In-code documentation for CVMix

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

August 14, 2024

Contents

1 Module Main Program (Stand-Alone)

1.0.1 cvmix_driver (Source File: cvmix_driver.F90)

The stand-alone driver for the CVMix package. This 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.

INTERFACE

USES

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

1.0.2 cvmix_BL_driver (Source File: cvmix_bgrnd_BL.F90)

A routine to test the Bryan-Lewis implementation of 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.

INTERFACE

```
Subroutine cvmix_BL_driver(nlev, max_nlev, ocn_depth)
```

USES

```
  use cvmix_kinds_and_types, only : cvmix_r8,                 &
                                    cvmix_zero,               &
                                    cvmix_data_type,          &
                                    cvmix_global_params_type
  use cvmix_background,      only : cvmix_init_bkgnd,         &
                                    cvmix_coeffs_bkgnd,       &
                                    cvmix_get_bkgnd_real_2D,  &
                                    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
                              max_nlev      ! number of columns in memory
real(cvmix_r8), intent(in) :: ocn_depth   ! Depth of ocn
```

1.0.3 cvmix_shear_driver (Source File: cvmix_shear_drv.F90)

A routine to test the Large, et al., implementation of 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.

INTERFACE

```
 Subroutine cvmix_shear_driver(nlev, max_nlev)
```

USES

```
   use cvmix_kinds_and_types, only : cvmix_r8,                &
                                     cvmix_zero,              &
                                     cvmix_one,               &
                                     cvmix_data_type
   use cvmix_shear,           only : cvmix_init_shear,        &
                                     cvmix_coeffs_shear
   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
                       max_nlev             ! number of columns in memory
```

1.0.4 cvmix_tidal_driver (Source File: cvmix_tidal_Simmons.F90)

A routine to test the Simmons implementation of tidal

INTERFACE

```
 Subroutine cvmix_tidal_driver()
```

USES

```
   use cvmix_kinds_and_types, only : cvmix_r8,                             &
                                     cvmix_zero,                           &
                                     cvmix_strlen,                         &
                                     cvmix_data_type,                      &
                                     cvmix_global_params_type
   use cvmix_tidal,           only : cvmix_init_tidal,                     &
                                     cvmix_coeffs_tidal,                   &
                                     cvmix_compute_Simmons_invariant,      &
                                     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
```

1.0.5 cvmix_ddiff_driver (Source File: cvmix_ddiff_drv.F90)

A routine to test the double diffusion mixing

INTERFACE

```
 Subroutine cvmix_ddiff_driver(nlev, max_nlev)
```

USES

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

```
  use cvmix_ddiff,           only : cvmix_init_ddiff,        &
                                    cvmix_coeffs_ddiff,      &
                                    cvmix_get_ddiff_real
  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
                         max_nlev             ! number of columns in memory
```

1.0.6 cvmix_kpp_driver (Source File: cvmix_kpp_drv.F90)

A routine to test the KPP

INTERFACE

```
Subroutine cvmix_kpp_driver()
```

USES

```
  use cvmix_kinds_and_types, only : cvmix_r8,                 &
                                    cvmix_zero,               &
                                    cvmix_one,                &
                                    cvmix_strlen,             &
                                    cvmix_data_type
  use cvmix_kpp,             only : cvmix_init_kpp,                          &
                                    cvmix_get_kpp_real,                      &
                                    cvmix_kpp_compute_OBL_depth,             &
                                    cvmix_kpp_compute_kOBL_depth,            &
                                    cvmix_kpp_compute_bulk_Richardson,       &
                                    cvmix_kpp_compute_unresolved_shear,      &
                                    cvmix_kpp_compute_turbulent_scales,      &
                                    cvmix_kpp_compute_shape_function_coeffs, &
                                    cvmix_coeffs_kpp
  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
```

2 Module 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.

USES

```
   use cvmix_kinds_and_types, only : cvmix_data_type,                          &
                                     cvmix_r8,                                 &
                                     cvmix_zero,                               &
                                     cvmix_strlen
   use cvmix_utils,           only : cvmix_att_name
#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_2d_double
    module procedure cvmix_output_write_3d_double
  end interface

  interface cvmix_output_write_att
    module procedure cvmix_output_write_att_integer
    module procedure cvmix_output_write_att_real
    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(:)
```

2.1 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.

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
```

2.2 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.

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
```

2.3 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.

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
```

2.4 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.

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
```

2.5 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.

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
```

2.6 cvmix_output_write_single_col

INTERFACE

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

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).

INPUT PARAMETERS

```
    integer,                                   intent(in) :: file_id
    type(cvmix_data_type)    ,                 intent(in) :: CVmix_vars
    character(len=*),             dimension(:), intent(in) :: var_names
    real(cvmix_r8), optional,                                              &
                    dimension(CVmix_vars%nlev), intent(in) :: buoyancy_cntr
```

LOCAL VARIABLES

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

2.7 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).

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 :: nlev, ncol, icol, kw, var
     logical :: z_err
 #ifdef _NETCDF
     integer                     :: nt_id, nw_id, ncol_id
     character(len=cvmix_strlen) :: var_name
     integer,             dimension(:),   allocatable :: var_id
     real(kind=cvmix_r8), dimension(:,:), allocatable :: lcl_Mdiff, lcl_Tdiff, &
                                                         lcl_Sdiff
 #endif
```

2.8 cvmix_write_2d_double

INTERFACE

```
   subroutine cvmix_output_write_2d_double(file_id, var_name, dim_names,       &
                                           field, FillVal)
```

DESCRIPTION:

Routine to write a 2d field to a netcdf file. Called with cvmix_output_ write (see interface in PUBLIC MEMBER FUNCTIONS above).

INPUT PARAMETERS

```
    integer,                           intent(in) :: file_id
    character(len=*),                  intent(in) :: var_name
    character(len=*), dimension(2),    intent(in) :: dim_names
    real(cvmix_r8),   dimension(:,:),  intent(in) :: field
    real(cvmix_r8), optional,          intent(in) :: FillVal
```

LOCAL VARIABLES

```
    integer, dimension(2) :: dims
    integer               :: i,j
    logical               :: add_fill
#ifdef _NETCDF
    integer, dimension(2) :: dimids
    integer               :: varid
#endif
```

2.9 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).

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
```

2.10 cvmix_write_att_integer

INTERFACE

```
subroutine cvmix_output_write_att_integer(file_id, att_name, att_val,       &
                                          var_name)
```

DESCRIPTION:

Routine to write an attribute with an integer 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).

INPUT PARAMETERS

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

LOCAL VARIABLES

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

2.11 cvmix_write_att_real

INTERFACE

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

DESCRIPTION:

Routine to write an attribute with a real 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).

INPUT PARAMETERS

```
integer,          intent(in)           :: file_id
character(len=*), intent(in)           :: att_name
real(cvmix_r8),   intent(in)           :: att_val
character(len=*), intent(in), optional :: var_name
```

LOCAL VARIABLES

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

2.12 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).

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
```

2.13 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.

INPUT PARAMETERS

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

LOCAL VARIABLES

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

2.14 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)

LOCAL VARIABLES

```
integer :: fid
```

2.15 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.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

LOCAL VARIABLES

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

2.16 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.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
integer             :: get_file_type
```

LOCAL VARIABLES

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

2.17 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.

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
```

2.18 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.

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
```

3 Module cvmix_kinds_and_types

AUTHOR

```
Michael Levy, NCAR (mlevy@ucar.edu)
```

DESCRIPTION:

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.

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_log_kind = kind(.true.),                &
                              cvmix_strlen   = 256

! Parameters to allow CVMix to store integers instead of strings
integer, parameter, public :: CVMIX_OVERWRITE_OLD_VAL    = 1
integer, parameter, public :: CVMIX_SUM_OLD_AND_NEW_VALS = 2
integer, parameter, public :: CVMIX_MAX_OLD_AND_NEW_VALS = 3

! 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 :: cvmix_zero = real(0,cvmix_r8),         &
                                     cvmix_one  = real(1,cvmix_r8)
real(cvmix_r8), parameter, public :: cvmix_PI   = &
                                     3.14159265358979323846_cvmix_r8
```

PUBLIC TYPES

```
! cvmix_data_type contains variables for time-dependent and column-specific
! mixing. Time-independent physical parameters should be stored in
! cvmix_global_params_type and *-mixing specific parameters should be
! stored in cvmix_*_params_type (found in the cvmix_* module).
type, public :: cvmix_data_type
  integer        :: nlev = -1      ! Number of active levels in column
  integer        :: max_nlev = -1  ! Number of levels in column
                                   ! Setting defaults to -1 might be F95...

  ! Scalar quantities
  ! distance from sea level to ocean bottom (positive => below sea level)
  real(cvmix_r8) :: OceanDepth
                  ! units: m
  ! distance from sea level to OBL bottom (positive => below sea level)
  real(cvmix_r8) :: BoundaryLayerDepth
                  ! units: m
  ! sea surface height (positive => above sea level)
  real(cvmix_r8) :: SeaSurfaceHeight
                  ! units: m
  ! turbulent friction velocity at surface
  real(cvmix_r8) :: SurfaceFriction
                  ! units: m/s
  ! buoyancy forcing at surface
  real(cvmix_r8) :: SurfaceBuoyancyForcing
                  ! units: m^2 s^-3
  ! latitude of column
  real(cvmix_r8) :: lat
```

```
                ! units: degrees
! longitude of column
real(cvmix_r8) :: lon
                ! units: degrees
! Coriolis parameter
real(cvmix_r8) :: Coriolis
                ! units: s^-1
! Index of cell containing OBL (fraction > .5 => below cell center)
real(cvmix_r8) :: kOBL_depth
                ! units: unitless
! Langmuir mixing induced enhancement factor to turbulent velocity scale
real(cvmix_r8) :: LangmuirEnhancementFactor
                ! units: unitless
! Langmuir number
real(cvmix_r8) :: LangmuirNumber
                ! units: unitless
! Stokes Similarity Parameter
real(cvmix_r8) :: StokesMostXi
                ! units: unitless
! Numerical limit of Ocean Boundary Layer Depth
real(cvmix_r8) :: zBottomOceanNumerics
                ! units: m
! A time-invariant coefficient needed for Simmons, et al. tidal mixing
real(cvmix_r8) :: SimmonsCoeff

! Values on interfaces (dimsize = nlev+1)
! height of interfaces in column (positive up => most are negative)
real(cvmix_r8), dimension(:), pointer :: zw_iface => NULL()
                                       ! units: m

! distance between neighboring cell centers (first value is top of ocean to
! middle of first cell, last value is middle of last cell to ocean bottom
real(cvmix_r8), dimension(:), pointer :: dzw                   => NULL()
                                       ! units: m

! diffusivity coefficients at interfaces
! different coefficients for momentum (Mdiff), temperature (Tdiff), and
! salinity / non-temp tracers (Sdiff)
real(cvmix_r8), dimension(:), pointer :: Mdiff_iface => NULL()
real(cvmix_r8), dimension(:), pointer :: Tdiff_iface => NULL()
real(cvmix_r8), dimension(:), pointer :: Sdiff_iface => NULL()
                                       ! units: m^2/s

! shear Richardson number at column interfaces
real(cvmix_r8), dimension(:), pointer :: ShearRichardson_iface => NULL()
                                       ! units: unitless

! For tidal mixing, we need the squared buoyancy frequency and vertical
! deposition function
real(cvmix_r8), dimension(:), pointer :: SqrBuoyancyFreq_iface => NULL()
                                       ! units: s^-2
real(cvmix_r8), dimension(:), pointer :: VertDep_iface => NULL()
                                       ! units: unitless

! A time-dependent coefficient needed for Schmittner 2014
real(cvmix_r8), dimension(:), pointer   :: SchmittnerCoeff => NULL()

! A time-invariant coefficient needed in Schmittner tidal mixing
real(cvmix_r8), dimension(:), pointer   :: SchmittnerSouthernOcean => NULL()

! Another time-invariant coefficient needed in Schmittner tidal mixing
real(cvmix_r8), dimension(:,:), pointer :: exp_hab_zetar => NULL()



! For KPP, need to store non-local transport term
real(cvmix_r8), dimension(:), pointer :: kpp_Tnonlocal_iface => NULL()
real(cvmix_r8), dimension(:), pointer :: kpp_Snonlocal_iface => NULL()
                                       ! units: unitless (see note below)
! Note that kpp_transport_iface is the value of K_x*gamma_x/flux_x: in
! other words, the user must multiply this value by either the freshwater
! flux or the penetrative shortwave heat flux to come the values in Eqs.
! (7.128) and (7.129) of the CVMix manual.
! Currently only provide nonlocal term for temperature tracer and salinity
! (non-temperature) tracers. Eventually may add support for momentum terms
! (would be 2D for x- and y-, respectively) but current implementation
! assumes momentum term is 0 everywhere.

! Values at tracer points (dimsize = nlev)
! height of cell centers in column (positive up => most are negative)
real(cvmix_r8), dimension(:), pointer :: zt_cntr => NULL()
                                       ! units: m

! level thicknesses (positive semi-definite)
real(cvmix_r8), dimension(:), pointer :: dzt => NULL()
                                       ! units: m

! Two density values are stored: the actual density of water at a given
! level and the the density of water after adiabatic displacement to the
! level below where the water actually is
real(cvmix_r8), dimension(:), pointer :: WaterDensity_cntr      => NULL()
real(cvmix_r8), dimension(:), pointer :: AdiabWaterDensity_cntr => NULL()
                                       ! units: kg m^-3

! bulk Richardson number
real(cvmix_r8), dimension(:), pointer :: BulkRichardson_cntr => NULL()
```

```
                                          ! units: unitless

    ! 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()
                                          ! units: unitless

    ! For KPP we need velocity (in both x direction and y direction)
    real(cvmix_r8), dimension(:), pointer :: Vx_cntr => NULL()
    real(cvmix_r8), dimension(:), pointer :: Vy_cntr => NULL()
                                          ! units: m/s
  end type cvmix_data_type

  ! cvmix_global_params_type contains global parameters used by multiple
  ! mixing methods.
  type, public :: cvmix_global_params_type
    ! maximum number of levels for any column
    integer :: max_nlev
              ! units: unitless

    real(cvmix_r8) :: Gravity = 9.80616_cvmix_r8

    ! Prandtl number
    real(cvmix_r8) :: prandtl
                     ! units: unitless

    ! Fresh water and salt water densities
    real(cvmix_r8) :: FreshWaterDensity
    real(cvmix_r8) :: SaltWaterDensity
                     ! units: kg m^-3

  end type cvmix_global_params_type
```

4 Module cvmix_background

AUTHOR

```
    Michael N. Levy, NCAR (mlevy@ucar.edu)
```

DESCRIPTION:

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.

References:
* K Bryan and LJ Lewis. A Water Mass Model of the World Ocean. Journal of Geophysical Research, 1979.

USES

```
   use cvmix_kinds_and_types, only : cvmix_PI,                                 &
                                     cvmix_r8,                                 &
                                     cvmix_strlen,                             &
                                     cvmix_zero,                               &
                                     cvmix_data_type,                          &
                                     cvmix_global_params_type,                 &
                                     CVMIX_OVERWRITE_OLD_VAL,                  &
                                     CVMIX_SUM_OLD_AND_NEW_VALS,               &
                                     CVMIX_MAX_OLD_AND_NEW_VALS
   use cvmix_put_get,         only : cvmix_put
   use cvmix_utils,           only : cvmix_update_wrap
```

PUBLIC MEMBER FUNCTIONS

```
   public :: cvmix_init_bkgnd
   public :: cvmix_coeffs_bkgnd
   public :: cvmix_bkgnd_lvary_horizontal
   public :: cvmix_bkgnd_static_Mdiff
   public :: cvmix_bkgnd_static_Tdiff
   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_wrap
     module procedure cvmix_init_bkgnd_BryanLewis_low
```

```
end interface cvmix_init_bkgnd

interface cvmix_coeffs_bkgnd
  module procedure cvmix_coeffs_bkgnd_low
  module procedure cvmix_coeffs_bkgnd_wrap
end interface cvmix_coeffs_bkgnd

interface cvmix_put_bkgnd
  module procedure cvmix_put_bkgnd_int
  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
    ! 3D viscosity field (horizontal dimensions are collapsed into first
    ! dimension, vertical is second dimension)
    real(cvmix_r8), allocatable :: static_Mdiff(:,:) ! ncol, max_nlev+1
                                                      ! units: m^2/s
    ! 3D diffusivity field (horizontal dimensions are collapsed into first
    ! dimension, vertical is second dimension)
    real(cvmix_r8), allocatable :: static_Tdiff(:,:) ! ncol, max_nlev+1
                                                      ! units: m^2/s

    ! Flag for what to do with old values of CVmix_vars%[MTS]diff
    integer :: handle_old_vals

    ! Note: need to include some logic to avoid excessive memory use
    !       when static_[MT]diff are constant or 1-D
    logical :: lvary_vertical   ! True => multiple levels
    logical :: lvary_horizontal ! True => multiple columns
end type cvmix_bkgnd_params_type
```

4.1 cvmix_init_bkgnd_scalar

INTERFACE

```
subroutine cvmix_init_bkgnd_scalar(bkgnd_Tdiff, bkgnd_Mdiff, old_vals,      &
                                   CVmix_bkgnd_params_user)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
real(cvmix_r8),             intent(in) :: bkgnd_Tdiff
real(cvmix_r8),             intent(in) :: bkgnd_Mdiff
character(len=*), optional, intent(in) :: old_vals
```

OUTPUT PARAMETERS

```
type(cvmix_bkgnd_params_type), optional, target, intent(inout) :: &
                                          CVmix_bkgnd_params_user
```

4.2 cvmix_init_bkgnd_1D

INTERFACE

```
subroutine cvmix_init_bkgnd_1D(bkgnd_Tdiff, bkgnd_Mdiff, ncol, old_vals,    &
                               CVmix_params_user, CVmix_bkgnd_params_user)
```

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!

INPUT PARAMETERS

```
real(cvmix_r8), dimension(:),          intent(in) :: bkgnd_Tdiff
real(cvmix_r8), dimension(:),          intent(in) :: bkgnd_Mdiff
integer,                     optional, intent(in) :: ncol
character(len=cvmix_strlen), optional, intent(in) :: old_vals
type(cvmix_global_params_type), optional, target, intent(in) :: &
                                              CVmix_params_user
```

OUTPUT PARAMETERS

```
type(cvmix_bkgnd_params_type),  optional, target, intent(inout) :: &
                                           CVmix_bkgnd_params_user
```

4.3 cvmix_init_bkgnd_2D

INTERFACE

```
subroutine cvmix_init_bkgnd_2D(bkgnd_Tdiff, bkgnd_Mdiff, ncol,             &
                               CVmix_params_in, old_vals,                  &
                               CVmix_bkgnd_params_user)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
real(cvmix_r8), dimension(:,:),         intent(in) :: bkgnd_Tdiff
real(cvmix_r8), dimension(:,:),         intent(in) :: bkgnd_Mdiff
integer,                                intent(in) :: ncol
character(len=cvmix_strlen), optional, intent(in) :: old_vals
type(cvmix_global_params_type),         intent(in) :: CVmix_params_in
```

OUTPUT PARAMETERS

```
type(cvmix_bkgnd_params_type),  target, optional, intent(inout) ::        &
                                              CVmix_bkgnd_params_user
```

4.4 cvmix_init_bkgnd_BryanLewis_wrap

INTERFACE

```
subroutine cvmix_init_bkgnd_BryanLewis_wrap(CVmix_vars, bl1, bl2, bl3, bl4, &
                                            CVmix_params_in, old_vals,      &
                                            CVmix_bkgnd_params_user)
```

DESCRIPTION:

Calls cvmix_init_bkgnd_BryanLewis_low

INPUT PARAMETERS

```
! Contains depth and nlev
type(cvmix_data_type), intent(in) :: CVmix_vars
! 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
```

```
     character(len=cvmix_strlen),           optional, intent(in) :: old_vals
     type(cvmix_global_params_type), intent(in) :: CVmix_params_in
```

OUTPUT PARAMETERS

```
     type(cvmix_bkgnd_params_type),  target, optional, intent(inout) ::        &
                                                   CVmix_bkgnd_params_user
```

4.5 cvmix_coeffs_bkgnd_low

INTERFACE

```
   subroutine cvmix_init_bkgnd_BryanLewis_low(max_nlev, zw, bl1, bl2, bl3, bl4, &
                                          prandtl, old_vals, CVmix_bkgnd_params_user)
```

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:

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

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

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

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

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

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

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

INPUT PARAMETERS

```
integer,                      intent(in) :: max_nlev
real(cvmix_r8), dimension(max_nlev+1), intent(in) :: zw
! 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
                              prandtl   ! nondim
character(len=cvmix_strlen),            optional, intent(in) :: old_vals
```

OUTPUT PARAMETERS

```
type(cvmix_bkgnd_params_type),  target, optional, intent(inout) ::        &
                                           CVmix_bkgnd_params_user
```

4.6 cvmix_coeffs_bkgnd_wrap

INTERFACE

```
subroutine cvmix_coeffs_bkgnd_wrap(CVmix_vars, colid,                     &
                                   CVmix_bkgnd_params_user)
```

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.

INPUT PARAMETERS

```
! Need to know column for pulling data from static_[MT]diff
integer,                                optional, intent(in) :: colid
type(cvmix_bkgnd_params_type), target, optional, intent(in) ::            &
                                        CVmix_bkgnd_params_user
```

INPUT/OUTPUT PARAMETERS

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

4.7 cvmix_coeffs_bkgnd_low

INTERFACE

```
subroutine cvmix_coeffs_bkgnd_low(Mdiff_out, Tdiff_out, nlev, max_nlev,     &
                                  colid, CVmix_bkgnd_params_user)
```

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.

INPUT PARAMETERS

```
    ! Need to know column for pulling data from static_[MT]diff
    integer,                                         intent(in) :: nlev,      &
                                                                   max_nlev
    integer,                               optional, intent(in) :: colid
    type(cvmix_bkgnd_params_type), target, optional, intent(in) ::            &
                                           CVmix_bkgnd_params_user
```

OUTPUT PARAMETERS

```
    ! Using intent(inout) because memory should already be allocated
    real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Mdiff_out,        &
                                                            Tdiff_out
```

4.8 cvmix_bkgnd_lvary_horizontal

INTERFACE

```
  function cvmix_bkgnd_lvary_horizontal(CVmix_bkgnd_params_test)
```

DESCRIPTION:

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

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
  logical :: cvmix_bkgnd_lvary_horizontal
```

4.9 cvmix_bkgnd_static_Mdiff

INTERFACE

```
function cvmix_bkgnd_static_Mdiff(CVmix_bkgnd_params_user,kw,colid)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
type(cvmix_bkgnd_params_type), target, optional, intent(in) ::            &
                                       CVmix_bkgnd_params_user
integer, optional, intent(in) :: kw, colid
```

OUTPUT PARAMETERS

```
real(cvmix_r8) :: cvmix_bkgnd_static_Mdiff
```

4.10 cvmix_bkgnd_static_Tdiff

INTERFACE

```
function cvmix_bkgnd_static_Tdiff(CVmix_bkgnd_params_user,kw,colid)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
type(cvmix_bkgnd_params_type), target, optional, intent(in) ::            &
                                       CVmix_bkgnd_params_user
integer, optional, intent(in) :: kw, colid
```

OUTPUT PARAMETERS

```
real(cvmix_r8) :: cvmix_bkgnd_static_Tdiff
```

4.11 cvmix_put_bkgnd_int

INTERFACE

```
subroutine cvmix_put_bkgnd_int(varname, val, CVmix_bkgnd_params_user)
```

DESCRIPTION:

Write a real value into a cvmix_bkgnd_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_bkgnd_params_type), target, optional, intent(inout) ::         &
                                                  CVmix_bkgnd_params_user
```

4.12 cvmix_put_bkgnd_real

INTERFACE

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

DESCRIPTION:

Write a real value into a cvmix_bkgnd_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_bkgnd_params_type), target, optional, intent(inout) ::         &
                                                  CVmix_bkgnd_params_user
```

4.13 cvmix_put_bkgnd_real_1D

INTERFACE

```
subroutine cvmix_put_bkgnd_real_1D(varname, val, CVmix_bkgnd_params_user,   &
                                   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.

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), target, optional, intent(inout) ::         &
                                              CVmix_bkgnd_params_user
```

4.14 cvmix_put_bkgnd_real_2D

INTERFACE

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

DESCRIPTION:

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

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),  optional, target, intent(inout) ::        &
                                               CVmix_bkgnd_params_user
```

4.15 cvmix_get_bkgnd_real_2D

INTERFACE

```
   function cvmix_get_bkgnd_real_2D(varname, CVmix_bkgnd_params_user)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
character(len=*),                                intent(in) :: varname
type(cvmix_bkgnd_params_type), target, optional, intent(in) ::           &
                                            CVmix_bkgnd_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8), allocatable, dimension(:,:) :: cvmix_get_bkgnd_real_2D
```

5 Module cvmix_shear

AUTHOR

```
Michael N. Levy, NCAR (mlevy@ucar.edu)
```

DESCRIPTION:

This module contains routines to initialize the derived types needed for shear mixing, and to set the viscosity and diffusivity coefficients.

References:
* RC Pacanowski and SGH Philander. Parameterizations of Vertical Mixing in Numerical Models of Tropical Oceans. Journal of Physical Oceanography, 1981.
* WG Large, JC McWilliams, and SC Doney. Oceanic Vertical Mixing: A Review and a Model with a Nonlocal Boundary Layer Parameterization. Review of Geophysics, 1994.

USES

```
use cvmix_kinds_and_types, only : cvmix_r8,                                 &
                                  cvmix_zero,                               &
                                  cvmix_one,                                &
                                  cvmix_strlen,                             &
                                  cvmix_data_type,                          &
                                  CVMIX_OVERWRITE_OLD_VAL,                  &
                                  CVMIX_SUM_OLD_AND_NEW_VALS,               &
                                  CVMIX_MAX_OLD_AND_NEW_VALS
use cvmix_put_get,         only : cvmix_put
use cvmix_utils,           only : cvmix_update_wrap
```

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_coeffs_shear
  module procedure cvmix_coeffs_shear_low
  module procedure cvmix_coeffs_shear_wrap
end interface cvmix_coeffs_shear

interface cvmix_put_shear
  module procedure cvmix_put_shear_int
  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
    ! Type of shear mixing to run (PP => Pacanowski-Philander, KPP => LMD94)
    character(len=cvmix_strlen) :: mix_scheme

    ! Pacanowski - Philander parameters
    ! See Eqs. (1) and (2) in 1981 paper

    ! numerator in viscosity term (O(5e-3) in PP81; default here is 0.01)
    real(cvmix_r8) :: PP_nu_zero  ! units: m^2/s

    ! coefficient of Richardson number in denominator of visc / diff terms
    ! (5 in PP81)
    real(cvmix_r8) :: PP_alpha    ! units: unitless

    ! exponent of denominator in viscosity term (2 in PP81)
    real(cvmix_r8) :: PP_exp      ! units: unitless

    ! background coefficients for visc / diff terms
    ! (1e-4 and 1e-5, respectively, in PP81; default here is 0 for both)
    real(cvmix_r8) :: PP_nu_b     ! units: m^2/s
    real(cvmix_r8) :: PP_kappa_b  ! units: m^2/s

    ! Large et al parameters
    ! See Eq. (28b) in 1994 paper

    ! leading coefficient of shear mixing formula (5e-3 in LMD94)
    real(cvmix_r8) :: KPP_nu_zero ! units: m^2/s

    ! critical Richardson number value (0.7 in LMD94)
    real(cvmix_r8) :: KPP_Ri_zero ! units: unitless

    ! Exponent of unitless factor of diffusities (3 in LMD94)
    real(cvmix_r8) :: KPP_exp     ! units: unitless

    ! Flag for what to do with old values of CVmix_vars%[MTS]diff
    integer :: handle_old_vals
end type cvmix_shear_params_type
```

5.1 cvmix_init_shear

INTERFACE

```
subroutine cvmix_init_shear(CVmix_shear_params_user, mix_scheme,          &
                                     PP_nu_zero, PP_alpha, PP_exp, PP_nu_b,         &
                                     PP_kappa_b, KPP_nu_zero, KPP_Ri_zero, KPP_exp, &
                                     old_vals)
```

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), α (`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 0 by default, with the assumption that background diffusivities are computed in the `cvmix_background` module

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

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

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
    type(cvmix_shear_params_type), optional, target, intent(inout) ::      &
                                              CVmix_shear_params_user
```

5.2 cvmix_coeffs_shear_wrap

INTERFACE

```
subroutine cvmix_coeffs_shear_wrap(CVmix_vars, CVmix_shear_params_user)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
type(cvmix_shear_params_type), target, optional, intent(in) ::           &
                                          CVmix_shear_params_user
```

INPUT/OUTPUT PARAMETERS

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

5.3 cvmix_coeffs_shear_low

INTERFACE

```
subroutine cvmix_coeffs_shear_low(Mdiff_out, Tdiff_out, RICH, nlev,        &
                                  max_nlev, CVmix_shear_params_user)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
type(cvmix_shear_params_type), target, optional, intent(in) ::           &
                                          CVmix_shear_params_user
integer, intent(in) :: nlev, max_nlev
real(cvmix_r8), dimension(max_nlev+1), intent(in) :: RICH
```

INPUT/OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Mdiff_out,       &
                                                        Tdiff_out
```

5.4 cvmix_put_shear_int

INTERFACE

```
subroutine cvmix_put_shear_int(varname, val, CVmix_shear_params_user)
```

DESCRIPTION:

Write an integer value into a cvmix_shear_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_shear_params_type), optional, target, intent(inout) ::         &
                                          CVmix_shear_params_user
```

5.5 cvmix_put_shear_real

INTERFACE

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

DESCRIPTION:

Write a real value into a cvmix_shear_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_shear_params_type), optional, target, intent(inout) ::         &
                                          CVmix_shear_params_user
```

5.6 cvmix_put_shear_str

INTERFACE

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

DESCRIPTION:

Write a string into a cvmix_shear_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_shear_params_type), optional, target, intent(inout) ::         &
                                          CVmix_shear_params_user
```

5.7 cvmix_get_shear_real

INTERFACE

```
function cvmix_get_shear_real(varname, CVmix_shear_params_user)
```

DESCRIPTION:

Read the real value of a cvmix_shear_params_type variable.

INPUT PARAMETERS

```
character(len=*),                                intent(in) :: varname
type(cvmix_shear_params_type), optional, target, intent(in) ::            &
                                       CVmix_shear_params_user
```

OUTPUT PARAMETERS

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

5.8 cvmix_get_shear_str

INTERFACE

```
function cvmix_get_shear_str(varname, CVmix_shear_params_user)
```

DESCRIPTION:

Read the string contents of a cvmix_shear_params_type variable.

INPUT PARAMETERS

```
character(len=*),                                  intent(in) :: varname
type(cvmix_shear_params_type), optional, target, intent(in) ::           &
                                               CVmix_shear_params_user
```

OUTPUT PARAMETERS

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

6 Module cvmix_tidal

AUTHOR

```
Michael N. Levy, NCAR (mlevy@ucar.edu)
```

DESCRIPTION:

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.

References:
* HL Simmons, SR Jayne, LC St. Laurent, and AJ Weaver. Tidally Driven Mixing in a Numerical Model of the Ocean General Circulation. Ocean Modelling, 2004.

USES

```
use cvmix_kinds_and_types, only : cvmix_r8,                                &
                                  cvmix_log_kind,                          &
                                  cvmix_zero,                              &
                                  cvmix_one,                               &
                                  cvmix_data_type,                         &
                                  cvmix_strlen,                            &
                                  cvmix_global_params_type,                &
                                  CVMIX_OVERWRITE_OLD_VAL,                 &
                                  CVMIX_SUM_OLD_AND_NEW_VALS,              &
                                  CVMIX_MAX_OLD_AND_NEW_VALS
use cvmix_utils,           only : cvmix_update_wrap
use cvmix_put_get,         only : cvmix_put
```

PUBLIC MEMBER FUNCTIONS

```
public :: cvmix_init_tidal
public :: cvmix_coeffs_tidal
public :: cvmix_coeffs_tidal_schmittner
public :: cvmix_compute_Simmons_invariant
public :: cvmix_compute_Schmittner_invariant
public :: cvmix_compute_SchmittnerCoeff
public :: cvmix_compute_socn_tidal_invariant
public :: cvmix_compute_vert_dep
public :: cvmix_compute_vert_dep_Schmittner
public :: cvmix_put_tidal
public :: cvmix_get_tidal_real
public :: cvmix_get_tidal_str

interface cvmix_coeffs_tidal
```

```
    module procedure cvmix_coeffs_tidal_low
    module procedure cvmix_coeffs_tidal_schmittner
    module procedure cvmix_coeffs_tidal_wrap
  end interface cvmix_coeffs_tidal

  interface cvmix_compute_Simmons_invariant
    module procedure cvmix_compute_Simmons_invariant_low
    module procedure cvmix_compute_Simmons_invariant_wrap
  end interface cvmix_compute_Simmons_invariant

  interface cvmix_compute_Schmittner_invariant
    module procedure cvmix_compute_Schmittner_invariant_low
    module procedure cvmix_compute_Schmittner_invariant_wrap
  end interface cvmix_compute_Schmittner_invariant

  interface cvmix_compute_SchmittnerCoeff
    module procedure cvmix_compute_SchmittnerCoeff_low
    module procedure cvmix_compute_SchmittnerCoeff_wrap
  end interface cvmix_compute_SchmittnerCoeff

  interface cvmix_compute_socn_tidal_invariant
    module procedure cvmix_compute_socn_tidal_invariant_low
    module procedure cvmix_compute_socn_tidal_invariant_wrap
  end interface cvmix_compute_socn_tidal_invariant

  interface cvmix_put_tidal
    module procedure cvmix_put_tidal_int
    module procedure cvmix_put_tidal_logical
    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
      ! Tidal mixing scheme being used (currently only support Simmons et al)
      character(len=cvmix_strlen) :: mix_scheme

      ! efficiency is the mixing efficiency (Gamma in Simmons)
      real(cvmix_r8) :: efficiency           ! units: unitless (fraction)

      ! local_mixing_frac is the tidal dissipation efficiency (q in Simmons)
      real(cvmix_r8) :: local_mixing_frac    ! units: unitless (fraction)

      ! vertical_decay_scale is zeta in the Simmons paper (used to compute the
      ! vertical deposition function)
      real(cvmix_r8) :: vertical_decay_scale ! units: m

      ! vertical_decay_scaleR is zetar in Schmittner method (used to compute the
      ! vertical deposition function)
      real(cvmix_r8) :: vertical_decay_scaleR ! units: m

      ! depth_cutoff is depth of the shallowest column where tidal mixing is
      ! computed (like all depths, positive => below the surface)
      real(cvmix_r8) :: depth_cutoff         ! units: m

      ! max_coefficient is the largest acceptable value for diffusivity
      real(cvmix_r8) :: max_coefficient      ! units: m^2/s

      ! Flag for what to do with old values of CVmix_vars%[MTS]diff
      integer :: handle_old_vals

      ! Flag for controlling application of Schmittner Southern-Ocean mods
      logical(cvmix_log_kind)  :: ltidal_Schmittner_socn

      ! Note: need to include some logic to avoid excessive memory use
  end type cvmix_tidal_params_type
```

6.1 cvmix_init_tidal

INTERFACE

```
  subroutine cvmix_init_tidal(CVmix_tidal_params_user, mix_scheme, efficiency,&
                              vertical_decay_scale, max_coefficient,         &
                              local_mixing_frac, depth_cutoff,               &
                              ltidal_Schmittner_socn, old_vals)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
    character(len=*),        optional, intent(in) :: mix_scheme, old_vals
    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
    logical(cvmix_log_kind), optional, intent(in) :: ltidal_Schmittner_socn
```

OUTPUT PARAMETERS

```
    type(cvmix_tidal_params_type), optional, target, intent(inout) ::         &
                                          CVmix_tidal_params_user
```

6.2 cvmix_coeffs_tidal_wrap

INTERFACE

```
  subroutine cvmix_coeffs_tidal_wrap(CVmix_vars, &
                                     CVmix_params,               &
                                     CVmix_tidal_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for tidal mixing parameterizations.

INPUT PARAMETERS

```
    type(cvmix_tidal_params_type),  target, optional, intent(in) ::           &
                                        CVmix_tidal_params_user
    type(cvmix_global_params_type), intent(in) :: CVmix_params
```

INPUT/OUTPUT PARAMETERS

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

6.3 cvmix_coeffs_tidal_low

INTERFACE

```
  subroutine cvmix_coeffs_tidal_low(Mdiff_out, Tdiff_out, Nsqr, OceanDepth,   &
                                    SimmonsCoeff, vert_dep, nlev, max_nlev,   &
                                    CVmix_params,                             &
                                    SchmittnerSouthernOcean,                  &
                                    CVmix_tidal_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for tidal mixing parameterizations.

INPUT PARAMETERS

```
type(cvmix_tidal_params_type),  target, optional, intent(in) ::          &
                                         CVmix_tidal_params_user
type(cvmix_global_params_type),        intent(in) :: CVmix_params
integer,                               intent(in) :: nlev, max_nlev
real(cvmix_r8), dimension(max_nlev+1), intent(in) :: Nsqr, vert_dep
real(cvmix_r8),                        intent(in) :: OceanDepth
real(cvmix_r8),                        intent(in) :: SimmonsCoeff
real(cvmix_r8), dimension(max_nlev+1), intent(in), &
                optional                          :: SchmittnerSouthernOcean
```

INPUT/OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Mdiff_out
real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Tdiff_out
```

6.4 cvmix_coeffs_tidal_schmittner

INTERFACE

```
subroutine cvmix_coeffs_tidal_schmittner                                   &
                                        (Mdiff_out, Tdiff_out, Nsqr,        &
                                         OceanDepth, nlev, max_nlev,        &
                                         SchmittnerCoeff,                   &
                                         SchmittnerSouthernOcean,           &
                                         CVmix_params,                      &
                                         CVmix_tidal_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for tidal mixing parameterizations.

INPUT PARAMETERS

```
type(cvmix_tidal_params_type),  target, optional, intent(in) ::          &
                                         CVmix_tidal_params_user
integer,                               intent(in) :: nlev, max_nlev
type(cvmix_global_params_type),        intent(in) :: CVmix_params
real(cvmix_r8),                        intent(in) :: OceanDepth
real(cvmix_r8), dimension(max_nlev+1), intent(in) :: Nsqr
real(cvmix_r8), dimension(max_nlev+1), intent(in) :: SchmittnerSouthernOcean
real(cvmix_r8), dimension(max_nlev+1), intent(in) :: SchmittnerCoeff
```

INPUT/OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Mdiff_out
real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Tdiff_out
```

6.5 cvmix_compute_vert_dep

INTERFACE

```
function cvmix_compute_vert_dep(zw, zt, nlev, CVmix_tidal_params)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
type(cvmix_tidal_params_type),     intent(in) :: CVmix_tidal_params
integer,                           intent(in) :: nlev
real(cvmix_r8), dimension(nlev+1), intent(in) :: zw
real(cvmix_r8), dimension(nlev),   intent(in) :: zt
```

OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(nlev+1) :: cvmix_compute_vert_dep
```

6.6 cvmix_compute_vert_dep_Schmittner

INTERFACE

```
function cvmix_compute_vert_dep_Schmittner(zw, nlev, CVmix_tidal_params)
```

DESCRIPTION:

Computes the vertical deposition function needed for Schmittner 2014 tidal mixing.

INPUT PARAMETERS

```
type(cvmix_tidal_params_type),     intent(in) :: CVmix_tidal_params
integer,                           intent(in) :: nlev
real(cvmix_r8), dimension(nlev+1), intent(in) :: zw
```

OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(nlev+1) :: cvmix_compute_vert_dep_Schmittner
```

6.7 cvmix_compute_Simmons_invariant_wrap

INTERFACE

```
subroutine cvmix_compute_Simmons_invariant_wrap(CVmix_vars, CVmix_params,   &
                                                energy_flux,                &
                                                CVmix_tidal_params_user)
```

DESCRIPTION:

Compute the time-invariant portion of the tidal mixing coefficient using the Simmons, et al., scheme.

INPUT PARAMETERS

```
type(cvmix_global_params_type), intent(in) :: CVmix_params
real(cvmix_r8), intent(in) :: energy_flux
type(cvmix_tidal_params_type),  target, optional, intent(in) ::           &
                                        CVmix_tidal_params_user
```

INPUT/OUTPUT PARAMETERS

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

6.8 cvmix_compute_Simmons_invariant_low

INTERFACE

```
subroutine cvmix_compute_Simmons_invariant_low(nlev, energy_flux, rho,      &
                                               SimmonsCoeff, VertDep, zw,   &
                                               zt, CVmix_tidal_params_user)
```

DESCRIPTION:

Compute the time-invariant portion of the tidal mixing coefficient using the Simmons, et al., scheme.

INPUT PARAMETERS

```
integer,        intent(in) :: nlev
real(cvmix_r8), intent(in) :: energy_flux, rho
real(cvmix_r8), dimension(:), intent(in) :: zw, zt
type(cvmix_tidal_params_type),  target, optional, intent(in) ::           &
                                        CVmix_tidal_params_user
```

OUTPUT PARAMETERS

```
    real(cvmix_r8), intent(out) :: SimmonsCoeff
    real(cvmix_r8), dimension(nlev+1), intent(inout) :: VertDep
```

6.9 cvmix_compute_Schmittner_invariant_wrap

INTERFACE

```
  subroutine cvmix_compute_Schmittner_invariant_wrap(CVmix_vars,  &
                                                     CVmix_params,&
                                                     CVmix_tidal_params_user)
```

DESCRIPTION:

Compute the time-invariant portion of the tidal mixing coefficient using the Schmittner 2014 scheme.

INPUT PARAMETERS

```
    type(cvmix_global_params_type), intent(in) :: CVmix_params
    type(cvmix_tidal_params_type),  target, optional, intent(in) ::           &
                                          CVmix_tidal_params_user
```

INPUT/OUTPUT PARAMETERS

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

6.10 cvmix_compute_Schmittner_invariant_low

INTERFACE

```
  subroutine cvmix_compute_Schmittner_invariant_low(nlev, VertDep, efficiency, rho,  &
                                                    exp_hab_zetar, zw,               &
                                                    CVmix_tidal_params_user)
```

DESCRIPTION:

Compute the time-invariant portion of the tidal mixing coefficient using the Schmittner 2014 scheme.

INPUT PARAMETERS

```
    integer,        intent(in) :: nlev
    real(cvmix_r8), intent(in) :: efficiency
```

```
real(cvmix_r8), intent(in) :: rho
real(cvmix_r8), dimension(:), intent(in) :: zw
type(cvmix_tidal_params_type),  target, optional, intent(in) ::           &
                                        CVmix_tidal_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(1:nlev+1), intent(inout) :: VertDep
real(cvmix_r8), dimension(2:nlev+1,2:nlev+1), intent(inout) :: exp_hab_zetar
```

6.11 cvmix_compute_SchmittnerCoeff_wrap

INTERFACE

```
subroutine cvmix_compute_SchmittnerCoeff_wrap(CVmix_vars, nlev, energy_flux, &
                                              CVmix_tidal_params_user)
```

DESCRIPTION:

Compute the full time-dependent tidal mixing coefficient using the Schmittner 2014 scheme.

INPUT PARAMETERS

```
integer, intent(in)  :: nlev
real(cvmix_r8), dimension(2:nlev+1),   intent(in) :: energy_flux
type(cvmix_tidal_params_type),  target, optional, intent(in) ::           &
                                        CVmix_tidal_params_user
```

INPUT/OUTPUT PARAMETERS

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

6.12 cvmix_compute_SchmittnerCoeff_low

INTERFACE

```
subroutine cvmix_compute_SchmittnerCoeff_low(nlev, energy_flux,          &
                                             SchmittnerCoeff,            &
                                             exp_hab_zetar,              &
                                             CVmix_tidal_params_user)
```

DESCRIPTION:

Compute the time-dependent portion of the tidal mixing coefficient using the Schmittner 2014 scheme.

INPUT PARAMETERS

```
integer,        intent(in) :: nlev
real(cvmix_r8), dimension(2:nlev+1,2:nlev+1), intent(in) :: exp_hab_zetar
real(cvmix_r8), dimension(2:nlev+1),   intent(in) :: energy_flux
type(cvmix_tidal_params_type),  target, optional, intent(in) ::        &
                                        CVmix_tidal_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(:), intent(out) :: SchmittnerCoeff
```

6.13 cvmix_compute_socn_tidal_invariant_wrap

INTERFACE

```
subroutine cvmix_compute_socn_tidal_invariant_wrap(CVmix_vars,            &
                                                   CVmix_tidal_params_user)
```

DESCRIPTION:

Compute the time-invariant Schmittner Southern-Ocean tidal mixing terms

INPUT PARAMETERS

```
type(cvmix_tidal_params_type),  target, optional, intent(in) ::           &
                                        CVmix_tidal_params_user
```

INPUT/OUTPUT PARAMETERS

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

6.14 cvmix_compute_socn_tidal_invariant_low

INTERFACE

```
subroutine cvmix_compute_socn_tidal_invariant_low(nlev,                   &
                                                  lat,                    &
                                                  zw,                     &
```

```
                                                        SchmittnerSouthernOcean, &
                                                        CVmix_tidal_params_user  )
```

DESCRIPTION:

Compute the time-invariant Schmittner Southern-Ocean tidal mixing terms

INPUT PARAMETERS

```
integer,         intent(in) :: nlev
real(cvmix_r8), intent(in) :: lat
real(cvmix_r8), dimension(:), intent(in) :: zw
type(cvmix_tidal_params_type),  target, optional, intent(in) ::  &
                                        CVmix_tidal_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8),dimension(:),intent(inout) :: SchmittnerSouthernOcean
```

6.15 cvmix_put_tidal_int

INTERFACE

```
subroutine cvmix_put_tidal_int(varname, val, CVmix_tidal_params_user)
```

DESCRIPTION:

Write an integer value into a cvmix_tidal_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_tidal_params_type), optional, target, intent(inout) ::         &
                                          CVmix_tidal_params_user
```

6.16 cvmix_put_tidal_logical

INTERFACE

```
subroutine cvmix_put_tidal_logical(varname, val, CVmix_tidal_params_user)
```

DESCRIPTION:

Write a logical value into a cvmix_tidal_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_tidal_params_type), optional, target, intent(inout) ::         &
                                          CVmix_tidal_params_user
```

6.17 cvmix_put_tidal_real

INTERFACE

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

DESCRIPTION:

Write a real value into a cvmix_tidal_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_tidal_params_type), optional, target, intent(inout) ::         &
                                          CVmix_tidal_params_user
```

6.18 cvmix_put_tidal_str

INTERFACE

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

DESCRIPTION:

Write a string into a cvmix_tidal_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_tidal_params_type), optional, target, intent(inout) ::          &
                                          CVmix_tidal_params_user
```

6.19 cvmix_get_tidal_real

INTERFACE

```
function cvmix_get_tidal_real(varname, CVmix_tidal_params_user)
```

DESCRIPTION:

Returns the real value of a cvmix_tidal_params_type variable.

INPUT PARAMETERS

```
character(len=*),                               intent(in) :: varname
type(cvmix_tidal_params_type), optional, target, intent(in) ::            &
                                       CVmix_tidal_params_user
```

OUTPUT PARAMETERS

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

6.20 cvmix_get_tidal_str

INTERFACE

```
function cvmix_get_tidal_str(varname, CVmix_tidal_params_user)
```

DESCRIPTION:

Returns the string value of a cvmix_tidal_params_type variable.

INPUT PARAMETERS

```
character(len=*),                                 intent(in) :: varname
type(cvmix_tidal_params_type), optional, target, intent(in) ::            &
                                            CVmix_tidal_params_user
```

OUTPUT PARAMETERS

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

7 Module cvmix_ddiff

AUTHOR

```
Michael N. Levy, NCAR (mlevy@ucar.edu)
```

DESCRIPTION:

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

References:
* RW Schmitt. Double Diffusion in Oceanography. Annual Review of Fluid Mechanics, 1994.
* WG Large, JC McWilliams, and SC Doney. Oceanic Vertical Mixing: A Review and a Model with a Nonlocal Boundary Layer Parameterization. Review of Geophysics, 1994.
* G Danabasoglu, WG Large, JJ Tribbia, PR Gent, BP Briegleb, and JC McWilliams. Diurnal Coupling in the Tropical Oceans of CCSM3. Journal of Climate, 2006.

USES

```
use cvmix_kinds_and_types, only : cvmix_r8,                               &
                                  cvmix_strlen,                           &
                                  cvmix_zero,                             &
                                  cvmix_one,                              &
                                  cvmix_data_type,                        &
                                  CVMIX_OVERWRITE_OLD_VAL,                &
                                  CVMIX_SUM_OLD_AND_NEW_VALS,             &
                                  CVMIX_MAX_OLD_AND_NEW_VALS
use cvmix_put_get,         only : cvmix_put
use cvmix_utils,           only : cvmix_update_wrap
```

PUBLIC MEMBER FUNCTIONS

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

interface cvmix_coeffs_ddiff
  module procedure cvmix_coeffs_ddiff_low
  module procedure cvmix_coeffs_ddiff_wrap
end interface cvmix_coeffs_ddiff

interface cvmix_put_ddiff
  module procedure cvmix_put_ddiff_str
  module procedure cvmix_put_ddiff_real
```

```
    module procedure cvmix_put_ddiff_int
  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
      ! Max value of the stratification parameter (diffusivity = 0 for values
      ! that exceed this constant). R_p^0 in LMD94.
      real(cvmix_r8) :: strat_param_max    ! units: unitless

      ! Type of diffusive convection to use
      ! Options are Marmorino and Caldwell 1976 ("MC76"; default)
      ! and Kelley 1988, 1990 ("K90")
      character(len=cvmix_strlen) :: diff_conv_type

      ! leading coefficient in formula for salt-fingering regime for salinity
      ! diffusion (nu_f in LMD94, kappa_0 in Gokhan's paper)
      real(cvmix_r8) :: kappa_ddiff_s      ! units: m^2/s

      ! interior exponent in salt-fingering regime formula (2 in LMD94, 1 in
      ! Gokhan's paper)
      real(cvmix_r8) :: ddiff_exp1         ! units: unitless

      ! exterior exponent in salt-fingering regime formula (p2 in LMD94, 3 in
      ! Gokhan's paper)
      real(cvmix_r8) :: ddiff_exp2         ! units: unitless

      ! Exterior coefficient in diffusive convection regime (0.909 in LMD94)
      real(cvmix_r8) :: kappa_ddiff_param1 ! units: unitless

      ! Middle coefficient in diffusive convection regime (4.6 in LMD94)
      real(cvmix_r8) :: kappa_ddiff_param2 ! units: unitless

      ! Interior coefficient in diffusive convection regime (-0.54 in LMD94)
      real(cvmix_r8) :: kappa_ddiff_param3 ! units: unitless

      ! Molecular diffusivity (leading coefficient in diffusive convection
      real(cvmix_r8) :: mol_diff           ! units: m^2/s

      ! Flag for what to do with old values of CVmix_vars%[MTS]diff
      integer :: handle_old_vals

  end type cvmix_ddiff_params_type
```

7.1 cvmix_init_ddiff

INTERFACE

```
  subroutine cvmix_init_ddiff(CVmix_ddiff_params_user, strat_param_max,        &
                              kappa_ddiff_s, ddiff_exp1, ddiff_exp2, mol_diff, &
                              kappa_ddiff_param1, kappa_ddiff_param2,          &
                              kappa_ddiff_param3, diff_conv_type, old_vals)
```

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

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

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

$$\kappa^0 = \begin{cases} 7 \cdot 10^{-5} \text{ m}^2\text{/s} & \text{for temperature } (0.7 \cdot \texttt{kappa\_ddiff\_s} \text{ in this routine}) \\ 10^{-4} \text{ m}^2\text{/s} & \text{for salinity and other tracers } (\texttt{kappa\_ddiff\_s} \text{ in this routine}). \end{cases}$$

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

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

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)`. Note that CVMix assumes units are —'mks'—.

INPUT PARAMETERS

```
  real(cvmix_r8),   optional, intent(in) :: strat_param_max,               &
                                            kappa_ddiff_s,                 &
                                            ddiff_exp1,                    &
                                            ddiff_exp2,                    &
                                            mol_diff,                      &
                                            kappa_ddiff_param1,            &
                                            kappa_ddiff_param2,            &
                                            kappa_ddiff_param3
  character(len=*), optional, intent(in) :: diff_conv_type, old_vals
```

OUTPUT PARAMETERS

```
  type(cvmix_ddiff_params_type), optional, target, intent(inout) ::         &
                                            CVmix_ddiff_params_user
```

7.2 cvmix_coeffs_ddiff

INTERFACE

```
subroutine cvmix_coeffs_ddiff_wrap(CVmix_vars, CVmix_ddiff_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for the double diffusion mixing parameterization.

INPUT PARAMETERS

```
  type(cvmix_ddiff_params_type), optional, target, intent(in) ::            &
                                         CVmix_ddiff_params_user
```

INPUT/OUTPUT PARAMETERS

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

7.3 cvmix_coeffs_ddiff_low

INTERFACE

```
subroutine cvmix_coeffs_ddiff_low(Tdiff_out, Sdiff_out, strat_param_num,    &
                                  strat_param_denom, nlev, max_nlev,        &
                                  CVmix_ddiff_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for the double diffusion mixing parameterization.

INPUT PARAMETERS

```
type(cvmix_ddiff_params_type), optional, target, intent(in) ::            &
                                       CVmix_ddiff_params_user
integer,                              intent(in) :: nlev, max_nlev
real(cvmix_r8), dimension(max_nlev), intent(in) :: strat_param_num,       &
                                                    strat_param_denom
```

INPUT/OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Tdiff_out,        &
                                                        Sdiff_out
```

LOCAL VARIABLES

```
integer :: k
real(cvmix_r8) :: ddiff, Rrho
```

7.4 cvmix_put_ddiff_str

INTERFACE

```
subroutine cvmix_put_ddiff_str(varname, val, CVmix_ddiff_params_user)
```

DESCRIPTION:

Write a string value into a cvmix_ddiff_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_ddiff_params_type), optional, target, intent(inout) ::         &
                                          CVmix_ddiff_params_user
```

7.5 cvmix_put_ddiff_real

INTERFACE

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

DESCRIPTION:

Write a real value into a cvmix_ddiff_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_ddiff_params_type), optional, target, intent(inout) ::         &
                                             CVmix_ddiff_params_user
```

7.6 cvmix_put_ddiff_int

INTERFACE

```
subroutine cvmix_put_ddiff_int(varname, val, CVmix_ddiff_params_user)
```

DESCRIPTION:

Write an integer value into a cvmix_ddiff_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_ddiff_params_type), optional, target, intent(inout) ::         &
                                             CVmix_ddiff_params_user
```

7.7 cvmix_get_ddiff_real

INTERFACE

```
function cvmix_get_ddiff_real(varname, CVmix_ddiff_params_user)
```

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.

INPUT PARAMETERS

```
character(len=*),                                intent(in)    :: varname
type(cvmix_ddiff_params_type), optional, target, intent(inout) ::         &
                                            CVmix_ddiff_params_user
```

OUTPUT PARAMETERS

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

8 Module cvmix_kpp

AUTHOR

Michael N. Levy, NCAR (mlevy@ucar.edu)

DESCRIPTION:

This module contains routines to initialize the derived types needed for KPP mixing and to set the viscosity and diffusivity coefficients accordingly.

References:
* WG Large, JC McWilliams, and SC Doney. Oceanic Vertical Mixing: A Review and a Model with a Nonlocal Boundary Layer Parameterization. Review of Geophysics, 1994.

USES

```
use cvmix_kinds_and_types, only : cvmix_r8,                                 &
                                  cvmix_strlen,                             &
                                  cvmix_zero,                               &
                                  cvmix_one,                                &
                                  cvmix_PI,                                 &
                                  cvmix_data_type,                          &
                                  cvmix_global_params_type,                 &
                                  CVMIX_OVERWRITE_OLD_VAL,                  &
                                  CVMIX_SUM_OLD_AND_NEW_VALS,               &
                                  CVMIX_MAX_OLD_AND_NEW_VALS
use cvmix_math, only :            CVMIX_MATH_INTERP_LINEAR,                 &
                                  CVMIX_MATH_INTERP_QUAD,                   &
                                  CVMIX_MATH_INTERP_CUBE_SPLINE,            &
                                  cvmix_math_poly_interp,                   &
                                  cvmix_math_cubic_root_find,               &
                                  cvmix_math_evaluate_cubic
use cvmix_put_get,         only : cvmix_put
use cvmix_utils,           only : cvmix_update_wrap
```

DEFINED PARAMETERS

```
integer, parameter :: CVMIX_KPP_INTERP_LMD94       = -1
integer, parameter :: CVMIX_KPP_MATCH_BOTH         = 1
integer, parameter :: CVMIX_KPP_MATCH_GRADIENT     = 2
integer, parameter :: CVMIX_KPP_SIMPLE_SHAPES      = 3
integer, parameter :: CVMIX_KPP_PARABOLIC_NONLOCAL = 4
integer, parameter :: NO_LANGMUIR_MIXING           = -1
integer, parameter :: LANGMUIR_MIXING_LWF16        = 1
integer, parameter :: LANGMUIR_MIXING_RWHGK16      = 2
integer, parameter :: NO_LANGMUIR_ENTRAINMENT      = -1
```

```
integer, parameter :: LANGMUIR_ENTRAINMENT_LWF16   = 1
integer, parameter :: LANGMUIR_ENTRAINMENT_LF17    = 2
integer, parameter :: LANGMUIR_ENTRAINMENT_RWHGK16 = 3
```

PUBLIC MEMBER FUNCTIONS

```
public :: cvmix_init_kpp
! Note: cvmix_kpp_compute_OBL_depth would be part of cvmix_coeffs_kpp but
!       CVMix can not smooth the boundary layer depth or correct the
!       buoyancy flux term
public :: cvmix_kpp_compute_OBL_depth
public :: cvmix_coeffs_kpp
public :: cvmix_put_kpp
public :: cvmix_get_kpp_real
public :: cvmix_kpp_compute_bulk_Richardson
public :: cvmix_kpp_compute_turbulent_scales
public :: cvmix_kpp_compute_unresolved_shear
public :: cvmix_kpp_composite_Gshape                     !STOKES_MOST
public :: cvmix_kpp_compute_StokesXi                    !STOKES_MOST
! These are public for testing, may end up private later
public :: cvmix_kpp_compute_shape_function_coeffs
public :: cvmix_kpp_compute_kOBL_depth
public :: cvmix_kpp_compute_enhanced_diff
public :: cvmix_kpp_compute_nu_at_OBL_depth_LMD94
public :: cvmix_kpp_EFactor_model
public :: cvmix_kpp_ustokes_SL_model


interface cvmix_coeffs_kpp
  module procedure cvmix_coeffs_kpp_low
  module procedure cvmix_coeffs_kpp_wrap
end interface cvmix_coeffs_kpp

interface cvmix_put_kpp
  module procedure cvmix_put_kpp_int
  module procedure cvmix_put_kpp_real
  module procedure cvmix_put_kpp_logical
end interface cvmix_put_kpp

interface cvmix_kpp_compute_OBL_depth
  module procedure cvmix_kpp_compute_OBL_depth_low
  module procedure cvmix_kpp_compute_OBL_depth_wrap
end interface cvmix_kpp_compute_OBL_depth

interface cvmix_kpp_compute_turbulent_scales
  module procedure cvmix_kpp_compute_turbulent_scales_0d
  module procedure cvmix_kpp_compute_turbulent_scales_1d_sigma
  module procedure cvmix_kpp_compute_turbulent_scales_1d_OBL
```

```
end interface cvmix_kpp_compute_turbulent_scales
```

PUBLIC TYPES

```
! cvmix_kpp_params_type contains the necessary parameters for KPP mixing
type, public :: cvmix_kpp_params_type
  private
    real(cvmix_r8) :: Ri_crit         ! Critical Richardson number
                                      ! (OBL_depth = where bulk Ri = Ri_crit)

    real(cvmix_r8) :: minOBLdepth     ! Minimum allowable OBL depth
                                      ! (Default is 0 m => no minimum)
    real(cvmix_r8) :: maxOBLdepth     ! Maximum allowable OBL depth
                                      ! (Default is 0 m => no maximum)
    real(cvmix_r8) :: minVtsqr        ! Minimum allowable unresolved shear
                                      ! (Default is 1e-10 m^2/s^2)

    real(cvmix_r8) :: vonkarman       ! von Karman constant

    real(cvmix_r8) :: Cstar           ! coefficient for nonlinear transport
    real(cvmix_r8) :: nonlocal_coeff ! Cs from Eq (20) in LMD94
                                      ! Default value comes from paper, but
                                      ! some users may set it = 1.

    ! For velocity scale function, _m => momentum and _s => scalar (tracer)
    real(cvmix_r8) :: zeta_m          ! parameter for computing vel scale func
    real(cvmix_r8) :: zeta_s          ! parameter for computing vel scale func
    real(cvmix_r8) :: a_m             ! parameter for computing vel scale func
    real(cvmix_r8) :: c_m             ! parameter for computing vel scale func
    real(cvmix_r8) :: a_s             ! parameter for computing vel scale func
    real(cvmix_r8) :: c_s             ! parameter for computing vel scale func

    real(cvmix_r8) :: surf_layer_ext ! nondimensional extent of surface layer
                                      ! (expressed in sigma-coordinates)

    integer        :: interp_type     ! interpolation type used to interpolate
                                      ! bulk Richardson number
    integer        :: interp_type2    ! interpolation type used to interpolate
                                      ! diff and visc at OBL_depth

    ! Cv is a parameter used to compute the unresolved shear. By default, the
    ! formula from Eq. (A3) of Danabasoglu et al. is used, but a single
    ! scalar value can be set instead.
    real(cvmix_r8) :: Cv

    ! MatchTechnique is set by a string of the same name as an argument in
    ! cvmix_init_kpp. It determines how matching between the boundary layer
    ! and ocean interior is handled at the interface. Note that this also
    ! controls whether the shape function used to compute the coefficient in
    ! front of the nonlocal term is the same as that used to compute the
    ! gradient term.
    ! Options (for cvmix_init_kpp) are
    ! (i) SimpleShapes => Shape functions for both the gradient and nonlocal
    !                     terms vanish at interface
    ! (ii) MatchGradient => Shape function for nonlocal term vanishes at
    !                       interface, but gradient term matches interior
    ! (iii) MatchBoth => Shape functions for both the gradient and nonlocal
    !                    term match interior values at interface
    ! (iv) ParabolicNonLocal => Shape function for the nonlocal term is
    !                         (1-sigma)^2, gradient term is sigma*(1-sigma)^2
    integer :: MatchTechnique

    ! Flag for what to do with old values of CVmix_vars%[MTS]diff
    integer :: handle_old_vals

    ! Logic flags to dictate if / how various terms are computed
    logical         :: lStokesMOST    ! True => use Stokes Similarty package
    logical         :: lscalar_Cv     ! True => use the scalar Cv value
    logical         :: lEkman         ! True => compute Ekman depth limit
    logical         :: lMonOb         ! True => compute Monin-Obukhov limit
    logical         :: lnoDGat1       ! True => G'(1) = 0 (shape function)
                                      ! False => compute G'(1) as in LMD94
    logical         :: lenhanced_diff ! True => enhance diffusivity at OBL
    integer         :: Langmuir_Mixing_Opt
                                      ! Option of Langmuir enhanced mixing
                                      ! - apply an enhancement factor to the
                                      ! turbulent velocity scale
    integer         :: Langmuir_Entrainment_Opt
                                      ! Option of Langmuir turbulence enhanced
                                      ! entrainment - modify the unresolved shear
    logical         :: l_LMD_ws       ! flag to use original Large et al. (1994)
                                      ! equations for computing turbulent scales
                                      ! rather than the updated methodology in
                                      ! Danabasoglu et al. (2006). The latter
                                      ! limits sigma to be < surf_layer_extent
                                      ! when computing turbulent scales while
                                      ! the former only imposes this restriction
                                      ! in unstable regimes.
    real(cvmix_r8) :: c_LT, c_ST, c_CT  ! Empirical constants in the scaling of the
                                        ! entrainment buoyancy flux
                                        ! (20) in Li and Fox-Kemper, 2017, JPO
    real(cvmix_r8) :: p_LT              ! Power of Langmuir number in the above
    !BGR
    real(cvmix_r8) :: RWHGK_ENTR_COEF,& ! Coefficient and exponent from
                      RWHGK_ENTR_EXP    ! RWHGK16 Langmuir parameterization
```

```
   end type cvmix_kpp_params_type
```

8.1 cvmix_init_kpp

INTERFACE

```
   subroutine cvmix_init_kpp(ri_crit, minOBLdepth, maxOBLdepth, minVtsqr,      &
                             vonkarman, Cstar, zeta_m, zeta_s, surf_layer_ext, &
                             Cv, interp_type, interp_type2, MatchTechnique,    &
                             old_vals, lEkman, lStokesMOST, lMonOb, lnoDGat1,  &
                             lenhanced_diff, lnonzero_surf_nonlocal,           &
                             Langmuir_mixing_str, Langmuir_entrainment_str,    &
                             l_LMD_ws, CVmix_kpp_params_user)
```

DESCRIPTION:

Initialization routine for KPP mixing.

INPUT PARAMETERS

```
     real(cvmix_r8),   optional, intent(in) :: ri_crit,                        &
                                               minOBLdepth,                    &
                                               maxOBLdepth,                    &
                                               minVtsqr,                       &
                                               vonkarman,                      &
                                               Cstar,                          &
                                               zeta_m,                         &
                                               zeta_s,                         &
                                               surf_layer_ext,                 &
                                               Cv
     character(len=*), optional, intent(in) :: interp_type,                    &
                                               interp_type2,                   &
                                               MatchTechnique,                 &
                                               old_vals,                       &
                                               Langmuir_mixing_str,            &
                                               Langmuir_entrainment_str
     logical,          optional, intent(in) :: lEkman,                         &
                                               lStokesMOST,                    &
                                               lMonOb,                         &
                                               lnoDGat1,                       &
                                               lenhanced_diff,                 &
                                               lnonzero_surf_nonlocal,         &
                                               l_LMD_ws
```

OUTPUT PARAMETERS

```
    type(cvmix_kpp_params_type), intent(inout), target, optional ::           &
                                                   CVmix_kpp_params_user
```

8.2 cvmix_coeffs_kpp_wrap

INTERFACE

```
  subroutine cvmix_coeffs_kpp_wrap(CVmix_vars, CVmix_kpp_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for the KPP boundary layer mixing parameterization.

INPUT PARAMETERS

```
    type(cvmix_kpp_params_type), intent(in), optional, target ::              &
                                                CVmix_kpp_params_user
```

INPUT/OUTPUT PARAMETERS

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

8.3 cvmix_coeffs_kpp_low

INTERFACE

```
  subroutine cvmix_coeffs_kpp_low(Mdiff_out, Tdiff_out, Sdiff_out, zw, zt,    &
                                  old_Mdiff, old_Tdiff, old_Sdiff, OBL_depth, &
                                  kOBL_depth, Tnonlocal, Snonlocal, surf_fric,&
                                  surf_buoy, nlev, max_nlev, Langmuir_EFactor,&
                                  StokesXI,CVmix_kpp_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for the KPP boundary layer mixing parameterization.

INPUT PARAMETERS

```
    type(cvmix_kpp_params_type),  intent(in), optional, target ::             &
                                                  CVmix_kpp_params_user
```

```
integer,                               intent(in) :: nlev, max_nlev
real(cvmix_r8), dimension(max_nlev+1), intent(in) :: old_Mdiff,          &
                                                     old_Tdiff,          &
                                                     old_Sdiff,          &
                                                     zw
real(cvmix_r8), dimension(max_nlev),   intent(in) :: zt
real(cvmix_r8),                        intent(in) :: OBL_depth,          &
                                                     surf_fric,          &
                                                     surf_buoy,          &
                                                     kOBL_depth
! Langmuir enhancement factor
real(cvmix_r8), intent(in), optional :: Langmuir_EFactor
real(cvmix_r8), intent(in), optional :: StokesXI
```

INPUT/OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Mdiff_out,        &
                                                        Tdiff_out,        &
                                                        Sdiff_out,        &
                                                        Tnonlocal,        &
                                                        Snonlocal
```

8.4 cvmix_put_kpp_real

INTERFACE

```
subroutine cvmix_put_kpp_real(varname, val, CVmix_kpp_params_user)
```

DESCRIPTION:

Write a real value into a cvmix_kpp_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_kpp_params_type), intent(inout), target, optional ::          &
                                          CVmix_kpp_params_user
```

8.5 cvmix_put_kpp_int

INTERFACE

```
subroutine cvmix_put_kpp_int(varname, val, CVmix_kpp_params_user)
```

DESCRIPTION:

Write an integer value into a cvmix_kpp_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_kpp_params_type), intent(inout), target, optional ::          &
                                          CVmix_kpp_params_user
```

8.6 cvmix_put_kpp_logical

INTERFACE

```
subroutine cvmix_put_kpp_logical(varname, val, CVmix_kpp_params_user)
```

DESCRIPTION:

Write a Boolean value into a cvmix_kpp_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type(cvmix_kpp_params_type), intent(inout), target, optional ::          &
                                          CVmix_kpp_params_user
```

8.7 cvmix_get_kpp_real

INTERFACE

```
function cvmix_get_kpp_real(varname, CVmix_kpp_params_user)
```

DESCRIPTION:

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

INPUT PARAMETERS

```
character(len=*),                              intent(in) :: varname
type(cvmix_kpp_params_type), optional, target, intent(in) ::              &
                                           CVmix_kpp_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8) :: cvmix_get_kpp_real
```

8.8 cvmix_kpp_compute_OBL_depth_low

INTERFACE

```
subroutine cvmix_kpp_compute_OBL_depth_low(Ri_bulk, zw_iface, OBL_depth,    &
                                           kOBL_depth, zt_cntr, surf_fric,  &
                                           surf_buoy, Coriolis,Xi, zBottom, &
                                           CVmix_kpp_params_user)
```

DESCRIPTION:

Computes the depth of the ocean boundary layer (`OBL_depth`) for a given column. Ri_bulk(h) = Ricr; h ¡ -zBottom, (stable+lMonOb) 0 ¡ h ¡ vonKaraman Lstar

INPUT PARAMETERS

```
real(cvmix_r8), dimension(:),                    intent(in) :: Ri_bulk
real(cvmix_r8), dimension(:),            target, intent(in) :: zw_iface,   &
                                                               zt_cntr
real(cvmix_r8), dimension(:), optional,  intent(in) ::  Xi, surf_buoy
real(cvmix_r8),               optional,          intent(in) :: surf_fric,  &
                                                               Coriolis,   &
                                                               zBottom
type(cvmix_kpp_params_type),  optional, target, intent(in) ::              &
                                        CVmix_kpp_params_user
```

OUTPUT PARAMETERS

```
    real(cvmix_r8),               intent(out) :: OBL_depth, kOBL_depth
```

8.9 cvmix_kpp_compute_kOBL_depth

INTERFACE

```
  function cvmix_kpp_compute_kOBL_depth(zw_iface, zt_cntr, OBL_depth)
```

DESCRIPTION:

Computes the index of the level and interface above `OBL_depth`. The index is stored as a real number, and the integer index can be solved for in the following way:
`kt` = index of cell center above `OBL_depth` = `nint(kOBL_depth)-1` `kw` = index of interface above `OBL_depth` = `floor(kOBL_depth)`

INPUT PARAMETERS

```
    real(cvmix_r8), dimension(:), intent(in) :: zw_iface, zt_cntr
    real(cvmix_r8), intent(in)  :: OBL_depth
```

OUTPUT PARAMETERS

```
    real(cvmix_r8) :: cvmix_kpp_compute_kOBL_depth
```

8.10 cvmix_kpp_compute_enhanced_diff

INTERFACE

```
  subroutine cvmix_kpp_compute_enhanced_diff(Mdiff_ktup, Tdiff_ktup,          &
                                             Sdiff_ktup, Mdiff, Tdiff, Sdiff, &
                                             OBL_Mdiff, OBL_Tdiff, OBL_Sdiff, &
                                             Tnonlocal, Snonlocal,            &
                                             delta, lkteqkw)
```

DESCRIPTION:

The enhanced mixing described in Appendix D of LMD94 changes the diffusivity values at the interface between the cell center above `OBL_depth` and the one below it, based on a weighted average of how close to each center `OBL_depth` is. Note that we need to know whether `OBL_depth` is above this interface or below it - we do this by comparing the indexes of the cell center above `OBL_depth` (`ktup`) and the cell interface above `OBL_depth`(`kwup`).

INPUT PARAMETERS

```
! Diffusivity and viscosity at cell center above OBL_depth
real(cvmix_r8), intent(in) :: Mdiff_ktup, Tdiff_ktup, Sdiff_ktup

! Weight to use in averaging (distance between OBL_depth and cell center
! above OBL_depth divided by distance between cell centers bracketing
real(cvmix_r8), intent(in) :: delta

logical, intent(in) :: lkteqkw ! .true.  => interface ktup+1 is outside OBL
                               !            (update diff and visc)
                               ! .false. => interface ktup+1 is inside OBL
                               !            (update OBL_diff and OBL_visc)
```

OUTPUT PARAMETERS

```
! Will change either diff & visc or OBL_diff & OBL_visc, depending on value
! of lkteqkw
real(cvmix_r8), intent(inout) :: Mdiff, Tdiff, Sdiff,                    &
                                 OBL_Mdiff, OBL_Tdiff, OBL_Sdiff,         &
                                 Tnonlocal, Snonlocal
```

8.11 cvmix_kpp_compute_OBL_depth_wrap

INTERFACE

```
subroutine cvmix_kpp_compute_OBL_depth_wrap(CVmix_vars, CVmix_kpp_params_user)
```

DESCRIPTION:

Computes the depth of the ocean boundary layer (`CVmix_vars%BoundaryLayerDepth`) for a given column.

INPUT PARAMETERS

```
type(cvmix_kpp_params_type), optional, target, intent(in) ::              &
                                        CVmix_kpp_params_user
```

OUTPUT PARAMETERS

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

8.12 cvmix_kpp_compute_bulk_Richardson

INTERFACE

```
function cvmix_kpp_compute_bulk_Richardson(zt_cntr, delta_buoy_cntr,        &
                                           delta_Vsqr_cntr, Vt_sqr_cntr,    &
                                           ws_cntr, N_iface, Nsqr_iface,    &
                                           EFactor, LaSL, bfsfc, ustar,     &
                                           CVmix_kpp_params_user)
```

DESCRIPTION:

Computes the bulk Richardson number at cell centers. If `Vt_sqr_cntr` is not present, this routine will call `compute_unresolved_shear`, a routine that requires `ws_cntr` and either `N_iface` or `Nsqr_iface`.

INPUT PARAMETERS

```
! * zt_cntr is level-center height (d in LMD94, units: m)
! * delta_buoy_cntr is the mean buoyancy estimate over surface layer minus
!   the level-center buoyancy ( (Br-B(d)) in LMD94, units: m/s^2)
! * delta_Vsqr_cntr is the square of the magnitude of the mean velocity
!   estimate over surface layer minus the level-center velocity
!   ( |Vr-V(d)|^2 in LMD94, units: m^2/s^2)
real(cvmix_r8), dimension(:), intent(in) :: zt_cntr, delta_buoy_cntr,     &
                                            delta_Vsqr_cntr
! * ws_cntr: w_s (turbulent scale factor) at center of cell (units: m/s)
! * N_iface: buoyancy frequency at interfaces (units: 1/s)
! * Nsqr_iface: squared buoyancy frequency at interfaces (units: 1/s^2)
! * Vt_sqr_cntr: squared unresolved shear term (units m^2/s^2)
! See note in description about what values should be passed in
! * bfsfc: surface buoyancy flux (units: m^2/s^3)
real(cvmix_r8), dimension(size(zt_cntr)), intent(in), optional ::         &
                                          bfsfc, ws_cntr, Vt_sqr_cntr
real(cvmix_r8), dimension(size(zt_cntr)+1), intent(in), optional ::       &
                                                N_iface, Nsqr_iface
! * EFactor: Langmuir enhancement factor for entrainment (units: none)
! * LaSL: surface layer averaged Langmuir number (units: none)
! * ustar: friction velocity (units: m/s)
real(cvmix_r8), intent(in), optional :: EFactor, LaSL, ustar
type(cvmix_kpp_params_type), intent(in), optional, target ::              &
                                        CVmix_kpp_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(size(zt_cntr)) ::                               &
                          cvmix_kpp_compute_bulk_Richardson
```

8.13 cvmix_kpp_compute_turbulent_scales_0d

INTERFACE

```
subroutine cvmix_kpp_compute_turbulent_scales_0d(sigma_coord, OBL_depth,    &
                                                   surf_buoy_force,          &
                                                   surf_fric_vel,            &
                                                   xi, w_m, w_s,             &
                                                   CVmix_kpp_params_user)
```

DESCRIPTION:

Computes the turbulent velocity scales for momentum (`w_m`) and scalars (`w_s`) at a single σ coordinate.

INPUT PARAMETERS

```
real(cvmix_r8), intent(in) :: sigma_coord
real(cvmix_r8), intent(in) :: OBL_depth, surf_buoy_force, surf_fric_vel
real(cvmix_r8), optional, intent(in) :: xi
type(cvmix_kpp_params_type), intent(in), optional, target ::              &
                                        CVmix_kpp_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8), optional, intent(inout) :: w_m
real(cvmix_r8), optional, intent(inout) :: w_s
```

8.14 cvmix_kpp_compute_turbulent_scales_1d

INTERFACE

```
subroutine cvmix_kpp_compute_turbulent_scales_1d_sigma(sigma_coord,         &
                                                         OBL_depth,           &
                                                         surf_buoy_force,     &
                                                         surf_fric_vel, xi,   &
                                                         w_m, w_s,            &
                                                         CVmix_kpp_params_user)
```

DESCRIPTION:

Computes the turbulent velocity scales for momentum (`w_m`) and scalars (`w_s`) given a 1d array of σ coordinates. Note that the turbulent scales are a continuous function, so there is no restriction to only evaluating this routine at interfaces or cell centers. Also, if $\sigma >$ `surf_layer_ext` (which is typically 0.1), `w_m` and `w_s` will be evaluated at the latter value.

INPUT PARAMETERS

```
real(cvmix_r8), dimension(:), intent(in) :: sigma_coord
real(cvmix_r8), intent(in) :: OBL_depth, surf_buoy_force, surf_fric_vel
real(cvmix_r8), optional, intent(in)           :: xi
type(cvmix_kpp_params_type), intent(in), optional, target ::             &
                                       CVmix_kpp_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8), optional, dimension(size(sigma_coord)), intent(inout) ::  &
                                                              w_m, w_s
```

8.15 cvmix_kpp_compute_turbulent_scales_1d_OBL

INTERFACE

```
subroutine cvmix_kpp_compute_turbulent_scales_1d_OBL(sigma_coord,          &
                                                     OBL_depth,            &
                                                     surf_buoy_force,      &
                                                     surf_fric_vel,        &
                                                     xi, w_m, w_s,         &
                                                     CVmix_kpp_params_user)
```

DESCRIPTION:

Computes the turbulent velocity scales for momentum (w_m) and scalars (w_s) given a single σ coordinate and an array of boundary layer depths. Note that the turbulent scales are a continuous function, so they are evaluated at sigma_coord * OBL_depth(z) using surf_buoy_force(z)

INPUT PARAMETERS

```
real(cvmix_r8), intent(in) :: sigma_coord
real(cvmix_r8), intent(in) :: surf_fric_vel
real(cvmix_r8), dimension(:), intent(in) ::  surf_buoy_force, OBL_depth
real(cvmix_r8), dimension(:), intent(in), optional :: xi
type(cvmix_kpp_params_type), intent(in), optional, target ::             &
                                       CVmix_kpp_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8), optional, dimension(size(surf_buoy_force)), intent(inout) &
                                                          :: w_m, w_s
```

8.16 cvmix_kpp_compute_unresolved_shear

INTERFACE

```
function cvmix_kpp_compute_unresolved_shear(zt_cntr, ws_cntr, N_iface,      &
                                             Nsqr_iface, EFactor,            &
                                             LaSL, bfsfc, ustar,             &
                                             CVmix_kpp_params_user)
```

DESCRIPTION:

Computes the square of the unresolved shear (V_t^2 in Eq. (23) of LMD94) at cell centers. Note that you must provide either the buoyancy frequency or its square at cell interfaces, this routine by default will use the lower cell interface value as the cell center, but you can instead take an average of the top and bottom interface values by setting lavg_N_or_Nsqr = .true. in cvmix_kpp_init(). If you pass in Nsqr then negative values are assumed to be zero (default POP behavior).

INPUT PARAMETERS

```
! zt_cntr: height at center of cell (units: m)
! ws_cntr: w_s (turbulent scale factor) at center of cell (units: m/s)
real(cvmix_r8), dimension(:), intent(in) :: zt_cntr,  ws_cntr
! N_iface: buoyancy frequency at cell interfaces (units: 1/s)
! Nsqr_iface: squared buoyancy frequency at cell interfaces (units: 1/s^2)
! note that you must provide exactly one of these two inputs!
real(cvmix_r8), dimension(size(zt_cntr)+1), intent(in), optional ::      &
                                                     N_iface, Nsqr_iface
! bfsfc: surface buoyancy flux above cell centers (units: m^2/s^3)
real(cvmix_r8), dimension(size(zt_cntr)), intent(in), optional :: bfsfc
! EFactor: Langmuir enhancement factor (units: none)
! LaSL: surface layer averaged Langmuir number (units: none)
! ustar: friction velocity (units: m/s)
real(cvmix_r8), intent(in), optional :: EFactor, LaSL, ustar
type(cvmix_kpp_params_type),  intent(in), optional, target ::            &
                                     CVmix_kpp_params_user
```

OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(size(zt_cntr)) ::                              &
                          cvmix_kpp_compute_unresolved_shear
```

8.17 compute_phi_inv

INTERFACE

```
function compute_phi_inv(zeta, CVmix_kpp_params_in, L_Lstokes, lphi_m, lphi_s)
```

DESCRIPTION:

Computes $\frac{1}{\phi_m}$ or $\frac{1}{\phi_s}$

INPUT PARAMETERS

```
real(cvmix_r8),               intent(in) :: zeta
type(cvmix_kpp_params_type), intent(in) :: CVmix_kpp_params_in
real(cvmix_r8), optional,    intent(in) :: L_Lstokes
logical, optional,           intent(in) :: lphi_m, lphi_s
```

OUTPUT PARAMETERS

```
real(cvmix_r8) :: compute_phi_inv
```

8.18 compute_Stokes_chi

INTERFACE

```
function compute_Stokes_chi( xi, lchi_m, lchi_s)
```

DESCRIPTION:

Compute Stokes similarity function chi, of Stokes parameter xi= Ps/(PU+PS+PB)

INPUT PARAMETERS

```
real(cvmix_r8),              intent(in) :: xi
logical, optional,           intent(in) :: lchi_m, lchi_s
```

OUTPUT PARAMETERS

```
real(cvmix_r8) :: compute_Stokes_chi
```

8.19 cvmix_kpp_compute_shape_function_coeffs

INTERFACE

```
subroutine cvmix_kpp_compute_shape_function_coeffs(GAT1, DGAT1, coeffs)
```

DESCRIPTION:

Computes the coefficients of the shape function $G(\sigma) = a_0 + a_1\sigma + a_2\sigma^2 + a_3\sigma^3$, where

$$\begin{aligned} a_0 &= 0 \\ a_1 &= 1 \\ a_2 &= 3G(1) - G'(1) - 2 \\ a_3 &= -2G(1) + G'(1) + 1 \end{aligned}$$

Note that $G(1)$ and $G'(1)$ come from Eq. (18) in Large, et al., and this routine returns coeffs(1:4) = $(/a_0, a_1, a_2, a_3/)$

INPUT PARAMETERS

```
real(cvmix_r8), intent(in) :: GAT1  ! G(1)
real(cvmix_r8), intent(in) :: DGAT1 ! G'(1)
```

OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(4), intent(inout) :: coeffs
```

8.20 cvmix_compute_nu_at_OBL_depth_LMD94

INTERFACE

```
function cvmix_kpp_compute_nu_at_OBL_depth_LMD94(depths_cntr, layer_widths, &
                                                  diffs_iface, OBL_depth,    &
                                                  diff_2above, dnu_dz)
```

DESCRIPTION:

Interpolate to find ν at `OBL_depth` from values at interfaces above and below.

INPUT PARAMETERS

```
! depths_cntr  = (/layer center containing OBL, layer center below/)
! diffs_iface  = diffusivity at interfaces of cell containing OBL
! layer_widths = (/width of layer containing OBL, width of layer below/)
real(cvmix_r8), dimension(2), intent(in) :: depths_cntr, diffs_iface,     &
                                            layer_widths
real(cvmix_r8),               intent(in) :: OBL_depth
! diffusivity at iface above the iface above OBL_depth (not needed if
! OBL is in top layer)
real(cvmix_r8), optional,     intent(in) :: diff_2above
```

OUTPUT PARAMETERS

```
    real(cvmix_r8), optional, intent(out) :: dnu_dz
    real(cvmix_r8) :: cvmix_kpp_compute_nu_at_OBL_depth_LMD94
```

8.21 cvmix_kpp_EFactor_model

INTERFACE

```
  function cvmix_kpp_EFactor_model(u10, ustar, hbl, CVmix_params_in)
```

DESCRIPTION:

This function returns the enhancement factor, given the 10-meter wind (m/s), friction velocity (m/s) and the boundary layer depth (m).

INPUT PARAMETERS

```
    real(cvmix_r8),                 intent(in) :: u10    ! 10 meter wind (m/s)
    real(cvmix_r8),                 intent(in) :: ustar  ! water-side surface friction veloci
    real(cvmix_r8),                 intent(in) :: hbl    ! boundary layer depth (m)
    type(cvmix_global_params_type), intent(in) :: CVmix_params_in
```

8.22 cvmix_kpp_ustokes_SL_model

INTERFACE

```
  function cvmix_kpp_ustokes_SL_model(u10, hbl, CVmix_params_in)
```

DESCRIPTION:

This function returns the surface layer averaged Stokes drift, given the 10-meter wind (m/s) and the boundary layer depth (m).

INPUT PARAMETERS

```
    real(cvmix_r8),                 intent(in) :: u10    ! 10 meter wind (m/s)
    real(cvmix_r8),                 intent(in) :: hbl    ! boundary layer depth (m)
    type(cvmix_global_params_type), intent(in) :: CVmix_params_in
```

8.23 cvmix_kpp_composite_shape

INTERFACE

```
function cvmix_kpp_composite_shape( sigma , Gat1)
```

DESCRIPTION:

This function returns the value of the composite shape function for both momentum and scalars at fractional depth sigma in the boundary layer. This shape function is a cubic for sigma¡sig_m; and a quadratic below, as fit to Fig. 6 of Large et al., 2020 (doi:10.1175/JPO-D-20-0308.1) The subroutine also returns the derivative dG / dsig

INPUT PARAMETERS

```
real(cvmix_r8),                    intent(in)  ::  sigma
real(cvmix_r8), optional,          intent(in)  ::  Gat1
```

OUTPUT PARAMETERS

```
real(cvmix_r8) :: cvmix_kpp_composite_shape
```

8.24 cvmix_kpp_composite_Gshape

INTERFACE

```
subroutine cvmix_kpp_composite_Gshape(sigma , Gat1, Gsig, dGdsig)
```

INPUT PARAMETERS

```
real(cvmix_r8),                    intent(in)  ::  sigma
real(cvmix_r8),                    intent(in)  ::  Gat1
```

INPUT/OUTPUT PARAMETERS

```
real(cvmix_r8),                 intent(inout) ::  Gsig
real(cvmix_r8),                 intent(inout) ::  dGdsig
```

8.25 cvmix_coeffs_bkgnd_wrap

INTERFACE

```
  subroutine cvmix_kpp_compute_StokesXi (zi, zk, kSL, SLDepth,          &
           surf_buoy_force, surf_fric_vel, omega_w2x, uE, vE, uS, vS,   &
           uSbar, vSbar, uS_SLD, vS_SLD, uSbar_SLD, vSbar_SLD,          &
           StokesXI, CVmix_kpp_params_user)
```

DESCRIPTION:

Compute the Stokes similarity parameter, StokesXI, and Entrainment Rule, BEdE_ER, from surface layer integrated TKE production terms as parameterized in Large et al., 2020 (doi:10.1175/JPO-D-20-0308.1)

INPUT PARAMETERS

```
    real(cvmix_r8), dimension(:), intent(in) :: zi, zk          !< Cell interface and center l
    integer,        intent(in) :: kSL                           !< cell index of Surface Laye
    real(cvmix_r8), intent(in) :: SLDepth                       !< Surface Layer Depth Integra
    real(cvmix_r8), intent(in) :: surf_buoy_force               !< Surface buoyancy flux forci
    real(cvmix_r8), intent(in) :: surf_fric_vel, omega_w2x      !< Surface wind forcing from x
    real(cvmix_r8), dimension(:), intent(in) :: uE, vE          !< Eulerian velocity at cente
    real(cvmix_r8), intent(in) :: uS_SLD, vS_SLD                !< Stokes drift at SLDepth
    real(cvmix_r8), intent(in) :: uSbar_SLD, vSbar_SLD          !< Average Stokes drift cell l

    type(cvmix_kpp_params_type), intent(in), optional, target :: CVmix_kpp_params_user
```

INPUT/OUTPUT PARAMETERS

```
    real(cvmix_r8), dimension(:), intent(inout) :: uS, vS       !< Stokes drift at interfaces
    real(cvmix_r8), dimension(:), intent(inout) :: uSbar, vSbar !< Cell average Stokes drift
    real(cvmix_r8), intent(inout) :: StokesXI                   !< Stokes similarity paramete
```

9 Module cvmix_convection

AUTHOR

Michael N. Levy, NCAR (mlevy@ucar.edu)

DESCRIPTION:

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.

References:
* Brunt-Vaisala?

USES

```
use cvmix_kinds_and_types, only : cvmix_r8,                               &
                                  cvmix_strlen,                           &
                                  cvmix_zero,                             &
                                  cvmix_one,                              &
                                  cvmix_data_type,                        &
                                  CVMIX_OVERWRITE_OLD_VAL,                &
                                  CVMIX_SUM_OLD_AND_NEW_VALS,             &
                                  CVMIX_MAX_OLD_AND_NEW_VALS
use cvmix_utils,           only : cvmix_update_wrap
use cvmix_put_get,         only : cvmix_put
```

PUBLIC MEMBER FUNCTIONS

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

interface cvmix_coeffs_conv
  module procedure cvmix_coeffs_conv_low
  module procedure cvmix_coeffs_conv_wrap
end interface cvmix_coeffs_conv

interface cvmix_put_conv
  module procedure cvmix_put_conv_int
  module procedure cvmix_put_conv_real
  module procedure cvmix_put_conv_logical
end interface cvmix_put_conv
```

PUBLIC TYPES

```
! cvmix_conv_params_type contains the necessary parameters for convective
type, public :: cvmix_conv_params_type
  private
    ! Convective diff
    ! diffusivity coefficient used in convective regime
    real(cvmix_r8) :: convect_diff ! units: m^2/s

    ! viscosity coefficient used in convective regime
    real(cvmix_r8) :: convect_visc ! units: m^2/s
    logical        :: lBruntVaisala

    ! Threshold for squared buoyancy frequency needed to trigger
    ! Brunt-Vaisala parameterization
    real(cvmix_r8) :: BVsqr_convect ! units: s^-2

    ! Only apply below the boundary layer?
    logical :: lnoOBL

    ! Flag for what to do with old values of CVmix_vars%[MTS]diff
    integer :: handle_old_vals
end type cvmix_conv_params_type
```

9.1 cvmix_init_conv

INTERFACE

```
subroutine cvmix_init_conv(convect_diff, convect_visc, lBruntVaisala,       &
                           BVsqr_convect, lnoOBL, old_vals,                 &
                           CVmix_conv_params_user)
```

DESCRIPTION:

Initialization routine for specifying convective mixing coefficients.

INPUT PARAMETERS

```
real(cvmix_r8), intent(in) :: &
  convect_diff,      &! diffusivity to parameterize convection
  convect_visc        ! viscosity to parameterize convection
logical,        intent(in), optional :: lBruntVaisala ! True => B-V mixing
real(cvmix_r8), intent(in), optional :: BVsqr_convect ! B-V parameter
logical,        intent(in), optional :: lnoOBL ! False => apply in OBL too
character(len=cvmix_strlen), optional, intent(in) :: old_vals
```

OUTPUT PARAMETERS

```
    type (cvmix_conv_params_type), optional, intent(inout) ::               &
                                       CVmix_conv_params_user
```

9.2 cvmix_coeffs_conv_wrap

INTERFACE

```
  subroutine cvmix_coeffs_conv_wrap(CVmix_vars, CVmix_conv_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for convective mixing.

INPUT PARAMETERS

```
    type (cvmix_conv_params_type), optional, target, intent(in)  ::         &
                                             CVmix_conv_params_user
```

INPUT/OUTPUT PARAMETERS

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

9.3 cvmix_coeffs_conv_low

INTERFACE

```
  subroutine cvmix_coeffs_conv_low(Mdiff_out, Tdiff_out, Nsqr, dens, dens_lwr,&
                                   nlev, max_nlev, OBL_ind,                 &
                                   CVmix_conv_params_user)
```

DESCRIPTION:

Computes vertical diffusion coefficients for convective mixing.

INPUT PARAMETERS

```
    integer,                               intent(in) :: nlev, max_nlev
    integer,                               intent(in) :: OBL_ind
    real(cvmix_r8), dimension(max_nlev+1), intent(in) :: Nsqr
    real(cvmix_r8), dimension(max_nlev),   intent(in) :: dens, dens_lwr
    type (cvmix_conv_params_type), optional, target, intent(in)  ::         &
                                             CVmix_conv_params_user
```

INPUT/OUTPUT PARAMETERS

```
     real(cvmix_r8), dimension(max_nlev+1), intent(inout) :: Mdiff_out,        &
                                                             Tdiff_out
```

9.4 cvmix_put_conv_int

INTERFACE

```
   subroutine cvmix_put_conv_int(varname, val, CVmix_conv_params_user)
```

DESCRIPTION:

Write a real value into a cvmix_conv_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
     type (cvmix_conv_params_type), optional, target, intent(inout) ::         &
                                                CVmix_conv_params_user
```

9.5 cvmix_put_conv_real

INTERFACE

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

DESCRIPTION:

Write a real value into a cvmix_conv_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
     type (cvmix_conv_params_type), optional, target, intent(inout) ::         &
                                                CVmix_conv_params_user
```

9.6 cvmix_put_conv_logical

INTERFACE

```
subroutine cvmix_put_conv_logical(varname, val, CVmix_conv_params_user)
```

DESCRIPTION:

Write a Boolean value into a cvmix_conv_params_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

```
type (cvmix_conv_params_type), optional, target, intent(inout) ::         &
                                          CVmix_conv_params_user
```

9.7 cvmix_get_conv_real

INTERFACE

```
function cvmix_get_conv_real(varname, CVmix_conv_params_user)
```

DESCRIPTION:

Read the real value of a cvmix_conv_params_type variable.

INPUT PARAMETERS

```
character(len=*),             intent(in) :: varname
type(cvmix_conv_params_type), optional, target, intent(in) ::             &
                                      CVmix_conv_params_user
```

OUTPUT PARAMETERS

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

---

10 Module cvmix_math

AUTHOR

```
Michael N. Levy, NCAR (mlevy@ucar.edu)
```

DESCRIPTION:

This module contains routines to compute polynomial interpolations (linear, quadratic, or cubic spline), evaluate third-order polynomials and their derivatives at specific values, and compute roots of these polynomials.

REVISION HISTORY

USES

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

DEFINED PARAMETERS

```
integer, parameter, public :: CVMIX_MATH_INTERP_LINEAR      = 1
integer, parameter, public :: CVMIX_MATH_INTERP_QUAD        = 2
integer, parameter, public :: CVMIX_MATH_INTERP_CUBE_SPLINE = 3

real(cvmix_r8), parameter :: CVMIX_MATH_NEWTON_TOL       = 1.0e-12_cvmix_r8
integer,        parameter :: CVMIX_MATH_MAX_NEWTON_ITERS = 100
```

PUBLIC MEMBER FUNCTIONS

```
public :: cvmix_math_poly_interp
public :: cvmix_math_cubic_root_find
public :: cvmix_math_evaluate_cubic
```

10.1 cvmix_math_poly_interp

INTERFACE

```
subroutine cvmix_math_poly_interp(coeffs, interp_type, x, y, x0, y0)
```

DESCRIPTION:

Given (x(1), y(1)), (x(2), y(2)), and possibly (x0, y0), compute coeffs = $(/a_0, a_1, a_2, a_3/)$ such that, for $f(x) = \sum a_n x^n$, the following hold: $f(x(1)) = y(1)$ and $f(x(2)) = y(2)$. For both quadratic and cubic interpolation, $f'(x(1)) = (y(1) - y0)/(x(1) - x0)$ as well, and for cubic splines $f'(x(2)) = (y(2) - y(1))/(x(2) - x(1))$.

INPUT PARAMETERS

```
integer,                         intent(in)    :: interp_type
real(cvmix_r8), dimension(2), intent(in)    :: x, y
real(cvmix_r8), optional,     intent(in)    :: x0, y0
```

OUTPUT PARAMETERS

```
real(cvmix_r8), dimension(4), intent(inout) :: coeffs
```

10.2 cvmix_math_evaluate_cubic

INTERFACE

```
function cvmix_math_evaluate_cubic(coeffs, x_in, fprime)
```

DESCRIPTION:

Computes $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ at $x =$`x_in`, where `coeffs`$= (/a_0, a_1, a_2, a_3/)$. If requested, can also return $f'(x)$

INPUT PARAMETERS

```
real(cvmix_r8), dimension(4), intent(in) :: coeffs
real(cvmix_r8),               intent(in) :: x_in
```

OUTPUT PARAMETERS

```
real(cvmix_r8) :: cvmix_math_evaluate_cubic
real(cvmix_r8), optional, intent(out) :: fprime
```

---

11 Module cvmix_put_get

AUTHOR

```
Michael N. Levy, NCAR (mlevy@ucar.edu)
```

DESCRIPTION:

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.

USES

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

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_real_2D
  module procedure cvmix_put_global_params_int
  module procedure cvmix_put_global_params_real
end interface cvmix_put
```

11.1 cvmix_put_int

INTERFACE

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

DESCRIPTION:

Write an integer value into a cvmix_data_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

11.2 cvmix_put_real

INTERFACE

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

DESCRIPTION:

Write a real value into a cvmix_data_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

11.3 cvmix_put_real_1D

INTERFACE

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

DESCRIPTION:

Write an array of real values into a cvmix_data_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

11.4 cvmix_put_real_2D

INTERFACE

```
subroutine cvmix_put_real_2D(CVmix_vars, varname, val, nlev_in)
```

DESCRIPTION:

Write an array of real values into a cvmix_data_type variable.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

11.5 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.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

11.6 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.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

12 Module cvmix_utils

AUTHOR

```
    Michael N. Levy, NCAR (mlevy@ucar.edu)
```

DESCRIPTION:

This module contains routines that are called by multiple modules but don't specifically compute anything mixing related.

USES

```
    use cvmix_kinds_and_types, only : cvmix_r8,                              &
                                      cvmix_strlen,                          &
                                      CVMIX_SUM_OLD_AND_NEW_VALS,            &
                                      CVMIX_MAX_OLD_AND_NEW_VALS,            &
                                      CVMIX_OVERWRITE_OLD_VAL
```

PUBLIC MEMBER FUNCTIONS

```
   public :: cvmix_update_wrap
   public :: cvmix_att_name
```

12.1 cvmix_update_wrap

INTERFACE

```
   subroutine cvmix_update_wrap(old_vals, nlev, Mdiff_out, new_Mdiff,          &
                                Tdiff_out, new_Tdiff, Sdiff_out, new_Sdiff)
```

DESCRIPTION:

Update diffusivity values based on `old_vals` (either overwrite, sum, or find the level-by-level max)

INPUT PARAMETERS

```
     integer, intent(in) :: old_vals, nlev
     real(cvmix_r8), dimension(nlev+1), optional, intent(in) :: new_Mdiff,     &
                                                                new_Tdiff,     &
                                                                new_Sdiff
```

OUTPUT PARAMETERS

```
    real(cvmix_r8), dimension(nlev+1), optional, intent(inout) :: Mdiff_out,  &
                                                                  Tdiff_out,  &
                                                                  Sdiff_out
```

12.2 cvmix_att_name

INTERFACE

```
  function cvmix_att_name(varname)
```

DESCRIPTION:

Given a variable short name, returns the precise name of the desired attribute in the cvmix_data_type structure.

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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