In-code documentation for CVMix

Many contributors from GFDL, LANL, and NCAR
GFDL, LANL, and NCAR

June 7, 2013

Contents

1 Routine/Function Prologues

1.0.1 cvmix_driver

The stand-alone driver for the CVMix package. This driver reads in the cvmix_nml namelist to determine what type of mixing has been requested, and also reads in mixing-specific parameters from a mixingtype_nml namelist.

REVISION HISTORY:

```
    SVN $Id: cvmix_driver.F90 147 2013-06-06 22:52:17Z mike.levy.work@gmail.com $
    SVN $URL: https://cvmix.googlecode.com/svn/trunk/src/cvmix_driver.F90 $
```

INTERFACE:

```
 Program cvmix_driver
```

USES:

```
   use cvmix_kinds_and_types, only : cvmix_r8, &
                                     cvmix_strlen
```

1.0.2 cvmix_BL_pointer_driver

A routine to test the Bryan-Lewis implementation of static background mixing. Inputs are BL coefficients in two columns, one that represents tropical latitudes and one that represents subtropical latitudes. All memory is declared in the driver, and the CVMix data type points to the local variables.

REVISION HISTORY:

```
    SVN:$Id: cvmix_bgrnd_BL_pointer.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
    SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_bgrnd_BL_pointer.F90 $
```

INTERFACE:

```
 Subroutine cvmix_BL_pointer_driver(nlev, ocn_depth)
```

USES:

```
   use cvmix_kinds_and_types, only : cvmix_r8,                &
                                     cvmix_data_type,         &
                                     cvmix_global_params_type
   use cvmix_background,      only : cvmix_init_bkgnd,        &
                                     cvmix_coeffs_bkgnd,      &
                                     cvmix_bkgnd_params_type
   use cvmix_put_get,         only : cvmix_put
   use cvmix_io,              only : cvmix_io_open,           &
                                     cvmix_output_write,      &
 #ifdef _NETCDF
                                     cvmix_output_write_att,  &
 #endif
                                     cvmix_io_close

   Implicit None
```

INPUT PARAMETERS:

```
   integer, intent(in)        :: nlev        ! number of levels for column
   real(cvmix_r8), intent(in) :: ocn_depth   ! Depth of ocn
```

1.0.3 cvmix_BL_memcopy_driver

A routine to test the Bryan-Lewis implementation of static background mixing. Inputs are BL coefficients in two columns, one that represents tropical latitudes and one that represents subtropical latitudes. All memory is declared in the driver and then copied into the CVMix data structures.

REVISION HISTORY:

```
    SVN:$Id: cvmix_bgrnd_BL_memcopy.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
    SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_bgrnd_BL_memcopy.F90 $
```

INTERFACE:

```
 Subroutine cvmix_BL_memcopy_driver(nlev, ocn_depth)
```

USES:

```
   use cvmix_kinds_and_types, only : cvmix_r8,                 &
                                     cvmix_data_type,          &
                                     cvmix_global_params_type
   use cvmix_background,      only : cvmix_init_bkgnd,         &
                                     cvmix_coeffs_bkgnd,       &
                                     cvmix_bkgnd_params_type
   use cvmix_put_get,         only : cvmix_put
   use cvmix_io,              only : cvmix_io_open,            &
                                     cvmix_output_write,       &
 #ifdef _NETCDF
                                     cvmix_output_write_att,   &
 #endif
                                     cvmix_io_close

   Implicit None
```

INPUT PARAMETERS:

```
   integer, intent(in)        :: nlev        ! number of levels for column
   real(cvmix_r8), intent(in) :: ocn_depth   ! Depth of ocn
```

1.0.4 cvmix_shear_driver

A routine to test the Large, et al., implementation of shear mixing. Inputs are the coefficients used in Equation (28) of the paper. The diffusivity coefficient is output from a single column to allow recreation of the paper's Figure 3. Note that here each "level" of the column denotes a different local gradient Richardson number rather than a physical ocean level. All memory is declared in the driver, and the CVMix data type points to the local variables.

REVISION HISTORY:

```
SVN:$Id: cvmix_shear_KPP.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_shear_KPP.F90 $
```

INTERFACE:

```
Subroutine cvmix_shear_driver(nlev)
```

USES:

```
  use cvmix_kinds_and_types, only : one,                    &
                                    cvmix_r8,               &
                                    cvmix_data_type,        &
                                    cvmix_global_params_type
  use cvmix_shear,           only : cvmix_init_shear,       &
                                    cvmix_coeffs_shear,     &
                                    cvmix_shear_params_type
  use cvmix_put_get,         only : cvmix_put
  use cvmix_io,              only : cvmix_io_open,          &
                                    cvmix_output_write,     &
#ifdef _NETCDF
                                    cvmix_output_write_att, &
#endif
                                    cvmix_io_close

  Implicit None
```

INPUT PARAMETERS:

```
  integer, intent(in) :: nlev      ! number of Ri points to sample
```

1.0.5 cvmix_tidal_driver

A routine to test the Simmons implementation of tidal mixing.

REVISION HISTORY:

```
    SVN:$Id: cvmix_tidal_Simmons.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
    SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_tidal_Simmons.F90 $
```

INTERFACE:

```
 Subroutine cvmix_tidal_driver()
```

USES:

```
   use cvmix_kinds_and_types, only : cvmix_r8,                  &
                                     cvmix_strlen,              &
                                     cvmix_data_type,           &
                                     cvmix_global_params_type
   use cvmix_tidal,           only : cvmix_init_tidal,          &
                                     cvmix_coeffs_tidal,        &
                                     cvmix_tidal_params_type,   &
                                     cvmix_get_tidal_str,       &
                                     cvmix_get_tidal_real
   use cvmix_put_get,         only : cvmix_put
   use cvmix_io,              only : cvmix_io_open,             &
                                     cvmix_input_read,          &
 #ifdef _NETCDF
                                     cvmix_input_get_netcdf_dim, &
 #endif
                                     cvmix_output_write,        &
                                     cvmix_output_write_att,    &
                                     cvmix_io_close

   Implicit None
```

1.0.6 cvmix_ddiff_driver

A routine to test the double diffusion mixing module.

REVISION HISTORY:

```
    SVN:$Id: cvmix_ddiff_drv.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
    SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_ddiff_drv.F90 $
```

INTERFACE:

```
 Subroutine cvmix_ddiff_driver(nlev)
```

USES:

```
   use cvmix_kinds_and_types, only : one,                    &
                                     cvmix_r8,               &
                                     cvmix_data_type
   use cvmix_ddiff,           only : cvmix_init_ddiff,       &
                                     cvmix_coeffs_ddiff,     &
                                     cvmix_get_ddiff_real,   &
                                     cvmix_ddiff_params_type
   use cvmix_put_get,         only : cvmix_put
   use cvmix_io,              only : cvmix_io_open,          &
                                     cvmix_output_write,     &
 #ifdef _NETCDF
                                     cvmix_output_write_att, &
 #endif
                                     cvmix_io_close

   Implicit None
```

INPUT PARAMETERS:

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

1.1 Fortran: Module Interface cvmix_io

This module contains routines to read CVmix variables from data files or output CVmix variables to data files. Currently only ascii and netCDF output are supported, as well as netCDF input, but the plan is to also include plain binary input / output as well.

REVISION HISTORY:

```
    SVN:$Id: cvmix_io.F90 147 2013-06-06 22:52:17Z mike.levy.work@gmail.com $
    SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/cvmix_io.F90 $
```

USES:

```
    use cvmix_kinds_and_types, only : cvmix_data_type, &
                                      cvmix_r8,        &
                                      cvmix_strlen
#ifdef _NETCDF
    use netcdf
#endif
```

PUBLIC MEMBER FUNCTIONS:

```
   public :: cvmix_io_open
   public :: cvmix_input_read
#ifdef _NETCDF
   public :: cvmix_input_get_netcdf_dim
#endif
   public :: cvmix_output_write
   public :: cvmix_io_close
   public :: cvmix_io_close_all
   public :: print_open_files
   public :: cvmix_output_write_att

   interface cvmix_input_read
     module procedure cvmix_input_read_1d_double
     module procedure cvmix_input_read_2d_integer
     module procedure cvmix_input_read_2d_double
     module procedure cvmix_input_read_3d_double
   end interface

   interface cvmix_output_write
     module procedure cvmix_output_write_single_col
     module procedure cvmix_output_write_multi_col
     module procedure cvmix_output_write_3d_double
   end interface

   interface cvmix_output_write_att
```

```
    module procedure cvmix_output_write_att_string
  end interface
```

DEFINED PARAMETERS:

```
  integer, parameter :: ASCII_FILE_TYPE  = 1
  integer, parameter :: BIN_FILE_TYPE    = 2
  integer, parameter :: NETCDF_FILE_TYPE = 3
  integer, parameter :: FILE_NOT_FOUND   = 404

  ! Probably not the best technique, but going to use a linked list to keep
  ! track of what files are open / what format they are (ascii, bin, or nc)
  type :: cvmix_file_entry
    integer :: file_id
    integer :: file_type
    character(len=cvmix_strlen) :: file_name
    type(cvmix_file_entry), pointer :: prev
    type(cvmix_file_entry), pointer :: next
  end type

  type(cvmix_file_entry), allocatable, target :: file_database(:)
```

1.1.1 cvmix_io_open

INTERFACE:

```
  subroutine cvmix_io_open(file_id, file_name, file_format, read_only)
```

DESCRIPTION:

Routine to open a file for reading and / or writing. The goal is to support plain text (currently working for writing only), netCDF (working for both reading and writing), and plain binary (not supported at this time). Besides opening the file, this routine also adds an entry to file_database, a linked list that keeps track of what files are open and what type of file each identifier refers to. So it will be possible to output the same data in ascii and netCDF, for example.

USES:

```
   Only those used by entire module.
```

INPUT PARAMETERS:

```
    character(len=*),  intent(in) :: file_name, file_format
    logical, optional, intent(in) :: read_only
```

OUTPUT PARAMETERS:

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

LOCAL VARIABLES:

```
    type(cvmix_file_entry), pointer :: file_index
    logical                         :: readonly
```

1.1.2 cvmix_input_read_1d_double

INTERFACE:

```
  subroutine cvmix_input_read_1d_double(file_id, var_name, local_copy)
```

DESCRIPTION:

Routine to read the requested 1D variable from a netcdf file and save it to a local array (file must be opened using cvmix_io_open with the optional argument readonly = .true.). Called with cvmix_input_read (see interface in PUBLIC MEMBER FUNCTIONS above). At this time, only works with netcdf files.

USES:

```
   Only those used by entire module.
```

INPUT PARAMETERS:

```
    integer,          intent(in)  :: file_id
    character(len=*), intent(in)  :: var_name
    real(cvmix_r8), dimension(:), intent(out) :: local_copy
```

LOCAL VARIABLES:

```
    logical :: lerr_in_read
#ifdef _NETCDF
    integer :: varid, ndims, xtype
    integer :: dims1, dims2
    integer, dimension(1) :: dims
#endif
```

1.1.3 cvmix_input_read_2d_integer

INTERFACE:

```
  subroutine cvmix_input_read_2d_integer(file_id, var_name, local_copy)
```

DESCRIPTION:

Routine to read the requested 2D variable from a netcdf file and save it to a local array (file must be opened using cvmix_io_open with the optional argument readonly = .true.). Called with cvmix_input_read (see interface in PUBLIC MEMBER FUNCTIONS above). At this time, only works with netcdf files.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     integer,          intent(in)  :: file_id
     character(len=*), intent(in)  :: var_name
     integer, dimension(:,:),  intent(out) :: local_copy
```

LOCAL VARIABLES:

```
     logical :: lerr_in_read
 #ifdef _NETCDF
     integer :: varid, ndims, xtype, i
     integer, dimension(2) :: dims1, dims2
 #endif
```

1.1.4 cvmix_input_read_2d_double

INTERFACE:

```
  subroutine cvmix_input_read_2d_double(file_id, var_name, local_copy)
```

DESCRIPTION:

Routine to read the requested 2D variable from a netcdf file and save it to a local array (file must be opened using cvmix_io_open with the optional argument readonly = .true.). Called with cvmix_input_read (see interface in PUBLIC MEMBER FUNCTIONS above). At this time, only works with netcdf files.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
    integer,          intent(in)  :: file_id
    character(len=*), intent(in)  :: var_name
    real(cvmix_r8), dimension(:,:),  intent(out) :: local_copy
```

LOCAL VARIABLES:

```
    logical :: lerr_in_read
#ifdef _NETCDF
    integer :: varid, i, ndims, xtype
    integer, dimension(2) :: dims1, dims2
#endif
```

1.1.5 cvmix_input_read_3d_double

INTERFACE:

```
  subroutine cvmix_input_read_3d_double(file_id, var_name, local_copy)
```

DESCRIPTION:

Routine to read the requested 2D variable from a netcdf file and save it to a local array (file must be opened using cvmix_io_open with the optional argument readonly = .true.). Called with cvmix_input_read (see interface in PUBLIC MEMBER FUNCTIONS above). At this time, only works with netcdf files.

USES:

```
   Only those used by entire module.
```

INPUT PARAMETERS:

```
    integer,          intent(in)  :: file_id
    character(len=*), intent(in)  :: var_name
    real(cvmix_r8), dimension(:,:,:),  intent(out) :: local_copy
```

LOCAL VARIABLES:

```
    logical :: lerr_in_read
#ifdef _NETCDF
    integer :: varid, i, ndims, xtype
    integer, dimension(3) :: dims1, dims2
#endif
```

1.1.6 cvmix_output_write_single_col

INTERFACE:

```
  subroutine cvmix_output_write_single_col(file_id, CVmix_vars, var_names)
```

DESCRIPTION:

Routine to write the requested variables from a single column to a file (file must be opened using cvmix_io_open to ensure it is written correctly). Called with cvmix_output_write (see interface in PUBLIC MEMBER FUNCTIONS above).

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     integer,                         intent(in) :: file_id
     type(cvmix_data_type),            intent(in) :: CVmix_vars
     character(len=*), dimension(:), intent(in) :: var_names
```

LOCAL VARIABLES:

```
     integer :: kw, var
 #ifdef _NETCDF
     integer                             :: nt, nt_id, nw, nw_id
     integer, dimension(:), allocatable :: var_id
 #endif
```

1.1.7 cvmix_output_write_multi_col

INTERFACE:

```
  subroutine cvmix_output_write_multi_col(file_id, CVmix_vars, var_names)
```

DESCRIPTION:

Routine to write the requested variables from multiple columns to a file (file must be opened using vmix_output_open to ensure it is written correctly). Called with vmix_output_write (see interface in PUBLIC MEMBER FUNCTIONS above).

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
    integer,                               intent(in) :: file_id
    type(cvmix_data_type), dimension(:), intent(in) :: CVmix_vars
    character(len=*),      dimension(:), intent(in) :: var_names
```

LOCAL VARIABLES:

```
    integer :: ncol, nw, icol, kw, var
    logical :: z_err
#ifdef _NETCDF
    integer                                            :: nt, nt_id, nw_id, ncol_id
    integer,             dimension(:),   allocatable :: var_id
    real(kind=cvmix_r8), dimension(:,:), allocatable :: lcl_visc, lcl_diff, lcl_Rrho
#endif
```

1.1.8 cvmix_write_3d_double

INTERFACE:

```
  subroutine cvmix_output_write_3d_double(file_id, var_name, dim_names,       &
                                          field, FillVal)
```

DESCRIPTION:

Routine to write a 3d field to a netcdf file. Called with cvmix_output_ write (see interface in PUBLIC MEMBER FUNCTIONS above).

USES:

```
   Only those used by entire module.
```

INPUT PARAMETERS:

```
    integer,                               intent(in) :: file_id
    character(len=*),                      intent(in) :: var_name
    character(len=*), dimension(3),        intent(in) :: dim_names
    real(cvmix_r8),   dimension(:,:,:),  intent(in) :: field
    real(cvmix_r8), optional,              intent(in) :: FillVal
```

LOCAL VARIABLES:

```
    integer, dimension(3) :: dims
    logical               :: add_fill
#ifdef _NETCDF
    integer, dimension(3) :: dimids
    integer               :: varid, i
#endif
```

1.1.9 cvmix_write_att_string

INTERFACE:

```
subroutine cvmix_output_write_att_string(file_id, att_name, att_val, var_name)
```

DESCRIPTION:

Routine to write an attribute with a string value to a netcdf file. If var_name is omitted, routine writes a global attribute. Called with cvmix_output_write_att (see interface in PUBLIC MEMBER FUNCTIONS above).

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
integer,          intent(in)           :: file_id
character(len=*), intent(in)           :: att_name, att_val
character(len=*), intent(in), optional :: var_name
```

LOCAL VARIABLES:

```
#ifdef _NETCDF
    integer :: varid
    logical :: var_found
#endif
```

1.1.10 cvmix_io_close

INTERFACE:

```
subroutine cvmix_io_close(file_id)
```

DESCRIPTION:

Routine to close a file once all writing has been completed. In addition to closing the file, this routine also deletes its entry in file_database to avoid trying to write to the file in the future.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

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

LOCAL VARIABLES:

```
     type(cvmix_file_entry), pointer :: ifile, file_to_close
     logical                         :: file_found
     integer                         :: file_type
```

1.1.11 cvmix_io_close_all

INTERFACE:

```
   subroutine cvmix_io_close_all
```

DESCRIPTION:

Routine to close all files open (meant to be called prior to an abort)

USES:

```
    Only those used by entire module.
```

LOCAL VARIABLES:

```
     integer :: fid
```

1.1.12 get_file_name

INTERFACE:

```
   function get_file_name(file_id)
```

DESCRIPTION:

Returns the name of the file associated with a given file_id. If the file is not in the database, returns FILE_NOT_FOUND.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
    character(len=cvmix_strlen) :: get_file_name
```

LOCAL VARIABLES:

```
    type(cvmix_file_entry), pointer :: ifile
```

1.1.13 get_file_type

INTERFACE:

```
  function get_file_type(file_id)
```

DESCRIPTION:

Returns the file format (enumerated in DEFINED PARAMETERS section) of a given file. If the file is not in the database, returns FILE_NOT_FOUND.

USES:

```
   Only those used by entire module.
```

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
    integer             :: get_file_type
```

LOCAL VARIABLES:

```
    type(cvmix_file_entry), pointer :: ifile
```

1.1.14 cvmix_input_get_netcdf_dim

INTERFACE:

```
  function cvmix_input_get_netcdf_dim(file_id, dim_name)
```

DESCRIPTION:

Returns the value of the dimension dim_name in the netcdf file file_id. If the dimension does not exist, returns -1.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
integer,          intent(in) :: file_id
character(len=*), intent(in) :: dim_name
```

OUTPUT PARAMETERS:

```
integer                        :: cvmix_input_get_netcdf_dim
```

LOCAL VARIABLES:

```
character(len=cvmix_strlen) :: tmp_name
integer                     :: i, ndim, dimid
```

1.1.15 get_netcdf_varid

INTERFACE:

```
function get_netcdf_varid(file_id, var_name, xtype, ndims)
```

DESCRIPTION:

Returns the varid associated with the variable var_name in the netcdf file file_id. If the variable does not exist, returns -1.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
integer,          intent(in) :: file_id
character(len=*), intent(in) :: var_name
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out) :: xtype, ndims
integer                        :: get_netcdf_varid
```

LOCAL VARIABLES:

```
character(len=cvmix_strlen) :: tmp_name
integer                     :: i, nvar
```

1.2 Fortran: Module Interface cvmix_kinds_and_types

This module contains the declarations for all required vertical mixing data types. It also contains several global parameters used by the cvmix package, such as kind numbers and string lengths.

REVISION HISTORY:

```
    SVN:$Id: cvmix_kinds_and_types.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
    SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_kinds_and_types.F90 $
```

USES:

```
    uses no other modules
```

DEFINED PARAMETERS:

```
   ! Kind Types:
   ! The cvmix package uses double precision for floating point computations.
   integer, parameter, public :: cvmix_r8     = selected_real_kind(15, 307), &
                                 cvmix_strlen = 256

   ! Global parameters:
   ! The constant 1 is used repeatedly in PP and double-diff mixing.
   ! The value for pi is needed for Bryan-Lewis mixing.
   real(cvmix_r8), parameter, public :: one = 1.0_cvmix_r8
   real(cvmix_r8), parameter, public :: cvmix_PI = &
                                        3.14159265358979323846_cvmix_r8
```

PUBLIC TYPES:

```
   ! cvmix_input_type contains every possible necessary input field for all
   ! supported types of mixing.
   type, public :: cvmix_data_type
       integer        :: nlev = -1  ! Number of levels in column
                                    ! Setting default to -1 might be F95...
       real(cvmix_r8) :: ocn_depth, & ! depth >= 0!
                         surf_hgt     ! pos => above sea level
                                      ! neg => below sea level

       ! Values on interfaces
       ! nlev+1, 2
       real(cvmix_r8), dimension(:,:), pointer :: diff_iface => NULL()
       ! nlev+1
       real(cvmix_r8), dimension(:),   pointer :: visc_iface => NULL()
       real(cvmix_r8), dimension(:),   pointer :: z_iface    => NULL()
       real(cvmix_r8), dimension(:),   pointer :: dw_iface   => NULL()
       real(cvmix_r8), dimension(:),   pointer :: Ri_iface   => NULL()
```

```
    ! For tidal mixing, we need to calculate the vertical deposition
    ! function on each column as well as squared buoyancy frequency
    real(cvmix_r8), dimension(:),   pointer :: vert_dep   => NULL()
    real(cvmix_r8), dimension(:),   pointer :: buoy       => NULL()

    ! Values at tracer points
    ! nlev
    real(cvmix_r8), dimension(:),   pointer :: dens     => NULL()
    real(cvmix_r8), dimension(:),   pointer :: dens_lwr => NULL()
    real(cvmix_r8), dimension(:),   pointer :: z        => NULL()
    real(cvmix_r8), dimension(:),   pointer :: dz       => NULL()
    ! For double diffusion mixing, we need to calculate the stratification
    ! parameter R_rho. Since the denominator of this ratio may be zero,
    ! we store the numerator and denominator separately and make sure the
    ! denominator is non-zero before performing the division.
    real(cvmix_r8), dimension(:),   pointer :: strat_param_num   => NULL()
    real(cvmix_r8), dimension(:),   pointer :: strat_param_denom => NULL()
end type cvmix_data_type

! cvmix_global_params_type contains global parameters used by multiple
! mixing methods.
type, public :: cvmix_global_params_type
    integer                        :: max_nlev  ! maximum number of levels
    real(cvmix_r8)                 :: prandtl   ! Prandtl number
    ! For densities, user must keep track of units (kg/m^3 vs g/cm^3)
    real(cvmix_r8)                 :: fw_rho    ! fresh water density
    real(cvmix_r8)                 :: sw_rho    ! salt water density
end type cvmix_global_params_type

! cvmix_kpp_params_type contains the necessary parameters for KPP mixing
type, public :: cvmix_kpp_params_type
    real(cvmix_r8) :: deleteme
end type cvmix_kpp_params_type
```

1.3 Fortran: Module Interface cvmix_background

This module contains routines to initialize the derived types needed for time independent static background mixing coefficients. It specifies either a scalar, 1D, or 2D field for viscosity and diffusivity. It also calculates the background diffusivity using the Bryan-Lewis method. It then sets the viscosity and diffusivity to the specified value.

REVISION HISTORY:

```
SVN:$Id: cvmix_background.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_background.F90 $
```

USES:

```
use cvmix_kinds_and_types, only : cvmix_PI,                 &
                                  cvmix_r8,                 &
                                  cvmix_data_type,          &
                                  cvmix_global_params_type
```

PUBLIC MEMBER FUNCTIONS:

```
 public :: cvmix_init_bkgnd
 public :: cvmix_coeffs_bkgnd
 public :: cvmix_bkgnd_lvary_horizontal
 public :: cvmix_bkgnd_static_diff
 public :: cvmix_bkgnd_static_visc
 public :: cvmix_put_bkgnd
 public :: cvmix_get_bkgnd_real_2D

interface cvmix_init_bkgnd
  module procedure cvmix_init_bkgnd_scalar
  module procedure cvmix_init_bkgnd_1D
  module procedure cvmix_init_bkgnd_2D
  module procedure cvmix_init_bkgnd_BryanLewis
end interface cvmix_init_bkgnd

interface cvmix_put_bkgnd
  module procedure cvmix_put_bkgnd_real
  module procedure cvmix_put_bkgnd_real_1D
  module procedure cvmix_put_bkgnd_real_2D
end interface cvmix_put_bkgnd
```

PUBLIC TYPES:

```
! cvmix_bkgnd_params_type contains the necessary parameters for background
! mixing. Background mixing fields can vary from level to level as well as
! over latitude and longitude.
type, public :: cvmix_bkgnd_params_type
```

```
    private
    real(cvmix_r8), allocatable :: static_visc(:,:) ! ncol, nlev+1
    real(cvmix_r8), allocatable :: static_diff(:,:) ! ncol, nlev+1

    ! Note: need to include some logic to avoid excessive memory use
    !       when static_visc and static_diff are constant or 1-D
    logical                     :: lvary_vertical   ! True => second dim not 1
    logical                     :: lvary_horizontal ! True => first dim not 1
end type cvmix_bkgnd_params_type
```

1.3.1 cvmix_init_bkgnd_scalar

INTERFACE:

```
subroutine cvmix_init_bkgnd_scalar(CVmix_bkgnd_params, bkgnd_visc, bkgnd_diff)
```

DESCRIPTION:

Initialization routine for static background mixing coefficients. For each column, this routine sets the static viscosity / diffusivity to the given scalar constants.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
real(cvmix_r8), intent(in) :: bkgnd_visc
real(cvmix_r8), intent(in) :: bkgnd_diff
```

OUTPUT PARAMETERS:

```
type (cvmix_bkgnd_params_type), intent(out) :: CVmix_bkgnd_params
```

1.3.2 cvmix_init_bkgnd_1D

INTERFACE:

```
subroutine cvmix_init_bkgnd_1D(CVmix_params, CVmix_bkgnd_params, &
                               bkgnd_visc, bkgnd_diff, ncol)
```

DESCRIPTION:

Initialization routine for static background mixing coefficients. For each column, this routine sets the static viscosity / diffusivity to the given 1D field. If field varies horizontally, need to include ncol!

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_global_params_type), intent(in) :: CVmix_params
real(cvmix_r8), dimension(:),   intent(in) :: bkgnd_visc
real(cvmix_r8), dimension(:),   intent(in) :: bkgnd_diff
integer, optional,              intent(in) :: ncol
```

OUTPUT PARAMETERS:

```
type(cvmix_bkgnd_params_type), intent(out) :: CVmix_bkgnd_params
```

1.3.3 cvmix_init_bkgnd_2D

INTERFACE:

```
subroutine cvmix_init_bkgnd_2D(CVmix_params, CVmix_bkgnd_params, &
                               bkgnd_visc, bkgnd_diff, ncol)
```

DESCRIPTION:

Initialization routine for static background mixing coefficients. For each column, this routine sets the static viscosity / diffusivity to the given 2D field.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_global_params_type), intent(in) :: CVmix_params
real(cvmix_r8), dimension(:,:), intent(in) :: bkgnd_visc
real(cvmix_r8), dimension(:,:), intent(in) :: bkgnd_diff
integer,                        intent(in) :: ncol
```

OUTPUT PARAMETERS:

```
type(cvmix_bkgnd_params_type), intent(out) :: CVmix_bkgnd_params
```

1.3.4 cvmix_init_bkgnd_BryanLewis

INTERFACE:

```
subroutine cvmix_init_bkgnd_BryanLewis(CVmix_vars, CVmix_params,            &
                                       CVmix_bkgnd_params, bl1, bl2, bl3, bl4)
```

DESCRIPTION:

Initialization routine for Bryan-Lewis diffusivity/viscosity calculation. For each column, this routine sets the static viscosity & diffusivity based on the specified parameters. Note that the units of these parameters must be consistent with the units of viscosity and diffusivity – either cgs or mks, but do not mix and match!

The Bryan-Lewis parameterization is based on the following:

$$\begin{aligned}\kappa_{BL} &= \text{bl1} + \frac{\text{bl2}}{\pi}\tan^{-1}\bigg(\text{bl3}(|z| - \text{bl4})\bigg)\\ \nu_{BL} &= \text{Pr}\cdot\kappa_{BL}\end{aligned}$$

This method is based on the following paper:

A Water Mass Model of the World Ocean
K. Bryan and L. J. Lewis
Journal of Geophysical Research, vol 84 (1979), pages 2503-2517.

In that paper, they recommend the parameters

$\text{bl1} = 8\cdot 10^{-5}\ \text{m}^2/\text{s}$

$\text{bl2} = 1.05\cdot 10^{-4}\ \text{m}^2/\text{s}$

$\text{bl3} = 4.5\cdot 10^{-3}\ \text{m}^{-1}$

$\text{bl4} = 2500\ \text{m}$

However, more recent usage of their scheme may warrant different settings. **USES:**

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_data_type),          intent(in) :: CVmix_vars   ! Depth, nlev
type(cvmix_global_params_type), intent(in) :: CVmix_params ! Prandtl

! Units are first column if CVmix_data%depth is m, second if cm
real(cvmix_r8), intent(in) :: bl1,     &! m^2/s or cm^2/s
                              bl2,     &! m^2/s or cm^2/s
                              bl3,     &! 1/m   or 1/cm
                              bl4       ! m     or cm
```

OUTPUT PARAMETERS:

```
type(cvmix_bkgnd_params_type), intent(out) :: CVmix_bkgnd_params
```

1.3.5 cvmix_coeffs_bkgnd

INTERFACE:

```
subroutine cvmix_coeffs_bkgnd(CVmix_vars, CVmix_bkgnd_params, colid)
```

DESCRIPTION:

Computes vertical tracer and velocity mixing coefficients for static background mixing. This routine simply copies viscosity / diffusivity values from CVmix_bkgnd_params to CVmix_vars.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_bkgnd_params_type), intent(in) :: CVmix_bkgnd_params
! Need to know column for pulling data from static_visc and _diff
integer,                       intent(in) :: colid
```

INPUT/OUTPUT PARAMETERS:

```
type(cvmix_data_type), intent(inout) :: CVmix_vars
```

1.3.6 cvmix_bkgnd_lvary_horizontal

INTERFACE:

```
function cvmix_bkgnd_lvary_horizontal(CVmix_bkgnd_params)
```

DESCRIPTION:

Returns whether the background viscosity and diffusivity are varying with horizontal position.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_bkgnd_params_type), intent(in) :: CVmix_bkgnd_params
```

OUTPUT PARAMETERS:

```
logical :: cvmix_bkgnd_lvary_horizontal
```

1.3.7 cvmix_bkgnd_static_diff

INTERFACE:

```
  function cvmix_bkgnd_static_diff(CVmix_bkgnd_params,kw,colid)
```

DESCRIPTION:

Obtain the background diffusivity value at a position in a water column.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     type(cvmix_bkgnd_params_type), intent(in) :: CVmix_bkgnd_params
     integer, optional, intent(in) :: kw, colid
```

OUTPUT PARAMETERS:

```
     real(cvmix_r8) :: cvmix_bkgnd_static_diff
```

1.3.8 cvmix_bkgnd_static_visc

INTERFACE:

```
  function cvmix_bkgnd_static_visc(CVmix_bkgnd_params,kw,colid)
```

DESCRIPTION:

Obtain the background viscosity value at a position in a water column.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     type(cvmix_bkgnd_params_type), intent(in) :: CVmix_bkgnd_params
     integer, optional, intent(in) :: kw, colid
```

OUTPUT PARAMETERS:

```
     real(cvmix_r8) :: cvmix_bkgnd_static_visc
```

1.3.9 cvmix_put_bkgnd_real

INTERFACE:

```
subroutine cvmix_put_bkgnd_real(CVmix_bkgnd_params, varname, val)
```

DESCRIPTION:

Write a real value into a cvmix_bkgnd_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: varname
real(cvmix_r8),   intent(in) :: val
```

OUTPUT PARAMETERS:

```
type(cvmix_bkgnd_params_type), intent(inout) :: CVmix_bkgnd_params
```

1.3.10 cvmix_put_bkgnd_real_1D

INTERFACE:

```
subroutine cvmix_put_bkgnd_real_1D(CVmix_bkgnd_params, varname, val, &
                                   ncol, nlev)
```

DESCRIPTION:

Write an array of real values into a cvmix_bkgnd_params_type variable. You must use `opt='horiz'` to specify that the field varies in the horizontal direction, otherwise it is assumed to vary in the vertical.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*),             intent(in) :: varname
real(cvmix_r8), dimension(:), intent(in) :: val
integer, optional,            intent(in) :: ncol, nlev
```

OUTPUT PARAMETERS:

```
type (cvmix_bkgnd_params_type), intent(inout) :: CVmix_bkgnd_params
```

1.3.11 cvmix_put_bkgnd_real_2D

INTERFACE:

```
subroutine cvmix_put_bkgnd_real_2D(CVmix_bkgnd_params, varname, val, &
                                   ncol, nlev)
```

DESCRIPTION:

Write a 2D array of real values into a cvmix_bkgnd_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*),                intent(in) :: varname
real(cvmix_r8), dimension(:,:), intent(in) :: val
integer,                         intent(in) :: ncol, nlev
```

OUTPUT PARAMETERS:

```
type (cvmix_bkgnd_params_type), intent(out) :: CVmix_bkgnd_params
```

1.3.12 cvmix_get_bkgnd_real_2D

INTERFACE:

```
function cvmix_get_bkgnd_real_2D(CVmix_bkgnd_params, varname)
```

DESCRIPTION:

Read the real values of a cvmix_bkgnd_params_type 2D array variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_bkgnd_params_type), intent(in) :: CVmix_bkgnd_params
character(len=*),              intent(in) :: varname
```

OUTPUT PARAMETERS:

```
real(cvmix_r8), dimension(size(CVmix_bkgnd_params%static_visc,1),    &
                          size(CVmix_bkgnd_params%static_visc,2)) :: &
                cvmix_get_bkgnd_real_2D
```

1.4 Fortran: Module Interface cvmix_shear

This module contains routines to initialize the derived types needed for shear mixing, and to set the viscosity and diffusivity coefficients. Presently this scheme has implemented the shear mixing parameterizations from Pacanowski & Philander (1981) and Large, McWilliams, & Doney (1994).

REVISION HISTORY:

```
SVN:$Id: cvmix_shear.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_shear.F90 $
```

USES:

```
use cvmix_kinds_and_types, only : one,                          &
                                  cvmix_r8,                     &
                                  cvmix_strlen,                 &
                                  cvmix_data_type
use cvmix_background, only :      cvmix_bkgnd_params_type,      &
                                  cvmix_bkgnd_lvary_horizontal, &
                                  cvmix_bkgnd_static_diff,      &
                                  cvmix_bkgnd_static_visc
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cvmix_init_shear
public :: cvmix_coeffs_shear
public :: cvmix_put_shear
public :: cvmix_get_shear_real
public :: cvmix_get_shear_str

interface cvmix_put_shear
  module procedure cvmix_put_shear_real
  module procedure cvmix_put_shear_str
end interface cvmix_put_shear
```

PUBLIC TYPES:

```
! cvmix_shear_params_type contains the necessary parameters for shear mixing
! (currently Pacanowski-Philander or Large et al)
type, public :: cvmix_shear_params_type
    private
    character(len=cvmix_strlen) :: mix_scheme
    real(cvmix_r8)              :: PP_nu_zero
    real(cvmix_r8)              :: PP_alpha
    real(cvmix_r8)              :: PP_exp
    real(cvmix_r8)              :: KPP_nu_zero
    real(cvmix_r8)              :: KPP_Ri_zero
    real(cvmix_r8)              :: KPP_exp
  end type cvmix_shear_params_type
```

1.4.1 cvmix_init_shear

INTERFACE:

```
  subroutine cvmix_init_shear(CVmix_shear_params, mix_scheme,  &
                              PP_nu_zero, PP_alpha, PP_exp, &
                              KPP_nu_zero, KPP_Ri_zero, KPP_exp)
```

DESCRIPTION:

Initialization routine for shear (Richardson number-based) mixing. There are currently two supported schemes - set `mix_scheme = 'PP'` to use the Pacanowski-Philander mixing scheme or set `mix_scheme = 'KPP'` to use the interior mixing scheme laid out in Large et al.

PP requires setting ν_0 (`PP_nu_zero` in this routine), *alpha* (`PP_alpha`), and n (`PP_exp`), and returns

$$\begin{aligned}\nu_{PP} &= \frac{\nu_0}{(1+\alpha\mathrm{Ri})^n} + \nu_b \\ \kappa_{PP} &= \frac{\nu}{1+\alpha\mathrm{Ri}} + \kappa_b\end{aligned}$$

Note that ν_b and κ_b are set in `cvmix_init_bkgnd()`, which needs to be called separately from this routine.

KPP requires setting ν^0 (`KPP_nu_zero`, Ri_0(`KPP_Ri_zero`), and p_1 (`KPP_exp`), and returns

$$\nu_{KPP} = \begin{cases} \nu^0 & \mathrm{Ri} < 0 \\ \nu^0 \left[1 - \frac{\mathrm{Ri}}{\mathrm{Ri}_0}^2\right]^{p_1} & 0 < \mathrm{Ri} < \mathrm{Ri}_0 \\ 0 & \mathrm{Ri}_0 < \mathrm{Ri} \end{cases}$$

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
    character(len=*),          intent(in) :: mix_scheme
    real(cvmix_r8), optional, intent(in) :: PP_nu_zero, PP_alpha, PP_exp, &
                                            KPP_nu_zero, KPP_Ri_zero, KPP_exp
```

OUTPUT PARAMETERS:

```
    type(cvmix_shear_params_type), intent(inout) :: CVmix_shear_params
```

1.4.2 cvmix_coeffs_shear

INTERFACE:

```
subroutine cvmix_coeffs_shear(CVmix_vars, CVmix_shear_params,    &
                              CVmix_bkgnd_params, colid, no_diff)
```

DESCRIPTION:

Computes vertical tracer and velocity mixing coefficients for shear-type mixing parameterizatiions. Note that Richardson number is needed at both T-points and U-points.

USES:

```
only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_shear_params_type),           intent(in) :: CVmix_shear_params
! PP mixing requires CVmix_bkgnd_params
type(cvmix_bkgnd_params_type), optional, intent(in) :: CVmix_bkgnd_params
! colid is only needed if CVmix_bkgnd_params%lvary_horizontal is true
integer,                       optional, intent(in) :: colid
logical,                       optional, intent(in) :: no_diff
```

INPUT/OUTPUT PARAMETERS:

```
type(cvmix_data_type), intent(inout) :: CVmix_vars
```

1.4.3 cvmix_put_shear_real

INTERFACE:

```
subroutine cvmix_put_shear_real(CVmix_shear_params, varname, val)
```

DESCRIPTION:

Write a real value into a cvmix_shear_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: varname
real(cvmix_r8),   intent(in) :: val
```

OUTPUT PARAMETERS:

```
     type(cvmix_shear_params_type), intent(inout) :: CVmix_shear_params
```

1.4.4 cvmix_put_shear_str

INTERFACE:

```
   subroutine cvmix_put_shear_str(CVmix_shear_params, varname, val)
```

DESCRIPTION:

Write a string into a cvmix_shear_params_type variable.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     character(len=*), intent(in) :: varname
     character(len=*), intent(in) :: val
```

OUTPUT PARAMETERS:

```
     type(cvmix_shear_params_type), intent(inout) :: CVmix_shear_params
```

1.4.5 cvmix_get_shear_real

INTERFACE:

```
   function cvmix_get_shear_real(CVmix_shear_params, varname)
```

DESCRIPTION:

Read the real value of a cvmix_shear_params_type variable.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     type(cvmix_shear_params_type), intent(in) :: CVmix_shear_params
     character(len=*),              intent(in) :: varname
```

OUTPUT PARAMETERS:

```
     real(cvmix_r8) :: cvmix_get_shear_real
```

1.4.6 cvmix_get_shear_str

INTERFACE:

```
function cvmix_get_shear_str(CVmix_shear_params, varname)
```

DESCRIPTION:

Read the string contents of a cvmix_shear_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_shear_params_type), intent(in) :: CVmix_shear_params
character(len=*),              intent(in) :: varname
```

OUTPUT PARAMETERS:

```
character(len=cvmix_strlen) :: cvmix_get_shear_str
```

1.5 Fortran: Module Interface cvmix_tidal

This module contains routines to initialize the derived types needed for tidal mixing (currently just the Simmons scheme) and to set the viscosity and diffusivity coefficients accordingly.

REVISION HISTORY:

```
SVN:$Id: cvmix_tidal.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_tidal.F90 $
```

USES:

```
use cvmix_kinds_and_types, only : cvmix_r8,                 &
                                  cvmix_data_type,          &
                                  cvmix_strlen,             &
                                  cvmix_global_params_type
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cvmix_init_tidal
public :: cvmix_compute_vert_dep
public :: cvmix_coeffs_tidal
public :: cvmix_put_tidal
public :: cvmix_get_tidal_real
public :: cvmix_get_tidal_str

interface cvmix_put_tidal
  module procedure cvmix_put_tidal_real
  module procedure cvmix_put_tidal_str
end interface cvmix_put_tidal
```

PUBLIC TYPES:

```
! cvmix_tidal_params_type contains the necessary parameters for tidal mixing
! (currently just Simmons)
type, public :: cvmix_tidal_params_type
   private
   character(len=cvmix_strlen) :: mix_scheme
   real(cvmix_r8)              :: efficiency
   real(cvmix_r8)              :: vertical_decay_scale
   real(cvmix_r8)              :: max_coefficient
   real(cvmix_r8)              :: local_mixing_frac
   real(cvmix_r8)              :: depth_cutoff
end type cvmix_tidal_params_type
```

1.5.1 cvmix_init_tidal

INTERFACE:

```
subroutine cvmix_init_tidal(CVmix_tidal_params, mix_scheme, units,          &
                            efficiency, vertical_decay_scale,               &
                            max_coefficient, local_mixing_frac, depth_cutoff)
```

DESCRIPTION:

Initialization routine for tidal mixing. There is currently just one supported schemes - set `mix_scheme = 'simmons'` to use the Simmons mixing scheme. **USES:**

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*),           intent(in) :: mix_scheme
character(len=*),           intent(in) :: units
real(cvmix_r8), optional, intent(in) :: efficiency
real(cvmix_r8), optional, intent(in) :: vertical_decay_scale
real(cvmix_r8), optional, intent(in) :: max_coefficient
real(cvmix_r8), optional, intent(in) :: local_mixing_frac
real(cvmix_r8), optional, intent(in) :: depth_cutoff
```

OUTPUT PARAMETERS:

```
type(cvmix_tidal_params_type), intent(inout) :: CVmix_tidal_params
```

1.5.2 cvmix_coeffs_tidal

INTERFACE:

```
subroutine cvmix_coeffs_tidal(CVmix_vars, CVmix_tidal_params, CVmix_params, &
                              energy_flux)
```

DESCRIPTION:

Computes vertical diffusion coefficients for tidal mixing parameterizatiions.

USES:

```
only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_tidal_params_type),  intent(in) :: CVmix_tidal_params
type(cvmix_global_params_type), intent(in) :: CVmix_params
real(cvmix_r8),                 intent(in) :: energy_flux
```

INPUT/OUTPUT PARAMETERS:

```
type(cvmix_data_type), intent(inout) :: CVmix_vars
```

1.5.3 cvmix_compute_vert_dep

INTERFACE:

```
subroutine cvmix_compute_vert_dep(CVmix_vars, CVmix_tidal_params)
```

DESCRIPTION:

Computes the vertical deposition function needed for Simmons et al tidal mixing.

USES:

```
only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_tidal_params_type), intent(in) :: CVmix_tidal_params
```

OUTPUT PARAMETERS:

```
type(cvmix_data_type), intent(inout) :: CVmix_vars
```

1.5.4 cvmix_put_tidal_real

INTERFACE:

```
subroutine cvmix_put_tidal_real(CVmix_tidal_params, varname, val)
```

DESCRIPTION:

Write a real value into a cvmix_tidal_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
     character(len=*), intent(in) :: varname
     real(cvmix_r8),   intent(in) :: val
```

OUTPUT PARAMETERS:

```
     type(cvmix_tidal_params_type), intent(inout) :: CVmix_tidal_params
```

1.5.5 cvmix_put_tidal_str

INTERFACE:

```
   subroutine cvmix_put_tidal_str(CVmix_tidal_params, varname, val)
```

DESCRIPTION:

Write a string into a cvmix_tidal_params_type variable.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     character(len=*), intent(in) :: varname
     character(len=*), intent(in) :: val
```

OUTPUT PARAMETERS:

```
     type(cvmix_tidal_params_type), intent(inout) :: CVmix_tidal_params
```

1.5.6 cvmix_get_tidal_real

INTERFACE:

```
   function cvmix_get_tidal_real(CVmix_tidal_params, varname)
```

DESCRIPTION:

Returns the real value of a cvmix_tidal_params_type variable.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: varname
type(cvmix_tidal_params_type), intent(in) :: CVmix_tidal_params
```

OUTPUT PARAMETERS:

```
real(cvmix_r8) :: cvmix_get_tidal_real
```

1.5.7 cvmix_get_tidal_str

INTERFACE:

```
function cvmix_get_tidal_str(CVmix_tidal_params, varname)
```

DESCRIPTION:

Returns the string value of a cvmix_tidal_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: varname
type(cvmix_tidal_params_type), intent(inout) :: CVmix_tidal_params
```

OUTPUT PARAMETERS:

```
character(len=cvmix_strlen) :: cvmix_get_tidal_str
```

1.6 Fortran: Module Interface cvmix_ddiff

This module contains routines to initialize the derived types needed for double diffusion mixing and to set the diffusivity coefficient accordingly.

REVISION HISTORY:

```
 SVN:$Id: cvmix_ddiff.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
 SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_ddiff.F90 $
```

USES:

```
use cvmix_kinds_and_types, only : one,                     &
                                  cvmix_r8,                &
                                  cvmix_data_type
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cvmix_init_ddiff
public :: cvmix_coeffs_ddiff
public :: cvmix_put_ddiff
public :: cvmix_get_ddiff_real

interface cvmix_put_ddiff
  module procedure cvmix_put_ddiff_real
end interface cvmix_put_ddiff
```

PUBLIC TYPES:

```
! cvmix_ddiff_params_type contains the necessary parameters for double
! diffusion mixing
type, public :: cvmix_ddiff_params_type
  private
  real(cvmix_r8) :: strat_param_max
  real(cvmix_r8) :: kappa_ddiff_t
  real(cvmix_r8) :: kappa_ddiff_s
  real(cvmix_r8) :: ddiff_exp1
  real(cvmix_r8) :: ddiff_exp2
  real(cvmix_r8) :: kappa_ddiff_param1
  real(cvmix_r8) :: kappa_ddiff_param2
  real(cvmix_r8) :: kappa_ddiff_param3
  real(cvmix_r8) :: mol_diff
end type cvmix_ddiff_params_type
```

1.6.1 cvmix_init_ddiff

INTERFACE:

```
  subroutine cvmix_init_ddiff(CVmix_ddiff_params, units, strat_param_max, &
                              kappa_ddiff_t, kappa_ddiff_s, ddiff_exp1,   &
                              ddiff_exp2, mol_diff, kappa_ddiff_param1,   &
                              kappa_ddiff_param2, kappa_ddiff_param3)
```

DESCRIPTION:

Initialization routine for double diffusion mixing. This mixing technique looks for two unstable cases in a column - salty water over fresher water and colder water over warmer water - and computes different diffusivity coefficients in each of these two locations. The parameter

$$R_\rho = \frac{\alpha(\partial\Theta/\partial z)}{\beta(\partial S/\partial z)}$$

to determine as a stratification parameter. If $(\partial S/\partial z)$ is positive and $1 < R_\rho < R_\rho^0$ then salt water sits on top of fresh water and the diffusivity is given by

$$\kappa = \kappa^0 \left[1 - \left(\frac{R_\rho - 1}{R_\rho^0 - 1}\right)^{p_1}\right]^{p_2}$$

The user must specify which set of units to use, either `'mks'` or `'cgs'`. By default, $R_\rho^0 = 2.55$, but that can be changed by setting `strat_param_max` in the code. Similarly, by default $p_1 = 1$ (`ddiff_exp1`), $p_2 = 3$ (`ddiff_exp2`), and

$$\kappa^0 = \begin{cases} 7 \cdot 10^{-5}\ \mathrm{m}^2/\mathrm{s} & \text{for temperature (\texttt{kappa\_ddiff\_t} in this routine)} \\ 10^{-4}\ \mathrm{m}^2/\mathrm{s} & \text{for salinity and other tracers (\texttt{kappa\_ddiff\_s} in this routine).} \end{cases}$$

On the other hand, if $(\partial\Theta/\partial z)$ is negative and $0 < R_\rho < 1$ then cold water sits on warm warm water and the diffusivity for temperature is given by

$$\kappa = \nu_{\text{molecular}} \cdot 0.909 \exp\left\{4.6 \exp\left[-0.54\left(\frac{1}{R_\rho} - 1\right)\right]\right\}$$

where $\nu_{\text{molecular}}$ Is the molecular viscosity of water. By default it is set to $1.5 \cdot 10^{-6}$ m^2/s, but it can be changed through `mol_diff` in the code. Similarly, 0.909, 4.6, and -0.54 are the default values of `kappa_ddiff_param1`, `kappa_ddiff_param2`, and `kappa_ddiff_param3`, respectively.

For salinity and other tracers, κ above is multiplied by the factor

$$\text{factor} = \begin{cases} 0.15R_\rho & R_\rho < 0.5 \\ 1.85R_\rho - 0.85 & 0.5 \le R_\rho < 1 \end{cases}$$

κ is stored in `CVmix_vars%diff_iface(:,1)`, while the modified value for non-temperature tracers is stored in `CVmix_vars%diff_iface(:,2)`.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*),           intent(in) :: units ! "mks" or "cgs"
real(cvmix_r8),   optional, intent(in) :: strat_param_max, &
                                          kappa_ddiff_t, &
                                          kappa_ddiff_s, &
                                          ddiff_exp1, &
                                          ddiff_exp2, &
                                          mol_diff, &
                                          kappa_ddiff_param1, &
                                          kappa_ddiff_param2, &
                                          kappa_ddiff_param3
```

OUTPUT PARAMETERS:

```
type(cvmix_ddiff_params_type), intent(inout) :: CVmix_ddiff_params
```

1.6.2 cvmix_coeffs_ddiff

INTERFACE:

```
subroutine cvmix_coeffs_ddiff(CVmix_vars, CVmix_ddiff_params)
```

DESCRIPTION:

Computes vertical diffusion coefficients for the double diffusion mixing parameterizatiion.

USES:

```
only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_ddiff_params_type), intent(in) :: CVmix_ddiff_params
```

INPUT/OUTPUT PARAMETERS:

```
type(cvmix_data_type), intent(inout) :: CVmix_vars
```

LOCAL VARIABLES:

```
integer :: k ! column index
real(cvmix_r8) :: ddiff, Rrho
```

1.6.3 cvmix_put_ddiff_real

INTERFACE:

```
subroutine cvmix_put_ddiff_real(CVmix_ddiff_params, varname, val)
```

DESCRIPTION:

Write a real value into a cvmix_ddiff_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: varname
real(cvmix_r8),   intent(in) :: val
```

OUTPUT PARAMETERS:

```
type(cvmix_ddiff_params_type), intent(inout) :: CVmix_ddiff_params
```

1.6.4 cvmix_get_ddiff_real

INTERFACE:

```
function cvmix_get_ddiff_real(CVmix_ddiff_params, varname)
```

DESCRIPTION:

Return the real value of a cvmix_ddiff_params_type variable. NOTE: This function is not efficient and is only for infrequent queries of ddiff parameters, such as at initialization.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
type(cvmix_ddiff_params_type), intent(in) :: CVmix_ddiff_params
character(len=*),              intent(in) :: varname
```

OUTPUT PARAMETERS:

```
real(cvmix_r8) :: cvmix_get_ddiff_real
```

1.7 Fortran: Module Interface cvmix_convection

This module contains routines to initialize the derived types needed for specifying mixing coefficients to parameterize vertical convective mixing, and to set the viscosity and diffusivity in gravitationally unstable portions of the water column.

REVISION HISTORY:

```
SVN:$Id: cvmix_convection.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_convection.F90 $
```

USES:

```
use cvmix_kinds_and_types, only : cvmix_r8,               &
                                  cvmix_data_type
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cvmix_init_conv
public :: cvmix_coeffs_conv
public :: cvmix_put_conv
public :: cvmix_get_conv_real

interface cvmix_put_conv
  module procedure cvmix_put_conv_real
end interface cvmix_put_conv
```

PUBLIC TYPES:

```
! cvmix_conv_params_type contains the necessary parameters for convective
! mixing.
type, public :: cvmix_conv_params_type
  private
  real(cvmix_r8) :: convect_diff
  real(cvmix_r8) :: convect_visc
end type cvmix_conv_params_type
```

1.7.1 cvmix_init_conv

INTERFACE:

```
subroutine cvmix_init_conv(CVmix_conv_params, convect_diff, convect_visc)
```

DESCRIPTION:

Initialization routine for specifying convective mixing coefficients.

USES:

```
    Only those used by entire module.
```

OUTPUT PARAMETERS:

```
     type (cvmix_conv_params_type), intent(out) :: CVmix_conv_params
```

INPUT PARAMETERS:

```
    real(cvmix_r8), intent(in) :: &
       convect_diff,      &! diffusivity to parameterize convection
       convect_visc        ! viscosity to parameterize convection
```

1.7.2 cvmix_coeffs_conv

INTERFACE:

```
   subroutine cvmix_coeffs_conv(CVmix_vars, CVmix_conv_params)
```

DESCRIPTION:

Computes vertical diffusion coefficients for convective mixing.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     type (cvmix_conv_params_type), intent(in)  :: CVmix_conv_params
```

INPUT/OUTPUT PARAMETERS:

```
     type (cvmix_data_type), intent(inout) :: CVmix_vars
```

1.7.3 cvmix_put_conv_real

INTERFACE:

```
   subroutine cvmix_put_conv_real(CVmix_conv_params, varname, val)
```

DESCRIPTION:

Write a real value into a cvmix_conv_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
 character(len=*), intent(in) :: varname
 real(cvmix_r8),   intent(in) :: val
```

OUTPUT PARAMETERS:

```
 type(cvmix_conv_params_type), intent(inout) :: CVmix_conv_params
```

1.7.4 cvmix_get_conv_real

INTERFACE:

```
function cvmix_get_conv_real(CVmix_conv_params, varname)
```

DESCRIPTION:

Read the real value of a cvmix_conv_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
 type(cvmix_conv_params_type), intent(in) :: CVmix_conv_params
 character(len=*),             intent(in) :: varname
```

OUTPUT PARAMETERS:

```
 real(cvmix_r8) :: cvmix_get_conv_real
```

1.8 Fortran: Module Interface cvmix_put_get

This module contains routines to pack data into the cvmix datatypes (allocating memory as necessary) and then unpack the data out. If we switch to pointers, the pack will just point at the right target and the unpack will be un-necessary.

REVISION HISTORY:

```
    SVN:$Id: cvmix_put_get.F90 155 2013-06-07 19:08:49Z mike.levy.work@gmail.com $
    SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_put_get.F90 $
```

USES:

```
    use cvmix_kinds_and_types, only : cvmix_r8,                 &
                                      cvmix_data_type,          &
                                      cvmix_global_params_type
```

PUBLIC MEMBER FUNCTIONS:

```
   public :: cvmix_put

   interface cvmix_put
     module procedure cvmix_put_int
     module procedure cvmix_put_real
     module procedure cvmix_put_real_1D
     module procedure cvmix_put_global_params_int
     module procedure cvmix_put_global_params_real
   end interface cvmix_put
```

1.8.1 cvmix_put_int

INTERFACE:

```
   subroutine cvmix_put_int(CVmix_vars, varname, val, opts)
```

DESCRIPTION:

Write an integer value into a cvmix_data_type variable.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     character(len=*),           intent(in) :: varname
     integer,                    intent(in) :: val
     character(len=*), optional, intent(in) :: opts
```

OUTPUT PARAMETERS:

```
     type(cvmix_data_type), intent(inout) :: CVmix_vars
```

1.8.2 cvmix_put_real

INTERFACE:

```
   subroutine cvmix_put_real(CVmix_vars, varname, val, opts)
```

DESCRIPTION:

Write a real value into a cvmix_data_type variable.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
     character(len=*),           intent(in) :: varname
     real(cvmix_r8),             intent(in) :: val
     character(len=*), optional, intent(in) :: opts
```

OUTPUT PARAMETERS:

```
     type(cvmix_data_type), intent(inout) :: CVmix_vars
```

1.8.3 cvmix_put_real_1D

INTERFACE:

```
   subroutine cvmix_put_real_1D(CVmix_vars, varname, val, opts)
```

DESCRIPTION:

Write an array of real values into a cvmix_data_type variable.

USES:

```
    Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*),             intent(in) :: varname
real(cvmix_r8), dimension(:), intent(in) :: val
character(len=*), optional,   intent(in) :: opts
```

OUTPUT PARAMETERS:

```
type(cvmix_data_type), intent(inout) :: CVmix_vars
```

1.8.4 cvmix_put_global_params_int

INTERFACE:

```
subroutine cvmix_put_global_params_int(CVmix_params, varname, val)
```

DESCRIPTION:

Write an integer value into a cvmix_global_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: varname
integer,          intent(in) :: val
```

OUTPUT PARAMETERS:

```
type (cvmix_global_params_type), intent(inout) :: CVmix_params
```

1.8.5 cvmix_put_global_params_real

INTERFACE:

```
subroutine cvmix_put_global_params_real(CVmix_params, varname, val)
```

DESCRIPTION:

Write a real value into a cvmix_global_params_type variable.

USES:

```
Only those used by entire module.
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: varname
real(cvmix_r8),   intent(in) :: val
```

OUTPUT PARAMETERS:

```
type(cvmix_global_params_type), intent(inout) :: CVmix_params
```