pointer   :: att(:)
  end type filetype

  type(axistype), public  :: default_axis !provided to users with default components
  type(fieldtype), public :: default_field !provided to users with default components
  type(atttype), public   :: default_att !provided to users with default components
!action on open
  integer, parameter, public :: MPP_WRONLY=100, MPP_RDONLY=101, MPP_APPEND=102, MPP_OVERWR
!format
  integer, parameter, public :: MPP_ASCII=200,  MPP_IEEE32=201, MPP_NATIVE=202, MPP_NETCDF
!access
  integer, parameter, public :: MPP_SEQUENTIAL=300, MPP_DIRECT=301
!threading, fileset
  integer, parameter, public :: MPP_SINGLE=400, MPP_MULTI=401, MPP_PARALLEL=402
!action on close
```

```
  integer, parameter, public :: MPP_DELETE=501, MPP_COLLECT=502

  type(filetype), private, allocatable :: mpp_file(:)
  integer, private :: records_per_pe
  integer, private :: maxunits, unit_begin, unit_end
  integer, private :: varnum=0
  integer, private :: error
  character(len=256) :: text
!null unit: returned by PEs not participating in IO after a collective call
  integer, parameter, private :: NULLUNIT=-1
  real(DOUBLE_KIND), parameter, private :: NULLTIME=-1.
#ifdef DEBUG
  logical, private :: verbose=.FALSE., debug=.TRUE., module_is_initialized=.FALSE.
#else
  logical, private :: verbose=.FALSE., debug=.FALSE., module_is_initialized=.FALSE.
#endif

  real(DOUBLE_KIND), private, allocatable :: mpp_io_stack(:)
  integer, private :: mpp_io_stack_size=0, mpp_io_stack_hwm=0

  interface mpp_write_meta
     module procedure mpp_write_meta_var
     module procedure mpp_write_meta_scalar_r
     module procedure mpp_write_meta_scalar_i
     module procedure mpp_write_meta_axis
     module procedure mpp_write_meta_field
     module procedure mpp_write_meta_global
     module procedure mpp_write_meta_global_scalar_r
     module procedure mpp_write_meta_global_scalar_i
  end interface

  interface mpp_copy_meta
     module procedure mpp_copy_meta_axis
     module procedure mpp_copy_meta_field
     module procedure mpp_copy_meta_global
  end interface

  interface mpp_write
     module procedure mpp_write_2ddecomp_r1d
     module procedure mpp_write_2ddecomp_r2d
     module procedure mpp_write_2ddecomp_r3d
     module procedure mpp_write_2ddecomp_r4d
     module procedure mpp_write_r0D
     module procedure mpp_write_r1D
     module procedure mpp_write_r2D
     module procedure mpp_write_r3D
     module procedure mpp_write_r4D
     module procedure mpp_write_axis
  end interface

  interface mpp_read
     module procedure mpp_read_2ddecomp_r1d
     module procedure mpp_read_2ddecomp_r2d
     module procedure mpp_read_2ddecomp_r3d
     module procedure mpp_read_2ddecomp_r4d
     module procedure mpp_read_r0D
     module procedure mpp_read_r1D
     module procedure mpp_read_r2D
     module procedure mpp_read_r3D
     module procedure mpp_read_r4D
  end interface

  interface mpp_get_id
```

```
     module procedure mpp_get_axis_id
     module procedure mpp_get_field_id
  end interface

  interface mpp_get_atts
     module procedure mpp_get_global_atts
     module procedure mpp_get_field_atts
     module procedure mpp_get_axis_atts
  end interface

  interface mpp_modify_meta
!     module procedure mpp_modify_att_meta
     module procedure mpp_modify_field_meta
     module procedure mpp_modify_axis_meta
  end interface

  public :: mpp_close, mpp_flush, mpp_get_iospec, mpp_get_id, mpp_get_ncid, mpp_get_unit_r
            mpp_open, mpp_set_unit_range, mpp_write, mpp_write_meta, mpp_read, mpp_get_inf
            mpp_get_fields, mpp_get_times, mpp_get_axes, mpp_copy_meta, mpp_get_recdimid,
            mpp_io_set_stack_size, mpp_get_field_index, mpp_nullify_axistype, mpp_nullify_

  private :: read_record, mpp_read_meta, lowercase

#ifdef use_netCDF
#include <pnetcdf.inc>
!rr
#ifdef NAG_COMPILER
    use mpi
#else
#include <mpif.h>
!!include 'mpif.h'
#endif

  integer(kind=MPI_OFFSET_KIND), private :: idim
#endif

  contains

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                       !
!               mpp_io_init: initialize parallel I/O                    !
!                                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine mpp_io_init( flags, maxunit,maxresunit )
      integer, intent(in), optional :: flags, maxunit ,maxresunit
!rv
!rv I introduced the variable to indentify that the top max_reserved_units
!rv of maxunits are reserved for OASIS coupler specific things like the trace
!rv files. This variable is active only if one specifies explicitely  the
!rv argument maxunit.
      integer::max_reserved_units
!rv
!initialize IO package: initialize mpp_file array, set valid range of units for fortran IC

      if( module_is_initialized )return
      call mpp_init(flags)           !if mpp_init has been called, this call will merely r
      pe = mpp_pe()
      npes = mpp_npes()
      call mpp_domains_init(flags)

      maxunits = 64
      if( PRESENT(maxunit) )maxunits = maxunit
      max_reserved_units=5
      if( PRESENT(maxresunit) )max_reserved_units = maxresunit

      if( PRESENT(flags) )then
          debug   = flags.EQ.MPP_DEBUG
          verbose = flags.EQ.MPP_VERBOSE .OR. debug
      end if
!initialize default_field
      default_field%name = 'noname'
      default_field%units = 'nounits'
      default_field%longname = 'noname'
      default_field%id = -1
      default_field%type = -1
      default_field%natt = -1
      default_field%ndim = -1
!largest possible 4-byte reals
      default_field%min = -huge(1._ip_single_mpp)
      default_field%max =  huge(1._ip_single_mpp)
      default_field%missing = -1e36
      default_field%fill = -1e36
      default_field%scale = 0.
      default_field%add = huge(1._ip_single_mpp)
      default_field%pack = 1
      default_field%time_axis_index = -1 !this value will never match any index
      Nullify(default_field%axes)
      Nullify(default_field%size)
      Nullify(default_field%att)
! Initialize default axis
      default_axis%name = 'noname'
      default_axis%units = 'nounits'
      default_axis%longname = 'noname'
      default_axis%cartesian = 'none'
      default_axis%sense = 0
      default_axis%len = -1
      default_axis%id = -1
      default_axis%did = -1
      default_axis%type = -1
      default_axis%natt = -1
      Nullify(default_axis%data)
! Initialize default attribute
      default_att%name = 'noname'
      default_att%type = -1
      default_att%len = -1
      default_att%catt = 'none'
      Nullify(default_att%fatt)

!up to MAXUNITS fortran units and MAXUNITS netCDF units are supported
!file attributes (opened, format, access, threading, fileset) are saved against the unit r
!external handles to netCDF units are saved from maxunits+1:2*maxunits
      allocate( mpp_file(NULLUNIT:2*maxunits) ) !starts at NULLUNIT=-1, used by non-partic
      mpp_file(:)%name   = ' '
      mpp_file(:)%action    = -1
      mpp_file(:)%format    = -1
      mpp_file(:)%threading = -1
      mpp_file(:)%fileset   = -1
      mpp_file(:)%record    = -1
      mpp_file(:)%ncid      = -1
      mpp_file(:)%opened = .FALSE.
      mpp_file(:)%initialized = .FALSE.
      mpp_file(:)%time_level = 0
      mpp_file(:)%time = NULLTIME
      mpp_file(:)%id = -1
!
```

```
      mpp_file(:)%ndim = -1
      mpp_file(:)%nvar = -1
!NULLUNIT "file" is always single-threaded, open and initialized (to pass checks in mpp_wr
      mpp_file(NULLUNIT)%threading = MPP_SINGLE
      mpp_file(NULLUNIT)%opened = .TRUE.
      mpp_file(NULLUNIT)%initialized = .TRUE.
!declare the stdunits to be open
      mpp_file(stdin ())%opened = .TRUE.
      mpp_file(stdout())%opened = .TRUE.
      mpp_file(stderr())%opened = .TRUE.
      mpp_file(stdout())%opened = .TRUE.
!set range of allowed fortran unit numbers: could be compiler-dependent (should not overla
!
!rv For OASIS 3 I consider the top max_reserved_units to be excluded from
!rv the list of files ito closed during mpp_io_exit.
!rv      call mpp_set_unit_range( 7, maxunits )
      if(present(maxunit)) then
        call mpp_set_unit_range( 7, maxunits-max_reserved_units )
      else
        call mpp_set_unit_range( 7, maxunits )
      endif
!rv

!rr   if( pe.EQ.mpp_root_pe() )then
          write( stdout(),'(/a)' )'MPP_IO module '//trim(version)
#ifdef use_netCDF
!rr not yet supported
!rr       text = NFMPI_INQ_LIBVERS()
!rr       write( stdout(),'(a)' )'Using netCDF library version '//trim(text)
#endif
!rr   endif

#ifdef CRAYPVP
!we require every file to be assigned threadwise: PVPs default to global, and are reset he
      call ASSIGN( 'assign -P thread p:%', error )
#endif

      call mpp_io_set_stack_size(131072) ! default initial value
      call mpp_sync()
      module_is_initialized = .TRUE.
      return
    end subroutine mpp_io_init

    subroutine mpp_io_exit()
      integer :: unit

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_IO_EXIT: must first call
!close all open fortran units
      do unit = unit_begin,unit_end
         if( mpp_file(unit)%opened )call FLUSH(unit)
      end do
      call mpp_sync()
      do unit = unit_begin,unit_end
         if( mpp_file(unit)%opened )close(unit)
      end do
#ifdef use_netCDF
!close all open netCDF units
      do unit = maxunits+1,2*maxunits
         if( mpp_file(unit)%opened )error = NFMPI_CLOSE(mpp_file(unit)%ncid)
      end do
#endif

      call mpp_max(mpp_io_stack_hwm)
```

```
!rr   if( pe.EQ.mpp_root_pe() )then
!          write( stdout,'(/a)' )'Exiting MPP_IO module...'
!          write( stdout,* )'MPP_IO_STACK high water mark=', mpp_io_stack_hwm
!rr   end if
      deallocate(mpp_file)
      module_is_initialized = .FALSE.
      return
    end subroutine mpp_io_exit

    subroutine mpp_io_set_stack_size(n)
!set the mpp_io_stack variable to be at least n LONG words long
      integer, intent(in) :: n
      character(len=8) :: text

      if( n.GT.mpp_io_stack_size .AND. allocated(mpp_io_stack) )deallocate(mpp_io_stack)
      if( .NOT.allocated(mpp_io_stack) )then
          allocate( mpp_io_stack(n) )
          mpp_io_stack_size = n
          write( text,'(i8)' )n
!rr       if( pe.EQ.mpp_root_pe() ) &
          call mpp_error( NOTE, 'MPP_IO_SET_STACK_SIZE: stack size set to '//text//'.' )
      end if

      return
    end subroutine mpp_io_set_stack_size

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                            !
!           OPENING AND CLOSING FILES: mpp_open() and mpp_close()            !
!                                                                            !
! mpp_open( unit, file, action, form, access, threading, &                   !
!           fileset, iospec, nohdrs, recl, pelist )                          !
!      integer, intent(out) :: unit                                          !
!      character(len=*), intent(in) :: file                                  !
!      integer, intent(in), optional :: action, form, access, threading,     !
!                                       fileset, recl                        !
!      character(len=*), intent(in), optional :: iospec                      !
!      logical, intent(in), optional :: nohdrs                               !
!      integer, optional, intent(in) :: pelist(:) !default ALL               !
!                                                                            !
!  unit is intent(OUT): always _returned_by_ mpp_open()                      !
!  file is the filename: REQUIRED                                            !
!    we append .nc to filename if it is a netCDF file                        !
!    we append .<pppp> to filename if fileset is private (pppp is PE number) !
!  iospec is a system hint for I/O organization                              !
!         e.g assign(1) on SGI/Cray systems.                                 !
!  if nohdrs is .TRUE. headers are not written on non-netCDF writes.         !
!  nohdrs has no effect when action=MPP_RDONLY|MPP_APPEND                    !
!                    or when form=MPP_NETCDF                                 !
! FLAGS:                                                                     !
!    action is one of MPP_RDONLY, MPP_APPEND or MPP_WRONLY                   !
!    form is one of MPP_ASCII:  formatted read/write                         !
!                   MPP_NATIVE: unformatted read/write, no conversion        !
!                   MPP_IEEE32: unformatted read/write, conversion to IEEE32 !
!                   MPP_NETCDF: unformatted read/write, conversion to netCDF !
!    access is one of MPP_SEQUENTIAL or MPP_DIRECT (ignored for netCDF)      !
!      RECL argument is REQUIRED for direct access IO                        !
!    threading is one of MPP_SINGLE or MPP_MULTI                             !
!      single-threaded IO in a multi-PE run is done by PE0                   !
!    fileset is one of MPP_MULTI and MPP_SINGLE                              !
!      fileset is only used for multi-threaded I/O                           !
!      if all I/O PEs in <pelist> use a single fileset,                      !
!              they write to the same file                                    !
!      if all I/O PEs in <pelist> use a multi  fileset,                       !
!              they each write an independent file                            !
!  recl is the record length in bytes                                         !
!  pelist is the list of I/O PEs (currently ALL)                              !
!                                                                             !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine mpp_open( unit, file, action, mpp_comm, form, access, threading, &
                                 fileset, iospec, nohdrs, recl, pelist )

      integer, intent(out) :: unit
      character(len=*), intent(in) :: file
      integer, intent(in), optional :: action, form, access, threading, &
           fileset, recl, mpp_comm
      character(len=*), intent(in), optional :: iospec
      logical, intent(in), optional :: nohdrs
      integer, intent(in), optional :: pelist(:) !default ALL

      character(len=16) :: act, acc, for, pos
      integer :: action_flag, form_flag, access_flag, threading_flag, fileset_flag, lengt
      logical :: exists
      character(len=64) :: filespec
      type(axistype) :: unlim    !used by netCDF with mpp_append

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_OPEN: must first call mp
!set flags
      action_flag = MPP_WRONLY        !default
      if( PRESENT(action) )action_flag = action
      form_flag = MPP_ASCII
      if( PRESENT(form) )form_flag = form
#ifndef use_netCDF
      if( form_flag.EQ.MPP_NETCDF ) &
           call mpp_error( FATAL, 'MPP_OPEN: To open a file with form=MPP_NETCDF, you must
#endif
      access_flag = MPP_SEQUENTIAL
      if( PRESENT(access) )access_flag = access
      threading_flag = MPP_SINGLE
      if( npes.GT.1 .AND. PRESENT(threading) )threading_flag = threading
      fileset_flag = MPP_MULTI
      if( PRESENT(fileset) )fileset_flag = fileset
      if( threading_flag.EQ.MPP_SINGLE )fileset_flag = MPP_SINGLE
!rr
      fileset_flag = MPP_PARALLEL
      threading_flag = MPP_PARALLEL

!get a unit number
      if( threading_flag.EQ.MPP_SINGLE )then
          if( pe.NE.mpp_root_pe() .AND. action_flag.NE.MPP_RDONLY )then
              unit = NULLUNIT           !PEs not participating in IO from this mpp_open()
              return
          end if
      end if
      if( form_flag.EQ.MPP_NETCDF )then
          do unit = maxunits+1,2*maxunits
             if( .NOT.mpp_file(unit)%opened )exit
          end do
          if( unit.GT.2*maxunits )call mpp_error( FATAL, 'MPP_OPEN: too many open netCDF f
      else
          do unit = unit_begin, unit_end
             inquire( unit,OPENED=mpp_file(unit)%opened )
             if( .NOT.mpp_file(unit)%opened )exit
          end do
          if( unit.GT.unit_end )call mpp_error( FATAL, 'MPP_OPEN: no available units.' )
      end if

!get a filename
      text = file
      length = len(file)

!RV   I dropped the automatic file name extension. PSMILE will always
!RV   provide netcdf file names with an extension .nc or names containing .nc.
!RV      if( form_flag.EQ.MPP_NETCDF.AND. file(length-2:length) /= '.nc' ) &
!RV         text = trim(file)//'.nc'

      if( fileset_flag.EQ.MPP_MULTI )write( text,'(a,i4.4)' )trim(text)//'.', pe
      mpp_file(unit)%name = text
      if( verbose ) write (stdout(), '(a,2i3,1x,a,5i5)') &
           'MPP_OPEN: PE, unit, filename, action, format, access, threading, fileset=', &
           pe, unit, trim(mpp_file(unit)%name), action_flag, form_flag, access_flag, threa

!action: read, write, overwrite, append: act and pos are ignored by netCDF
      if( action_flag.EQ.MPP_RDONLY )then
          act = 'READ'
          pos = 'REWIND'
!          if( form_flag.EQ.MPP_NETCDF )call mpp_error( FATAL, 'MPP_OPEN: only writes are
      else if( action_flag.EQ.MPP_WRONLY .OR. action_flag.EQ.MPP_OVERWR )then
          act = 'WRITE'
          pos = 'REWIND'
      else if( action_flag.EQ.MPP_APPEND )then
          act = 'WRITE'
          pos = 'APPEND'
      else
          call mpp_error( FATAL, 'MPP_OPEN: action must be one of MPP_WRONLY, MPP_APPEND c
      end if

!access: sequential or direct: ignored by netCDF
      if( form_flag.NE.MPP_NETCDF )then
          if( access_flag.EQ.MPP_SEQUENTIAL )then
              acc = 'SEQUENTIAL'
          else if( access_flag.EQ.MPP_DIRECT )then
              acc = 'DIRECT'
              if( form_flag.EQ.MPP_ASCII )call mpp_error( FATAL, 'MPP_OPEN: formatted dire
              if( .NOT.PRESENT(recl) ) &
                   call mpp_error( FATAL, 'MPP_OPEN: recl (record length in bytes) must be
              mpp_file(unit)%record = 1
              records_per_pe = 1 !each PE writes 1 record per mpp_write
          else
              call mpp_error( FATAL, 'MPP_OPEN: access must be one of MPP_SEQUENTIAL or MP
          end if
      end if

!threading: SINGLE or MULTI
      if( threading_flag.EQ.MPP_MULTI )then
!fileset: MULTI or SINGLE (only for multi-threaded I/O
          if( fileset_flag.EQ.MPP_SINGLE )then
              if( form_flag.EQ.MPP_NETCDF .AND. act.EQ.'WRITE' ) &
                   call mpp_error( FATAL, 'MPP_OPEN: netCDF currently does not support sin

#ifdef _CRAYT3E
              call ASSIGN( 'assign -I -F global.privpos f:'//trim(mpp_file(unit)%name), er
#endif
          else if( fileset_flag.NE.MPP_PARALLEL )then
              call mpp_error( FATAL, 'MPP_OPEN: fileset must be one of MPP_PARALLEL.' )
          end if
      else if( threading_flag.NE.MPP_PARALLEL )then
          call mpp_error( FATAL, 'MPP_OPEN: threading must be MPP_PARALLEL.' )
      end if

!apply I/O specs before opening the file
!note that -P refers to the scope of a fortran unit, which is always thread-private even i
#ifdef CRAYPVP
      call ASSIGN( 'assign -I -P thread  f:'//trim(mpp_file(unit)%name), error )
#endif
#ifdef _CRAYT3E
      call ASSIGN( 'assign -I -P private f:'//trim(mpp_file(unit)%name), error )
#endif
      if( PRESENT(iospec) )then
!iospec provides hints to the system on how to organize I/O
!on Cray systems this is done through 'assign', see assign(1) and assign(3F)
!on other systems this will be expanded as needed
!no error checks here on whether the supplied iospec is valid
#ifdef SGICRAY
          call ASSIGN( 'assign -I '//trim(iospec)//' f:'//trim(mpp_file(unit)%name), error
          if( form_flag.EQ.MPP_NETCDF )then
!for netCDF on SGI/Cray systems we pass it to the environment variable NETCDF_XFFIOSPEC
!ideally we should parse iospec, pass the argument of -F to NETCDF_FFIOSPEC, and the rest
!maybe I'll get around to it someday
!PXFSETENV is a POSIX-standard routine for setting environment variables from fortran
              call PXFSETENV( 'NETCDF_XFFIOSPEC', 0, trim(iospec), 0, 1, error )
          end if
#endif
      end if

!open the file as specified above for various formats
      if( form_flag.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          if( action_flag.EQ.MPP_WRONLY )then
              error = NFMPI_CREATE( mpp_comm, trim(mpp_file(unit)%name), NF_NOCLOBBER, MPI
              call netcdf_err(error)
              if( verbose ) write (stdout(), '(a,i3,i16)') 'MPP_OPEN: new netCDF file: pe,
          else if( action_flag.EQ.MPP_OVERWR )then
              error = NFMPI_CREATE( mpp_comm, trim(mpp_file(unit)%name), NF_CLOBBER, MPI_I
              call netcdf_err(error)
              action_flag = MPP_WRONLY !after setting clobber, there is no further distinc
              if( verbose ) write (stdout(), '(a,i3,i16)') 'MPP_OPEN: overwrite netCDF fil
          else if( action_flag.EQ.MPP_APPEND )then
              error = NFMPI_OPEN( mpp_comm, trim(mpp_file(unit)%name), NF_WRITE, MPI_INFO_
              call netcdf_err(error)
!get the current time level of the file: writes to this file will be at next time level
              error = NFMPI_INQ_UNLIMDIM( mpp_file(unit)%ncid, unlim%did )
              if( error.EQ.NF_NOERR )then
                  error = NFMPI_INQ_DIM( mpp_file(unit)%ncid, unlim%did, unlim%name, idim
                  mpp_file(unit)%time_level = idim
                  call netcdf_err(error)
                  error = NFMPI_INQ_VARID( mpp_file(unit)%ncid, unlim%name, mpp_file(unit)
              end if
              if( verbose ) write (stdout(), '(a,i3,i16,i4)') 'MPP_OPEN: append to existir
                   pe, mpp_file(unit)%ncid, mpp_file(unit)%id
          else if( action_flag.EQ.MPP_RDONLY )then
              error = NFMPI_OPEN( mpp_comm, trim(mpp_file(unit)%name), NF_NOWRITE, MPI_INF
              call netcdf_err(error)
              if( verbose ) write (stdout(), '(a,i3,i16,i4)') 'MPP_OPEN: opening existing
                   pe, mpp_file(unit)%ncid, mpp_file(unit)%id
              mpp_file(unit)%format=form_flag ! need this for mpp_read
              call mpp_read_meta(unit)
          end if
          mpp_file(unit)%opened = .TRUE.
#endif
      else
```

```
!format: ascii, native, or IEEE 32 bit
     if( form_flag.EQ.MPP_ASCII )then
       for = 'FORMATTED'
     else if( form_flag.EQ.MPP_IEEE32 )then
       for = 'UNFORMATTED'
!assign -N is currently unsupported on SGI
#ifdef _CRAY
       call ASSIGN( 'assign -I -N ieee_32 f:'//trim(mpp_file(unit)%name), error )
#endif
     else if( form_flag.EQ.MPP_NATIVE )then
       for = 'UNFORMATTED'
     else
       call mpp_error( FATAL, 'MPP_OPEN: form must be one of MPP_ASCII, MPP_NATIVE,
     end if
     inquire( file=trim(mpp_file(unit)%name), EXIST=exists )
     if( exists .AND. action_flag.EQ.MPP_WRONLY ) &
        call mpp_error( WARNING, 'MPP_OPEN: File '//trim(mpp_file(unit)%name)//' op
     if( action_flag.EQ.MPP_OVERWR )action_flag = MPP_WRONLY
!perform the OPEN here
     if( PRESENT(recl) )then
       if( verbose ) write (stdout(), '(2(1x,a,i3),5(1x,a),a,i8)') 'MPP_OPEN: PE=',
          'unit=', unit, trim(mpp_file(unit)%name), 'attributes=', trim(acc), tri
       open( unit, file=trim(mpp_file(unit)%name), access=acc, form=for, action=act
     else
       if( verbose ) write (stdout(), '(2(1x,a,i3),6(1x,a))')      'MPP_OPEN: PE=',
          'unit=', unit, trim(mpp_file(unit)%name), 'attributes=', trim(acc), tri
       open( unit, file=trim(mpp_file(unit)%name), access=acc, form=for, action=act
     end if
!check if OPEN worked
     inquire( unit,OPENED=mpp_file(unit)%opened )
     if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_OPEN: error in OPEN(
   end if
   mpp_file(unit)%action = action_flag
   mpp_file(unit)%format = form_flag
   mpp_file(unit)%access = access_flag
   mpp_file(unit)%threading = threading_flag
   mpp_file(unit)%fileset = fileset_flag
   if( PRESENT(nohdrs) )mpp_file(unit)%nohdrs = nohdrs

   if( action_flag.EQ.MPP_WRONLY )then
     if( form_flag.NE.MPP_NETCDF .AND. access_flag.EQ.MPP_DIRECT )call mpp_write_meta
!actual file name
     call mpp_write_meta( unit, 'filename', cval=mpp_file(unit)%name )
!MPP_IO package version
     call mpp_write_meta( unit, 'MPP_IO_VERSION', cval=trim(version) )
!filecount for multifileset
     if( threading_flag.EQ.MPP_MULTI .AND. fileset_flag.EQ.MPP_MULTI ) &
        call mpp_write_meta( unit, 'NumFilesInSet', ival=npes )
   end if

   return
  end subroutine mpp_open

  subroutine mpp_close( unit, action )
   integer, intent(in) :: unit
   integer, intent(in), optional :: action
   character(len=8) :: status
   logical :: collect

   if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_CLOSE: must first call m
   if( unit.EQ.NULLUNIT )return !nothing was actually opened on this unit

!action on close
```

```
      status = 'KEEP'
!collect is supposed to launch the post-processing collector tool for multi-fileset
      collect = .FALSE.
      if( PRESENT(action) )then
          if( action.EQ.MPP_DELETE )then
              status = 'DELETE'
          else if( action.EQ.MPP_COLLECT )then
              collect = .FALSE.         !should be TRUE but this is not yet ready
              call mpp_error( WARNING, 'MPP_CLOSE: the COLLECT operation is not yet implem
          else
              call mpp_error( FATAL, 'MPP_CLOSE: action must be one of MPP_DELETE or MPP_C
          end if
      end if
      if( mpp_file(unit)%fileset.NE.MPP_MULTI )collect = .FALSE.
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          error = NFMPI_CLOSE(mpp_file(unit)%ncid); call netcdf_err(error)
#endif
      else
          close(unit,status=status)
      end if
#ifdef SGICRAY
!this line deleted: since the FILENV is a shared file, this might cause a problem in
! multi-threaded I/O if one PE does assign -R before another one has opened it.
!      call ASSIGN( 'assign -R f:'//trim(mpp_file(unit)%name), error )
#endif
      mpp_file(unit)%name = ' '
      mpp_file(unit)%action    = -1
      mpp_file(unit)%format    = -1
      mpp_file(unit)%access    = -1
      mpp_file(unit)%threading = -1
      mpp_file(unit)%fileset   = -1
      mpp_file(unit)%record    = -1
      mpp_file(unit)%ncid      = -1
      mpp_file(unit)%opened = .FALSE.
      mpp_file(unit)%initialized = .FALSE.
      mpp_file(unit)%id = -1
      mpp_file(unit)%time_level = 0
      mpp_file(unit)%time = NULLTIME
      return
    end subroutine mpp_close

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                             !
!                             MPP_WRITE_META                                  !
!                                                                             !
!This series of routines is used to describe the contents of the file         !
!being written on <unit>. Each file can contain any number of fields,         !
!which can be functions of 0-3 spatial axes and 0-1 time axes. Axis           !
!descriptors are stored in the <axistype> structure and field                 !
!descriptors in the <fieldtype> structure.                                    !
!                                                                             !
!  type, public :: axistype                                                   !
!     sequence                                                                !
!     character(len=128) :: name                                              !
!     character(len=128) :: units                                             !
!     character(len=256) :: longname                                          !
!     integer :: sense           !+/-1, depth or height?                      !
!     type(domain1D) :: domain                                                !
!     real, pointer :: data(:) !axis values (not used if time axis)           !
!     integer :: id                                                           !
!  end type axistype                                                          !
!                                                                             !
!  type, public :: fieldtype                                                 !
!     sequence                                                               !
!     character(len=128) :: name                                             !
!     character(len=128) :: units                                            !
!     character(len=256) :: longname                                         !
!     real :: min, max, missing, fill, scale, add                            !
!     type(axistype), pointer :: axis(:)                                     !
!     integer :: id                                                          !
!  end type fieldtype                                                        !
!                                                                            !
!The metadata contained in the type is always written for each axis and      !
!field. Any other metadata one wishes to attach to an axis or field          !
!can subsequently be passed to mpp_write_meta using the ID, as shown below.  !
!                                                                            !
!mpp_write_meta can take several forms:                                      !
!                                                                            !
!  mpp_write_meta( unit, name, rval=rval, pack=pack )                        !
!  mpp_write_meta( unit, name, ival=ival )                                   !
!  mpp_write_meta( unit, name, cval=cval )                                   !
!      integer, intent(in) :: unit                                           !
!      character(len=*), intent(in) :: name                                  !
!      real, intent(in), optional :: rval(:)                                 !
!      integer, intent(in), optional :: ival(:)                              !
!      character(len=*), intent(in), optional :: cval                        !
!                                                                            !
!    This form defines global metadata associated with the file as a         !
!    whole. The attribute is named <name> and can take on a real, integer    !
!    or character value. <rval> and <ival> can be scalar or 1D arrays.       !
!                                                                            !
!  mpp_write_meta( unit, id, name, rval=rval, pack=pack )                    !
!  mpp_write_meta( unit, id, name, ival=ival )                               !
!  mpp_write_meta( unit, id, name, cval=cval )                               !
!      integer, intent(in) :: unit, id                                       !
!      character(len=*), intent(in) :: name                                  !
!      real, intent(in), optional :: rval(:)                                 !
!      integer, intent(in), optional :: ival(:)                              !
!      character(len=*), intent(in), optional :: cval                        !
!                                                                            !
!    This form defines metadata associated with a previously defined         !
!    axis or field, identified to mpp_write_meta by its unique ID <id>.      !
!    The attribute is named <name> and can take on a real, integer           !
!    or character value. <rval> and <ival> can be scalar or 1D arrays.       !
!    This need not be called for attributes already contained in             !
!    the type.                                                               !
!                                                                            !
!    PACK can take values 1,2,4,8. This only has meaning when writing        !
!    floating point numbers. The value of PACK defines the number of words   !
!    written into 8 bytes. For pack=4 and pack=8, an integer value is        !
!    written: rval is assumed to have been scaled to the appropriate dynamic !
!    range.                                                                  !
!    PACK currently only works for netCDF files, and is ignored otherwise.   !
!                                                                            !
!   subroutine mpp_write_meta_axis( unit, axis, name, units, longname, &     !
!        cartesian, sense, domain, data )                                    !
!     integer, intent(in) :: unit                                            !
!     type(axistype), intent(inout) :: axis                                  !
!     character(len=*), intent(in) :: name, units, longname                  !
!     character(len=*), intent(in), optional :: cartesian                    !
!     integer, intent(in), optional :: sense                                 !
!     type(domain1D), intent(in), optional :: domain                         !
!     real, intent(in), optional :: data(:)                                  !
!                                                                            !
!    This form defines a time or space axis. Metadata corresponding to the   !
!    type above are written to the file on <unit>. A unique ID for subsequent!
!    references to this axis is returned in axis%id. If the <domain>         !
!    element is present, this is recognized as a distributed data axis       !
!    and domain decomposition information is also written if required (the   !
!    domain decomposition info is required for multi-fileset multi-threaded  !
!    I/O). If the <data> element is allocated, it is considered to be a space!
!    axis, otherwise it is a time axis with an unlimited dimension. Only one !
!    time axis is allowed per file.                                          !
!                                                                            !
!   subroutine mpp_write_meta_field( unit, field, axes, name, units, longname!
!        min, max, missing, fill, scale, add, pack )                         !
!     integer, intent(in) :: unit                                            !
!     type(fieldtype), intent(out) :: field                                  !
!     type(axistype), intent(in) :: axes(:)                                  !
!     character(len=*), intent(in) :: name, units, longname                  !
!     real, intent(in), optional :: min, max, missing, fill, scale, add      !
!     integer, intent(in), optional :: pack                                  !
!                                                                            !
!    This form defines a field. Metadata corresponding to the type           !
!    above are written to the file on <unit>. A unique ID for subsequent     !
!    references to this field is returned in field%id. At least one axis     !
!    must be associated, 0D variables are not considered. mpp_write_meta     !
!    must previously have been called on all axes associated with this       !
!    field.                                                                  !
!                                                                            !
! The mpp_write_meta package also includes subroutines write_attribute and   !
! write_attribute_netcdf, that are private to this module.                   !
!                                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine mpp_write_meta_global( unit, name, rval, ival, cval, pack )
!writes a global metadata attribute to unit <unit>
!attribute <name> can be an real, integer or character
!one and only one of rval, ival, and cval should be present
!the first found will be used
!for a non-netCDF file, it is encoded into a string "GLOBAL <name> <val>"
      integer, intent(in) :: unit
      character(len=*), intent(in) :: name
      real,             intent(in), optional :: rval(:)
      integer,          intent(in), optional :: ival(:)
      character(len=*), intent(in), optional :: cval
      integer, intent(in), optional :: pack

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          call write_attribute_netcdf( unit, NF_GLOBAL, name, rval, ival, cval, pack )
#endif
      else
          call write_attribute( unit, 'GLOBAL '//trim(name), rval, ival, cval, pack )
      end if

      return
    end subroutine mpp_write_meta_global

!versions of above to support <rval> and <ival> as scalars (because of f90 strict rank mat
    subroutine mpp_write_meta_global_scalar_r( unit, name, rval, pack )
```

```
      integer, intent(in) :: unit
      character(len=*), intent(in) :: name
      real, intent(in) :: rval
      integer, intent(in), optional :: pack

      call mpp_write_meta_global( unit, name, rval=(/rval/), pack=pack )
      return
    end subroutine mpp_write_meta_global_scalar_r

    subroutine mpp_write_meta_global_scalar_i( unit, name, ival )
      integer, intent(in) :: unit
      character(len=*), intent(in) :: name
      integer, intent(in) :: ival

      call mpp_write_meta_global( unit, name, ival=(/ival/) )
      return
    end subroutine mpp_write_meta_global_scalar_i

    subroutine mpp_write_meta_var( unit, id, name, rval, ival, cval, pack )
!writes a metadata attribute for variable <id> to unit <unit>
!attribute <name> can be an real, integer or character
!one and only one of rval, ival, and cval should be present
!the first found will be used
!for a non-netCDF file, it is encoded into a string "<id> <name> <val>"
      integer, intent(in) :: unit, id
      character(len=*), intent(in) :: name
      real,             intent(in), optional :: rval(:)
      integer,          intent(in), optional :: ival(:)
      character(len=*), intent(in), optional :: cval
      integer, intent(in), optional :: pack

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
          call write_attribute_netcdf( unit, id, name, rval, ival, cval, pack )
      else
          write( text, '(a,i4,a)' )'VARIABLE ', id, ' '//name
          call write_attribute( unit, trim(text), rval, ival, cval, pack )
      end if

      return
    end subroutine mpp_write_meta_var

!versions of above to support <rval> and <ival> as scalar (because of f90 strict rank matc
    subroutine mpp_write_meta_scalar_r( unit, id, name, rval, pack )
      integer, intent(in) :: unit, id
      character(len=*), intent(in) :: name
      real, intent(in) :: rval
      integer, intent(in), optional :: pack

      call mpp_write_meta( unit, id, name, rval=(/rval/), pack=pack )
      return
    end subroutine mpp_write_meta_scalar_r

    subroutine mpp_write_meta_scalar_i( unit, id, name, ival )
      integer, intent(in) :: unit, id
      character(len=*), intent(in) :: name
      integer, intent(in) :: ival

      call mpp_write_meta( unit, id, name, ival=(/ival/) )
      return
    end subroutine mpp_write_meta_scalar_i

    subroutine mpp_write_meta_axis( unit, axis, name, units, longname, cartesian, sense, c
!load the values in an axistype (still need to call mpp_write)
!write metadata attributes for axis
!it is declared intent(inout) so you can nullify pointers in the incoming object if needec
!the f90 standard doesn't guarantee that intent(out) on a type guarantees that its pointer
      integer, intent(in) :: unit
      type(axistype), intent(inout) :: axis
      character(len=*), intent(in) :: name, units, longname
      character(len=*), intent(in), optional :: cartesian
      integer, intent(in), optional :: sense
      type(domain1D), intent(in), optional :: domain
      real, intent(in), optional :: data(:)
      character(len=*), intent(in), optional :: cdata(:) !RV,bundles
      integer :: is, ie, isg, ieg

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

!pre-existing pointers need to be nullified
      if( ASSOCIATED(axis%data) )NULLIFY(axis%data)
      if( ASSOCIATED(axis%cdata) )NULLIFY(axis%cdata) !RV,bundles
!load axistype
      axis%name     = name
      axis%units    = units
      axis%longname = longname
      if( PRESENT(cartesian) )axis%cartesian = cartesian
      if( PRESENT(sense)     )axis%sense     = sense
      if( PRESENT(domain)    )then
          axis%domain = domain
          call mpp_get_global_domain( domain, isg, ieg )
          call mpp_get_compute_domain( domain, is, ie )
      else
          axis%domain = NULL_DOMAIN1D
          if( PRESENT(data) )then
             isg=1; ieg=size(data); is=isg; ie=ieg
          endif
          if( PRESENT(cdata) )then !!RV,bundles
             isg=1; ieg=size(cdata); is=isg; ie=ieg !!RV,bundles
          endif !!RV,bundles
      end if
      if( PRESENT(data) )then
          if( PRESENT(domain) )then
              if( size(data).NE.ieg-isg+1 ) &
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: size(data).NE.domain%globa
              allocate( axis%data(isg:ieg) )
          else
              allocate( axis%data(size(data)) )
          end if
          axis%data = data
      end if
      if( PRESENT(cdata) )then !RV,bundles
          if( PRESENT(domain) )then !RV,bundles
```

```
              if( size(cdata).NE.ieg-isg+1 ) & !RV,bundles
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: size(cdata).NE.domain%glob
              allocate( axis%cdata(isg:ieg) ) !RV,bundles
              allocate( axis%data(isg:ieg) ) !RV,bundles
          else !RV,bundles
              allocate( axis%cdata(size(cdata)) ) !RV,bundles
              allocate( axis%data(size(cdata)) ) !RV,bundles
          end if !RV,bundles
          axis%cdata = cdata !RV,bundles
      end if !RV,bundles


!write metadata
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
!write axis def
!space axes are always floats, time axis is always double
          if( ASSOCIATED(axis%data).or. ASSOCIATED(axis%cdata) )then !space axisRV,bundles
              if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )
                  idim = ie-is+1
                  error = NFMPI_DEF_DIM( mpp_file(unit)%ncid, axis%name, idim, axis%did )
              else
                  if(  ASSOCIATED(axis%data).and.(.not.present(cdata)))then !!RV,bundles
                    idim = size(axis%data)
                    error = NFMPI_DEF_DIM( mpp_file(unit)%ncid, axis%name, idim, axis%did
                  else !!RV,bundles
                    idim = len(axis%cdata)
                    error = NFMPI_DEF_DIM( mpp_file(unit)%ncid, 'MAX_STRLEN', idim, axis%c
                    call netcdf_err(error)
                    idim = size(axis%cdata)
                    error = NFMPI_DEF_DIM( mpp_file(unit)%ncid, axis%name, idim, axis%did
                  endif !!RV,bundles
              end if
              call netcdf_err(error)
              if(present(cdata)) then !!RV, bundles
              error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_CHAR, 2,(/axis%cle
              call netcdf_err(error) !!Bundles
              else !!Bundles
              error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_FLOAT, 1, axis%dic
              endif !!Bundles

          else                            !time axis
              if( mpp_file(unit)%id.NE.-1 ) &
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: There is already a time ax
              idim = NF_UNLIMITED
              error = NFMPI_DEF_DIM( mpp_file(unit)%ncid, axis%name, idim, axis%did ); cal
              error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_DOUBLE, 1, axis%di
              mpp_file(unit)%id = axis%id !file ID is the same as time axis varID
          end if
#endif
      else
          varnum = varnum + 1
          axis%id = varnum
          axis%did = varnum
!write axis def
          write( text, '(a,i4,a)' )'AXIS ', axis%id, ' name'
          call write_attribute( unit, trim(text), cval=axis%name )
          write( text, '(a,i4,a)' )'AXIS ', axis%id, ' size'
          if( ASSOCIATED(axis%data) )then !space axis
              if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )
                  call write_attribute( unit, trim(text), ival=(/ie-is+1/) )
              else
                  if(ASSOCIATED(axis%data).and.(.not.present(cdata))) then !!RV,bundles
                  call write_attribute( unit, trim(text), ival=(/size(axis%data)/) )
                  else !!RV,bundles
                  call write_attribute( unit, trim(text), ival=(/size(axis%cdata)/) ) !!RV
                  endif !!RV, bundles
              end if
          else                            !time axis
              if( mpp_file(unit)%id.NE.-1 ) &
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: There is already a time ax
              call write_attribute( unit, trim(text), ival=(/0/) ) !a size of 0 indicates
              mpp_file(unit)%id = axis%id
          end if
      end if

!write axis attributes
      call mpp_write_meta( unit, axis%id, 'long_name', cval=axis%longname )
      call mpp_write_meta( unit, axis%id, 'units',     cval=axis%units    )
      if( PRESENT(cartesian) )call mpp_write_meta( unit, axis%id, 'cartesian_axis', cval=a
      if( PRESENT(sense) )then
          if( sense.EQ.-1 )then
              call mpp_write_meta( unit, axis%id, 'positive', cval='down' )
          else if( sense.EQ.1 )then
              call mpp_write_meta( unit, axis%id, 'positive', cval='up' )
          end if
!silently ignore values of sense other than +/-1.
      end if
      if( mpp_file(unit)%threading.EQ.MPP_MULTI .AND. mpp_file(unit)%fileset.EQ.MPP_MULTI
          call mpp_write_meta( unit, axis%id, 'domain_decomposition', ival=(/isg,ieg,is,ie
      end if
      if( verbose ) write (stdout(), '(a,2i3,1x,a,2i3)') &
           'MPP_WRITE_META: Wrote axis metadata, pe, unit, axis%name, axis%id, axis%did=',
           pe, unit, trim(axis%name), axis%id, axis%did

      return
    end subroutine mpp_write_meta_axis

    subroutine mpp_write_meta_field( unit, field, axes, name, units, longname, min, max, m
!define field: must have already called mpp_write_meta(axis) for each axis
      integer, intent(in) :: unit
      type(fieldtype), intent(out) :: field
      type(axistype), intent(in) :: axes(:)
      character(len=*), intent(in) :: name, units, longname
      real, intent(in), optional :: min, max, missing, fill, scale, add
      integer, intent(in), optional :: pack
!this array is required because of f77 binding on netCDF interface
      integer, allocatable :: axis_id(:)
      real :: a, b
      integer :: i

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

!pre-existing pointers need to be nullified
      if( ASSOCIATED(field%axes) )NULLIFY(field%axes)
!fill in field metadata
      field%name = name
      field%units = units
      field%longname = longname
      allocate( field%axes(size(axes)) )
      field%axes = axes
      field%time_axis_index = -1 !this value will never match any axis index
!size is buffer area for the corresponding axis info: it is required to buffer this info i
!because axis might be reused in different files
      allocate( field%size(size(axes)) )
      do i = 1,size(axes)
         if( ASSOCIATED(axes(i)%data) )then !space axis
             field%size(i) = size(axes(i)%data)
         else              !time
             field%size(i) = 1
             field%time_axis_index = i
         end if
      end do
!attributes
      if( PRESENT(min) )field%min = min
      if( PRESENT(max) )field%max = max
      if( PRESENT(missing) )field%missing = missing
      if( PRESENT(fill) )field%fill = fill
      if( PRESENT(scale) )field%scale = scale
      if( PRESENT(add) )field%add = add

!pack is currently used only for netCDF
      field%pack = 2        !default write 32-bit floats
      if( PRESENT(pack) )field%pack = pack
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          allocate( axis_id(size(field%axes)) )
          do i = 1,size(field%axes)
             axis_id(i) = field%axes(i)%did
          end do
!write field def
          select case (field%pack)
              case(1)
                  error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_DOUBLE, size(
              case(2)
                  error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_FLOAT,  size(
              case(4)
                  if( .NOT.PRESENT(scale) .OR. .NOT.PRESENT(add) ) &
                       call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: scale and add must be
                  error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_SHORT,  size(
              case(8)
                  if( .NOT.PRESENT(scale) .OR. .NOT.PRESENT(add) ) &
                       call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: scale and add must be
                  error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_BYTE,   size(
              case default
                  call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: only legal packing values
          end select
          call netcdf_err(error)
#endif
      else
          varnum = varnum + 1
          field%id = varnum
          if( PRESENT(pack) )call mpp_error( WARNING, 'MPP_WRITE_META: Packing is currentl
!write field def
          write( text, '(a,i4,a)' )'FIELD ', field%id, ' name'
          call write_attribute( unit, trim(text), cval=field%name )
          write( text, '(a,i4,a)' )'FIELD ', field%id, ' axes'
          call write_attribute( unit, trim(text), ival=field%axes(:)%did )
      end if
!write field attributes: these names follow netCDF conventions
      call mpp_write_meta( unit, field%id, 'long_name', cval=field%longname )
      call mpp_write_meta( unit, field%id, 'units',     cval=field%units    )
```

```
!all real attributes must be written as packed
      if( PRESENT(min) .AND. PRESENT(max) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_range', rval=(/min,max/), pack=p
          else
              a = nint((min-add)/scale)
              b = nint((max-add)/scale)
              call mpp_write_meta( unit, field%id, 'valid_range', rval=(/a,  b  /), pack=p
          end if
      else if( PRESENT(min) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_min', rval=field%min, pack=pack
          else
              a = nint((min-add)/scale)
              call mpp_write_meta( unit, field%id, 'valid_min', rval=a, pack=pack )
          end if
      else if( PRESENT(max) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_max', rval=field%max, pack=pack
          else
              a = nint((max-add)/scale)
              call mpp_write_meta( unit, field%id, 'valid_max', rval=a, pack=pack )
          end if
      end if
      if( PRESENT(missing) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'missing_value', rval=field%missing, pa
          else
              a = nint((missing-add)/scale)
              call mpp_write_meta( unit, field%id, 'missing_value', rval=a, pack=pack )
          end if
      end if
      if( PRESENT(fill) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, '_FillValue', rval=field%missing, pack=
          else
              a = nint((fill-add)/scale)
              call mpp_write_meta( unit, field%id, '_FillValue', rval=a, pack=pack )
          end if
      end if
      if( field%pack.NE.1 .AND. field%pack.NE.2 )then
          call mpp_write_meta( unit, field%id, 'packing', ival=field%pack )
          if( PRESENT(scale) )call mpp_write_meta( unit, field%id, 'scale_factor',  rval=f
          if( PRESENT(add)   )call mpp_write_meta( unit, field%id, 'add_offset',    rval=f
      end if
      if( verbose ) write (stdout(), '(a,2i3,1x,a,i3)') 'MPP_WRITE_META: Wrote field metad
           pe, unit, trim(field%name), field%id

      return
    end subroutine mpp_write_meta_field

    subroutine write_attribute( unit, name, rval, ival, cval, pack )
!called to write metadata for non-netCDF I/O
      integer, intent(in) :: unit
      character(len=*), intent(in) :: name
      real, intent(in), optional :: rval(:)
      integer, intent(in), optional :: ival(:)
      character(len=*), intent(in), optional :: cval
!pack is currently ignored in this routine: only used by netCDF I/O
      integer, intent(in), optional :: pack

      if( mpp_file(unit)%nohdrs )return
!encode text string
```

```
      if( PRESENT(rval) )then
          write( text,* )trim(name)//'=', rval
      else if( PRESENT(ival) )then
          write( text,* )trim(name)//'=', ival
      else if( PRESENT(cval) )then
          text = ' '//trim(name)//'='//trim(cval)
      else
          call mpp_error( FATAL, 'WRITE_ATTRIBUTE: one of rval, ival, cval must be present
      end if
      if( mpp_file(unit)%format.EQ.MPP_ASCII )then
!implies sequential access
          write( unit,fmt='(a)' )trim(text)//char(10)
      else                      !MPP_IEEE32 or MPP_NATIVE
          if( mpp_file(unit)%access.EQ.MPP_SEQUENTIAL )then
              write(unit)trim(text)//char(10)
          else                  !MPP_DIRECT
              write( unit,rec=mpp_file(unit)%record )trim(text)//char(10)
              if( verbose ) write (stdout(), '(a,i3,a,i3)') 'WRITE_ATTRIBUTE: PE=', pe, '
              mpp_file(unit)%record = mpp_file(unit)%record + 1
          end if
      end if
      return
    end subroutine write_attribute

    subroutine write_attribute_netcdf( unit, id, name, rval, ival, cval, pack )
!called to write metadata for netCDF I/O
      integer, intent(in) :: unit
      integer, intent(in) :: id
      character(len=*), intent(in) :: name
      real,             intent(in), optional :: rval(:)
      integer,          intent(in), optional :: ival(:)
      character(len=*), intent(in), optional :: cval
      integer, intent(in), optional :: pack
      integer :: lenc
      integer, allocatable :: rval_i(:)
#ifdef use_netCDF
      integer :: ii, il_bytesize, il_iosize
      integer :: il_int_iosize, il_rbyt
!
      if( PRESENT(rval) )then
          il_bytesize = BIT_SIZE(ii)/8
          INQUIRE (iolength=il_iosize) ii
          il_int_iosize = il_iosize
          INQUIRE (iolength=il_iosize) rval(1)
          il_rbyt = il_iosize/il_int_iosize*il_bytesize
!pack is only meaningful for FP numbers
          if( PRESENT(pack) )then
              if( pack.EQ.1 )then
                  idim = size(rval)
                  if( il_rbyt .EQ. DOUBLE_KIND )then
                      error = NFMPI_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_DOUB
                  else if( il_rbyt .EQ. FLOAT_KIND )then
                      call mpp_error( WARNING, &
                           'WRITE_ATTRIBUTE_NETCDF: attempting to write internal 32-bit re
                      error = NFMPI_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_DOUB
                  end if
                  call netcdf_err(error)
              else if( pack.EQ.2 )then
                  idim =  size(rval)
                  if( il_rbyt.EQ.DOUBLE_KIND )then
                      error = NFMPI_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_FLOA
                  else if( il_rbyt.EQ.FLOAT_KIND )then
                      error = NFMPI_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_FLOA
                  end if
                  call netcdf_err(error)
              else if( pack.EQ.4 )then
                  allocate( rval_i(size(rval)) )
                  rval_i = rval
                  idim = size(rval_i)
                  if( il_rbyt.EQ.DOUBLE_KIND )then
                      error = NFMPI_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_SHOR
                  else if( il_rbyt.EQ.FLOAT_KIND )then
                      error = NFMPI_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_SHOR
                  end if
                  call netcdf_err(error)
                  deallocate(rval_i)
              else if( pack.EQ.8 )then
                  allocate( rval_i(size(rval)) )
                  rval_i = rval
                  idim = size(rval_i)
                  if( il_rbyt.EQ.DOUBLE_KIND )then
                      error = NFMPI_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_BYTE
                  else if( il_rbyt.EQ.FLOAT_KIND )then
                      error = NFMPI_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_BYTE
                  end if
                  call netcdf_err(error)
                  deallocate(rval_i)
              else
                  call mpp_error( FATAL, 'WRITE_ATTRIBUTE_NETCDF: only legal packing value
              end if
          else
!default is to write FLOATs (32-bit)
              idim = size(rval)
              if( il_rbyt.EQ.DOUBLE_KIND )then
                  error = NFMPI_PUT_ATT_DOUBLE( mpp_file(unit)%ncid, id, name, NF_FLOAT, i
              else if( il_rbyt.EQ.FLOAT_KIND )then
                  error = NFMPI_PUT_ATT_REAL  ( mpp_file(unit)%ncid, id, name, NF_FLOAT, i
              end if
              call netcdf_err(error)
          end if
      else if( PRESENT(ival) )then
          idim = size(ival)
          error = NFMPI_PUT_ATT_INT ( mpp_file(unit)%ncid, id, name, NF_INT, idim, ival );
      else if( present(cval) )then
          idim = len_trim(cval)
          error = NFMPI_PUT_ATT_TEXT( mpp_file(unit)%ncid, id, name, idim, cval ); call ne
      else
          call mpp_error( FATAL, 'WRITE_ATTRIBUTE_NETCDF: one of rval, ival, cval must be
      end if
#endif /* use_netCDF */
      return
    end subroutine write_attribute_netcdf

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!                             MPP_WRITE                                !
!                                                                      !
! mpp_write is used to write data to the file on <unit> using the      !
! file parameters supplied by mpp_open(). Axis and field definitions   !
! must have previously been written to the file using mpp_write_meta.  !
!                                                                      !
! mpp_write can take 2 forms, one for distributed data and one for     !
! non-distributed data. Distributed data refer to arrays whose two     !
! fastest-varying indices are domain-decomposed. Distributed data      !
! must be 2D or 3D (in space). Non-distributed data can be 0-3D.       !
!                                                                      !
! In all calls to mpp_write, tstamp is an optional argument. It is to  !
! be omitted if the field was defined not to be a function of time.    !
! Results are unpredictable if the argument is supplied for a time-    !
! independent field, or omitted for a time-dependent field. Repeated   !
! writes of a time-independent field are also not recommended. One     !
! time level of one field is written per call.                         !
!                                                                      !
!                                                                      !
! For non-distributed data, use                                        !
!                                                                      !
!  mpp_write( unit, field, data, tstamp )                              !
!     integer, intent(in) :: unit                                      !
!     type(fieldtype), intent(in) :: field                             !
!     real, optional :: tstamp                                         !
!     data is real and can be scalar or of rank 1-3.                   !
!                                                                      !
! For distributed data, use                                            !
!                                                                      !
!  mpp_write( unit, field, domain, data, tstamp )                      !
!     integer, intent(in) :: unit                                      !
!     type(fieldtype), intent(in) :: field                             !
!     type(domain2D), intent(in) :: domain                             !
!     real, optional :: tstamp                                         !
!     data is real and can be of rank 2 or 3.                          !
!                                                                      !
!  mpp_write( unit, axis )                                             !
!     integer, intent(in) :: unit                                      !
!     type(axistype), intent(in) :: axis                               !
!                                                                      !
! This call writes the actual co-ordinate values along each space      !
! axis. It must be called once for each space axis after all other     !
! metadata has been written.                                           !
!                                                                      !
! The mpp_write package also includes the routine write_record which   !
! performs the actual write. This routine is private to this module.   !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#define MPP_WRITE_2DDECOMP_1D_ mpp_write_2ddecomp_r1d
#define MPP_WRITE_2DDECOMP_2D_ mpp_write_2ddecomp_r2d
#define MPP_WRITE_2DDECOMP_3D_ mpp_write_2ddecomp_r3d
#define MPP_WRITE_2DDECOMP_4D_ mpp_write_2ddecomp_r4d
#define MPP_TYPE_ real
#include <mpp_write_2Ddecomp.h>

#define MPP_WRITE_ mpp_write_r0D
#define MPP_TYPE_ real
#define MPP_RANK_ !
#define MPP_WRITE_RECORD_ call write_record( unit, field, 1, (/data/), tstamp )
#include <mpp_write.h>

#define MPP_WRITE_ mpp_write_r1D
#define MPP_TYPE_ real
#define MPP_WRITE_RECORD_ call write_record( unit, field, size(data), data, tstamp )
#define MPP_RANK_ (:)
#include <mpp_write.h>

#define MPP_WRITE_ mpp_write_r2D
#define MPP_TYPE_ real
#define MPP_WRITE_RECORD_ call write_record( unit, field, size(data), data, tstamp )
#define MPP_RANK_ (:,:)
#include <mpp_write.h>

#define MPP_WRITE_ mpp_write_r3D
```

```
#define MPP_TYPE_ real
#define MPP_WRITE_RECORD_ call write_record( unit, field, size(data), data, tstamp )
#define MPP_RANK_ (:,:,:)
#include <mpp_write.h>

#define MPP_WRITE_ mpp_write_r4D
#define MPP_TYPE_ real
#define MPP_WRITE_RECORD_ call write_record( unit, field, size(data), data, tstamp )
#define MPP_RANK_ (:,:,:,:)
#include <mpp_write.h>

    subroutine mpp_write_axis( unit, axis )
      integer, intent(in) :: unit
      type(axistype), intent(in) :: axis
      type(fieldtype) :: field
      integer :: is, ie


      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_WRITE: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE: invalid unit numb
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset  .EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
!we convert axis to type(fieldtype) in order to call write_record
      field = default_field
      allocate( field%axes(1) )
      field%axes(1) = axis
      allocate( field%size(1) )
      field%id = axis%id
      if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )then
          call mpp_get_compute_domain( axis%domain, is, ie )
          field%size(1) = ie-is+1
!!RV,bundles
          if(associated( axis%cdata)) then
          call write_record_c( unit, field, field%size(1), axis%cdata(is:) )
          else
          call write_record( unit, field, field%size(1), axis%data(is:) )
          endif
!!RV,bundles
      else
!!RV,bundles
          if(associated( axis%cdata)) then
          field%size(1) = size(axis%cdata)
          call write_record_c(unit,field, field%size(1), axis%cdata )
          else
          field%size(1) = size(axis%data)
          call write_record( unit, field, field%size(1), axis%data )
          endif
!!RV,bundles
      end if
      return
    end subroutine mpp_write_axis

    subroutine write_record_c( unit, field, nwords, cdata, time_in, domain ) !!RV,bundles
!routine that is finally called by all mpp_write routines to perform the write
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
```

```
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
!RV,bundles
      character(len=64), intent(in) :: cdata(nwords)
      real(DOUBLE_KIND), intent(in), optional :: time_in
      type(domain2D), intent(in), optional :: domain
!RV      integer, dimension(size(field%axes)) :: start, axsiz

      integer,allocatable,dimension(:) :: start, axsiz
!RV
      real :: time
      integer :: time_level
      logical :: newtime
      integer :: subdomain(4)
      integer :: packed_data(nwords)
      integer :: i, is, ie, js, je, isg, ieg, jsg, jeg, isizc, jsizc, isizg, jsizg
#ifdef use_netCDF
      integer :: ii, il_bytesize, il_iosize
      integer :: il_int_iosize, il_rbyt
#endif

#ifdef use_CRI_pointers
      real(FLOAT_KIND) :: data_r4(nwords)
      pointer( ptr1, data_r4)
      pointer( ptr2, packed_data)

      if (mpp_io_stack_size < 2*nwords) call mpp_io_set_stack_size(2*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
#endif

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_WRITE: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE: invalid unit numb
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset  .EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return

!RV
      allocate(start(size(field%axes)))
      allocate(axsiz(size(field%axes)))
!RV
      if( .NOT.mpp_file(unit)%initialized )then
!this is the first call to mpp_write
!we now declare the file to be initialized
!if this is netCDF we switch file from DEFINE mode to DATA mode
          if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
!NOFILL is probably required for parallel: any circumstances in which not advisable?
!rr not yet supported
!rr           error = NFMPI_SET_FILL( mpp_file(unit)%ncid, NF_NOFILL, i ); call netcdf_err
              if( mpp_file(unit)%action.EQ.MPP_WRONLY )error = NFMPI_ENDDEF(mpp_file(unit)
#endif
          else
              call mpp_write_meta( unit, 'END', cval='metadata' )
          end if
          mpp_file(unit)%initialized = .TRUE.
          if( verbose ) write (stdout(), '(a,i3,a)') 'MPP_WRITE: PE=', pe, ' initialized f
      end if

!initialize time: by default assume NULLTIME
      time = NULLTIME
      time_level = -1
      newtime = .FALSE.
      if( PRESENT(time_in) )time = time_in
!increment time level if new time
      if( time.GT.mpp_file(unit)%time+EPSILON(time) )then !new time
          mpp_file(unit)%time_level = mpp_file(unit)%time_level + 1
          mpp_file(unit)%time = time
          newtime = .TRUE.
      end if
      if( verbose ) write (stdout(), '(a,2i3,2i5,es13.5)') 'MPP_WRITE: PE, unit, %id, %tim
           pe, unit, mpp_file(unit)%id, mpp_file(unit)%time_level, mpp_file(unit)%time

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be written:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NFMPI_PUT_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1! NOTE: pa
!       since that attempts to gather all data on PE 0.
          start = 1
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( i.EQ.field%time_axis_index )start(i) = mpp_file(unit)%time_level
             start(i) = max(start(i),1)
          end do
          if( PRESENT(domain) )then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je,  xsize=isizc, ysize=
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg, xsize=isizg, ysize=
              axsiz(1) = isizc
              axsiz(2) = jsizc
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( isizc.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( jsizc.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
          end if
          if( debug ) &
           write (stdout(), '(a,2i3,12i4)') 'd WRITE_RECORD: PE, unit, start, axsiz=', pe,
#ifdef use_netCDF
!write time information if new time
          if( newtime )then
              il_bytesize = BIT_SIZE(ii)/8
              INQUIRE (iolength=il_iosize) ii
              il_int_iosize = il_iosize
              INQUIRE (iolength=il_iosize) time
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt.EQ.DOUBLE_KIND )then
                  idim = mpp_file(unit)%time_level
                  error = NFMPI_PUT_VAR1_DOUBLE( mpp_file(unit)%ncid, mpp_file(unit)%id, i
              else if( il_rbyt.EQ.FLOAT_KIND )then
                  idim = mpp_file(unit)%time_level
                  error = NFMPI_PUT_VAR1_REAL  ( mpp_file(unit)%ncid, mpp_file(unit)%id, i
              end if
          end if
          if( field%pack.LE.2 )then
                 write(6,*) ' Iam here 6!'
                 call mpp_flushstd(6)
               error = NFMPI_PUT_VARA_TEXT_ALL( mpp_file(unit)%ncid, field%id, (/1,start/)
                 write(6,*) ' Iam here 7!'
                 call mpp_flushstd(6)
          else  !!RV, bundles
                 write(6,*) ' Iam here 8!'
                 call mpp_flushstd(6)
            call mpp_error( FATAL, 'MPP_WRITE_RECORD_C: pack on text !' )
          end if !!RV, bundles
                 write(6,*) ' Iam here 9!',error
                 call mpp_flushstd(6)
          call netcdf_err(error)
#endif
      else                      !non-netCDF
!subdomain contains (/is,ie,js,je/)
          if( PRESENT(domain) )then
              subdomain(:) = (/ is, ie, js, je /)
          else
              subdomain(:) = -1    ! -1 means use global value from axis metadata
          end if
          if( mpp_file(unit)%format.EQ.MPP_ASCII )then
!implies sequential access
              write( unit,* )field%id, subdomain, time_level, time, cdata
          else                      !MPP_IEEE32 or MPP_NATIVE
              if( mpp_file(unit)%access.EQ.MPP_SEQUENTIAL )then
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      write(unit)field%id, subdomain, time_level, time, cdata
                  else
                      write(unit)field%id, subdomain, time_level, time, cdata
                  end if
#else
                  write(unit)field%id, subdomain, time_level, time, cdata
#endif
              else                  !MPP_DIRECT
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  else
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  end if
#else
                  write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_level,
#endif
                  if( debug ) write (stdout(), '(a,i3,a,i3)') 'MPP_WRITE: PE=', pe, ' wrot
              end if
          end if
      end if

!recompute current record for direct access I/O
      if( mpp_file(unit)%access.EQ.MPP_DIRECT )then
          if( mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
!assumes all PEs participate in I/O: modify later
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe*npes
          else
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe
          end if
      end if
!RV
      deallocate(start)
      deallocate(axsiz)
!RV
      return
    end subroutine write_record_c

    subroutine write_record_b( unit, field, nwords, data, time_in, domain,block_id )
!routine that is finally called by all mpp_write routines to perform the write
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.
!RV,SGI:
!   The routine write_record_b is a special clone of write_record.
!   The assumption is here that the user has declared a data structure
!   like a(:,:,:,1:no_of_blocks). For whatever reason that arrray is written
!   is not written in a big chunk but on a per block basis for a certain time
!   stamp: At t_i write a(:,:,:,block_id).After all block are written the data structure c
!   should look like as if array a was written in one big chunk.
!   Moreover, I assume that the time axis is always the last one and that the block axis
!   comes befor the time axis, means the block axis is the last pseudo spatial axis.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
      real, intent(in) :: data(nwords)
      real(DOUBLE_KIND), intent(in), optional :: time_in
      integer,intent(in),optional :: block_id
      type(domain2D), intent(in), optional :: domain
!RV      integer, dimension(size(field%axes)) :: start, axsiz

      integer,allocatable,dimension(:) :: start, axsiz
!RV
      real :: time
      integer :: time_level
      logical :: newtime
      integer :: subdomain(4)
      integer :: packed_data(nwords)
      integer :: i, is, ie, js, je, isg, ieg, jsg, jeg, isizc, jsizc, isizg, jsizg
#ifdef use_netCDF
      integer :: ii, il_bytesize, il_iosize
      integer :: il_int_iosize, il_rbyt
#endif

#ifdef use_CRI_pointers
      real(FLOAT_KIND) :: data_r4(nwords)
      pointer( ptr1, data_r4)
      pointer( ptr2, packed_data)

      if (mpp_io_stack_size < 2*nwords) call mpp_io_set_stack_size(2*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
#endif

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_WRITE: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE: invalid unit numb
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset  .EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return

!RV
      allocate(start(size(field%axes)))
      allocate(axsiz(size(field%axes)))
!RV
      if( .NOT.mpp_file(unit)%initialized )then
!this is the first call to mpp_write
!we now declare the file to be initialized
!if this is netCDF we switch file from DEFINE mode to DATA mode
          if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
!NOFILL is probably required for parallel: any circumstances in which not advisable?
!rr not yet supported
!rr           error = NFMPI_SET_FILL( mpp_file(unit)%ncid, NF_NOFILL, i ); call netcdf_err
              if( mpp_file(unit)%action.EQ.MPP_WRONLY )error = NFMPI_ENDDEF(mpp_file(unit)
#endif
          else
              call mpp_write_meta( unit, 'END', cval='metadata' )
          end if
          mpp_file(unit)%initialized = .TRUE.
          if( verbose ) write (stdout(), '(a,i3,a)') 'MPP_WRITE: PE=', pe, ' initialized f
      end if

!initialize time: by default assume NULLTIME
      time = NULLTIME
      time_level = -1
      newtime = .FALSE.
      if( PRESENT(time_in) )time = time_in
!increment time level if new time
      if( time.GT.mpp_file(unit)%time+EPSILON(time) )then !new time
          mpp_file(unit)%time_level = mpp_file(unit)%time_level + 1
          mpp_file(unit)%time = time
          newtime = .TRUE.
      end if
      if( verbose ) write (stdout(), '(a,2i3,2i5,es13.5)') 'MPP_WRITE: PE, unit, %id, %tim
           pe, unit, mpp_file(unit)%id, mpp_file(unit)%time_level, mpp_file(unit)%time

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be written:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NFMPI_PUT_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1
! NOTE: passing of subdomains will fail for multi-PE single-threaded I/O,
!       since that attempts to gather all data on PE 0.
          start = 1
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( i.EQ.field%time_axis_index )start(i) = mpp_file(unit)%time_level
             start(i) = max(start(i),1)
          end do
          if( PRESENT(domain) )then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je,  xsize=isizc, ysize=
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg, xsize=isizg, ysize=
              axsiz(1) = isizc
              axsiz(2) = jsizc
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( isizc.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( jsizc.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
          end if
!RV,SGI
          if( PRESENT(block_id) )then
            if (block_id.le.0) then
               call mpp_error( FATAL, 'MPP_RECORD_B: block_id <= 0!' )
            endif
            if( PRESENT(time_in) )then

               if(block_id.gt. axsiz(size(field%axes)-1)) &
                 call mpp_error( FATAL, 'MPP_RECORD_B: block_id > axis range!' )

               start(size(field%axes)-1)=block_id

            else

               if(block_id.gt. axsiz(size(field%axes))) &
                 call mpp_error( FATAL, 'MPP_RECORD_B: block_id > axis range!' )

               start(size(field%axes))=block_id

            endif
          endif
!RV,SGI
          if( debug ) &
           write (stdout(), '(a,2i3,12i4)') 'e WRITE_RECORD: PE, unit, start, axsiz=', pe,
#ifdef use_netCDF
!write time information if new time
          if( newtime )then
              il_bytesize = BIT_SIZE(ii)/8
              INQUIRE (iolength=il_iosize) ii
              il_int_iosize = il_iosize
              INQUIRE (iolength=il_iosize) time
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt .EQ. DOUBLE_KIND )then
                  idim = mpp_file(unit)%time_level
                  error = NFMPI_PUT_VAR1_DOUBLE( mpp_file(unit)%ncid, mpp_file(unit)%id, i
              else if( il_rbyt .EQ. FLOAT_KIND )then
                  idim = mpp_file(unit)%time_level
                  error = NFMPI_PUT_VAR1_REAL  ( mpp_file(unit)%ncid, mpp_file(unit)%id, i
              end if
          end if
          if( field%pack.LE.2 )then
             INQUIRE (iolength=il_iosize) data(1)
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt.EQ.DOUBLE_KIND )then
!                  write(stderr,*)data
                  error = NFMPI_PUT_VARA_DOUBLE_ALL( mpp_file(unit)%ncid, field%id, start,
              else if( il_rbyt.EQ.FLOAT_KIND )then
                  error = NFMPI_PUT_VARA_REAL_ALL  ( mpp_file(unit)%ncid, field%id, start,
              end if
          else              !convert to integer using scale and add: no error check on pac
              packed_data = nint((data-field%add)/field%scale)
              error = NFMPI_PUT_VARA_INT_ALL   ( mpp_file(unit)%ncid, field%id, start, axs
          end if
          call netcdf_err(error)
#endif
      else                          !non-netCDF
!subdomain contains (/is,ie,js,je/)
          if( PRESENT(domain) )then
              subdomain(:) = (/ is, ie, js, je /)
          else
              subdomain(:) = -1    ! -1 means use global value from axis metadata
          end if
          if( mpp_file(unit)%format.EQ.MPP_ASCII )then
!implies sequential access
              write( unit,* )field%id, subdomain, time_level, time, data
          else                      !MPP_IEEE32 or MPP_NATIVE
              if( mpp_file(unit)%access.EQ.MPP_SEQUENTIAL )then
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      data_r4 = data !IEEE conversion layer on SGI until assign -N ieee_32
                      write(unit)field%id, subdomain, time_level, time, data_r4
                  else
                      write(unit)field%id, subdomain, time_level, time, data
                  end if
#else
                  write(unit)field%id, subdomain, time_level, time, data
#endif
              else                  !MPP_DIRECT
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      data_r4 = data !IEEE conversion layer on SGI until assign -N ieee_32
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  else
```

```
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  end if
#else
                  write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_level,
#endif
                  if( debug ) write (stdout(), '(a,i3,a,i3)') 'MPP_WRITE: PE=', pe, ' wrot
              end if
          end if
      end if

!recompute current record for direct access I/O
      if( mpp_file(unit)%access.EQ.MPP_DIRECT )then
          if( mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
!assumes all PEs participate in I/O: modify later
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe*npes
          else
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe
          end if
      end if

!RV
      deallocate(start)
      deallocate(axsiz)
!RV
      return
    end subroutine write_record_b

    subroutine write_record( unit, field, nwords, data, time_in, domain )
!routine that is finally called by all mpp_write routines to perform the write
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
      real, intent(in) :: data(nwords)
      real(DOUBLE_KIND), intent(in), optional :: time_in
      type(domain2D), intent(in), optional :: domain
!RV   Very unsafe!!!! One can not use size(field%axes) before it
!RV   is clear that every thing has been initialized.
!RV   The code crashes in a multi-PE run.
!RV      integer, dimension(size(field%axes)) :: start, axsiz
!rr
      integer(kind=MPI_OFFSET_KIND),allocatable,dimension(:) :: start, axsiz
!RV
      real :: time
      integer :: time_level
      logical :: newtime
      integer :: subdomain(4)
      integer :: packed_data(nwords)
      integer :: i, is, ie, js, je, isg, ieg, jsg, jeg, isizc, jsizc, isizg, jsizg
!rv,sgi<
```

```
      integer :: icount_domains
!rv,sgi>
#ifdef use_netCDF
      integer :: ii, il_bytesize, il_iosize
      integer :: il_int_iosize, il_rbyt
#endif

#ifdef use_CRI_pointers
      real(FLOAT_KIND) :: data_r4(nwords)
      pointer( ptr1, data_r4)
      pointer( ptr2, packed_data)

      if (mpp_io_stack_size < 2*nwords) call mpp_io_set_stack_size(2*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
#endif

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_WRITE: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE: invalid unit numb
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset  .EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return

!RV
      allocate(start(size(field%axes)))
      allocate(axsiz(size(field%axes)))
!RV
      if( .NOT.mpp_file(unit)%initialized )then
!this is the first call to mpp_write
!we now declare the file to be initialized
!if this is netCDF we switch file from DEFINE mode to DATA mode
          if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
!NOFILL is probably required for parallel: any circumstances in which not advisable?
!rr not yet supported
!rr        error = NFMPI_SET_FILL( mpp_file(unit)%ncid, NF_NOFILL, i ); call netcdf_err(er
           if( mpp_file(unit)%action.EQ.MPP_WRONLY )error = NFMPI_ENDDEF(mpp_file(unit)%nc
#endif
          else
              call mpp_write_meta( unit, 'END', cval='metadata' )
          end if
          mpp_file(unit)%initialized = .TRUE.
          if( verbose ) write (stdout(), '(a,i3,a)') 'MPP_WRITE: PE=', pe, ' initialized f
      end if

!initialize time: by default assume NULLTIME
      time = NULLTIME
      time_level = -1
      newtime = .FALSE.
      if( PRESENT(time_in) )time = time_in
!increment time level if new time
      if( time.GT.mpp_file(unit)%time+EPSILON(time) )then !new time
          mpp_file(unit)%time_level = mpp_file(unit)%time_level + 1
          mpp_file(unit)%time = time
          newtime = .TRUE.
      end if
      if( verbose ) write (stdout(), '(a,2i3,2i5,es13.5)') 'MPP_WRITE: PE, unit, %id, %tim
           pe, unit, mpp_file(unit)%id, mpp_file(unit)%time_level, mpp_file(unit)%time

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be written:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NFMPI_PUT_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1
! NOTE: passing of subdomains will fail for multi-PE single-threaded I/O,
!       since that attempts to gather all data on PE 0.
          start = 1
!
!rv,sgi<
!Treatment of the case x(k,i,j) where k is a common, non-decompsoed axis of
!all PEs and i,j are 2D decomposed .
!the array x(k,i,j) is collapsed allong the two first axis. It is treated 2D.
!A corresponding domain is defined as well which is used for stitching.
!However, for writing to a file the decomposition information is taken
!from the field axes rather then from the domain 'domain'.
!If icount_domains is 2 we have exactly that case.
          icount_domains=0
!rv,sgi<
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( i.EQ.field%time_axis_index )start(i) = mpp_file(unit)%time_level
!rr
             start(i) = max(start(i),1)
             if ( start(i) < 1 ) start(i) = 1
!rv,sgi<
             if((field%axes(i)%domain .ne.  NULL_DOMAIN1D) .and. &
                (field%axes(1)%domain .eq.  NULL_DOMAIN1D)) &
                icount_domains=icount_domains+1
!rv,sgi>
          end do
          if( PRESENT(domain) )then
            if(icount_domains .ne. 2 ) then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je &
                                           ,  xsize=isizc, ysize=jsizc )
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg &
                                           , xsize=isizg, ysize=jsizg )
#ifdef __PARNETCDF
!rr for longitudes, latitudes, and data
              start(1) = is - isg + 1
              start(2) = js - jsg + 1
#endif
              axsiz(1) = isizc
              axsiz(2) = jsizc
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( isizc.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( jsizc.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
!rv,sgi<
            else

              call mpp_get_compute_domain( field%axes(2)%domain, is,  ie &
                                         ,  size=isizc)
              call mpp_get_global_domain ( field%axes(2)%domain, isg, ieg &
                                         , size=isizg )
              call mpp_get_compute_domain( field%axes(3)%domain, js,  je &
                                         ,  size=jsizc)
              call mpp_get_global_domain ( field%axes(3)%domain, jsg, jeg &
                                         , size=jsizg )


#ifdef __PARNETCDF
!rr for bounds

              start(2) = is - isg + 1
              start(3) = js - jsg + 1
#endif

              axsiz(2) = isizc
              axsiz(3) = jsizc
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(2) = is - isg + 1
                  start(3) = js - jsg + 1
              else
                  if( isizc.NE.ie-is+1 )then
                      start(2) = is - isg + 1
                      axsiz(2) = ie - is + 1
                  end if
                  if( jsizc.NE.je-js+1 )then
                      start(3) = js - jsg + 1
                      axsiz(3) = je - js + 1
                  end if
              end if

            endif
!rv,sgi>
          end if
          if( debug ) write (stdout(),'(a,3i5,12i4)') &
                      'f WRITE_RECORD: PE, unit, icount_domains, start, axsiz=' &
                    , pe, unit, icount_domains, start, axsiz
#ifdef use_netCDF
!write time information if new time
          il_bytesize = BIT_SIZE(ii)/8
          INQUIRE (iolength=il_iosize) ii
          il_int_iosize = il_iosize
          if( newtime )then
              INQUIRE (iolength=il_iosize) time
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt .EQ. DOUBLE_KIND )then
                  idim = mpp_file(unit)%time_level
                  error = NFMPI_PUT_VAR1_DOUBLE( mpp_file(unit)%ncid, mpp_file(unit)%id, i
              else if( il_rbyt .EQ. FLOAT_KIND )then
                  idim = mpp_file(unit)%time_level
                  error = NFMPI_PUT_VAR1_REAL  ( mpp_file(unit)%ncid, mpp_file(unit)%id, i
              end if
          end if
          if( field%pack.LE.2 )then
              INQUIRE (iolength=il_iosize) data(1)
              il_rbyt = il_iosize/il_int_iosize*il_bytesize
              if( il_rbyt.EQ.DOUBLE_KIND )then
                  error = NFMPI_PUT_VARA_DOUBLE_ALL( mpp_file(unit)%ncid, field%id, start,
              else if( il_rbyt.EQ.FLOAT_KIND )then
                  error = NFMPI_PUT_VARA_REAL_ALL  ( mpp_file(unit)%ncid, field%id, start,
              end if
          else              !convert to integer using scale and add: no error check on pac
              packed_data = nint((data-field%add)/field%scale)
              error = NFMPI_PUT_VARA_INT_ALL  ( mpp_file(unit)%ncid, field%id, start, axsi
          end if
          call netcdf_err(error)
#endif
      else                         !non-netCDF
!subdomain contains (/is,ie,js,je/)
          if( PRESENT(domain) )then
              subdomain(:) = (/ is, ie, js, je /)
          else
              subdomain(:) = -1    ! -1 means use global value from axis metadata
          end if
          if( mpp_file(unit)%format.EQ.MPP_ASCII )then
!implies sequential access
              write( unit,* )field%id, subdomain, time_level, time, data
          else                      !MPP_IEEE32 or MPP_NATIVE
              if( mpp_file(unit)%access.EQ.MPP_SEQUENTIAL )then
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      data_r4 = data !IEEE conversion layer on SGI until assign -N ieee_32
                      write(unit)field%id, subdomain, time_level, time, data_r4
                  else
                      write(unit)field%id, subdomain, time_level, time, data
                  end if
#else
                  write(unit)field%id, subdomain, time_level, time, data
#endif
              else                  !MPP_DIRECT
#ifdef __sgi
                  if( mpp_file(unit)%format.EQ.MPP_IEEE32 )then
                      data_r4 = data !IEEE conversion layer on SGI until assign -N ieee_32
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  else
                      write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_le
                  end if
#else
                  write( unit, rec=mpp_file(unit)%record )field%id, subdomain, time_level,
#endif
                  if( debug ) write (stdout(), '(a,i3,a,i3)') 'MPP_WRITE: PE=', pe, ' wrot
              end if
          end if
      end if

!recompute current record for direct access I/O
      if( mpp_file(unit)%access.EQ.MPP_DIRECT )then
          if( mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
!assumes all PEs participate in I/O: modify later
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe*npes
          else
              mpp_file(unit)%record = mpp_file(unit)%record + records_per_pe
          end if
      end if
!RV
```

```
      deallocate(start)
      deallocate(axsiz)
!RV

      return
    end subroutine write_record

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!                          MPP_COPY_META                               !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    subroutine mpp_copy_meta_global( unit, gatt )
!writes a global metadata attribute to unit <unit>
!attribute <name> can be an real, integer or character
!one and only one of rval, ival, and cval should be present
!the first found will be used
!for a non-netCDF file, it is encoded into a string "GLOBAL <name> <val>"
      integer, intent(in) :: unit
      type(atttype), intent(in) :: gatt
      integer :: len

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an
#ifdef use_netCDF
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
         if( gatt%type.EQ.NF_CHAR )then
            len = gatt%len
            call write_attribute_netcdf( unit, NF_GLOBAL, gatt%name, cval=gatt%catt(1:len)
         else
            call write_attribute_netcdf( unit, NF_GLOBAL, gatt%name, rval=gatt%fatt )
         endif
      else
         if( gatt%type.EQ.NF_CHAR )then
            len=gatt%len
            call write_attribute( unit, 'GLOBAL '//trim(gatt%name), cval=gatt%catt(1:len)
         else
            call write_attribute( unit, 'GLOBAL '//trim(gatt%name), rval=gatt%fatt )
         endif
     end if
#else
     call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine mpp_copy_meta_global

    subroutine mpp_copy_meta_axis( unit, axis, domain )
!load the values in an axistype (still need to call mpp_write)
!write metadata attributes for axis.  axis is declared inout
!because the variable and dimension ids are altered

      integer, intent(in) :: unit
      type(axistype), intent(inout) :: axis
      type(domain1D), intent(in), optional :: domain
      character(len=512) :: text
      integer :: i, len, is, ie, isg, ieg

      integer(kind=mpi_offset_kind) :: idim
      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

! redefine domain if present
      if( PRESENT(domain) )then
          axis%domain = domain
      else
          axis%domain = NULL_DOMAIN1D
      end if

#ifdef use_netCDF
!write metadata
      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then

!write axis def
          if( ASSOCIATED(axis%data) )then !space axis
              if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )
                  call mpp_get_compute_domain( axis%domain, is, ie )
                  call mpp_get_global_domain( axis%domain, isg, ieg )
                  idim = ie-is+1
              else
                  idim =  size(axis%data)
              end if
              error = NFMPI_DEF_DIM( mpp_file(unit)%ncid, axis%name, idim, axis%did )
              call netcdf_err(error)
              error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_FLOAT, 1, axis%dic
          else                            !time axis
              idim = NF_UNLIMITED
              error = NFMPI_DEF_DIM( mpp_file(unit)%ncid, axis%name, idim, axis%did ); cal
              error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, axis%name, NF_DOUBLE, 1, axis%di
              mpp_file(unit)%id = axis%id !file ID is the same as time axis varID
              mpp_file(unit)%recdimid = axis%did ! record dimension id
          end if
      else
          varnum = varnum + 1
          axis%id = varnum
          axis%did = varnum
!write axis def
          write( text, '(a,i4,a)' )'AXIS ', axis%id, ' name'
          call write_attribute( unit, trim(text), cval=axis%name )
          write( text, '(a,i4,a)' )'AXIS ', axis%id, ' size'
          if( ASSOCIATED(axis%data) )then !space axis
              if( mpp_file(unit)%fileset.EQ.MPP_MULTI .AND. axis%domain.NE.NULL_DOMAIN1D )
                  call write_attribute( unit, trim(text), ival=(/ie-is+1/) )
              else
                  call write_attribute( unit, trim(text), ival=(/size(axis%data)/) )
              end if
          else                            !time axis
              if( mpp_file(unit)%id.NE.-1 ) &
                   call mpp_error( FATAL, 'MPP_WRITE_META_AXIS: There is already a time ax
              call write_attribute( unit, trim(text), ival=(/0/) ) !a size of 0 indicates
              mpp_file(unit)%id = axis%id
          end if
      end if
!write axis attributes

      do i=1,axis%natt
```

```
         if( axis%Att(i)%name.NE.default_att%name )then
            if( axis%Att(i)%type.EQ.NF_CHAR )then
               len = axis%Att(i)%len
               call mpp_write_meta( unit, axis%id, axis%Att(i)%name, cval=axis%Att(i)%catt
            else
               call mpp_write_meta( unit, axis%id, axis%Att(i)%name, rval=axis%Att(i)%fatt
            endif
         endif
      enddo

      if( mpp_file(unit)%threading.EQ.MPP_MULTI .AND. mpp_file(unit)%fileset.EQ.MPP_MULTI
          call mpp_write_meta( unit, axis%id, 'domain_decomposition', ival=(/isg,ieg,is,ie
      end if
      if( verbose ) write (stdout(), '(a,2i3,1x,a,2i3)') &
           'MPP_WRITE_META: Wrote axis metadata, pe, unit, axis%name, axis%id, axis%did=',
           pe, unit, trim(axis%name), axis%id, axis%did
#else
      call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine mpp_copy_meta_axis

    subroutine mpp_copy_meta_field( unit, field, axes )
!useful for copying field metadata from a previous call to mpp_read_meta
!define field: must have already called mpp_write_meta(axis) for each axis
      integer, intent(in) :: unit
      type(fieldtype), intent(inout) :: field
      type(axistype), intent(in), optional :: axes(:)
!this array is required because of f77 binding on netCDF interface
      integer, allocatable :: axis_id(:)
      real :: a, b
      integer :: i

      if( .NOT.module_is_initialized    )call mpp_error( FATAL, 'MPP_WRITE_META: must firs
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_WRITE_META: invalid unit
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%fileset.EQ.MPP_SINGLE   .AND. pe.NE.mpp_root_pe() )return
      if( mpp_file(unit)%action.NE.MPP_WRONLY )return !no writing metadata on APPEND
      if( mpp_file(unit)%initialized ) &
           call mpp_error( FATAL, 'MPP_WRITE_META: cannot write metadata to file after an

       if( field%pack.NE.1 .AND. field%pack.NE.2 )then
            if( field%pack.NE.4 .AND. field%pack.NE.8 ) &
               call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: only legal packing values are
      end if

      if (PRESENT(axes)) then
         deallocate(field%axes)
         deallocate(field%size)
         allocate(field%axes(size(axes)))
         allocate(field%size(size(axes)))
         field%axes = axes
         do i=1,size(axes)
            if (ASSOCIATED(axes(i)%data)) then
               field%size(i) = size(axes(i)%data)
            else
               field%size(i) = 1
               field%time_axis_index = i
            endif
         enddo
      endif

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
```

```
#ifdef use_netCDF
          allocate( axis_id(size(field%axes)) )
          do i = 1,size(field%axes)
             axis_id(i) = field%axes(i)%did
          end do
!write field def
          select case (field%pack)
              case(1)
                  error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_DOUBLE, size(
              case(2)
                  error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_FLOAT,  size(
              case(4)
                  if( field%scale.EQ.default_field%scale .OR. field%add.EQ.default_field%a
                       call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: scale and add must be
                  error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_SHORT,  size(
              case(8)
                  if( field%scale.EQ.default_field%scale .OR. field%add.EQ.default_field%a
                       call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: scale and add must be
                  error = NFMPI_DEF_VAR( mpp_file(unit)%ncid, field%name, NF_BYTE,   size(
              case default
                  call mpp_error( FATAL, 'MPP_WRITE_META_FIELD: only legal packing values
          end select
#endif
      else
          varnum = varnum + 1
          field%id = varnum
          if( field%pack.NE.default_field%pack ) &
           call mpp_error( WARNING, 'MPP_WRITE_META: Packing is currently available only c
!write field def
          write( text, '(a,i4,a)' )'FIELD ', field%id, ' name'
          call write_attribute( unit, trim(text), cval=field%name )
          write( text, '(a,i4,a)' )'FIELD ', field%id, ' axes'
          call write_attribute( unit, trim(text), ival=field%axes(:)%did )
      end if
!write field attributes: these names follow netCDF conventions
      call mpp_write_meta( unit, field%id, 'long_name', cval=field%longname )
      call mpp_write_meta( unit, field%id, 'units',     cval=field%units    )
!all real attributes must be written as packed
      if( (field%min.NE.default_field%min) .AND. (field%max.NE.default_field%max) )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_range', rval=(/field%min,field%m
          else
              a = nint((field%min-field%add)/field%scale)
              b = nint((field%max-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, 'valid_range', rval=(/a,  b  /), pack=f
          end if
      else if( field%min.NE.default_field%min )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_min', rval=field%min, pack=fielc
          else
              a = nint((field%min-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, 'valid_min', rval=a, pack=field%pack )
          end if
      else if( field%max.NE.default_field%max )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, 'valid_max', rval=field%max, pack=fielc
          else
              a = nint((field%max-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, 'valid_max', rval=a, pack=field%pack )
          end if
      end if
      if( field%missing.NE.default_field%missing )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
```

```
              call mpp_write_meta( unit, field%id, 'missing_value', rval=field%missing, pa
          else
              a = nint((field%missing-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, 'missing_value', rval=a, pack=field%pac
          end if
      end if
      if( field%fill.NE.default_field%fill )then
          if( field%pack.EQ.1 .OR. field%pack.EQ.2 )then
              call mpp_write_meta( unit, field%id, '_FillValue', rval=field%missing, pack=
          else
              a = nint((field%fill-field%add)/field%scale)
              call mpp_write_meta( unit, field%id, '_FillValue', rval=a, pack=field%pack )
          end if
      end if
      if( field%pack.NE.1 .AND. field%pack.NE.2 )then
          call mpp_write_meta( unit, field%id, 'packing', ival=field%pack )
          if( field%scale.NE.default_field%scale )call mpp_write_meta( unit, field%id, 'sc
          if( field%add.NE.default_field%add   )call mpp_write_meta( unit, field%id, 'add_
      end if
      if( verbose ) write (stdout(), '(a,2i3,1x,a,i3)') 'MPP_WRITE_META: Wrote field metac
           pe, unit, trim(field%name), field%id

      return
    end subroutine mpp_copy_meta_field

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!                               MPP_READ                               !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

#define MPP_READ_2DDECOMP_1D_ mpp_read_2ddecomp_r1d
#define MPP_READ_2DDECOMP_2D_ mpp_read_2ddecomp_r2d
#define MPP_READ_2DDECOMP_3D_ mpp_read_2ddecomp_r3d
#define MPP_READ_2DDECOMP_4D_ mpp_read_2ddecomp_r4d
#define MPP_TYPE_ real
#include <mpp_read_2Ddecomp.h>

    subroutine read_record( unit, field, nwords, data, time_level, domain )
!routine that is finally called by all mpp_read routines to perform the read
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(nwords)
      integer, intent(in), optional  :: time_level
      type(domain2D), intent(in), optional :: domain
      integer(kind=MPI_OFFSET_KIND), dimension(size(field%axes)) :: start, axsiz
      real :: time
      logical :: newtime
      integer :: subdomain(4), tlevel

      integer(SHORT_KIND) :: i2vals(nwords)
!#ifdef __sgi
      integer(INT_KIND) :: ivals(nwords)
      real(FLOAT_KIND) :: rvals(nwords)
!#else
!      integer :: ivals(nwords)
!      real :: rvals(nwords)
!#endif

      real(DOUBLE_KIND) :: r8vals(nwords)

      integer :: i, error, is, ie, js, je, isg, ieg, jsg, jeg

#ifdef use_CRI_pointers
      pointer( ptr1, i2vals )
      pointer( ptr2, ivals )
      pointer( ptr3, rvals )
      pointer( ptr4, r8vals )

      if (mpp_io_stack_size < 4*nwords) call mpp_io_set_stack_size(4*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
      ptr3 = LOC(mpp_io_stack(2*nwords+1))
      ptr4 = LOC(mpp_io_stack(3*nwords+1))
#endif
      if (.not.PRESENT(time_level)) then
          tlevel = 0
      else
          tlevel = time_level
      endif

#ifdef use_netCDF
      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'READ_RECORD: must first call
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'READ_RECORD: invalid unit nu
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
!RV
      if( mpp_file(unit)%fileset.EQ.MPP_MULTI .and. present(domain)) &
        call mpp_error( FATAL, 'READ_RECORD: multiple filesets not supported for MPP_READ'

      if( .NOT.mpp_file(unit)%initialized ) call mpp_error( FATAL, 'MPP_READ: must first c



      if( verbose ) write (stdout(), '(a,2i3,2i5)') 'MPP_READ: PE, unit, %id, %time_level
           pe, unit, mpp_file(unit)%id, tlevel

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be read:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NFMPI_GET_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1
! NOTE: passing of subdomains will fail for multi-PE single-threaded I/O,
!       since that attempts to gather all data on PE 0.
          start = 1
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( field%axes(i)%did.EQ.field%time_axis_index )start(i) = tlevel
          end do
          if( PRESENT(domain) )then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je  )
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg )
              axsiz(1) = ie-is+1
              axsiz(2) = je-js+1
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( ie-is+1.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( je-js+1.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
          end if

          if( verbose ) write (stdout(), '(a,2i3,i6,12i4)') 'READ_RECORD: PE, unit, nwords
                           pe, unit, nwords, start, axsiz

          select case (field%type)
             case(NF_BYTE)
! use type conversion
                call mpp_error( FATAL, 'MPP_READ: does not support NF_BYTE packing' )
             case(NF_SHORT)
                error = NFMPI_GET_VARA_INT2_ALL  ( mpp_file(unit)%ncid, field%id, start, a
                 data(:)=i2vals(:)*field%scale + field%add
             case(NF_INT)
                error = NFMPI_GET_VARA_INT_ALL   ( mpp_file(unit)%ncid, field%id, start, a
                data(:)=ivals(:)
             case(NF_FLOAT)
                error = NFMPI_GET_VARA_REAL_ALL  ( mpp_file(unit)%ncid, field%id, start, a
                data(:)=rvals(:)
             case(NF_DOUBLE)
                error = NFMPI_GET_VARA_DOUBLE_ALL( mpp_file(unit)%ncid, field%id, start, a
                data(:)=r8vals(:)
             case default
                call mpp_error( FATAL, 'MPP_READ: invalid pack value' )
          end select
      else                        !non-netCDF
!subdomain contains (/is,ie,js,je/)
          call mpp_error( FATAL, 'Currently dont support non-NetCDF mpp read' )

      end if
#else
```

```
      call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine read_record
    subroutine read_record_b(unit,field,nwords,data,time_level,domain,block_id)
!routine that is finally called by all mpp_read routines to perform the read
!a non-netCDF record contains:
!      field ID
!      a set of 4 coordinates (is:ie,js:je) giving the data subdomain
!      a timelevel and a timestamp (=NULLTIME if field is static)
!      3D real data (stored as 1D)
!if you are using direct access I/O, the RECL argument to OPEN must be large enough for th
!in a global direct access file, record position on PE is given by %record.

!Treatment of timestamp:
!   We assume that static fields have been passed without a timestamp.
!   Here that is converted into a timestamp of NULLTIME.
!   For non-netCDF fields, field is treated no differently, but is written
!   with a timestamp of NULLTIME. There is no check in the code to prevent
!   the user from repeatedly writing a static field.

      integer, intent(in) :: unit, nwords
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(nwords)
      integer, intent(in), optional  :: time_level
!RV
      integer, intent(in), optional  ::  block_id
!RV
      type(domain2D), intent(in), optional :: domain
      integer(kind=MPI_OFFSET_KIND), dimension(size(field%axes)) :: start, axsiz
      real :: time

      logical :: newtime
      integer :: subdomain(4), tlevel

      integer(SHORT_KIND) :: i2vals(nwords)
!#ifdef __sgi
      integer(INT_KIND) :: ivals(nwords)
      real(FLOAT_KIND) :: rvals(nwords)
!#else
!      integer :: ivals(nwords)
!      real :: rvals(nwords)
!#endif

      real(DOUBLE_KIND) :: r8vals(nwords)

      integer :: i, error, is, ie, js, je, isg, ieg, jsg, jeg

#ifdef use_CRI_pointers
      pointer( ptr1, i2vals )
      pointer( ptr2, ivals )
      pointer( ptr3, rvals )
      pointer( ptr4, r8vals )

      if (mpp_io_stack_size < 4*nwords) call mpp_io_set_stack_size(4*nwords)

      ptr1 = LOC(mpp_io_stack(1))
      ptr2 = LOC(mpp_io_stack(nwords+1))
      ptr3 = LOC(mpp_io_stack(2*nwords+1))
      ptr4 = LOC(mpp_io_stack(3*nwords+1))
#endif
      if (.not.PRESENT(time_level)) then
          tlevel = 0
      else
          tlevel = time_level
      endif

#ifdef use_netCDF
      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'READ_RECORD: must first call
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'READ_RECORD: invalid unit nu
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return
!RV
      if( mpp_file(unit)%fileset.EQ.MPP_MULTI .and. present(domain)) &
        call mpp_error( FATAL, 'READ_RECORD: multiple filesets not supported for MPP_READ'

      if( .NOT.mpp_file(unit)%initialized ) call mpp_error( FATAL, 'MPP_READ: must first c



      if( verbose ) write (stdout(), '(a,2i3,2i5)') 'MPP_READ: PE, unit, %id, %time_level
           pe, unit, mpp_file(unit)%id, tlevel

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
!define netCDF data block to be read:
!  time axis: START = time level
!             AXSIZ = 1
!  space axis: if there is no domain info
!              START = 1
!              AXSIZ = field%size(axis)
!          if there IS domain info:
!              start of domain is compute%start_index for multi-file I/O
!                                 global%start_index for all other cases
!              this number must be converted to 1 for NFMPI_GET_VAR
!                  (netCDF fortran calls are with reference to 1),
!          So, START = compute%start_index - <start of domain> + 1
!              AXSIZ = usually compute%size
!          However, if compute%start_index-compute%end_index+1.NE.compute%size,
!              we assume that the call is passing a subdomain.
!              To pass a subdomain, you must pass a domain2D object that satisfies the fol
!                  global%start_index must contain the <start of domain> as defined above;
!                  the data domain and compute domain must refer to the subdomain being pa
!              In this case, START = compute%start_index - <start of domain> + 1
!                            AXSIZ = compute%start_index - compute%end_index + 1
! NOTE: passing of subdomains will fail for multi-PE single-threaded I/O,
!       since that attempts to gather all data on PE 0.
          start = 1
          do i = 1,size(field%axes)
             axsiz(i) = field%size(i)
             if( field%axes(i)%did.EQ.field%time_axis_index )start(i) = tlevel
          end do
          if( PRESENT(domain) )then
              call mpp_get_compute_domain( domain, is,  ie,  js,  je  )
              call mpp_get_global_domain ( domain, isg, ieg, jsg, jeg )
              axsiz(1) = ie-is+1
              axsiz(2) = je-js+1
              if( npes.GT.1 .AND. mpp_file(unit)%fileset.EQ.MPP_SINGLE )then
                  start(1) = is - isg + 1
                  start(2) = js - jsg + 1
              else
                  if( ie-is+1.NE.ie-is+1 )then
                      start(1) = is - isg + 1
                      axsiz(1) = ie - is + 1
                  end if
                  if( je-js+1.NE.je-js+1 )then
                      start(2) = js - jsg + 1
                      axsiz(2) = je - js + 1
                  end if
              end if
          end if
!RV,SGI
          if( PRESENT(block_id) )then
            if (block_id.le.0) then
               call mpp_error( FATAL, 'READ_RECORD_B: block_id <= 0!' )
            endif
            if( PRESENT(time_level) )then

               if(block_id.gt. axsiz(size(field%axes)-1)) &
                 call mpp_error( FATAL, 'READ_RECORD_B: block_id > axis range!' )
               start(size(field%axes)-1)=block_id

            else

               if(block_id.gt. axsiz(size(field%axes))) &
                 call mpp_error( FATAL, 'READ_RECORD_B: block_id > axis range!' )
               start(size(field%axes))=block_id

            endif
          endif

!RV,SGI

          if( verbose ) write (stdout(), '(a,2i3,i6,12i4)') 'READ_RECORD: PE, unit, nwords
                          pe, unit, nwords, start, axsiz

          select case (field%type)
             case(NF_BYTE)
! use type conversion
                call mpp_error( FATAL, 'MPP_READ: does not support NF_BYTE packing' )
             case(NF_SHORT)
                error = NFMPI_GET_VARA_INT2_ALL  ( mpp_file(unit)%ncid, field%id, start, a
                 data(:)=i2vals(:)*field%scale + field%add
             case(NF_INT)
                error = NFMPI_GET_VARA_INT_ALL   ( mpp_file(unit)%ncid, field%id, start, a
                data(:)=ivals(:)
             case(NF_FLOAT)
                error = NFMPI_GET_VARA_REAL_ALL  ( mpp_file(unit)%ncid, field%id, start, a
                data(:)=rvals(:)
             case(NF_DOUBLE)
                error = NFMPI_GET_VARA_DOUBLE_ALL( mpp_file(unit)%ncid, field%id, start, a
                data(:)=r8vals(:)
             case default
                call mpp_error( FATAL, 'MPP_READ: invalid pack value' )
          end select
      else                      !non-netCDF
!subdomain contains (/is,ie,js,je/)
          call mpp_error( FATAL, 'Currently dont support non-NetCDF mpp read' )

      end if
#else
      call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine read_record_b

    subroutine mpp_read_r4D( unit, field, data, tindex,blockid)
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(:,:,:,:)
      integer, intent(in), optional :: tindex
      integer, intent(in), optional :: blockid

      if(present(blockid)) then
        call read_record_b(unit,field,size(data),data,tindex,block_id=blockid )
      else
        call read_record( unit, field, size(data), data, tindex )
      endif
    end subroutine mpp_read_r4D

    subroutine mpp_read_r3D( unit, field, data, tindex,blockid)
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(:,:,:)
      integer, intent(in), optional :: tindex
      integer, intent(in), optional :: blockid

      if(present(blockid)) then
        call read_record_b(unit,field,size(data),data,tindex,block_id=blockid )
      else
        call read_record( unit, field, size(data), data, tindex )
      endif
    end subroutine mpp_read_r3D

    subroutine mpp_read_r2D( unit, field, data, tindex )
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(:,:)
      integer, intent(in), optional :: tindex

      call read_record( unit, field, size(data), data, tindex )
    end subroutine mpp_read_r2D

    subroutine mpp_read_r1D( unit, field, data, tindex )
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data(:)
      integer, intent(in), optional :: tindex

      call read_record( unit, field, size(data), data, tindex )
    end subroutine mpp_read_r1D

    subroutine mpp_read_r0D( unit, field, data, tindex )
      integer, intent(in) :: unit
      type(fieldtype), intent(in) :: field
      real, intent(inout) :: data
      integer, intent(in), optional :: tindex
      real, dimension(1) :: data_tmp

      data_tmp(1)=data
      call read_record( unit, field, 1, data_tmp, tindex )
      data=data_tmp(1)
    end subroutine mpp_read_r0D

    subroutine mpp_read_meta(unit)
!
! read file attributes including dimension and variable attributes
! and store in filetype structure.  All of the file information
! with the exception of the (variable) data is stored.  Attributes
! are supplied to the user by get_info,get_atts,get_axes and get_fields
!
! every PE is eligible to call mpp_read_meta
!
      integer, parameter :: MAX_DIMVALS = 100000
      integer, intent(in) :: unit

      integer         :: ncid,ndim,nvar_total,natt,recdim,nv,nvar,len
      integer :: error,i,j
      integer         :: type,nvdims,nvatts, dimid
      integer, allocatable, dimension(:) :: dimids
      type(axistype) , allocatable, dimension(:) :: Axis
      character(len=128) :: name, attname, unlimname, attval
      logical :: isdim

      integer(SHORT_KIND) :: i2vals(MAX_DIMVALS)
!#ifdef __sgi
      integer(INT_KIND) :: ivals(MAX_DIMVALS)
      real(FLOAT_KIND)  :: rvals(MAX_DIMVALS)
!#else
!      integer :: ivals(MAX_DIMVALS)
!      real    :: rvals(MAX_DIMVALS)
!#endif
      real(DOUBLE_KIND) :: r8vals(MAX_DIMVALS)

#ifdef use_netCDF

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
        ncid = mpp_file(unit)%ncid
        error = NFMPI_INQ(ncid,ndim, nvar_total,&
                      natt, recdim);call netcdf_err(error)


        mpp_file(unit)%ndim = ndim
        mpp_file(unit)%natt = natt
        mpp_file(unit)%recdimid = recdim
!
! if no recdim exists, recdimid = -1
! variable id of unlimdim and length
!
        if( recdim.NE.-1 )then
           error = NFMPI_INQ_DIM( ncid, recdim, unlimname, idim );call netcdf_err(error)
           mpp_file(unit)%time_level = idim
           error = NFMPI_INQ_VARID( ncid, unlimname, mpp_file(unit)%id ); call netcdf_err(
        else
           mpp_file(unit)%time_level = -1 ! set to zero so mpp_get_info returns ntime=0 if
        endif

        if ( natt .gt. 0 ) allocate(mpp_file(unit)%Att(natt))
        allocate(Axis(ndim))
        allocate(dimids(ndim))
        allocate(mpp_file(unit)%Axis(ndim))

!
! initialize fieldtype and axis type
!



        do i=1,ndim
           Axis(i) = default_axis
           mpp_file(unit)%Axis(i) = default_axis
        enddo

        do i=1,natt
           mpp_file(unit)%Att(i) = default_att
        enddo

!
```

```
! assign global attributes
!
        do i=1,natt
           error=NFMPI_INQ_ATTNAME(ncid,NF_GLOBAL,i,name);call netcdf_err(error)
           error=NFMPI_INQ_ATT(ncid,NF_GLOBAL,trim(name),type,idim);call netcdf_err(error)
           len = idim
           mpp_file(unit)%Att(i)%name = name
           mpp_file(unit)%Att(i)%len = len
           mpp_file(unit)%Att(i)%type = type
!
!  allocate space for att data and assign
!
           select case (type)

              case (NF_CHAR)
                 if (len.gt.512) then
                    call mpp_error(NOTE,'GLOBAL ATT too long - not reading this metadata')
                    len=7
                    mpp_file(unit)%Att(i)%len=len
                    mpp_file(unit)%Att(i)%catt = 'unknown'
                 else
                     error=NFMPI_GET_ATT_TEXT(ncid,NF_GLOBAL,name,mpp_file(unit)%Att(i)%ca
                     if (verbose.and.pe == 0) write (stdout(),*)  'GLOBAL ATT ',trim(name)
                 endif
!
! store integers in float arrays
!
              case (NF_SHORT)
                 allocate(mpp_file(unit)%Att(i)%fatt(len))
                 error=NFMPI_GET_ATT_INT2(ncid,NF_GLOBAL,name,i2vals);call netcdf_err(errc
                 if( verbose .and. pe == 0 )write (stdout(),*)  'GLOBAL ATT ',trim(name),'
                 mpp_file(unit)%Att(i)%fatt(1:len)=i2vals(1:len)
              case (NF_INT)
                 allocate(mpp_file(unit)%Att(i)%fatt(len))
                 error=NFMPI_GET_ATT_INT(ncid,NF_GLOBAL,name,ivals);call netcdf_err(error)
                 if( verbose .and. pe == 0 )write (stdout(),*)  'GLOBAL ATT ',trim(name),'
                 mpp_file(unit)%Att(i)%fatt(1:len)=ivals(1:len)
              case (NF_FLOAT)
                 allocate(mpp_file(unit)%Att(i)%fatt(len))
                 error=NFMPI_GET_ATT_REAL(ncid,NF_GLOBAL,name,rvals);call netcdf_err(error
                 mpp_file(unit)%Att(i)%fatt(1:len)=rvals(1:len)
                 if( verbose .and. pe == 0)write (stdout(),*)  'GLOBAL ATT ',trim(name),'
              case (NF_DOUBLE)
                 allocate(mpp_file(unit)%Att(i)%fatt(len))
                 error=NFMPI_GET_ATT_DOUBLE(ncid,NF_GLOBAL,name,r8vals);call netcdf_err(er
                 mpp_file(unit)%Att(i)%fatt(1:len)=r8vals(1:len)
                 if( verbose .and. pe == 0)write (stdout(),*)  'GLOBAL ATT ',trim(name),'
           end select

        enddo
!
! assign dimension name and length
!
        do i=1,ndim
           error = NFMPI_INQ_DIM(ncid,i,name,idim);call netcdf_err(error)
           len = idim
           Axis(i)%name = name
           Axis(i)%len = len
        enddo

        nvar=0
        do i=1, nvar_total
           error=NFMPI_INQ_VAR(ncid,i,name,type,nvdims,dimids,nvatts);call netcdf_err(errc
           isdim=.false.
           do j=1,ndim
              if( trim(lowercase(name)).EQ.trim(lowercase(Axis(j)%name)) )isdim=.true.
           enddo
           if (.not.isdim) nvar=nvar+1
        enddo
        mpp_file(unit)%nvar = nvar
        allocate(mpp_file(unit)%Var(nvar))

        do i=1,nvar
           mpp_file(unit)%Var(i) = default_field
        enddo

!
! assign dimension info
!
        do i=1, nvar_total
           error=NFMPI_INQ_VAR(ncid,i,name,type,nvdims,dimids,nvatts);call netcdf_err(errc
           isdim=.false.
           do j=1,ndim
              if( trim(lowercase(name)).EQ.trim(lowercase(Axis(j)%name)) )isdim=.true.
           enddo

           if( isdim )then
              error=NFMPI_INQ_DIMID(ncid,name,dimid);call netcdf_err(error)
              Axis(dimid)%type = type
              Axis(dimid)%did = dimid
              Axis(dimid)%id = i
              Axis(dimid)%natt = nvatts
              ! get axis values
              if( i.NE.mpp_file(unit)%id )then   ! non-record dims
                 select case (type)
                 case (NF_INT)
                    len=Axis(dimid)%len
                    allocate(Axis(dimid)%data(len))
                    error = NFMPI_GET_VAR_INT_ALL(ncid,i,ivals);call netcdf_err(error)
                    Axis(dimid)%data(1:len)=ivals(1:len)
                 case (NF_FLOAT)
                    len=Axis(dimid)%len
                    allocate(Axis(dimid)%data(len))
                    error = NFMPI_GET_VAR_REAL_ALL(ncid,i,rvals);call netcdf_err(error)
                    Axis(dimid)%data(1:len)=rvals(1:len)
                 case (NF_DOUBLE)
                    len=Axis(dimid)%len
                    allocate(Axis(dimid)%data(len))
                    error = NFMPI_GET_VAR_DOUBLE_ALL(ncid,i,r8vals);call netcdf_err(error)
                    Axis(dimid)%data(1:len) = r8vals(1:len)
                 case (NF_CHAR) !RV,bundle
                    len=Axis(dimid)%len !RV,bundle
                    allocate(Axis(dimid)%cdata(len)) !RV,bundle
                    error = NFMPI_GET_VAR_TEXT_ALL(ncid,i,Axis(dimid)%cdata) !RV,bundle
                    print*,'cdata',Axis(dimid)%cdata !RV,bundle
                    call netcdf_err(error) !RV,bundle
                 case default
                    call mpp_error( FATAL, 'Invalid data type for dimension' )
                 end select
             else
                 len = mpp_file(unit)%time_level
                 allocate(mpp_file(unit)%time_values(len))
                 select case (type)
                 case (NF_FLOAT)
                    error = NFMPI_GET_VAR_REAL_ALL(ncid,i,rvals);call netcdf_err(error)
                    mpp_file(unit)%time_values(1:len) = rvals(1:len)
```

```
         case (NF_DOUBLE)
            error = NFMPI_GET_VAR_DOUBLE_ALL(ncid,i,r8vals);call netcdf_err(error)
            mpp_file(unit)%time_values(1:len) = r8vals(1:len)
         case default
            call mpp_error( FATAL, 'Invalid data type for dimension' )
         end select
      endif
      ! assign dimension atts
      if( nvatts.GT.0 )allocate(Axis(dimid)%Att(nvatts))

      do j=1,nvatts
         Axis(dimid)%Att(j) = default_att
      enddo

      do j=1,nvatts
         error=NFMPI_INQ_ATTNAME(ncid,i,j,attname);call netcdf_err(error)
         error=NFMPI_INQ_ATT(ncid,i,trim(attname),type,idim);call netcdf_err(error
         len = idim

         Axis(dimid)%Att(j)%name = trim(attname)
         Axis(dimid)%Att(j)%type = type
         Axis(dimid)%Att(j)%len = len

         select case (type)
         case (NF_CHAR)
            if (len.gt.512) call mpp_error(FATAL,'DIM ATT too long')
            error=NFMPI_GET_ATT_TEXT(ncid,i,trim(attname),Axis(dimid)%Att(j)%catt)
            if( verbose .and. pe == 0 ) &
                 write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
            ! store integers in float arrays
            ! assume dimension data not packed
         case (NF_SHORT)
            allocate(Axis(dimid)%Att(j)%fatt(len))
            error=NFMPI_GET_ATT_INT2(ncid,i,trim(attname),i2vals);call netcdf_err(
            Axis(dimid)%Att(j)%fatt(1:len)=i2vals(1:len)
            if( verbose .and. pe == 0  ) &
                 write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
         case (NF_INT)
            allocate(Axis(dimid)%Att(j)%fatt(len))
            error=NFMPI_GET_ATT_INT(ncid,i,trim(attname),ivals);call netcdf_err(er
            Axis(dimid)%Att(j)%fatt(1:len)=ivals(1:len)
            if( verbose .and. pe == 0  ) &
                 write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
         case (NF_FLOAT)
            allocate(Axis(dimid)%Att(j)%fatt(len))
            error=NFMPI_GET_ATT_REAL(ncid,i,trim(attname),rvals);call netcdf_err(e
            Axis(dimid)%Att(j)%fatt(1:len)=rvals(1:len)
            if( verbose  .and. pe == 0 ) &
                 write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
         case (NF_DOUBLE)
            allocate(Axis(dimid)%Att(j)%fatt(len))
            error=NFMPI_GET_ATT_DOUBLE(ncid,i,trim(attname),r8vals);call netcdf_er
            Axis(dimid)%Att(j)%fatt(1:len)=r8vals(1:len)
            if( verbose  .and. pe == 0 ) &
                 write (stdout(),*)  'AXIS ',trim(Axis(dimid)%name),' ATT ',trim(a
         case default
            call mpp_error( FATAL, 'Invalid data type for dimension at' )
         end select
         ! assign pre-defined axis attributes
         select case(trim(attname))
         case('long_name')
            Axis(dimid)%longname=Axis(dimid)%Att(j)%catt(1:len)
         case('units')
```

```
                    Axis(dimid)%units=Axis(dimid)%Att(j)%catt(1:len)
                 case('cartesian_axis')
                    Axis(dimid)%cartesian=Axis(dimid)%Att(j)%catt(1:len)
                 case('positive')
                    attval = Axis(dimid)%Att(j)%catt(1:len)
                    if( attval.eq.'down' )then
                       Axis(dimid)%sense=-1
                    else if( attval.eq.'up' )then
                       Axis(dimid)%sense=1
                    endif
                 end select

              enddo
              ! store axis info in filetype
              mpp_file(unit)%Axis(dimid) = Axis(dimid)
           endif
        enddo
! assign variable info
        nv = 0
        do i=1, nvar_total
           error=NFMPI_INQ_VAR(ncid,i,name,type,nvdims,dimids,nvatts);call netcdf_err(errc
!
! is this a dimension variable?
!
           isdim=.false.
           do j=1,ndim
              if( trim(lowercase(name)).EQ.trim(lowercase(Axis(j)%name)) )isdim=.true.
           enddo

           if( .not.isdim )then
! for non-dimension variables
              nv=nv+1; if( nv.GT.mpp_file(unit)%nvar )call mpp_error( FATAL, 'variable inc
              mpp_file(unit)%Var(nv)%type = type
              mpp_file(unit)%Var(nv)%id = i
              mpp_file(unit)%Var(nv)%name = name
              mpp_file(unit)%Var(nv)%natt = nvatts
! determine packing attribute based on NetCDF variable type
             select case (type)
             case(NF_SHORT)
                 mpp_file(unit)%Var(nv)%pack = 4
             case(NF_FLOAT)
                 mpp_file(unit)%Var(nv)%pack = 2
             case(NF_DOUBLE)
                 mpp_file(unit)%Var(nv)%pack = 1
             case (NF_INT)
                 mpp_file(unit)%Var(nv)%pack = 2
             case default
                   call mpp_error( FATAL, 'Invalid variable type in NetCDF file' )
             end select
! assign dimension ids
              mpp_file(unit)%Var(nv)%ndim = nvdims
              allocate(mpp_file(unit)%Var(nv)%axes(nvdims))
              do j=1,nvdims
                 mpp_file(unit)%Var(nv)%axes(j) = Axis(dimids(j))
              enddo
              allocate(mpp_file(unit)%Var(nv)%size(nvdims))

              do j=1,nvdims
                 if( dimids(j).eq.mpp_file(unit)%recdimid )then
                    mpp_file(unit)%Var(nv)%time_axis_index = dimids(j)
                    mpp_file(unit)%Var(nv)%size(j)=1    ! dimid length set to 1 here for c
                 else
                    mpp_file(unit)%Var(nv)%size(j)=Axis(dimids(j))%len
                 endif
              enddo
! assign variable atts
              if( nvatts.GT.0 )allocate(mpp_file(unit)%Var(nv)%Att(nvatts))

              do j=1,nvatts
                 mpp_file(unit)%Var(nv)%Att(j) = default_att
              enddo

              do j=1,nvatts
                 error=NFMPI_INQ_ATTNAME(ncid,i,j,attname);call netcdf_err(error)
                 error=NFMPI_INQ_ATT(ncid,i,attname,type,idim);call netcdf_err(error)
                 len = idim
                 mpp_file(unit)%Var(nv)%Att(j)%name = trim(attname)
                 mpp_file(unit)%Var(nv)%Att(j)%type = type
                 mpp_file(unit)%Var(nv)%Att(j)%len = len

                 select case (type)
                   case (NF_CHAR)
                     if (len.gt.512) call mpp_error(FATAL,'VAR ATT too long')
                     error=NFMPI_GET_ATT_TEXT(ncid,i,trim(attname),mpp_file(unit)%Var(nv)%
                     if (verbose .and. pe == 0 )&
                           write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_fil
! store integers as float internally
                   case (NF_SHORT)
                     allocate(mpp_file(unit)%Var(nv)%Att(j)%fatt(len))
                     error=NFMPI_GET_ATT_INT2(ncid,i,trim(attname),i2vals);call netcdf_err
                     mpp_file(unit)%Var(nv)%Att(j)%fatt(1:len)= i2vals(1:len)
                     if( verbose  .and. pe == 0 )&
                          write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_file
                   case (NF_INT)
                     allocate(mpp_file(unit)%Var(nv)%Att(j)%fatt(len))
                     error=NFMPI_GET_ATT_INT(ncid,i,trim(attname),ivals);call netcdf_err(e
                     mpp_file(unit)%Var(nv)%Att(j)%fatt(1:len)=ivals(1:len)
                     if( verbose .and. pe == 0  )&
                          write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_file
                   case (NF_FLOAT)
                     allocate(mpp_file(unit)%Var(nv)%Att(j)%fatt(len))
                     error=NFMPI_GET_ATT_REAL(ncid,i,trim(attname),rvals);call netcdf_err(
                     mpp_file(unit)%Var(nv)%Att(j)%fatt(1:len)=rvals(1:len)
                     if( verbose  .and. pe == 0 )&
                          write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_file
                   case (NF_DOUBLE)
                     allocate(mpp_file(unit)%Var(nv)%Att(j)%fatt(len))
                     error=NFMPI_GET_ATT_DOUBLE(ncid,i,trim(attname),r8vals);call netcdf_e
                     mpp_file(unit)%Var(nv)%Att(j)%fatt(1:len)=r8vals(1:len)
                     if( verbose .and. pe == 0  ) &
                          write (stdout(),*)  'Var ',nv,' ATT ',trim(attname),' ',mpp_file
                   case default
                        call mpp_error( FATAL, 'Invalid data type for variable att' )
                 end select
! assign pre-defined field attributes
                 select case (trim(attname))
                    case ('long_name')
                      mpp_file(unit)%Var(nv)%longname=mpp_file(unit)%Var(nv)%Att(j)%catt(1
                    case('units')
                      mpp_file(unit)%Var(nv)%units=mpp_file(unit)%Var(nv)%Att(j)%catt(1:le
                    case('scale_factor')
                       mpp_file(unit)%Var(nv)%scale=mpp_file(unit)%Var(nv)%Att(j)%fatt(1)
                    case('missing')
                       mpp_file(unit)%Var(nv)%missing=mpp_file(unit)%Var(nv)%Att(j)%fatt(1
                    case('add_offset')
                       mpp_file(unit)%Var(nv)%add=mpp_file(unit)%Var(nv)%Att(j)%fatt(1)
                    case('valid_range')
                       mpp_file(unit)%Var(nv)%min=mpp_file(unit)%Var(nv)%Att(j)%fatt(1)
                       mpp_file(unit)%Var(nv)%max=mpp_file(unit)%Var(nv)%Att(j)%fatt(2)
                 end select
              enddo
           endif
        enddo   ! end variable loop
      else
        call mpp_error( FATAL,  'MPP READ CURRENTLY DOES NOT SUPPORT NON-NETCDF' )
      endif

      mpp_file(unit)%initialized = .TRUE.
#else
      call mpp_error( FATAL, 'MPP_READ currently requires use_netCDF option' )
#endif
      return
    end subroutine mpp_read_meta


    subroutine mpp_get_info( unit, ndim, nvar, natt, ntime )

      integer, intent(in) :: unit
      integer, intent(out) :: ndim, nvar, natt, ntime


      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_INFO: must first cal
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_INFO: invalid unit r

      ndim = mpp_file(unit)%ndim
      nvar = mpp_file(unit)%nvar
      natt = mpp_file(unit)%natt
      ntime = mpp_file(unit)%time_level

      return

    end subroutine mpp_get_info


    subroutine mpp_get_global_atts( unit, global_atts )
!
!  copy global file attributes for use by user
!
!  global_atts is an attribute type which is allocated from the
!  calling routine

      integer,       intent(in)    :: unit
      type(atttype), intent(inout) :: global_atts(:)
      integer :: natt,i

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_INFO: must first cal
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_INFO: invalid unit r

      if (size(global_atts).lt.mpp_file(unit)%natt) &
      call mpp_error(FATAL, 'MPP_GET_ATTS: atttype not dimensioned properly in calling rou

      natt = mpp_file(unit)%natt
      global_atts = default_att

      do i=1,natt
         global_atts(i) = mpp_file(unit)%Att(i)
      enddo

      return
```

```
end subroutine mpp_get_global_atts

subroutine mpp_get_field_atts( field, name, units, longname, min, max, missing, ndim, s

  type(fieldtype), intent(in) :: field
  character(len=*), intent(out) , optional :: name, units
  character(len=*), intent(out), optional :: longname
  real,intent(out), optional :: min,max,missing
  integer, intent(out), optional :: ndim
  integer, intent(out), dimension(:), optional :: siz

  type(atttype), intent(out), optional, dimension(:) :: atts
  type(axistype), intent(out), optional, dimension(:) :: axes

  integer :: n,m

  if (PRESENT(name)) name = field%name
  if (PRESENT(units)) units = field%units
  if (PRESENT(longname)) longname = field%longname
  if (PRESENT(min)) min = field%min
  if (PRESENT(max)) max = field%max
  if (PRESENT(missing)) missing = field%missing
  if (PRESENT(ndim)) ndim = field%ndim
  if (PRESENT(atts)) then
     atts = default_att
     n = size(atts);m=size(field%Att)
     if (n.LT.m) call mpp_error(FATAL,'attribute array not large enough in mpp_get_fiel
     atts(1:m) = field%Att(1:m)
  end if
  if (PRESENT(axes)) then
     axes = default_axis
     n = size(axes);m=field%ndim
     if (n.LT.m) call mpp_error(FATAL,'axis array not large enough in mpp_get_field_att
     axes(1:m) = field%axes(1:m)
  end if
  if (PRESENT(siz)) then
     siz = -1
     n = size(siz);m=field%ndim
     if (n.LT.m) call mpp_error(FATAL,'size array not large enough in mpp_get_field_att
     siz(1:m) = field%size(1:m)
  end if
  return
end subroutine mpp_get_field_atts

subroutine mpp_get_axis_atts( axis, name, units, longname, cartesian, sense, len, natts

  type(axistype), intent(in) :: axis
  character(len=*), intent(out) , optional :: name, units
  character(len=*), intent(out), optional :: longname, cartesian
  integer,intent(out), optional :: sense, len , natts
  type(atttype), intent(out), optional, dimension(:) :: atts

  integer :: n,m

  if (PRESENT(name)) name = axis%name
  if (PRESENT(units)) units = axis%units
  if (PRESENT(longname)) longname = axis%longname
  if (PRESENT(cartesian)) cartesian = axis%cartesian
  if (PRESENT(sense)) sense = axis%sense
  if (PRESENT(len)) len = axis%len
  if (PRESENT(atts)) then
     atts = default_att
     n = size(atts);m=size(axis%Att)
```

```
        if (n.LT.m) call mpp_error(FATAL,'attribute array not large enough in mpp_get_fiel
        atts(1:m) = axis%Att(1:m)
     end if
     if (PRESENT(natts)) natts = size(axis%Att)

     return
   end subroutine mpp_get_axis_atts


    subroutine mpp_get_fields( unit, variables )
!
!  copy variable information from file (excluding data)
!  global_atts is an attribute type which is allocated from the
!  calling routine
!
      integer,         intent(in)    :: unit
      type(fieldtype), intent(inout) :: variables(:)

      integer :: nvar,i

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_FIELDS: must first c
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_FIELDS: invalid unit

      if (size(variables).ne.mpp_file(unit)%nvar) &
      call mpp_error(FATAL, 'MPP_GET_FIELDS: fieldtype not dimensioned properly in calling

      nvar = mpp_file(unit)%nvar

      do i=1,nvar
         variables(i) = mpp_file(unit)%Var(i)
      enddo

      return
   end subroutine mpp_get_fields

    subroutine mpp_get_axes( unit, axes, time_axis )
!
!  copy variable information from file (excluding data)
!  global_atts is an attribute type which is allocated from the
!  calling routine
!
      integer, intent(in) :: unit
      type(axistype), intent(out) :: axes(:)
      type(axistype), intent(out), optional :: time_axis
      character(len=128) :: name
      logical :: save
      integer :: ndim,i, nvar, j, num_dims, k

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_AXES: must first cal
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_AXES: invalid unit r

      if (size(axes).ne.mpp_file(unit)%ndim) &
      call mpp_error(FATAL, 'MPP_GET_AXES: axistype not dimensioned properly in calling rc


      if (PRESENT(time_axis)) time_axis = default_axis
      ndim = mpp_file(unit)%ndim
      do i=1,ndim
        if (ASSOCIATED(mpp_file(unit)%Axis(i)%data)) then
           axes(i)=mpp_file(unit)%Axis(i)
       else
           axes(i)=mpp_file(unit)%Axis(i)
           if (PRESENT(time_axis)) time_axis = mpp_file(unit)%Axis(i)
        endif
      enddo

      return
   end subroutine mpp_get_axes

   subroutine mpp_get_times( unit, time_values )
!
!  copy time information from file and convert to time_type
!
      integer, intent(in) :: unit
      real(DOUBLE_KIND), intent(inout) :: time_values(:)

      integer :: ntime,i

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_TIMES: must first ca
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_TIMES: invalid unit

      if (size(time_values).ne.mpp_file(unit)%time_level) &
      call mpp_error(FATAL, 'MPP_GET_TIMES: time_values not dimensioned properly in callin

      ntime = mpp_file(unit)%time_level

      do i=1,ntime
         time_values(i) = mpp_file(unit)%time_values(i)
      enddo



      return
   end subroutine mpp_get_times

   function mpp_get_field_index(fields,fieldname)

     type(fieldtype), dimension(:) :: fields
     character(len=*) :: fieldname
     integer :: mpp_get_field_index

     integer :: n

     mpp_get_field_index = -1

     do n=1,size(fields)
        if (lowercase(fields(n)%name) == lowercase(fieldname)) then
           mpp_get_field_index = n
           exit
        endif
     enddo

     return
   end function mpp_get_field_index

   function mpp_get_field_size(field)

     type(fieldtype) :: field
     integer :: mpp_get_field_size(4)

     integer :: n

     mpp_get_field_size = -1

     mpp_get_field_size(1) = field%size(1)
     mpp_get_field_size(2) = field%size(2)
     mpp_get_field_size(3) = field%size(3)
     mpp_get_field_size(4) = field%size(4)

     return
   end function mpp_get_field_size


   subroutine mpp_get_axis_data( axis, data )

     type(axistype), intent(in) :: axis
     real, dimension(:), intent(out) :: data


     if (size(data).lt.axis%len) call mpp_error(FATAL,'MPP_GET_AXIS_DATA: data array not l
     if (.NOT.ASSOCIATED(axis%data)) then
        call mpp_error(NOTE,'MPP_GET_AXIS_DATA: use mpp_get_times for record dims')
        data = 0.
     else
        data(1:axis%len) = axis%data
     endif

     return
   end subroutine mpp_get_axis_data


   function mpp_get_recdimid(unit)
!
      integer, intent(in) :: unit
      integer  :: mpp_get_recdimid


      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_RECDIMID: must first
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_RECDIMID: invalid ur

      mpp_get_recdimid = mpp_file(unit)%recdimid

      return
   end function mpp_get_recdimid

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!         mpp_get_iospec, mpp_flush: OS-dependent calls                !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    subroutine mpp_flush(unit)
!flush the output on a unit, syncing with disk
      integer, intent(in) :: unit

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_FLUSH: must first call m
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_FLUSH: invalid unit numb
      if( .NOT.mpp_file(unit)%initialized )call mpp_error( FATAL, 'MPP_FLUSH: cannot flush
      if( mpp_file(unit)%threading.EQ.MPP_SINGLE .AND. pe.NE.mpp_root_pe() )return

      if( mpp_file(unit)%format.EQ.MPP_NETCDF )then
#ifdef use_netCDF
          error = NFMPI_SYNC(mpp_file(unit)%ncid); call netcdf_err(error)
#endif
      else
          call FLUSH(unit)
      end if
      return
    end subroutine mpp_flush
    subroutine mpp_get_iospec( unit, iospec )
      integer, intent(in) :: unit
      character(len=*), intent(out) :: iospec

      if( .NOT.module_is_initialized )call mpp_error( FATAL, 'MPP_GET_IOSPEC: must first c
      if( .NOT.mpp_file(unit)%opened )call mpp_error( FATAL, 'MPP_GET_IOSPEC: invalid unit
#ifdef SGICRAY
!currently will write to stdout: don't know how to trap and return as string to iospec
      call ASSIGN( 'assign -V f:'//trim(mpp_file(unit)%name), error )
#endif
      return
    end subroutine mpp_get_iospec

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!         netCDF-specific routines: mpp_get_id, netcdf_error         !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    function mpp_get_ncid(unit)
      integer :: mpp_get_ncid
      integer, intent(in) :: unit

      mpp_get_ncid = mpp_file(unit)%ncid
      return
    end function mpp_get_ncid

    function mpp_get_axis_id(axis)
      integer mpp_get_axis_id
      type(axistype), intent(in) :: axis
      mpp_get_axis_id = axis%id
      return
    end function mpp_get_axis_id

    function mpp_get_field_id(field)
      integer mpp_get_field_id
      type(fieldtype), intent(in) :: field
      mpp_get_field_id = field%id
      return
    end function mpp_get_field_id

    subroutine netcdf_err(err)
      integer, intent(in) :: err
      character(len=80) :: errmsg
      integer :: unit

#ifdef use_netCDF
      if( err.EQ.NF_NOERR )return
      errmsg = NFMPI_STRERROR(err)
      call mpp_io_exit()        !make sure you close all open files
      call mpp_error( FATAL, 'NETCDF ERROR: '//trim(errmsg) )
#endif
      return
    end subroutine netcdf_err

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!       minor routines: mpp_get_unit_range, mpp_set_unit_range        !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    subroutine mpp_get_unit_range( unit_begin_out, unit_end_out )
```

```
      integer, intent(out) ::      unit_begin_out, unit_end_out

      unit_begin_out = unit_begin; unit_end_out = unit_end
      return
    end subroutine mpp_get_unit_range

    subroutine mpp_set_unit_range( unit_begin_in, unit_end_in )
      integer, intent(in) ::       unit_begin_in, unit_end_in

      if( unit_begin_in.GT.unit_end_in )call mpp_error( FATAL, 'MPP_SET_UNIT_RANGE: unit_b
      if( unit_begin_in.LT.0           )call mpp_error( FATAL, 'MPP_SET_UNIT_RANGE: unit_b
      if( unit_end_in  .GT.maxunits    )call mpp_error( FATAL, 'MPP_SET_UNIT_RANGE: unit_e
      unit_begin = unit_begin_in; unit_end = unit_end_in
      return
    end subroutine mpp_set_unit_range

    subroutine mpp_modify_axis_meta( axis, name, units, longname, cartesian, data )

      type(axistype), intent(inout) :: axis
      character(len=*), intent(in), optional :: name, units, longname, cartesian
      real, dimension(:), intent(in), optional :: data

      if (PRESENT(name)) axis%name = trim(name)
      if (PRESENT(units)) axis%units = trim(units)
      if (PRESENT(longname)) axis%longname = trim(longname)
      if (PRESENT(cartesian)) axis%cartesian = trim(cartesian)
      if (PRESENT(data)) then
         axis%len = size(data)
         if (ASSOCIATED(axis%data)) deallocate(axis%data)
         allocate(axis%data(axis%len))
         axis%data = data
      endif

      return
    end subroutine mpp_modify_axis_meta

    subroutine mpp_modify_field_meta( field, name, units, longname, min, max, missing, axe

      type(fieldtype), intent(inout) :: field
      character(len=*), intent(in), optional :: name, units, longname
      real, intent(in), optional :: min, max, missing
      type(axistype), dimension(:), intent(inout), optional :: axes

      if (PRESENT(name)) field%name = trim(name)
      if (PRESENT(units)) field%units = trim(units)
      if (PRESENT(longname)) field%longname = trim(longname)
      if (PRESENT(min)) field%min = min
      if (PRESENT(max)) field%max = max
      if (PRESENT(missing)) field%missing = missing
!      if (PRESENT(axes)) then
!         axis%len = size(data)
!         deallocate(axis%data)
!         allocate(axis%data(axis%len))
!         axis%data = data
!      endif

      return
    end subroutine mpp_modify_field_meta

    function lowercase (cs)
    character(len=*), intent(in) :: cs
#ifdef __crayx1
    character(len=128)       :: lowercase
    integer :: ido, i
    integer, save :: &
!
    & down_map_ascii(0:127)=(/ (ido, ido=0,64), (ido+32, ido=65,90), (ido, ido=91,127) /)
        do i = 1, len( cs )
            lowercase(i:i) = achar(down_map_ascii(iachar(cs(i:i))))
        end do
#else
      character(len=len(cs))       :: lowercase
      character :: ca(len(cs))

      integer, parameter :: co=iachar('a')-iachar('A') ! case offset

      ca = transfer(cs,"x",len(cs))
      where (ca >= "A" .and. ca <= "Z") ca = achar(iachar(ca)+co)
          lowercase = transfer(ca,cs)
#endif
    end function lowercase

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                      !
!       minor routines: mpp_nullify_axistype,                          !
!                       mpp_nullify_axistype_array                     !
!                                                                      !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    subroutine mpp_nullify_axistype(axis)
      type(axistype), intent(inout) :: axis

      Nullify(axis%data)
      Nullify(axis%cdata)
      Nullify(axis%Att)
    end subroutine mpp_nullify_axistype

    subroutine mpp_nullify_axistype_array(axis)
      type(axistype), intent(inout), dimension(:) :: axis
      integer :: i

      do i=1, size(axis)
        Nullify(axis(i)%data)
        Nullify(axis(i)%cdata)
        Nullify(axis(i)%Att)
      enddo
    end subroutine mpp_nullify_axistype_array

end module mpp_io_mod_oa


#endif
#endif
```