In-code documentation for CVMix

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

August 12, 2024

Contents

1 Routine/Function Prologues

1.1 Fortran: Module Interface cvmix_kinds_and_types (Source File: cvmix_kinds_and_typ

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
```

1.2 Fortran: Module Interface cvmix_background (Source File: cvmix_background.F90)

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
```

1.3 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.

USES:

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

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
```

```
type(cvmix_bkgnd_params_type), pointer :: CVmix_bkgnd_params_out

CVmix_bkgnd_params_out => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_out => CVmix_bkgnd_params_user
end if

! Clean up memory in bkgnd_params_type (will be re-allocated in put call)
if (allocated(CVmix_bkgnd_params_out%static_Mdiff))                        &
  deallocate(CVmix_bkgnd_params_out%static_Mdiff)
if (allocated(CVmix_bkgnd_params_out%static_Tdiff))                        &
  deallocate(CVmix_bkgnd_params_out%static_Tdiff)

! Set static_Mdiff and static_Tdiff in background_input_type
call cvmix_put_bkgnd('static_Mdiff', bkgnd_Mdiff, CVmix_bkgnd_params_user)
call cvmix_put_bkgnd('static_Tdiff', bkgnd_Tdiff, CVmix_bkgnd_params_user)

if (present(old_vals)) then
  select case (trim(old_vals))
    case ("overwrite")
```

```
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,    &
                           cvmix_bkgnd_params_user)
    case ("sum")
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS, &
                           cvmix_bkgnd_params_user)
    case ("max")
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS, &
                           cvmix_bkgnd_params_user)
    case DEFAULT
      print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
              "handling old values of diff and visc."
      stop 1
  end select
else
  call cvmix_put_bkgnd('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,        &
                           cvmix_bkgnd_params_user)
end if
```

1.4 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!

USES:

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

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


! local vars
integer :: nlev
type(cvmix_global_params_type), pointer :: CVmix_params_in
type(cvmix_bkgnd_params_type),  pointer :: CVmix_bkgnd_params_out

nullify(CVmix_params_in)
if (present(CVmix_params_user)) then
  CVmix_params_in => CVmix_params_user
  nlev = CVmix_params_in%max_nlev
else
  if (.not.present(ncol)) then
    print*, "ERROR: You must specify either ncol or a global param type", &
            "containing max_nlev!"
    stop 1
  end if
end if

CVmix_bkgnd_params_out => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_out => CVmix_bkgnd_params_user
end if

! NOTE: need to verify that bkgnd_[MT]diff are ncol x 1 or 1 x nlev+1

! Clean up memory in bkgnd_params_type (will be re-allocated in put call)
if (allocated(CVmix_bkgnd_params_out%static_Mdiff))                        &
  deallocate(CVmix_bkgnd_params_out%static_Mdiff)
if (allocated(CVmix_bkgnd_params_out%static_Tdiff))                        &
  deallocate(CVmix_bkgnd_params_out%static_Tdiff)

! Set static_[MT]diff in background_input_type
if (present(ncol)) then
  call cvmix_put_bkgnd('static_Mdiff', bkgnd_Mdiff,                        &
                       CVmix_bkgnd_params_user, ncol=ncol)
  call cvmix_put_bkgnd('static_Tdiff', bkgnd_Tdiff,                        &
                       CVmix_bkgnd_params_user, ncol=ncol)
else
  call cvmix_put_bkgnd('static_Mdiff', bkgnd_Mdiff,                        &
                       CVmix_bkgnd_params_user, nlev=nlev)
```

```
  call cvmix_put_bkgnd('static_Tdiff', bkgnd_Tdiff,                        &
                       CVmix_bkgnd_params_user, nlev=nlev)
end if

if (present(old_vals)) then
  select case (trim(old_vals))
    case ("overwrite")
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,    &
                           cvmix_bkgnd_params_user)
    case ("sum")
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS, &
                           cvmix_bkgnd_params_user)
    case ("max")
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS, &
                           cvmix_bkgnd_params_user)
    case DEFAULT
      print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
              "handling old values of diff and visc."
      stop 1
  end select
else
  call cvmix_put_bkgnd('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,        &
                           cvmix_bkgnd_params_user)
end if
```

1.5 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.

USES:

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

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


! local vars
integer :: nlev
type(cvmix_bkgnd_params_type),  pointer :: CVmix_bkgnd_params_out

CVmix_bkgnd_params_out => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_out => CVmix_bkgnd_params_user
end if

! NOTE: need to verify that bkgnd_[MT]diff are ncol x nlev+1

nlev = CVmix_params_in%max_nlev

! Clean up memory in bkgnd_params_type (will be re-allocated in put call)
if (allocated(CVmix_bkgnd_params_out%static_Mdiff))                       &
  deallocate(CVmix_bkgnd_params_out%static_Mdiff)
if (allocated(CVmix_bkgnd_params_out%static_Tdiff))                       &
  deallocate(CVmix_bkgnd_params_out%static_Tdiff)

! Set static_[MT]diff in background_input_type
call cvmix_put_bkgnd("static_Mdiff", bkgnd_Mdiff, ncol, nlev,             &
                     CVmix_bkgnd_params_user)
call cvmix_put_bkgnd("static_Tdiff", bkgnd_Tdiff, ncol, nlev,             &
                     CVmix_bkgnd_params_user)

if (present(old_vals)) then
  select case (trim(old_vals))
    case ("overwrite")
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,    &
                           cvmix_bkgnd_params_user)
    case ("sum")
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS, &
                           cvmix_bkgnd_params_user)
    case ("max")
```

```
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS, &
                           cvmix_bkgnd_params_user)
    case DEFAULT
      print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
              "handling old values of diff and visc."
      stop 1
  end select
else
  call cvmix_put_bkgnd('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,        &
                           cvmix_bkgnd_params_user)
end if
```

1.6 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

USES:

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

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
```

```
call cvmix_init_bkgnd(CVmix_params_in%max_nlev,-CVMix_vars%zw_iface, bl1,      &
                      bl2, bl3, bl4, CVmix_params_in%prandtl,                  &
                      old_vals, CVmix_bkgnd_params_user)
```

1.7 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:

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

This method is based on the following paper:

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

In that paper, they recommend the parameters

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

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

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

$\text{bl4} = 2500$ m

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

USES:

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

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


! Pointers to parameter data type
type(cvmix_bkgnd_params_type),  pointer :: CVmix_bkgnd_params_out

! Local copies to make code easier to read
real(cvmix_r8), dimension(max_nlev+1) :: Mdiff, Tdiff

CVmix_bkgnd_params_out => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_out => CVmix_bkgnd_params_user
end if

! Clean up memory in bkgnd_params_type (will be re-allocated in put call)
if (allocated(CVmix_bkgnd_params_out%static_Mdiff))                       &
  deallocate(CVmix_bkgnd_params_out%static_Mdiff)
if (allocated(CVmix_bkgnd_params_out%static_Tdiff))                       &
  deallocate(CVmix_bkgnd_params_out%static_Tdiff)

! Set static_[MT]diff in background_input_type
Tdiff = bl1 + (bl2/cvmix_PI)*atan(bl3*(zw-bl4))
Mdiff = prandtl*Tdiff

call cvmix_put_bkgnd("static_Mdiff", Mdiff, CVmix_bkgnd_params_user,      &
                     nlev=max_nlev)
call cvmix_put_bkgnd("static_Tdiff", Tdiff, CVmix_bkgnd_params_user,      &
                     nlev=max_nlev)

if (present(old_vals)) then
  select case (trim(old_vals))
    case ("overwrite")
      call cvmix_put_bkgnd('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,    &
```

```
                            cvmix_bkgnd_params_user)
      case ("sum")
        call cvmix_put_bkgnd('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS, &
                            cvmix_bkgnd_params_user)
      case ("max")
        call cvmix_put_bkgnd('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS, &
                            cvmix_bkgnd_params_user)
      case DEFAULT
        print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
                "handling old values of diff and visc."
        stop 1
    end select
  else
    call cvmix_put_bkgnd('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,        &
                            cvmix_bkgnd_params_user)
  end if
```

1.8 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.

USES:

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

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
```

```
real(cvmix_r8), dimension(CVmix_vars%max_nlev+1) :: new_Mdiff, new_Tdiff
type(cvmix_bkgnd_params_type), pointer :: CVmix_bkgnd_params_in
integer :: nlev, max_nlev

CVmix_bkgnd_params_in => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_in => CVmix_bkgnd_params_user
end if

nlev = CVmix_vars%nlev
max_nlev = CVmix_vars%max_nlev

if (.not.associated(CVmix_vars%Mdiff_iface)) &
  call cvmix_put(CVmix_vars, "Mdiff", cvmix_zero, max_nlev)
if (.not.associated(CVmix_vars%Tdiff_iface)) &
  call cvmix_put(CVmix_vars, "Tdiff", cvmix_zero, max_nlev)

call cvmix_coeffs_bkgnd(new_Mdiff, new_Tdiff, nlev, max_nlev, colid,      &
                        CVmix_bkgnd_params_user)
call cvmix_update_wrap(CVmix_bkgnd_params_in%handle_old_vals, max_nlev,   &
                       Mdiff_out = CVmix_vars%Mdiff_iface,                &
                       new_Mdiff = new_Mdiff,                             &
                       Tdiff_out = CVmix_vars%Tdiff_iface,                &
                       new_Tdiff = new_Tdiff)
```

1.9 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.

USES:

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

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
```

```
!-----------------------------------------------------------------------
!
!  local variables
!
!-----------------------------------------------------------------------

    integer :: kw

    do kw=1,nlev+1
      Mdiff_out(kw) = cvmix_bkgnd_static_Mdiff(CVmix_bkgnd_params_user, kw,   &
                                               colid)
      Tdiff_out(kw) = cvmix_bkgnd_static_Tdiff(CVmix_bkgnd_params_user, kw,   &
                                               colid)
    end do
```

1.10 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
logical :: cvmix_bkgnd_lvary_horizontal
```

```
cvmix_bkgnd_lvary_horizontal = CVmix_bkgnd_params_test%lvary_horizontal
```

1.11 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.

USES:

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

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


type(cvmix_bkgnd_params_type),  pointer :: CVmix_bkgnd_params_in
integer :: cid, kid

! Error check
CVmix_bkgnd_params_in => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_in => CVmix_bkgnd_params_user
end if

if (CVmix_bkgnd_params_in%lvary_horizontal) then
  if (present(colid)) then
    cid = colid
  else
    print*, "ERROR: need to pass colid when static_Mdiff varies across",  &
            " columns."
    stop 1
  end if
else
  cid = 1
end if

if (CVmix_bkgnd_params_in%lvary_vertical) then
  if (present(kw)) then
    kid = kw
  else
    print*, "ERROR: need to pass kw (level id) when static_Mdiff varies", &
            "across levels columns."
    stop 1
  end if
else
  kid = 1
end if

cvmix_bkgnd_static_Mdiff = CVmix_bkgnd_params_in%static_Mdiff(cid, kid)
```

1.12 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.

USES:

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

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
```

```
type(cvmix_bkgnd_params_type),  pointer :: CVmix_bkgnd_params_in
integer :: cid, kid

! Error che
CVmix_bkgnd_params_in => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_in => CVmix_bkgnd_params_user
end if

if (CVmix_bkgnd_params_in%lvary_horizontal) then
  if (present(colid)) then
    cid = colid
  else
    print*, "ERROR: need to pass colid when static_Tdiff varies across",  &
            " columns."
    stop 1
  end if
else
  cid = 1
end if

if (CVmix_bkgnd_params_in%lvary_vertical) then
  if (present(kw)) then
    kid = kw
```

```
  else
    print*, "ERROR: need to pass kw (level id) when static_Tdiff varies",  &
            "across levels columns."
    stop 1
  end if
else
  kid = 1
end if

cvmix_bkgnd_static_Tdiff = CVmix_bkgnd_params_in%static_Tdiff(cid, kid)
```

1.13 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.

USES:

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

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


type(cvmix_bkgnd_params_type), pointer :: CVmix_bkgnd_params_out

CVmix_bkgnd_params_out => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
```

```
  CVmix_bkgnd_params_out => CVmix_bkgnd_params_user
end if

select case (trim(varname))
  case ('old_vals', 'handle_old_vals')
    CVmix_bkgnd_params_out%handle_old_vals = val
  case DEFAULT
    call cvmix_put_bkgnd(varname, real(val,cvmix_r8),                    &
                         CVmix_bkgnd_params_user)
end select
```

1.14 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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


type(cvmix_bkgnd_params_type), pointer :: CVmix_bkgnd_params_out

CVmix_bkgnd_params_out => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
```

```
  CVmix_bkgnd_params_out => CVmix_bkgnd_params_user
end if

select case (trim(varname))
  case ('static_Mdiff')
    if (.not.allocated(CVmix_bkgnd_params_out%static_Mdiff)) then
      allocate(CVmix_bkgnd_params_out%static_Mdiff(1,1))
      CVmix_bkgnd_params_out%lvary_horizontal=.false.
      CVmix_bkgnd_params_out%lvary_vertical=.false.
    else
      print*, "WARNING: overwriting static_Mdiff!"
    end if
    CVmix_bkgnd_params_out%static_Mdiff(:,:) = val

  case ('static_Tdiff')
    if (.not.allocated(CVmix_bkgnd_params_out%static_Tdiff)) then
      allocate(CVmix_bkgnd_params_out%static_Tdiff(1,1))
      CVmix_bkgnd_params_out%lvary_horizontal=.false.
      CVmix_bkgnd_params_out%lvary_vertical=.false.
    else
      print*, "WARNING: overwriting static_Tdiff!"
    end if
    CVmix_bkgnd_params_out%static_Tdiff(:,:) = val

  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.15 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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



! Local vars
integer, dimension(2) :: dims
integer               :: data_dims
logical               :: lvary_horizontal

type(cvmix_bkgnd_params_type), pointer :: CVmix_bkgnd_params_out

! Error checking to make sure dimension is specified
if ((.not.present(ncol)).and.(.not.present(nlev))) then
  print*, "ERROR: when putting 1D data in cvmix_bkgnd_params_type ", &
          "you must specify nlev or ncol!"
  stop 1
end if

if ((present(ncol)).and.(present(nlev))) then
  print*, "ERROR: when putting 1D data in cvmix_bkgnd_params_type ", &
          "you can not specify both nlev or ncol!"
  stop 1
end if

CVmix_bkgnd_params_out => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_out => CVmix_bkgnd_params_user
end if

data_dims = size(val)
if (present(ncol)) then
  if (data_dims.gt.ncol) then
    print*, "ERROR: data array is bigger than number of columns specified."
    stop 1
  end if
```

```
  lvary_horizontal=.true.
  dims(1) = ncol
  dims(2) = 1
else
  if (data_dims.gt.nlev+1) then
    print*, "ERROR: data array is bigger than number of levels specified."
    stop 1
  end if
  lvary_horizontal=.false.
  dims(1) = 1
  dims(2) = nlev+1
end if

select case (trim(varname))
  case ('static_Mdiff')
    if (.not.allocated(CVmix_bkgnd_params_out%static_Mdiff)) then
      allocate(CVmix_bkgnd_params_out%static_Mdiff(dims(1),dims(2)))
      CVmix_bkgnd_params_out%lvary_horizontal = lvary_horizontal
      CVmix_bkgnd_params_out%lvary_vertical = .not.lvary_horizontal
    else
      print*, "WARNING: overwriting static_Mdiff!"
    end if
    if (any(shape(CVmix_bkgnd_params_out%static_Mdiff).ne.dims)) then
      print*, "ERROR: dimensions of static_Mdiff do not match what was ", &
              "sent to cvmix_put"
      stop 1
    end if
    if (lvary_horizontal) then
      CVmix_bkgnd_params_out%static_Mdiff(:,1)           = cvmix_zero
      CVmix_bkgnd_params_out%static_Mdiff(1:data_dims,1) = val
    else
      CVmix_bkgnd_params_out%static_Mdiff(1,:)           = cvmix_zero
      CVmix_bkgnd_params_out%static_Mdiff(1,1:data_dims) = val
    end if

  case ('static_Tdiff')
    if (.not.allocated(CVmix_bkgnd_params_out%static_Tdiff)) then
      allocate(CVmix_bkgnd_params_out%static_Tdiff(dims(1),dims(2)))
      CVmix_bkgnd_params_out%lvary_horizontal = lvary_horizontal
      CVmix_bkgnd_params_out%lvary_vertical = .not.lvary_horizontal
    else
      print*, "WARNING: overwriting static_Tdiff!"
    end if
    if (any(shape(CVmix_bkgnd_params_out%static_Tdiff).ne.dims)) then
      print*, "ERROR: dimensions of static_Tdiff do not match what was ", &
              "sent to cvmix_put"
      stop 1
    end if
```

```
      if (lvary_horizontal) then
        CVmix_bkgnd_params_out%static_Tdiff(:,1)           = cvmix_zero
        CVmix_bkgnd_params_out%static_Tdiff(1:data_dims,1) = val
      else
        CVmix_bkgnd_params_out%static_Tdiff(1,:)           = cvmix_zero
        CVmix_bkgnd_params_out%static_Tdiff(1,1:data_dims) = val
      end if

    case DEFAULT
      print*, "ERROR: ", trim(varname), " not a valid choice!"
      stop 1

  end select
```

1.16 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
! Local vars
integer, dimension(2) :: dims, data_dims
type(cvmix_bkgnd_params_type),  pointer :: CVmix_bkgnd_params_out

CVmix_bkgnd_params_out => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
  CVmix_bkgnd_params_out => CVmix_bkgnd_params_user
end if

dims      = (/ncol, nlev+1/)
data_dims = shape(val)

if (any(data_dims.gt.dims)) then
  print*, "ERROR: data being put in cvmix_bkgnd_params_type is larger ", &
          "than (ncol, nlev+1)"
  stop 1
end if

select case (trim(varname))
  case ('static_Mdiff')
    if (.not.allocated(CVmix_bkgnd_params_out%static_Mdiff)) then
      allocate(CVmix_bkgnd_params_out%static_Mdiff(dims(1),dims(2)))
      CVmix_bkgnd_params_out%lvary_horizontal=.true.
      CVmix_bkgnd_params_out%lvary_vertical=.true.
    else
      print*, "WARNING: overwriting static_Mdiff!"
    end if
    if (any(shape(CVmix_bkgnd_params_out%static_Mdiff).ne.dims)) then
      print*, "ERROR: dimensions of static_Mdiff do not match what was ", &
              "sent to cvmix_put"
      stop 1
    end if
    CVmix_bkgnd_params_out%static_Mdiff = cvmix_zero
    CVmix_bkgnd_params_out%static_Mdiff(1:data_dims(1),1:data_dims(2))= val

  case ('static_Tdiff')
    if (.not.allocated(CVmix_bkgnd_params_out%static_Tdiff)) then
      allocate(CVmix_bkgnd_params_out%static_Tdiff(dims(1),dims(2)))
      CVmix_bkgnd_params_out%lvary_horizontal=.true.
      CVmix_bkgnd_params_out%lvary_vertical=.true.
    else
      print*, "WARNING: overwriting static_Tdiff!"
    end if
    if (any(shape(CVmix_bkgnd_params_out%static_Tdiff).ne.dims)) then
      print*, "ERROR: dimensions of static_Tdiff do not match what was ", &
              "sent to cvmix_put"
      stop 1
```

```
      end if
      CVmix_bkgnd_params_out%static_Tdiff = cvmix_zero
      CVmix_bkgnd_params_out%static_Tdiff(1:data_dims(1),1:data_dims(2))= val

    case DEFAULT
      print*, "ERROR: ", trim(varname), " not a valid choice!"
      stop 1

  end select
```

1.17 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.

USES:

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

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
```

```
type(cvmix_bkgnd_params_type), pointer :: CVmix_bkgnd_params_get
integer :: dim1, dim2

CVmix_bkgnd_params_get => CVmix_bkgnd_params_saved
if (present(CVmix_bkgnd_params_user)) then
```

```
  CVmix_bkgnd_params_get => CVmix_bkgnd_params_user
end if
dim1 = size(CVmix_bkgnd_params_get%static_Mdiff,1)
dim2 = size(CVmix_bkgnd_params_get%static_Mdiff,2)
allocate(cvmix_get_bkgnd_real_2D(dim1, dim2))

select case (trim(varname))
  case ('static_Mdiff')
    cvmix_get_bkgnd_real_2D = CVmix_bkgnd_params_get%static_Mdiff(:,:)
  case ('static_Tdiff')
    cvmix_get_bkgnd_real_2D = CVmix_bkgnd_params_get%static_Tdiff(:,:)
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1
end select
```

1.18 Fortran: Module Interface cvmix_shear (Source File: cvmix_shear.F90)

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
```

1.19 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), *alpha* (`PP_alpha`), and n (`PP_exp`), and returns

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

Note that ν_b and κ_b are 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

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

USES:

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

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



type(cvmix_shear_params_type), pointer :: CVmix_shear_params_out

if (present(CVmix_shear_params_user)) then
  CVmix_shear_params_out => CVmix_shear_params_user
else
  CVmix_shear_params_out => CVmix_shear_params_saved
end if

if (present(mix_scheme)) then
  call cvmix_put_shear("mix_scheme", trim(mix_scheme),                    &
                       CVmix_shear_params_user)
else
  call cvmix_put_shear("mix_scheme", "KPP", CVmix_shear_params_user)
end if

select case (trim(CVmix_shear_params_out%mix_scheme))
  case ('PP')
    if (present(PP_nu_zero)) then
      call cvmix_put_shear("PP_nu_zero", PP_nu_zero,                      &
                           CVmix_shear_params_user)
    else
      call cvmix_put_shear("PP_nu_zero", 0.01_cvmix_r8,                   &
                           CVmix_shear_params_user)
    end if

    if (present(PP_alpha)) then
      call cvmix_put_shear("PP_alpha", PP_alpha, CVmix_shear_params_user)
    else
      call cvmix_put_shear("PP_alpha", 5, CVmix_shear_params_user)
    end if

    if (present(PP_exp)) then
      call cvmix_put_shear("PP_exp", PP_exp, CVmix_shear_params_user)
    else
      call cvmix_put_shear("PP_exp", 2, CVmix_shear_params_user)
    end if

    if (present(PP_nu_b)) then
      call cvmix_put_shear("PP_nu_b", PP_nu_b, CVmix_shear_params_user)
    else
      call cvmix_put_shear("PP_nu_b", cvmix_zero, CVmix_shear_params_user)
    end if
```

```
    if (present(PP_kappa_b)) then
      call cvmix_put_shear("PP_kappa_b", PP_kappa_b, CVmix_shear_params_user)
    else
      call cvmix_put_shear("PP_kappa_b", cvmix_zero, CVmix_shear_params_user)
    end if

  case ('KPP')
    if (present(KPP_nu_zero)) then
      call cvmix_put_shear("KPP_nu_zero", KPP_nu_zero,                    &
                           CVmix_shear_params_user)
    else
      call cvmix_put_shear("KPP_nu_zero", 50e-4_cvmix_r8,                 &
                           CVmix_shear_params_user)
    end if

    if (present(KPP_Ri_zero)) then
      call cvmix_put_shear("KPP_Ri_zero", KPP_Ri_zero,                    &
                           CVmix_shear_params_user)
    else
      call cvmix_put_shear("KPP_Ri_zero", 0.7_cvmix_r8,                   &
                           CVmix_shear_params_user)
    end if

    if (present(KPP_exp)) then
      call cvmix_put_shear("KPP_exp", KPP_exp, CVmix_shear_params_user)
    else
      call cvmix_put_shear("KPP_exp", 3, CVmix_shear_params_user)
    end if

  case DEFAULT
    print*, "ERROR: ", trim(CVmix_shear_params_out%mix_scheme),           &
            " is not a valid choice for shear mixing."
    stop 1

end select

if (present(old_vals)) then
  select case (trim(old_vals))
    case ("overwrite")
      call cvmix_put_shear('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,    &
                           cvmix_shear_params_user)
    case ("sum")
      call cvmix_put_shear('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS, &
                           cvmix_shear_params_user)
    case ("max")
      call cvmix_put_shear('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS, &
                           cvmix_shear_params_user)
```

```
    case DEFAULT
      print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
              "handling old values of diff and visc."
      stop 1
  end select
else
  call cvmix_put_shear('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,        &
                           cvmix_shear_params_user)
end if
```

1.20 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.

USES:

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

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
```

```
real(cvmix_r8), dimension(CVmix_vars%max_nlev+1) :: new_Mdiff, new_Tdiff
integer :: nlev, max_nlev
type(cvmix_shear_params_type), pointer :: CVmix_shear_params_in

if (present(CVmix_shear_params_user)) then
  CVmix_shear_params_in => CVmix_shear_params_user
```

```
else
  CVmix_shear_params_in => CVmix_shear_params_saved
end if
nlev = CVmix_vars%nlev
max_nlev = CVmix_vars%max_nlev

if (.not.associated(CVmix_vars%Mdiff_iface)) &
  call cvmix_put(CVmix_vars, "Mdiff", cvmix_zero, max_nlev)
if (.not.associated(CVmix_vars%Tdiff_iface)) &
  call cvmix_put(CVmix_vars, "Tdiff", cvmix_zero, max_nlev)

call cvmix_coeffs_shear(new_Mdiff, new_Tdiff,                              &
                        CVmix_vars%ShearRichardson_iface, nlev, max_nlev, &
                        CVmix_shear_params_user)
call cvmix_update_wrap(CVmix_shear_params_in%handle_old_vals, max_nlev,   &
                       Mdiff_out = CVmix_vars%Mdiff_iface,                &
                       new_Mdiff = new_Mdiff,                             &
                       Tdiff_out = CVmix_vars%Tdiff_iface,                &
                       new_Tdiff = new_Tdiff)
```

1.21 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.

USES:

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

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
```

```
integer                    :: kw ! vertical cell index
! Parameters used in both PP81 and LMD94
real(cvmix_r8)             :: nu_zero, loc_exp
! Parameters only used in PP81
real(cvmix_r8)             :: PP_alpha, PP_nu_b, PP_kappa_b, denom
! Parameters only used in LMD94
real(cvmix_r8)             :: KPP_Ri_zero
type(cvmix_shear_params_type), pointer :: CVmix_shear_params

if (present(CVmix_shear_params_user)) then
  CVmix_shear_params => CVmix_shear_params_user
else
  CVmix_shear_params => CVmix_shear_params_saved
end if

select case (trim(CVmix_shear_params%mix_scheme))
  case ('PP')
    ! Copy parameters to make the code more legible
    nu_zero    = CVmix_shear_params%PP_nu_zero
    PP_alpha   = CVmix_shear_params%PP_alpha
    loc_exp    = CVmix_shear_params%PP_exp
    PP_nu_b    = CVmix_shear_params%PP_nu_b
    PP_kappa_b = CVmix_shear_params%PP_kappa_b

    ! Pacanowski-Philander
    do kw=1,nlev+1
      if (RICH(kw).gt.cvmix_zero) then
        denom = cvmix_one + PP_alpha * RICH(kw)
      else
        ! Treat non-negative Richardson number as Ri = 0
        denom = cvmix_one
      end if
      Mdiff_out(kw) = nu_zero / (denom**loc_exp) + PP_nu_b
      Tdiff_out(kw) = Mdiff_out(kw) / denom + PP_kappa_b
    end do

  case ('KPP')
    ! Copy parameters to make the code more legible
    nu_zero     = CVmix_shear_params%KPP_nu_zero
    KPP_Ri_zero = CVmix_shear_params%KPP_Ri_zero
    loc_exp     = CVmix_shear_params%KPP_exp
```

```
      ! Large, et al
      do kw=1,nlev+1
          if (RICH(kw).lt.cvmix_zero) then
            Tdiff_out(kw) = nu_zero
          else if (RICH(kw).lt.KPP_Ri_zero) then
            Tdiff_out(kw) = nu_zero * (cvmix_one - (RICH(kw)/KPP_Ri_zero)   &
                                                   **2)**loc_exp
          else ! Ri_g >= Ri_zero
            Tdiff_out(kw) = cvmix_zero
          end if
      end do
      ! to do: include global params for prandtl number!
      Mdiff_out = Tdiff_out

    case DEFAULT
      ! Note: this error should be caught in cvmix_init_shear
      print*, "ERROR: invalid choice for type of shear mixing."
      stop 1

  end select
```

1.22 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.

USES:

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

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


type(cvmix_shear_params_type), pointer :: CVmix_shear_params_out

if (present(CVmix_shear_params_user)) then
  CVmix_shear_params_out => CVmix_shear_params_user
else
  CVmix_shear_params_out => CVmix_shear_params_saved
end if

select case(trim(varname))
  case ('old_vals', 'handle_old_vals')
    CVmix_shear_params_out%handle_old_vals = val
  case DEFAULT
  call cvmix_put_shear(varname, real(val,cvmix_r8),                        &
                       CVmix_shear_params_user)
end select
```

1.23 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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


type(cvmix_shear_params_type), pointer :: CVmix_shear_params_out

if (present(CVmix_shear_params_user)) then
  CVmix_shear_params_out => CVmix_shear_params_user
else
  CVmix_shear_params_out => CVmix_shear_params_saved
end if

select case (trim(varname))
  case ('PP_nu_zero')
    CVmix_shear_params_out%PP_nu_zero = val
  case ('PP_alpha')
    CVmix_shear_params_out%PP_alpha = val
  case ('PP_exp')
    CVmix_shear_params_out%PP_exp = val
  case ('PP_nu_b')
    CVmix_shear_params_out%PP_nu_b = val
  case ('PP_kappa_b')
    CVmix_shear_params_out%PP_kappa_b = val
  case ('KPP_nu_zero')
    CVmix_shear_params_out%KPP_nu_zero = val
  case ('KPP_Ri_zero')
    CVmix_shear_params_out%KPP_Ri_zero = val
  case ('KPP_exp')
    CVmix_shear_params_out%KPP_exp = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.24 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
type(cvmix_shear_params_type), pointer :: CVmix_shear_params_out

if (present(CVmix_shear_params_user)) then
  CVmix_shear_params_out => CVmix_shear_params_user
else
  CVmix_shear_params_out => CVmix_shear_params_saved
end if

select case (trim(varname))
  case ('mix_scheme')
    CVmix_shear_params_out%mix_scheme = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.25 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.

USES:

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

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
```

```
type(cvmix_shear_params_type), pointer :: CVmix_shear_params_in

if (present(CVmix_shear_params_user)) then
  CVmix_shear_params_in => CVmix_shear_params_user
else
  CVmix_shear_params_in => CVmix_shear_params_saved
end if

cvmix_get_shear_real = cvmix_zero
select case (trim(varname))
  case ('PP_nu_zero')
    cvmix_get_shear_real =CVmix_shear_params_in%PP_nu_zero
  case ('PP_alpha')
    cvmix_get_shear_real =CVmix_shear_params_in%PP_alpha
  case ('PP_exp')
    cvmix_get_shear_real =CVmix_shear_params_in%PP_exp
  case ('KPP_nu_zero')
    cvmix_get_shear_real =CVmix_shear_params_in%KPP_nu_zero
  case ('KPP_Ri_zero')
    cvmix_get_shear_real =CVmix_shear_params_in%KPP_Ri_zero
  case ('KPP_exp')
    cvmix_get_shear_real =CVmix_shear_params_in%KPP_exp
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1
```

```
end select
```

1.26 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.

USES:

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

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
```

```
type(cvmix_shear_params_type), pointer :: CVmix_shear_params_in

if (present(CVmix_shear_params_user)) then
  CVmix_shear_params_in => CVmix_shear_params_user
else
  CVmix_shear_params_in => CVmix_shear_params_saved
end if

select case (trim(varname))
  case ('mix_scheme')
    cvmix_get_shear_str = trim(CVmix_shear_params_in%mix_scheme)
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
```

```
        stop 1

    end select
```

1.27 Fortran: Module Interface cvmix_tidal (Source File: cvmix_tidal.F90)

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
```

1.28 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.

USES:

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

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


type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params_out

if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params_out => CVmix_tidal_params_user
else
  CVmix_tidal_params_out => CVmix_tidal_params_saved
end if

if (present(mix_scheme)) then
  call cvmix_put_tidal("mix_scheme", trim(mix_scheme),                    &
                       CVmix_tidal_params_user)
else
  call cvmix_put_tidal("mix_scheme", "Simmons", CVmix_tidal_params_user)
end if

select case (trim(CVmix_tidal_params_out%mix_scheme))
  case ('simmons','Simmons')
    ! Unitless parameters
    if (present(efficiency)) then
      call cvmix_put_tidal("efficiency", efficiency,                      &
                           CVmix_tidal_params_user)
    else
      call cvmix_put_tidal("efficiency", 0.2_cvmix_r8,                    &
                           CVmix_tidal_params_user)
    end if

    if (present(local_mixing_frac)) then
      call cvmix_put_tidal("local_mixing_frac", local_mixing_frac,        &
                           CVmix_tidal_params_user)
    else
      call cvmix_put_tidal("local_mixing_frac", 3, CVmix_tidal_params_user)
    end if

    ! Parameters with units
```

```
  if (present(vertical_decay_scale)) then
    call cvmix_put_tidal("vertical_decay_scale", vertical_decay_scale,  &
                         CVmix_tidal_params_user)
  else
    call cvmix_put_tidal("vertical_decay_scale", 500,                   &
                         CVmix_tidal_params_user)
  end if

  if (present(depth_cutoff)) then
    call cvmix_put_tidal("depth_cutoff", depth_cutoff,                  &
                         CVmix_tidal_params_user)
  else
    ! Default: no cutoff depth => 0 m
    call cvmix_put_tidal("depth_cutoff", 0, CVmix_tidal_params_user)
  end if

  if (present(max_coefficient)) then
    call cvmix_put_tidal("max_coefficient", max_coefficient,            &
                         CVmix_tidal_params_user)
  else
    call cvmix_put_tidal("max_coefficient", 50e-4_cvmix_r8,             &
                         CVmix_tidal_params_user)
  end if

  if (present(ltidal_Schmittner_socn)) then
    call cvmix_put_tidal("ltidal_Schmittner_socn", ltidal_Schmittner_socn,        &
                         CVmix_tidal_params_user)
  else
    ! Default: do not apply Schmittner Southern Ocean mods
    call cvmix_put_tidal("ltidal_Schmittner_socn", .false., CVmix_tidal_params_user)
  end if

case ('schmittner','Schmittner')
  ! Unitless parameters
  if (present(efficiency)) then
    call cvmix_put_tidal("efficiency", efficiency,                      &
                         CVmix_tidal_params_user)
  else
    call cvmix_put_tidal("efficiency", 0.2_cvmix_r8,                    &
                         CVmix_tidal_params_user)
  end if

  ! Parameters with units
  if (present(vertical_decay_scale)) then
    call cvmix_put_tidal("vertical_decay_scaleR", cvmix_one/vertical_decay_scale,  &
                         CVmix_tidal_params_user)
  else
    call cvmix_put_tidal("vertical_decay_scaleR", cvmix_one/500.0_cvmix_r8,        &
```

```
                          CVmix_tidal_params_user)
    end if

    if (present(max_coefficient)) then
      call cvmix_put_tidal("max_coefficient", max_coefficient,           &
                          CVmix_tidal_params_user)
    else
      call cvmix_put_tidal("max_coefficient", 50e-4_cvmix_r8,            &
                          CVmix_tidal_params_user)
    end if

    if (present(ltidal_Schmittner_socn)) then
      call cvmix_put_tidal("ltidal_Schmittner_socn", ltidal_Schmittner_socn,       &
                          CVmix_tidal_params_user)
    else
      ! Default: do not apply Schmittner Southern Ocean mods
      call cvmix_put_tidal("ltidal_Schmittner_socn", .false., CVmix_tidal_params_user)
    end if

  case DEFAULT
    print*, "ERROR: ", trim(mix_scheme), " is not a valid choice for ", &
            "tidal mixing."
    stop 1

end select

if (present(old_vals)) then
  select case (trim(old_vals))
    case ("overwrite")
      call cvmix_put_tidal('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,    &
                          cvmix_tidal_params_user)
    case ("sum")
      call cvmix_put_tidal('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS, &
                          cvmix_tidal_params_user)
    case ("max")
      call cvmix_put_tidal('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS, &
                          cvmix_tidal_params_user)
    case DEFAULT
      print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
              "handling old values of diff and visc."
      stop 1
  end select
else
  call cvmix_put_tidal('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,        &
                          cvmix_tidal_params_user)
end if
```

1.29 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.

USES:

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

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
```

```
! Local variables
real(cvmix_r8), dimension(CVmix_vars%max_nlev+1) :: new_Mdiff, new_Tdiff
type(cvmix_tidal_params_type),  pointer :: CVmix_tidal_params_in
integer :: nlev, max_nlev

CVmix_tidal_params_in => CVmix_tidal_params_saved
if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params_in => CVmix_tidal_params_user
end if
nlev = CVmix_vars%nlev
max_nlev = CVmix_vars%max_nlev

select case (trim(CVmix_tidal_params_in%mix_scheme))
  case ('simmons','Simmons')
    call cvmix_coeffs_tidal_low                                          &
```

```
                              (new_Mdiff, new_Tdiff,                        &
                              CVmix_vars%SqrBuoyancyFreq_iface,             &
                              CVmix_vars%OceanDepth,                        &
                              CVmix_vars%SimmonsCoeff,                      &
                              CVmix_vars%VertDep_iface, nlev, max_nlev,     &
                              CVMix_params,                                 &
                              CVmix_vars%SchmittnerSouthernOcean,           &
                              CVmix_tidal_params_user)
    case ('schmittner','Schmittner')
      call cvmix_coeffs_tidal_schmittner                                    &
                              (new_Mdiff, new_Tdiff,                        &
                              CVmix_vars%SqrBuoyancyFreq_iface,             &
                              CVmix_vars%OceanDepth,                        &
                              nlev, max_nlev,                               &
                              CVmix_vars%SchmittnerCoeff,                   &
                              CVmix_vars%SchmittnerSouthernOcean,           &
                              CVmix_params,                                 &
                              CVmix_tidal_params_user)

  end select
  call cvmix_update_wrap(CVmix_tidal_params_in%handle_old_vals, max_nlev, &
                         Mdiff_out = CVmix_vars%Mdiff_iface,              &
                         Tdiff_out = CVmix_vars%Tdiff_iface,              &
                         new_Mdiff = new_Mdiff,                           &
                         new_Tdiff = new_Tdiff)
```

1.30 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.

USES:

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

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
```

```
! Local variables
integer        :: k
real(cvmix_r8), dimension(max_nlev+1) :: SchmittnerSouthernOceanLocal

type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params

if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params => CVmix_tidal_params_user
else
  CVmix_tidal_params => CVmix_tidal_params_saved
end if

if (present(SchmittnerSouthernOcean)) then
  SchmittnerSouthernOceanLocal = SchmittnerSouthernOcean
else
  SchmittnerSouthernOceanLocal = cvmix_zero
end if

select case (trim(CVmix_tidal_params%mix_scheme))
  case ('simmons','Simmons')
    Tdiff_out = cvmix_zero
    if (OceanDepth.ge.CVmix_tidal_params%depth_cutoff) then
      do k=1, nlev+1
        !*** compute tidal diffusion
        if (Nsqr(k).gt.cvmix_zero) &
          Tdiff_out(k) = SimmonsCoeff*vert_dep(k)/Nsqr(k)
```

```
        !*** apply Scmittner Southern Ocean modification
        if (CVmix_tidal_params%ltidal_Schmittner_socn .and. k<=nlev)&
          Tdiff_out(k) = max(Tdiff_out(k),SchmittnerSouthernOcean(k))

        !*** apply tidal diffusion cap
        if (Tdiff_out(k).gt.CVmix_tidal_params%max_coefficient) &
          Tdiff_out(k) = CVmix_tidal_params%max_coefficient

      end do
    end if

  case DEFAULT
    ! Note: this error should be caught in cvmix_init_tidal
    print*, "ERROR: invalid choice for type of tidal mixing."
    stop 1

end select
Mdiff_out = CVmix_params%Prandtl*Tdiff_out
```

1.31 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.

USES:

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

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
```

```
! Local variables
integer        :: k

type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params

if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params => CVmix_tidal_params_user
else
  CVmix_tidal_params => CVmix_tidal_params_saved
end if

select case (trim(CVmix_tidal_params%mix_scheme))

  case ('schmittner','Schmittner')
    Tdiff_out = cvmix_zero
    if (OceanDepth.ge.CVmix_tidal_params%depth_cutoff) then
      do k=1, nlev+1
        !*** compute tidal diffusion
        if (Nsqr(k).gt.cvmix_zero) &
          Tdiff_out(k) = SchmittnerCoeff(k)/Nsqr(k)

        !*** apply Scmittner Southern Ocean modification
        if (CVmix_tidal_params%ltidal_Schmittner_socn .and. k<=nlev)&
          Tdiff_out(k) = max(Tdiff_out(k),SchmittnerSouthernOcean(k))

        !*** apply tidal diffusion cap
        if (Tdiff_out(k).gt.CVmix_tidal_params%max_coefficient) &
          Tdiff_out(k) = CVmix_tidal_params%max_coefficient

      end do

    end if
```

```
    case DEFAULT
      ! Note: this error should be caught in cvmix_init_tidal
      print*, "ERROR: invalid choice for type of tidal mixing."
      stop 1

  end select
  Mdiff_out = CVmix_params%Prandtl*Tdiff_out
```

1.32 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.

USES:

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

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
```

```
! Local variables
real(cvmix_r8) :: tot_area, num, thick
integer        :: k

! Compute vertical deposition
```

```
tot_area = cvmix_zero
cvmix_compute_vert_dep(1) = cvmix_zero
cvmix_compute_vert_dep(nlev+1) = cvmix_zero
do k=2,nlev
  num = -zw(k)/CVmix_tidal_params%vertical_decay_scale
  ! Simmons vertical deposition
  ! Note that it is getting normalized (divide through by tot_area)
  ! So multiplicative constants that are independent of z are omitted
  cvmix_compute_vert_dep(k) = exp(num)

  ! Compute integral of vert_dep via trapezoid rule
  ! (looks like midpoint rule, but vert_dep = 0 at z=0 and z=-ocn_depth)
  thick = zt(k-1) - zt(k)
  tot_area = tot_area + cvmix_compute_vert_dep(k)*thick
end do
! Normalize vert_dep (need integral = 1.0D0)
cvmix_compute_vert_dep = cvmix_compute_vert_dep/tot_area
```

1.33 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.

USES:

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

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
```

```
    ! Local variables
    real(cvmix_r8) :: zetar
    integer        :: k

    zetar = CVmix_tidal_params%vertical_decay_scaleR

    ! Compute Schmittner-method vertical decay profile
    cvmix_compute_vert_dep_Schmittner(1)      = cvmix_zero
    do k=2,nlev+1
      cvmix_compute_vert_dep_Schmittner(k) = zetar/(cvmix_one-exp(zetar*zw(k)))
!     cvmix_compute_vert_dep_Schmittner(k) = zetar/(cvmix_one-exp(zetar*zw(k+1)))
!     cvmix_compute_vert_dep_Schmittner(k) = zetar/(cvmix_one-exp(zetar*zw(k-1)))
    end do
```

1.34 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.

USES:

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

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
```

1.35 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.

USES:

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

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
```

1.36 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.

USES:

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

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
```

1.37 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.

USES:

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

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
```

1.38 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.

USES:

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

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
```

1.39 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.

USES:

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

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
```

1.40 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

USES:

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

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
```

1.41 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

USES:

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

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
```

```
if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params => CVmix_tidal_params_user
else
  CVmix_tidal_params => CVmix_tidal_params_saved
end if

SchmittnerTanhLat = 0.5_cvmix_r8*(cvmix_one-tanh((lat+40.0_cvmix_r8)/8.0_cvmix_r8))

do k=1, nlev+1
  SchmittnerTanhZw = tanh((-zw(k)-500._cvmix_r8)/100.0_cvmix_r8)*1.0e-4_cvmix_r8
  SchmittnerSouthernOcean(k) = SchmittnerTanhLat*SchmittnerTanhZw
end do
```

1.42 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.

USES:

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

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
```

```
type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params_out

CVmix_tidal_params_out => CVmix_tidal_params_saved
if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params_out => CVmix_tidal_params_user
end if

select case (trim(varname))
  case ('old_vals', 'handle_old_vals')
    CVmix_tidal_params_out%handle_old_vals = val
  case DEFAULT
    call cvmix_put_tidal(varname, real(val,cvmix_r8),                    &
                         CVmix_tidal_params_user)
end select
```

1.43 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.

USES:

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

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
```

```
type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params_out

CVmix_tidal_params_out => CVmix_tidal_params_saved
if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params_out => CVmix_tidal_params_user
end if

select case (trim(varname))
  case ('ltidal_Schmittner_socn')
    CVmix_tidal_params_out%ltidal_Schmittner_socn = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " is not a boolean variable!"
    stop 1
end select
```

1.44 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params_out

if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params_out => CVmix_tidal_params_user
else
  CVmix_tidal_params_out => CVmix_tidal_params_saved
end if

select case (trim(varname))
  case ('efficiency')
    CVmix_tidal_params_out%efficiency = val
  case ('vertical_decay_scale')
    CVmix_tidal_params_out%vertical_decay_scale = val
  case ('vertical_decay_scaleR')
    CVmix_tidal_params_out%vertical_decay_scaleR = val
  case ('max_coefficient')
    CVmix_tidal_params_out%max_coefficient = val
  case ('local_mixing_frac')
    CVmix_tidal_params_out%local_mixing_frac = val
  case ('depth_cutoff')
    CVmix_tidal_params_out%depth_cutoff = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.45 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params_out

if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params_out => CVmix_tidal_params_user
else
  CVmix_tidal_params_out => CVmix_tidal_params_saved
end if

select case (trim(varname))
  case ('mix_scheme')
    CVmix_tidal_params_out%mix_scheme = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.46 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.

USES:

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

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
```

```
type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params_in

if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params_in => CVmix_tidal_params_user
else
  CVmix_tidal_params_in => CVmix_tidal_params_saved
end if

cvmix_get_tidal_real = cvmix_zero
select case (trim(varname))
  case ('efficiency')
    cvmix_get_tidal_real = CVmix_tidal_params_in%efficiency
  case ('vertical_decay_scale')
    cvmix_get_tidal_real = CVmix_tidal_params_in%vertical_decay_scale
  case ('max_coefficient')
    cvmix_get_tidal_real = CVmix_tidal_params_in%max_coefficient
  case ('local_mixing_frac')
    cvmix_get_tidal_real = CVmix_tidal_params_in%local_mixing_frac
  case ('depth_cutoff')
    cvmix_get_tidal_real = CVmix_tidal_params_in%depth_cutoff
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1
end select
```

1.47 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.

USES:

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

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
```

```
type(cvmix_tidal_params_type), pointer :: CVmix_tidal_params_in

if (present(CVmix_tidal_params_user)) then
  CVmix_tidal_params_in => CVmix_tidal_params_user
else
  CVmix_tidal_params_in => CVmix_tidal_params_saved
end if

select case (trim(varname))
  case ('mix_scheme')
    cvmix_get_tidal_str = trim(CVmix_tidal_params_in%mix_scheme)
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.48 Fortran: Module Interface cvmix_ddiff (Source File: cvmix_ddiff.F90)

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
      ! regime)
      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
```

1.49 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

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

κ is stored in `CVmix_vars%diff_iface(:,1)`, while the modified value for non-temperature tracers is stored in `CVmix_vars%diff_iface(:,2)`. Note that CVMix assumes units are —'mks'—.

USES:

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

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


! Unitless parameters
if (present(strat_param_max)) then
  call cvmix_put_ddiff("strat_param_max", strat_param_max,                 &
                       CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("strat_param_max", 2.55_cvmix_r8,                   &
                       CVmix_ddiff_params_user)
end if

if (present(diff_conv_type)) then
  call cvmix_put_ddiff("diff_conv_type", diff_conv_type,                  &
                        CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("diff_conv_type", "MC76", CVmix_ddiff_params_user)
end if

if (present(ddiff_exp1)) then
  call cvmix_put_ddiff("ddiff_exp1", ddiff_exp1, CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("ddiff_exp1", cvmix_one, CVmix_ddiff_params_user)
end if
```

```
if (present(ddiff_exp2)) then
  call cvmix_put_ddiff("ddiff_exp2", ddiff_exp2, CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("ddiff_exp2", 3, CVmix_ddiff_params_user)
end if

if (present(kappa_ddiff_param1)) then
  call cvmix_put_ddiff("kappa_ddiff_param1", kappa_ddiff_param1,          &
                       CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("kappa_ddiff_param1", 0.909_cvmix_r8,              &
                       CVmix_ddiff_params_user)
end if

if (present(kappa_ddiff_param2)) then
  call cvmix_put_ddiff("kappa_ddiff_param2", kappa_ddiff_param2,          &
                       CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("kappa_ddiff_param2", 4.6_cvmix_r8,                &
                       CVmix_ddiff_params_user)
end if

if (present(kappa_ddiff_param3)) then
  call cvmix_put_ddiff("kappa_ddiff_param3", kappa_ddiff_param3,          &
                       CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("kappa_ddiff_param3", -0.54_cvmix_r8,              &
                       CVmix_ddiff_params_user)
end if

! Parameters with physical units
if (present(kappa_ddiff_s)) then
  call cvmix_put_ddiff("kappa_ddiff_s", kappa_ddiff_s,                    &
                       CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("kappa_ddiff_s", 1e-4_cvmix_r8,                    &
                       CVmix_ddiff_params_user)
end if

if (present(mol_diff)) then
  call cvmix_put_ddiff("mol_diff", mol_diff, CVmix_ddiff_params_user)
else
  call cvmix_put_ddiff("mol_diff", 1.5e-6_cvmix_r8,                       &
                       CVmix_ddiff_params_user)
end if

if (present(old_vals)) then
```

```
    select case (trim(old_vals))
      case ("overwrite")
        call cvmix_put_ddiff('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,    &
                             cvmix_ddiff_params_user)
      case ("sum")
        call cvmix_put_ddiff('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS, &
                             cvmix_ddiff_params_user)
      case ("max")
        call cvmix_put_ddiff('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS, &
                             cvmix_ddiff_params_user)
      case DEFAULT
        print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
                "handling old values of diff and visc."
        stop 1
    end select
  else
    call cvmix_put_ddiff('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,        &
                             cvmix_ddiff_params_user)
  end if
```

1.50 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.

USES:

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

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


real(cvmix_r8), dimension(CVmix_vars%max_nlev+1) :: new_Tdiff, new_Sdiff
integer :: nlev, max_nlev
type(cvmix_ddiff_params_type), pointer :: CVmix_ddiff_params_in

if (present(CVmix_ddiff_params_user)) then
  CVmix_ddiff_params_in => CVmix_ddiff_params_user
else
  CVmix_ddiff_params_in => CVmix_ddiff_params_saved
end if
nlev = CVmix_vars%nlev
max_nlev = CVmix_vars%max_nlev

if (.not.associated(CVmix_vars%Tdiff_iface)) &
  call cvmix_put(CVmix_vars, "Tdiff", cvmix_zero, max_nlev)
if (.not.associated(CVmix_vars%Sdiff_iface)) &
  call cvmix_put(CVmix_vars, "Sdiff", cvmix_zero, max_nlev)

call cvmix_coeffs_ddiff(new_Tdiff, new_Sdiff, CVmix_vars%strat_param_num, &
                        CVmix_vars%strat_param_denom, nlev, max_nlev,     &
                        CVmix_ddiff_params_user)
call cvmix_update_wrap(CVmix_ddiff_params_in%handle_old_vals, max_nlev,   &
                       Tdiff_out = CVmix_vars%Tdiff_iface,                &
                       new_Tdiff = new_Tdiff,                             &
                       Sdiff_out = CVmix_vars%Sdiff_iface,                &
                       new_Sdiff = new_Sdiff)
```

1.51 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.

USES:

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

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
```

```
type(cvmix_ddiff_params_type), pointer :: CVmix_ddiff_params_in

if (present(CVmix_ddiff_params_user)) then
  CVmix_ddiff_params_in => CVmix_ddiff_params_user
else
  CVmix_ddiff_params_in => CVmix_ddiff_params_saved
end if

! Determine coefficients
Tdiff_out=cvmix_zero
Sdiff_out=cvmix_zero
do k = 1, nlev
  if ((strat_param_num(k).ge.strat_param_denom(k)).and.                    &
      (strat_param_denom(k).gt.cvmix_zero)) then
    ! Rrho > 1 and dS/dz < 0 => Salt fingering
    Rrho = strat_param_num(k) / strat_param_denom(k)
    if (Rrho.lt.CVmix_ddiff_params_in%strat_param_max) then
      ddiff = (cvmix_one - ((Rrho-cvmix_one)/                              &
              (CVmix_ddiff_params_in%strat_param_max-cvmix_one))**         &
        CVmix_ddiff_params_in%ddiff_exp1)**CVmix_ddiff_params_in%ddiff_exp2
      Sdiff_out(k) = CVmix_ddiff_params_in%kappa_ddiff_s*ddiff
    end if
    Tdiff_out(k) = Sdiff_out(k)*0.7_cvmix_r8
  end if
  if ((strat_param_num(k).ge.strat_param_denom(k)).and.                    &
```

```
      (strat_param_num(k).lt.cvmix_zero)) then
    ! Rrho < 1 and dT/dz > 0 => Diffusive convection
    Rrho = strat_param_num(k) / strat_param_denom(k)
    select case (trim(CVmix_ddiff_params_in%diff_conv_type))
      case ("MC76")
        ddiff = CVmix_ddiff_params_in%mol_diff *                          &
                CVmix_ddiff_params_in%kappa_ddiff_param1 *                &
                exp(CVmix_ddiff_params_in%kappa_ddiff_param2*exp(         &
                    CVmix_ddiff_params_in%kappa_ddiff_param3*             &
                    (cvmix_one/Rrho-cvmix_one)))
      case ("K88")
        ddiff = CVmix_ddiff_params_in%mol_diff * 8.7_cvmix_r8 *           &
                (Rrho**1.1_cvmix_r8)
      case DEFAULT
        print*, "ERROR: ", trim(CVmix_ddiff_params_in%diff_conv_type),    &
                " is not a valid value for diff_conv_type"
        stop 1
    end select
    Tdiff_out(k) = ddiff
    if (Rrho.lt.0.5_cvmix_r8) then
      Sdiff_out(k) = 0.15_cvmix_r8*Rrho*ddiff
    else
      Sdiff_out(k) = (1.85_cvmix_r8*Rrho-0.85_cvmix_r8)*ddiff
    end if
  end if
end do
Tdiff_out(nlev+1) = cvmix_zero
Sdiff_out(nlev+1) = cvmix_zero
```

1.52 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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


type(cvmix_ddiff_params_type), pointer :: CVmix_ddiff_params_out

if (present(CVmix_ddiff_params_user)) then
  CVmix_ddiff_params_out => CVmix_ddiff_params_user
else
  CVmix_ddiff_params_out => CVmix_ddiff_params_saved
end if

select case (trim(varname))
  case ('diff_conv_type')
    select case (trim(val))
      case ('MC76')
        CVmix_ddiff_params_out%diff_conv_type = 'MC76'
      case ('K88')
        CVmix_ddiff_params_out%diff_conv_type = 'K88'
      case DEFAULT
        print*, "ERROR: ", trim(val),                                    &
                " is not a valid value for diff_conv_type"
        stop 1
    end select

  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.53 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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


type(cvmix_ddiff_params_type), pointer :: CVmix_ddiff_params_out

if (present(CVmix_ddiff_params_user)) then
  CVmix_ddiff_params_out => CVmix_ddiff_params_user
else
  CVmix_ddiff_params_out => CVmix_ddiff_params_saved
end if

select case (trim(varname))
  case ('strat_param_max')
    CVmix_ddiff_params_out%strat_param_max = val
  case ('ddiff_exp1')
    CVmix_ddiff_params_out%ddiff_exp1 = val
  case ('ddiff_exp2')
    CVmix_ddiff_params_out%ddiff_exp2 = val
  case ('kappa_ddiff_param1')
    CVmix_ddiff_params_out%kappa_ddiff_param1 = val
  case ('kappa_ddiff_param2')
    CVmix_ddiff_params_out%kappa_ddiff_param2 = val
  case ('kappa_ddiff_param3')
    CVmix_ddiff_params_out%kappa_ddiff_param3 = val
  case ('kappa_ddiff_s')
    CVmix_ddiff_params_out%kappa_ddiff_s = val
  case ('mol_diff')
    CVmix_ddiff_params_out%mol_diff = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
```

```
      stop 1

  end select
```

1.54 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.

USES:

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

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



  type(cvmix_ddiff_params_type), pointer :: CVmix_ddiff_params_out

  if (present(CVmix_ddiff_params_user)) then
    CVmix_ddiff_params_out => CVmix_ddiff_params_user
  else
    CVmix_ddiff_params_out => CVmix_ddiff_params_saved
  end if

  select case (trim(varname))
    case ('old_vals', 'handle_old_vals')
      CVmix_ddiff_params_out%handle_old_vals = val
```

```
  case DEFAULT
    call cvmix_put_ddiff(varname, real(val,cvmix_r8),                    &
                         CVmix_ddiff_params_user)
end select
```

1.55 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.

USES:

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

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
```

```
type(cvmix_ddiff_params_type), pointer :: CVmix_ddiff_params_get

if (present(CVmix_ddiff_params_user)) then
  CVmix_ddiff_params_get => CVmix_ddiff_params_user
else
  CVmix_ddiff_params_get => CVmix_ddiff_params_saved
end if

cvmix_get_ddiff_real = cvmix_zero
select case (trim(varname))
```

```
  case ('strat_param_max')
    cvmix_get_ddiff_real = CVmix_ddiff_params_get%strat_param_max
  case ('ddiff_exp1')
    cvmix_get_ddiff_real = CVmix_ddiff_params_get%ddiff_exp1
  case ('ddiff_exp2')
    cvmix_get_ddiff_real = CVmix_ddiff_params_get%ddiff_exp2
  case ('kappa_ddiff_param1')
    cvmix_get_ddiff_real = CVmix_ddiff_params_get%kappa_ddiff_param1
  case ('kappa_ddiff_param2')
    cvmix_get_ddiff_real = CVmix_ddiff_params_get%kappa_ddiff_param2
  case ('kappa_ddiff_param3')
    cvmix_get_ddiff_real = CVmix_ddiff_params_get%kappa_ddiff_param3
  case ('kappa_ddiff_s')
    cvmix_get_ddiff_real = CVmix_ddiff_params_get%kappa_ddiff_s
  case ('mol_diff')
    cvmix_get_ddiff_real = CVmix_ddiff_params_get%mol_diff
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.56 Fortran: Module Interface cvmix_kpp (Source File: cvmix_kpp.F90)

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
!                       values.
! (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
                                    ! scaling
```

```
      !BGR
      real(cvmix_r8) :: RWHGK_ENTR_COEF,& ! Coefficient and exponent from
                        RWHGK_ENTR_EXP    ! RWHGK16 Langmuir parameterization

  end type cvmix_kpp_params_type
```

1.57 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.

USES:

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

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
```

```
real(cvmix_r8) :: zm, zs, a_m, a_s, c_m, c_s
real(cvmix_r8) :: Cstar_loc, vonkar_loc, surf_layer_ext_loc
real(cvmix_r8) :: nonlocal_coeff

if (present(ri_crit)) then
  if (ri_crit.lt.cvmix_zero) then
    print*, "ERROR: ri_crit can not be negative."
    stop 1
  end if
  call cvmix_put_kpp('Ri_crit', ri_crit, CVmix_kpp_params_user)
else
  call cvmix_put_kpp('Ri_crit', 0.3_cvmix_r8, CVmix_kpp_params_user)
end if

if (present(minOBLdepth)) then
  if (minOBLdepth.lt.cvmix_zero) then
    print*, "ERROR: minOBLdepth can not be negative."
    stop 1
  end if
  call cvmix_put_kpp('minOBLdepth', minOBLdepth, CVmix_kpp_params_user)
else
  call cvmix_put_kpp('minOBLdepth', 0, CVmix_kpp_params_user)
end if

if (present(maxOBLdepth)) then
  if (maxOBLdepth.lt.cvmix_zero) then
    print*, "ERROR: maxOBLdepth can not be negative."
    stop 1
  end if
  call cvmix_put_kpp('maxOBLdepth', maxOBLdepth, CVmix_kpp_params_user)
else
  call cvmix_put_kpp('maxOBLdepth', 0, CVmix_kpp_params_user)
```

```
end if

if (present(minVtsqr)) then
  if (minVtsqr.lt.cvmix_zero) then
    print*, "ERROR: minVtsqr can not be negative."
    stop 1
  end if
  call cvmix_put_kpp('minVtsqr', minVtsqr, CVmix_kpp_params_user)
else
  call cvmix_put_kpp('minVtsqr', 1e-10_cvmix_r8, CVmix_kpp_params_user)
end if

if (present(vonkarman)) then
  if (vonkarman.lt.cvmix_zero) then
    print*, "ERROR: vonkarman can not be negative."
    stop 1
  end if
  vonkar_loc = vonkarman
else
  vonkar_loc = 0.4_cvmix_r8
end if
call cvmix_put_kpp('vonkarman', vonkar_loc, CVmix_kpp_params_user)

if (present(Cstar)) then
  Cstar_loc = Cstar
else
  Cstar_loc = real(10,cvmix_r8)
end if
call cvmix_put_kpp('Cstar', Cstar_loc, CVmix_kpp_params_user)

if (present(zeta_m)) then
  if (zeta_m.ge.cvmix_zero) then
    print*, "ERROR: zeta_m must be negative."
    stop 1
  end if
  zm = zeta_m
else
  ! default value for zeta_m is -1/5
  zm = -0.2_cvmix_r8
end if
call cvmix_put_kpp('zeta_m', zm, CVmix_kpp_params_user)

if (present(zeta_s)) then
  if (zeta_s.ge.cvmix_zero) then
    print*, "ERROR: zeta_s must be negative."
    stop 1
  end if
  zs = zeta_s
```

```
else
  ! Default value for zeta_s is -1
  zs = -cvmix_one
end if
call cvmix_put_kpp('zeta_s', zs, CVmix_kpp_params_user)

! a_m, a_s, c_m, and c_s are computed from zeta_m and zeta_s
! a_m, c_m, and c_s are all non-negative. a_s may be negative depending
! on the value of zeta_s
a_m = ((cvmix_one - real(16,cvmix_r8)*zm)**(-0.25_cvmix_r8))*             &
      (cvmix_one - real(4,cvmix_r8)*zm)
c_m = ((cvmix_one - real(16,cvmix_r8)*zm)**(-0.25_cvmix_r8))*             &
      real(12,cvmix_r8)
call cvmix_put_kpp('a_m', a_m, CVmix_kpp_params_user)
call cvmix_put_kpp('c_m', c_m, CVmix_kpp_params_user)

a_s = sqrt(cvmix_one - real(16,cvmix_r8)*zs)*                              &
      (cvmix_one + real(8,cvmix_r8)*zs)
c_s = real(24,cvmix_r8)*sqrt(cvmix_one - real(16,cvmix_r8)*zs)
call cvmix_put_kpp('a_s', a_s, CVmix_kpp_params_user)
call cvmix_put_kpp('c_s', c_s, CVmix_kpp_params_user)

if (present(surf_layer_ext)) then
  if ((surf_layer_ext.lt.cvmix_zero).or.(surf_layer_ext.gt.cvmix_one))    &
  then
    print*, "surf_layer_ext must be between 0 and 1, inclusive."
    stop 1
  end if
  surf_layer_ext_loc = surf_layer_ext
else
  surf_layer_ext_loc = 0.1_cvmix_r8
end if
call cvmix_put_kpp('surf_layer_ext', surf_layer_ext_loc,                   &
                   CVmix_kpp_params_user)

if (present(Cv)) then
  ! Use scalar Cv parameter
  call cvmix_put_kpp('Cv', CV, CVmix_kpp_params_user)
  call cvmix_put_kpp('lscalar_Cv', .true., CVmix_kpp_params_user)
else
  ! Use Eq. (A3) from Danabasoglu et al.
  call cvmix_put_kpp('lscalar_Cv', .false., CVmix_kpp_params_user)
end if

if (present(interp_type)) then
  select case (trim(interp_type))
    case ('line', 'linear')
      call cvmix_put_kpp('interp_type', CVMIX_MATH_INTERP_LINEAR,          &
```

```
                        CVmix_kpp_params_user)
    case ('quad', 'quadratic')
      call cvmix_put_kpp('interp_type', CVMIX_MATH_INTERP_QUAD,           &
                        CVmix_kpp_params_user)
    case ('cube', 'cubic', 'cubic_spline', 'cubic spline')
      call cvmix_put_kpp('interp_type', CVMIX_MATH_INTERP_CUBE_SPLINE,    &
                        CVmix_kpp_params_user)
    case DEFAULT
      print*, "ERROR: ", trim(interp_type), " is not a valid type of ",   &
              "interpolation!"
      stop 1
  end select
else
  call cvmix_put_kpp('interp_type', CVMIX_MATH_INTERP_QUAD,               &
                     CVmix_kpp_params_user)
end if

if (present(interp_type2)) then
  select case (trim(interp_type2))
    case ('line', 'linear')
      call cvmix_put_kpp('interp_type2', CVMIX_MATH_INTERP_LINEAR,        &
                        CVmix_kpp_params_user)
    case ('quad', 'quadratic')
      call cvmix_put_kpp('interp_type2', CVMIX_MATH_INTERP_QUAD,          &
                        CVmix_kpp_params_user)
    case ('cube', 'cubic', 'cubic_spline', 'cubic spline')
      call cvmix_put_kpp('interp_type2', CVMIX_MATH_INTERP_CUBE_SPLINE,   &
                        CVmix_kpp_params_user)
    case ('POP','LMD94')
      call cvmix_put_kpp('interp_type2', CVMIX_KPP_INTERP_LMD94,          &
                        CVmix_kpp_params_user)
    case DEFAULT
      print*, "ERROR: ", trim(interp_type2), " is not a valid type of ",  &
              "interpolation!"
      stop 1
  end select
else
  call cvmix_put_kpp('interp_type2', CVMIX_KPP_INTERP_LMD94,              &
                     CVmix_kpp_params_user)
end if

if (present(MatchTechnique)) then
  select case (trim(MatchTechnique))
    case ('MatchBoth')
      call cvmix_put_kpp('MatchTechnique', CVMIX_KPP_MATCH_BOTH,          &
                        CVmix_kpp_params_user)
    case ('MatchGradient')
      call cvmix_put_kpp('MatchTechnique', CVMIX_KPP_MATCH_GRADIENT,      &
```

```
                        CVmix_kpp_params_user)
    case ('SimpleShapes')
      call cvmix_put_kpp('MatchTechnique', CVMIX_KPP_SIMPLE_SHAPES,       &
                        CVmix_kpp_params_user)
    case ('ParabolicNonLocal')
      call cvmix_put_kpp('MatchTechnique', CVMIX_KPP_PARABOLIC_NONLOCAL,  &
                        CVmix_kpp_params_user)
    case DEFAULT
      print*, "ERROR: ", trim(MatchTechnique), " is not a valid choice ", &
              "for MatchTechnique!"
      stop 1
    end select
else
  call cvmix_put_kpp('MatchTechnique', CVMIX_KPP_SIMPLE_SHAPES,           &
                     CVmix_kpp_params_user)
end if

if (present(old_vals)) then
  select case (trim(old_vals))
    case ("overwrite")
      call cvmix_put_kpp('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,      &
                         cvmix_kpp_params_user)
    case ("sum")
      call cvmix_put_kpp('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS,   &
                         cvmix_kpp_params_user)
    case ("max")
      call cvmix_put_kpp('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS,   &
                         cvmix_kpp_params_user)
    case DEFAULT
      print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
              "handling old values of diff and visc."
      stop 1
  end select
else
  call cvmix_put_kpp('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,          &
                         cvmix_kpp_params_user)
end if

if (present(lStokesMOST)) then
   call cvmix_put_kpp('lStokesMOST',lStokesMOST , CVmix_kpp_params_user)
else
  call cvmix_put_kpp('lStokesMOST', .false., CVmix_kpp_params_user)
end if

if (present(lEkman)) then
  call cvmix_put_kpp('lEkman', lEkman, CVmix_kpp_params_user)
else
  call cvmix_put_kpp('lEkman', .false., CVmix_kpp_params_user)
```

```
end if

if (present(lMonOb)) then
  call cvmix_put_kpp('lMonOb', lMonOb, CVmix_kpp_params_user)
else
  call cvmix_put_kpp('lMonOb', .false., CVmix_kpp_params_user)
end if

if (present(lnoDGat1)) then
  call cvmix_put_kpp('lnoDGat1', lnoDGat1, CVmix_kpp_params_user)
else
  call cvmix_put_kpp('lnoDGat1', .true., CVmix_kpp_params_user)
end if

if (present(lenhanced_diff)) then
  call cvmix_put_kpp('lenhanced_diff', lenhanced_diff,                    &
                     CVmix_kpp_params_user)
else
  call cvmix_put_kpp('lenhanced_diff', .true., CVmix_kpp_params_user)
end if

if (present(Langmuir_mixing_str)) then
   select case (trim(Langmuir_mixing_str))
   case ("LWF16")
      call cvmix_put_kpp('Langmuir_Mixing_Opt', LANGMUIR_MIXING_LWF16 ,   &
           CVmix_kpp_params_user)
   case ("RWHGK16")
      call cvmix_put_kpp('Langmuir_Mixing_Opt',                           &
           LANGMUIR_MIXING_RWHGK16, CVmix_kpp_params_user)
   case ("NONE")
      call cvmix_put_kpp('Langmuir_Mixing_Opt',                           &
           NO_LANGMUIR_MIXING, CVmix_kpp_params_user)
   case DEFAULT
      print*, "ERROR: ", trim(Langmuir_mixing_str), " is not a valid ",   &
              "option for Langmuir_mixing_str!"
      stop 1
   end select
else
   call cvmix_put_kpp('Langmuir_Mixing_Opt',                              &
        NO_LANGMUIR_MIXING, CVmix_kpp_params_user)
end if

if (present(Langmuir_entrainment_str)) then
   select case (trim(Langmuir_entrainment_str))
   case ("LWF16")
      call cvmix_put_kpp('Langmuir_Entrainment_Opt',                      &
           LANGMUIR_ENTRAINMENT_LWF16, CVmix_kpp_params_user)
   case ("LF17")
```

```
      call cvmix_put_kpp('Langmuir_Entrainment_Opt',                        &
           LANGMUIR_ENTRAINMENT_LF17, CVmix_kpp_params_user)
   case ("RWHGK16")
      call cvmix_put_kpp('Langmuir_Entrainment_Opt',                        &
           LANGMUIR_ENTRAINMENT_RWHGK16, CVmix_kpp_params_user)
   case ("NONE")
      call cvmix_put_kpp('Langmuir_Entrainment_Opt',                        &
           NO_LANGMUIR_ENTRAINMENT, CVmix_kpp_params_user)
   case DEFAULT
      print*, "ERROR: ", trim(Langmuir_entrainment_str), " is not a ",     &
              "valid option for Langmuir_entrainment_str!"
      stop 1
   end select
else
   call cvmix_put_kpp('Langmuir_Entrainment_Opt',                           &
        NO_LANGMUIR_ENTRAINMENT, CVmix_kpp_params_user)
end if

! By default, assume that G(0) = 0 for nonlocal term
nonlocal_coeff = (Cstar_loc*vonkar_loc*                                    &
                  (vonkar_loc*surf_layer_ext_loc*c_s)**                    &
                  (cvmix_one/real(3,cvmix_r8)))
if (present(lnonzero_surf_nonlocal)) then
  if (lnonzero_surf_nonlocal) then
    nonlocal_coeff = real(1,cvmix_r8)
  end if
end if
call cvmix_put_kpp('nonlocal_coeff',nonlocal_coeff,CVmix_kpp_params_user)

! By default, use sigma construction from Danabasoglu et al. when computing
! turbulent scales. Set l_LMD_ws = .true. to use Large et al. construction.
if (present(l_LMD_ws)) then
  call cvmix_put_kpp('l_LMD_ws', l_LMD_ws,                    &
                     CVmix_kpp_params_user)
else
  call cvmix_put_kpp('l_LMD_ws', .false., CVmix_kpp_params_user)
end if

! Initialize parameters for enhanced entrainment
call cvmix_put_kpp('c_ST', 0.17_cvmix_r8, CVmix_kpp_params_user)
call cvmix_put_kpp('c_CT', 0.15_cvmix_r8, CVmix_kpp_params_user)
call cvmix_put_kpp('c_LT', 0.083_cvmix_r8, CVmix_kpp_params_user)
call cvmix_put_kpp('p_LT', 2.0_cvmix_r8, CVmix_kpp_params_user)
call cvmix_put_kpp('RWHGK_ENTR_COEF', 2.3_cvmix_r8, CVmix_kpp_params_user)
call cvmix_put_kpp('RWHGK_ENTR_EXP', -0.5_cvmix_r8, CVmix_kpp_params_user)
```

1.58 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.

USES:

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

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
```

```
real(cvmix_r8), dimension(CVmix_vars%max_nlev+1) :: new_Mdiff, new_Tdiff, &
                                                    new_Sdiff
integer :: nlev, max_nlev
type(cvmix_kpp_params_type),  pointer :: CVmix_kpp_params_in

CVmix_kpp_params_in => CVmix_kpp_params_saved
if (present(CVmix_kpp_params_user)) then
  CVmix_kpp_params_in => CVmix_kpp_params_user
end if

nlev = CVmix_vars%nlev
max_nlev = CVmix_vars%max_nlev

if (.not.associated(CVmix_vars%Mdiff_iface)) &
  call cvmix_put(CVmix_vars, "Mdiff", cvmix_zero, max_nlev)
if (.not.associated(CVmix_vars%Tdiff_iface)) &
  call cvmix_put(CVmix_vars, "Tdiff", cvmix_zero, max_nlev)
if (.not.associated(CVmix_vars%Sdiff_iface)) &
  call cvmix_put(CVmix_vars, "Sdiff", cvmix_zero, max_nlev)
```

```
call cvmix_put(CVmix_vars, 'kpp_transport', cvmix_zero, max_nlev)

call cvmix_coeffs_kpp(new_Mdiff, new_Tdiff, new_Sdiff,                    &
                      CVmix_vars%zw_iface, CVmix_vars%zt_cntr,            &
                      CVmix_vars%Mdiff_iface, CVmix_vars%Tdiff_iface,     &
                      CVMix_vars%Sdiff_iface,                             &
                      CVmix_vars%BoundaryLayerDepth,                      &
                      CVmix_vars%kOBL_depth,                              &
                      CVmix_vars%kpp_Tnonlocal_iface,                     &
                      CVmix_vars%kpp_Snonlocal_iface,                     &
                      CVmix_vars%SurfaceFriction,                         &
                      CVmix_vars%SurfaceBuoyancyForcing,                  &
                      nlev, max_nlev,                                     &
                      CVmix_vars%LangmuirEnhancementFactor,               &
                      CVmix_vars%StokesMostXi,                            &
                      CVmix_kpp_params_user)

call cvmix_update_wrap(CVmix_kpp_params_in%handle_old_vals, max_nlev,     &
                       Mdiff_out = CVmix_vars%Mdiff_iface,                &
                       new_Mdiff = new_Mdiff,                             &
                       Tdiff_out = CVmix_vars%Tdiff_iface,                &
                       new_Tdiff = new_Tdiff,                             &
                       Sdiff_out = CVmix_vars%Sdiff_iface,                &
                       new_Sdiff = new_Sdiff)
```

1.59 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.

USES:

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

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



    ! Local variables
    type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_in

    ! OBL_[MTS]diff are the diffusivities in the whole OBL
    real(cvmix_r8), dimension(nint(kOBL_depth)) :: OBL_Mdiff, OBL_Tdiff,     &
                                                   OBL_Sdiff

    ! [MTS]diff_ktup are the enhanced diffusivity and viscosity values at the
    ! deepest cell center above OBL_depth. Other _ktup vars are intermediary
    ! variables needed to compute [MTS]diff_ktup
    real(cvmix_r8) :: Mdiff_ktup, Tdiff_ktup, Sdiff_ktup
    real(cvmix_r8) :: sigma_ktup, wm_ktup, ws_ktup

    real(cvmix_r8) :: delta

    real(cvmix_r8), dimension(nlev+1) :: sigma, w_m, w_s

    ! [MTS]shape are the coefficients of the shape function in the gradient
```

```
! term; [TS]shape2 are the coefficients for the nonlocal term;
! NMshape is the coefficient for the no-matching case, an option to shape
! a Langmuir enhancement
real(cvmix_r8), dimension(4) :: Mshape, Tshape, Sshape, Tshape2, Sshape2,&
                                NMshape

! [MTS]shapeAt1 is value of shape function at sigma = 1
! d[MTS]shapeAt1 is value of derivative of shape function at sigma = 1
! (Used for matching the shape function at OBL depth)
real(cvmix_r8) :: MshapeAt1, TshapeAt1, SshapeAt1
real(cvmix_r8) :: dMshapeAt1, dTshapeAt1, dSshapeAt1

! [MTS]shapeAtS is value of shape function at sigma = S
real(cvmix_r8) :: MshapeAtS, TshapeAtS, SshapeAtS, GAtS
! Storing the maximum value of shape function for no-matching case
!  that is used as an option for Langmuir mixing
real(cvmix_r8), parameter :: NMshapeMax = 4./27.

! [MTS]diff_OBL is value of diffusivity at OBL depth
! d[MTS]diff_OBL is value of derivative of diffusivity at OBL depth
! w[ms]_OBL is value of wm or ws at OBL depth
real(cvmix_r8) :: Mdiff_OBL, Tdiff_OBL, Sdiff_OBL
real(cvmix_r8) :: dMdiff_OBL, dTdiff_OBL, dSdiff_OBL
real(cvmix_r8) :: wm_OBL, ws_OBL, second_term

! coefficients used for interpolation if interp_type2 is not 'LMD94'
real(kind=cvmix_r8), dimension(4) :: coeffs

! Width of column kw_up and kw_up+1
real(cvmix_r8), dimension(2) :: col_widths, col_centers
real(cvmix_r8), dimension(2) :: Mdiff_vals, Tdiff_vals, Sdiff_vals

! Parameters for RWHGK16 Langmuir parameterization
real(cvmix_r8) :: MixingCoefEnhancement
real(cvmix_r8) :: ShapeNoMatchAtS

! Parameters for Stokes_MOST
real(cvmix_r8) :: Gcomposite, Hsigma, sigh, T_NLenhance , S_NLenhance , XIone

! Constant from params
integer :: interp_type2, MatchTechnique

integer :: kw
logical :: lstable
integer :: ktup, & ! kt index of cell center above OBL_depth
           kwup    ! kw index of iface above OBL_depth (= kt index of
                   ! cell containing OBL_depth)
```

```
  CVmix_kpp_params_in => CVmix_kpp_params_saved
  if (present(CVmix_kpp_params_user)) then
    CVmix_kpp_params_in => CVmix_kpp_params_user
  end if
  interp_type2   = CVmix_kpp_params_in%interp_type2
  MatchTechnique = CVmix_kpp_params_in%MatchTechnique

  ! Output values should be set to input values
  Mdiff_out = old_Mdiff
  Tdiff_out = old_Tdiff
  Sdiff_out = old_Sdiff

  ! (1) Column-specific parameters
  !
  ! Stability => positive surface buoyancy flux
  lstable = (surf_buoy.gt.cvmix_zero)

  kwup = floor(kOBL_depth)
  ktup = nint(kOBL_depth)-1

  if (ktup.eq.nlev) then
    ! OBL_depth between bottom cell center and ocean bottom, assume
    ! zt(ktup+1) = ocn_bottom (which is zw(nlev+1)
    delta = (OBL_depth+zt(ktup))/(zt(ktup)-zw(ktup+1))
  else
    delta = (OBL_depth+zt(ktup))/(zt(ktup)-zt(ktup+1))
  end if

if ( CVmix_kpp_params_in%lStokesMOST ) then             ! Stokes_MOST

      ! (2a) Compute turbulent scales at OBL depth
      call cvmix_kpp_compute_turbulent_scales(cvmix_one, OBL_depth,         &
                                              surf_buoy, surf_fric,         &
                              StokesXI,       wm_OBL, ws_OBL,               &
                                              CVmix_kpp_params_user)

      ! (2b) Compute diffusivities at OBL depth
        col_centers(1) = zt(kwup)
        col_widths(1) = zw(kwup) - zw(kwup+1)
        Mdiff_vals(1) = old_Mdiff(kwup+1)
        Tdiff_vals(1) = old_Tdiff(kwup+1)
        Sdiff_vals(1) = old_Sdiff(kwup+1)
        if (kwup.eq.nlev) then
          col_centers(2) = zw(kwup+1)
          col_widths(2)  = 1.0_cvmix_r8 !Value doesn't matter, will divide into zero
          Mdiff_vals(2)  = old_Mdiff(kwup+1)   !Mdiff_out(kwup+1)
          Tdiff_vals(2)  = old_Tdiff(kwup+1)
          Sdiff_vals(2)  = old_Sdiff(kwup+1)
```

```
      else
        col_centers(2) = zt(kwup+1)
        col_widths(2)  = zw(kwup+1) - zw(kwup+2)
        Mdiff_vals(2)  = old_Mdiff(kwup+2)
        Tdiff_vals(2)  = old_Tdiff(kwup+2)
        Sdiff_vals(2)  = old_Sdiff(kwup+2)
      end if
      if (kwup.eq.1) then
        Mdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,               &
                                                Mdiff_vals, OBL_depth,    &
                                                dnu_dz=dMdiff_OBL)
        Tdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,               &
                                                Tdiff_vals, OBL_depth,    &
                                                dnu_dz=dTdiff_OBL)
        Sdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,               &
                                                Sdiff_vals, OBL_depth,    &
                                                dnu_dz=dSdiff_OBL)
      else ! interp_type == 'LMD94' and kwup > 1
        Mdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,               &
                                                Mdiff_vals, OBL_depth,    &
                                                old_Mdiff(kwup),          &
                                                dnu_dz=dMdiff_OBL)
        Tdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,               &
                                                Tdiff_vals, OBL_depth,    &
                                                old_Tdiff(kwup),          &
                                                dnu_dz=dTdiff_OBL)
        Sdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,               &
                                                Sdiff_vals, OBL_depth,    &
                                                old_Sdiff(kwup),          &
                                                dnu_dz=dSdiff_OBL)
      end if

!     (3a) Compute turbulent scales at interfaces throughout column
sigma = -zw(1:nlev+1)/OBL_depth
call cvmix_kpp_compute_turbulent_scales(sigma, OBL_depth, surf_buoy, & !_1d
            surf_fric,   xi=StokesXI,    w_m=w_m, w_s=w_s,           &
            CVmix_kpp_params_user = CVmix_kpp_params_user)

do kw=2,kwup                                    ! OBL overwrite loop to kwup
!     (3b) Evaluate G(sigma) >= 0  at each cell interface
  Gcomposite = cvmix_kpp_composite_shape(sigma(kw))
  sigh   = MAX(CVmix_kpp_params_in%surf_layer_ext, MIN(sigma(kw) ,cvmix_one))
```

```
      Hsigma = ((sigh      - CVmix_kpp_params_in%surf_layer_ext) / &
                (cvmix_one - CVmix_kpp_params_in%surf_layer_ext) )**2
!     Hsigma = MAX( cvmix_zero , MIN( cvmix_one , Hsigma ) )

    !   (3c) Compute nonlocal term at each cell interface
      if (.not.lstable) then
        Tnonlocal(kw) = 4.7 * Gcomposite ! LMD 6.26 Gcubic
        Snonlocal(kw) = 4.7 * Gcomposite
      else
        Tnonlocal(kw) = cvmix_zero
        Snonlocal(kw) = cvmix_zero
      end if

    !   (3d) Diffusivity = (OBL_depth * turbulent scale * G(sigma) + Xdiff_OBL * Hsigma)
      OBL_Mdiff(kw) = OBL_depth * w_m(kw) * Gcomposite  + Mdiff_OBL * Hsigma
      OBL_Tdiff(kw) = OBL_depth * w_s(kw) * Gcomposite  + Tdiff_OBL * Hsigma
      OBL_Sdiff(kw) = OBL_depth * w_s(kw) * Gcomposite  + Sdiff_OBL * Hsigma

    end do

    ! (4) Compute the enhanced diffusivity
    !     (4a) Compute shape function at last cell center in OBL
    sigma_ktup = -zt(ktup)/OBL_depth
    Gcomposite =  cvmix_kpp_composite_shape(sigma_ktup)
    sigh   = MAX(CVmix_kpp_params_in%surf_layer_ext, MIN(sigma_ktup ,cvmix_one))
    Hsigma = ( (sigh - CVmix_kpp_params_in%surf_layer_ext) / &
                (cvmix_one - CVmix_kpp_params_in%surf_layer_ext) )**2

    !     (4b) Compute turbulent scales at last cell center in OBL
    call cvmix_kpp_compute_turbulent_scales(sigma_ktup, OBL_depth, surf_buoy, & !0d
                    surf_fric, StokesXI,  wm_ktup, ws_ktup,      &
                   CVmix_kpp_params_user)

    !     (4c) Diffusivity at last cell center in OBL
    Mdiff_ktup = OBL_depth * wm_ktup * Gcomposite  + Mdiff_OBL * Hsigma
    Tdiff_ktup = OBL_depth * ws_ktup * Gcomposite  + Tdiff_OBL * Hsigma
    Sdiff_ktup = OBL_depth * ws_ktup * Gcomposite  + Sdiff_OBL * Hsigma

    if (CVmix_kpp_params_in%lenhanced_diff) then
      if ((ktup.eq.kwup).or.(ktup.eq.kwup-1)) then
        T_NLenhance =  Tnonlocal(ktup+1)
        S_NLenhance =  Snonlocal(ktup+1)
        call cvmix_kpp_compute_enhanced_diff(Mdiff_ktup,                    &
                                             Tdiff_ktup,                    &
                                             Sdiff_ktup,                    &
                                             Mdiff_out(ktup+1),             &
                                             Tdiff_out(ktup+1),             &
                                             Sdiff_out(ktup+1),             &
```

```
                                                OBL_Mdiff(ktup+1),               &
                                                OBL_Tdiff(ktup+1),               &
                                                OBL_Sdiff(ktup+1),               &
                                                T_NLenhance      ,               &
                                                S_NLenhance      ,               &
                                                delta, lkteqkw=(ktup.eq.kwup))
    else
      print*, "ERROR: ktup should be either kwup or kwup-1!"
      print*, "ktup = ", ktup, " and kwup = ", kwup
      stop 1
    end if
  else
    if ( kwup .eq. ktup )  then
      OBL_Mdiff(ktup+1) = old_Mdiff(ktup+1)
      OBL_Tdiff(ktup+1) = old_Tdiff(ktup+1)
      OBL_Sdiff(ktup+1) = old_Sdiff(ktup+1)
    end if
  end if

  ! (5) Combine interior and boundary coefficients

  Mdiff_out(2:ktup+1) = OBL_Mdiff(2:ktup+1)
  Tdiff_out(2:ktup+1) = OBL_Tdiff(2:ktup+1)
  Sdiff_out(2:ktup+1) = OBL_Sdiff(2:ktup+1)
else ! not  Stokes_MOST

  XIone = cvmix_one
  ! (2) Compute coefficients of shape function
  !     A no-match case is stored for use in Langmuir scheme
  NMshape(1) = cvmix_zero
  NMshape(2) = cvmix_one
  NMshape(3) = -real(2,cvmix_r8)
  NMshape(4) = cvmix_one
  select case (MatchTechnique)
    case (CVMIX_KPP_SIMPLE_SHAPES)
      ! Simple shape function is sigma*(1-sigma)^2
      Mshape(1) =  cvmix_zero
      Mshape(2) =  cvmix_one
      Mshape(3) = -real(2,cvmix_r8)
      Mshape(4) =  cvmix_one
      Tshape    = Mshape
      Sshape    = Mshape
      Tshape2   = Tshape
      Sshape2   = Sshape
    case (CVMIX_KPP_PARABOLIC_NONLOCAL)
      ! Shape function is sigma*(1-sigma)^2 for gradient term
      ! and (1-sigma)^2 for non-local term
      Mshape(1) =  cvmix_zero
```

```
  Mshape(2) =  cvmix_one
  Mshape(3) = -real(2,cvmix_r8)
  Mshape(4) =  cvmix_one
  Tshape    = Mshape
  Sshape    = Mshape
  Tshape2(1) =  cvmix_one
  Tshape2(2) = -real(2,cvmix_r8)
  Tshape2(3) =  cvmix_one
  Tshape2(4) =  cvmix_zero
  Sshape2    = Tshape2
case DEFAULT
  ! (2a) Compute turbulent scales at OBL depth
  call cvmix_kpp_compute_turbulent_scales(cvmix_one, OBL_depth,         &
                                          surf_buoy, surf_fric,         &
                                          XIone, wm_OBL, ws_OBL,        &
                                          CVmix_kpp_params_user)
  if (CVMix_KPP_Params_in%Langmuir_Mixing_Opt &
     .eq. LANGMUIR_MIXING_LWF16) then
    ! enhance the turbulent velocity scale
    wm_OBL = wm_OBL * Langmuir_EFactor
    ws_OBL = ws_OBL * Langmuir_EFactor
  end if
  ! (2b) Compute diffusivities at OBL depth
  if (interp_type2.ne.CVMIX_KPP_INTERP_LMD94) then
    if (kwup.eq.1) then
      call cvmix_math_poly_interp(coeffs, interp_type2, zw(kwup:kwup+1),&
                                  old_Mdiff(kwup:kwup+1))
      Mdiff_OBL = cvmix_math_evaluate_cubic(coeffs, -OBL_depth,         &
                                            dMdiff_OBL)

      call cvmix_math_poly_interp(coeffs, interp_type2, zw(kwup:kwup+1),&
                                  old_Tdiff(kwup:kwup+1))
      Tdiff_OBL = cvmix_math_evaluate_cubic(coeffs, -OBL_depth,         &
                                            dTdiff_OBL)

      call cvmix_math_poly_interp(coeffs, interp_type2, zw(kwup:kwup+1),&
                                  old_Sdiff(kwup:kwup+1))
      Sdiff_OBL = cvmix_math_evaluate_cubic(coeffs, -OBL_depth,         &
                                            dSdiff_OBL)
    else ! interp_type2 != 'LMD94' and kwup > 1
      call cvmix_math_poly_interp(coeffs, interp_type2, zw(kwup:kwup+1),&
                                  old_Mdiff(kwup:kwup+1), zw(kwup-1),   &
                                  old_Mdiff(kwup-1))
      Mdiff_OBL = cvmix_math_evaluate_cubic(coeffs, -OBL_depth,         &
                                            dMdiff_OBL)

      call cvmix_math_poly_interp(coeffs, interp_type2, zw(kwup:kwup+1),&
                                  old_Tdiff(kwup:kwup+1), zw(kwup-1),   &
```

```
                                old_Tdiff(kwup-1))
    Tdiff_OBL = cvmix_math_evaluate_cubic(coeffs, -OBL_depth,         &
                                          dTdiff_OBL)

    call cvmix_math_poly_interp(coeffs, interp_type2, zw(kwup:kwup+1),&
                                old_Sdiff(kwup:kwup+1), zw(kwup-1),   &
                                old_Sdiff(kwup-1))
    Sdiff_OBL = cvmix_math_evaluate_cubic(coeffs, -OBL_depth,         &
                                          dSdiff_OBL)
  end if
else ! interp_type2 == 'LMD94'
  col_centers(1) = zt(kwup)
  col_widths(1) = zw(kwup) - zw(kwup+1)
  Mdiff_vals(1) = old_Mdiff(kwup+1)
  Tdiff_vals(1) = old_Tdiff(kwup+1)
  Sdiff_vals(1) = old_Sdiff(kwup+1)
  if (kwup.eq.nlev) then
    col_centers(2) = zw(kwup+1)
    col_widths(2)  = 1.0_cvmix_r8 ! Value doesn't matter, will divide
                                  ! into zero
    Mdiff_vals(2)  = old_Mdiff(kwup+1)
    Tdiff_vals(2)  = old_Tdiff(kwup+1)
    Sdiff_vals(2)  = old_Sdiff(kwup+1)
  else
    col_centers(2) = zt(kwup+1)
    col_widths(2)  = zw(kwup+1) - zw(kwup+2)
    Mdiff_vals(2)  = old_Mdiff(kwup+2)
    Tdiff_vals(2)  = old_Tdiff(kwup+2)
    Sdiff_vals(2)  = old_Sdiff(kwup+2)
  end if

  if (kwup.eq.1) then
    Mdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,           &
                                                Mdiff_vals, OBL_depth,  &
                                                dnu_dz=dMdiff_OBL)
    Tdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,           &
                                                Tdiff_vals, OBL_depth,  &
                                                dnu_dz=dTdiff_OBL)
    Sdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,           &
                                                Sdiff_vals, OBL_depth,  &
                                                dnu_dz=dSdiff_OBL)
  else ! interp_type == 'LMD94' and kwup > 1
    Mdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                col_widths,           &
                                                Mdiff_vals, OBL_depth,  &
```

```
                                                  old_Mdiff(kwup),        &
                                                  dnu_dz=dMdiff_OBL)
    Tdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                  col_widths,             &
                                                  Tdiff_vals, OBL_depth,  &
                                                  old_Tdiff(kwup),        &
                                                  dnu_dz=dTdiff_OBL)
    Sdiff_OBL = cvmix_kpp_compute_nu_at_OBL_depth_LMD94(col_centers,  &
                                                  col_widths,             &
                                                  Sdiff_vals, OBL_depth,  &
                                                  old_Sdiff(kwup),        &
                                                  dnu_dz=dSdiff_OBL)
  end if
end if ! interp_type != "LMD94"

! (2c) Compute G(1) [shape function when sigma = 1] and G'(1) for three
!      cases:
!      i) momentum diffusivity (viscosity)
!      ii) temperature diffusivity
!      iii) other tracers diffusivity
! Notes:
!   * We are computing G(1) and G'(1) so we can represent G(sigma) as a
!     cubic polynomial and then compute Kx = OBL_depth*wx*G. If either
!     OBL_depth or wx are 0, it doesn't matter what G is because Kx
!     will be zero everywhere... in these cases, we set G(1)=G'(1)=0.
!   * If OBL_depth = 0, the above note applies to all three situations
!     listed as (i), (ii), and (iii). If ws = 0, it applies only to (i)
!     and (ii). If wm = 0, it applies only to (iii).
if (OBL_depth.eq.cvmix_zero) then
  ! Values don't matter, K = 0
  MshapeAt1 = cvmix_zero
  TshapeAt1 = cvmix_zero
  SshapeAt1 = cvmix_zero
  dMshapeAt1 = cvmix_zero
  dTshapeAt1 = cvmix_zero
  dSshapeAt1 = cvmix_zero
else ! OBL_depth != 0
  if (wm_OBL.ne.cvmix_zero) then
    MshapeAt1 = Mdiff_OBL/(wm_OBL*OBL_depth)
  else
    MshapeAt1 = cvmix_zero ! value doesn't really matter, Km = 0
  end if
  if (ws_OBL.ne.cvmix_zero) then
    TshapeAt1 = Tdiff_OBL/(ws_OBL*OBL_depth)
    SshapeAt1 = Sdiff_OBL/(ws_OBL*OBL_depth)
  else
    TshapeAt1 = cvmix_zero ! value doesn't really matter, Ks = 0
    SshapeAt1 = cvmix_zero ! value doesn't really matter, Ks = 0
```

```
  end if
  if (CVmix_kpp_params_in%lnoDGat1) then
    ! Force G'(1) = 0
    dMshapeAt1 = cvmix_zero
    dTshapeAt1 = cvmix_zero
    dSshapeAt1 = cvmix_zero
  else
    second_term = real(5,cvmix_r8)*surf_buoy/(surf_fric**4)
    if (wm_OBL.ne.cvmix_zero) then
      dMshapeAt1 = -dMdiff_OBL/wm_OBL
      if (lstable) &
        dMshapeAt1 = dMshapeAt1 + second_term*Mdiff_OBL
    else
      dMshapeAt1 = cvmix_zero ! value doesn't really matter, Km = 0
    end if
    if (ws_OBL.ne.cvmix_zero) then
      dTshapeAt1 = -dTdiff_OBL/ws_OBL
      dSshapeAt1 = -dSdiff_OBL/ws_OBL
      if (lstable) then
        dTshapeAt1 = dTshapeAt1 + second_term*Tdiff_OBL
        dSshapeAt1 = dSshapeAt1 + second_term*Sdiff_OBL
      end if
    else
      dTshapeAt1 = cvmix_zero ! value doesn't really matter, Ks = 0
      dSshapeAt1 = cvmix_zero ! value doesn't really matter, Ks = 0
    end if
    dMshapeAt1 = min(dMshapeAt1, cvmix_zero) ! non-positive value!
    dTshapeAt1 = min(dTshapeAt1, cvmix_zero) ! non-positive value!
    dSshapeAt1 = min(dSshapeAt1, cvmix_zero) ! non-positive value!
  end if ! lnoDGat1
end if ! OBL_depth == 0

!   (2d) Compute coefficients of shape function
call cvmix_kpp_compute_shape_function_coeffs(MshapeAt1, dMshapeAt1,   &
                                             Mshape)
call cvmix_kpp_compute_shape_function_coeffs(TshapeAt1, dTshapeAt1,   &
                                             Tshape)
call cvmix_kpp_compute_shape_function_coeffs(SshapeAt1, dSshapeAt1,   &
                                             Sshape)
if (MatchTechnique.eq.CVMIX_KPP_MATCH_GRADIENT) then
  ! Only match for gradient term, use simple shape for nonlocal
  Tshape2(1) =  cvmix_zero
  Tshape2(2) =  cvmix_one
  Tshape2(3) = -real(2,cvmix_r8)
  Tshape2(4) =  cvmix_one
  Sshape2 = Tshape2
else
  ! Shape function is the same for gradient and nonlocal
```

```
      Tshape2 = Tshape
      Sshape2 = Sshape
    end if
end select

! (3) Use shape function to compute diffusivities throughout OBL
Tnonlocal = cvmix_zero
Snonlocal = cvmix_zero
OBL_Mdiff = cvmix_zero
OBL_Tdiff = cvmix_zero
OBL_Sdiff = cvmix_zero
sigma = -zw(1:nlev+1)/OBL_depth
!     (3a) Compute turbulent scales throghout column
call cvmix_kpp_compute_turbulent_scales(sigma, OBL_depth, surf_buoy,   &
                                        surf_fric, XIone, w_m, w_s,    &
                                        CVmix_kpp_params_user)
do kw=2,kwup
  !   (3b) Evaluate G(sigma) at each cell interface
  MshapeAtS = cvmix_math_evaluate_cubic(Mshape, sigma(kw))
  TshapeAtS = cvmix_math_evaluate_cubic(Tshape, sigma(kw))
  SshapeAtS = cvmix_math_evaluate_cubic(Sshape, sigma(kw))
  ! The RWHGK16 Langmuir uses the shape function to shape the
  !  enhancement to the mixing coefficient.
  ShapeNoMatchAtS = cvmix_math_evaluate_cubic(NMshape, sigma(kw))
  !   (3c) Compute nonlocal term at each cell interface
  if (.not.lstable) then
    GAtS = cvmix_math_evaluate_cubic(Tshape2, sigma(kw))
    Tnonlocal(kw) = CVmix_kpp_params_in%nonlocal_coeff*GAtS
    GAtS = cvmix_math_evaluate_cubic(Sshape2, sigma(kw))
    Snonlocal(kw) = CVmix_kpp_params_in%nonlocal_coeff*GAtS
  end if

  select case (CVMix_KPP_Params_in%Langmuir_Mixing_Opt)
  case (LANGMUIR_MIXING_LWF16)
     MixingCoefEnhancement = Langmuir_EFactor
  case (LANGMUIR_MIXING_RWHGK16)
     MixingCoefEnhancement = cvmix_one + ShapeNoMatchAtS/NMshapeMax * &
                             (Langmuir_EFactor - cvmix_one)
  case default
     MixingCoefEnhancement = cvmix_one
  end select
  !   (3d) Diffusivity = OBL_depth * (turbulent scale) * G(sigma)
  OBL_Mdiff(kw) = OBL_depth * w_m(kw) * MshapeAtS * MixingCoefEnhancement
  OBL_Tdiff(kw) = OBL_depth * w_s(kw) * TshapeAtS * MixingCoefEnhancement
  OBL_Sdiff(kw) = OBL_depth * w_s(kw) * SshapeAtS * MixingCoefEnhancement
end do

! (4) Compute the enhanced diffusivity
```

```
!     (4a) Compute shape function at last cell center in OBL
sigma_ktup = -zt(ktup)/OBL_depth
MshapeAtS = cvmix_math_evaluate_cubic(Mshape, sigma_ktup)
TshapeAtS = cvmix_math_evaluate_cubic(Tshape, sigma_ktup)
SshapeAtS = cvmix_math_evaluate_cubic(Sshape, sigma_ktup)
!     (4b) Compute turbulent scales at last cell center in OBL
call cvmix_kpp_compute_turbulent_scales(sigma_ktup, OBL_depth, surf_buoy, &
                                        surf_fric, XIone, wm_ktup, ws_ktup, &
                                        CVmix_kpp_params_user)
if (CVMix_KPP_Params_in%Langmuir_Mixing_Opt &
   .eq. LANGMUIR_MIXING_LWF16) then
  ! enhance the turbulent velocity scale
  wm_ktup = wm_ktup * Langmuir_EFactor
  ws_ktup = ws_ktup * Langmuir_EFactor
end if
!     (4c) Diffusivity = OBL_depth * (turbulent scale) * G(sigma)
Mdiff_ktup = OBL_depth * wm_ktup * MshapeAtS
Tdiff_ktup = OBL_depth * ws_ktup * TshapeAtS
Sdiff_ktup = OBL_depth * ws_ktup * SshapeAtS

if (CVmix_kpp_params_in%lenhanced_diff) then
  if ((ktup.eq.kwup).or.(ktup.eq.kwup-1)) then
    call cvmix_kpp_compute_enhanced_diff(Mdiff_ktup,                      &
                                         Tdiff_ktup,                      &
                                         Sdiff_ktup,                      &
                                         Mdiff_out(ktup+1),               &
                                         Tdiff_out(ktup+1),               &
                                         Sdiff_out(ktup+1),               &
                                         OBL_Mdiff(ktup+1),               &
                                         OBL_Tdiff(ktup+1),               &
                                         OBL_Sdiff(ktup+1),               &
                                         Tnonlocal(ktup+1),               &
                                         Snonlocal(ktup+1),               &
                                         delta, lkteqkw=(ktup.eq.kwup))
  else
    print*, "ERROR: ktup should be either kwup or kwup-1!"
    print*, "ktup = ", ktup, " and kwup = ", kwup
    stop 1
  end if
else
  if ( kwup .eq. ktup ) then
    OBL_Mdiff(ktup+1) = old_Mdiff(ktup+1)
    OBL_Tdiff(ktup+1) = old_Tdiff(ktup+1)
    OBL_Sdiff(ktup+1) = old_Sdiff(ktup+1)
  end if
end if

! (5) Combine interior and boundary coefficients
```

```
    Mdiff_out(2:ktup+1) = OBL_Mdiff(2:ktup+1)
    Tdiff_out(2:ktup+1) = OBL_Tdiff(2:ktup+1)
    Sdiff_out(2:ktup+1) = OBL_Sdiff(2:ktup+1)

  end if    ! lStokesMOST
```

1.60 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.

USES:

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

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



    type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_out

    CVmix_kpp_params_out => CVmix_kpp_params_saved
    if (present(CVmix_kpp_params_user)) then
      CVmix_kpp_params_out => CVmix_kpp_params_user
    end if

    select case (trim(varname))
      case ('Ri_crit')
```

```
    CVmix_kpp_params_out%Ri_crit = val
  case ('minOBLdepth')
    CVmix_kpp_params_out%minOBLdepth = val
  case ('maxOBLdepth')
    CVmix_kpp_params_out%maxOBLdepth = val
  case ('minVtsqr')
    CVmix_kpp_params_out%minVtsqr = val
  case ('vonkarman')
    CVmix_kpp_params_out%vonkarman = val
  case ('Cstar')
    CVmix_kpp_params_out%Cstar = val
  case ('zeta_m')
    CVmix_kpp_params_out%zeta_m = val
  case ('zeta_s')
    CVmix_kpp_params_out%zeta_s = val
  case ('a_m')
    CVmix_kpp_params_out%a_m = val
  case ('a_s')
    CVmix_kpp_params_out%a_s = val
  case ('c_m')
    CVmix_kpp_params_out%c_m = val
  case ('c_s')
    CVmix_kpp_params_out%c_s = val
  case ('surf_layer_ext')
    CVmix_kpp_params_out%surf_layer_ext = val
  case ('Cv')
    CVmix_kpp_params_out%Cv = val
  case ('nonlocal_coeff')
    CVmix_kpp_params_out%nonlocal_coeff = val
  case ('c_CT')
    CVmix_kpp_params_out%c_CT = val
  case ('c_ST')
    CVmix_kpp_params_out%c_ST = val
  case ('c_LT')
    CVmix_kpp_params_out%c_LT = val
  case ('p_LT')
     CVmix_kpp_params_out%p_LT = val
  case ('RWHGK_ENTR_COEF')
     CVmix_kpp_params_out%rwhgk_entr_coef = val
  case ('RWHGK_ENTR_EXP')
     CVmix_kpp_params_out%rwhgk_entr_exp = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1
end select
```

1.61 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.

USES:

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

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
```

```
type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_out

CVmix_kpp_params_out => CVmix_kpp_params_saved
if (present(CVmix_kpp_params_user)) then
  CVmix_kpp_params_out => CVmix_kpp_params_user
end if

select case (trim(varname))
  case ('interp_type')
    CVmix_kpp_params_out%interp_type = val
  case ('interp_type2')
    CVmix_kpp_params_out%interp_type2 = val
  case ('MatchTechnique')
    CVmix_kpp_params_out%MatchTechnique = val
  case ('old_vals', 'handle_old_vals')
    CVmix_kpp_params_out%handle_old_vals = val
  case ('Langmuir_Mixing_Opt')
```

```
      CVmix_kpp_params_out%Langmuir_Mixing_opt = val
    case ('Langmuir_Entrainment_Opt')
      CVmix_kpp_params_out%Langmuir_Entrainment_opt = val
    case DEFAULT
      call cvmix_put_kpp(varname, real(val, cvmix_r8), CVmix_kpp_params_out)
  end select
```

1.62 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.

USES:

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

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



type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_out

CVmix_kpp_params_out => CVmix_kpp_params_saved
if (present(CVmix_kpp_params_user)) then
  CVmix_kpp_params_out => CVmix_kpp_params_user
end if

select case (trim(varname))
```

```
    case ('lscalar_Cv')
      CVmix_kpp_params_out%lscalar_Cv = val
    case ('lEkman')
      CVmix_kpp_params_out%lEkman = val
    case ('lStokesMOST')
      CVmix_kpp_params_out%lStokesMOST = val
    case ('lMonOb')
      CVmix_kpp_params_out%lMonOb = val
    case ('lnoDGat1')
      CVmix_kpp_params_out%lnoDGat1 = val
    case ('lenhanced_diff')
      CVmix_kpp_params_out%lenhanced_diff = val
    case ('l_LMD_ws')
      CVmix_kpp_params_out%l_LMD_ws = val
    case DEFAULT
      print*, "ERROR: ", trim(varname), " is not a boolean variable!"
      stop 1
  end select
```

1.63 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.

USES:

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

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


type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_get

CVmix_kpp_params_get => CVmix_kpp_params_saved
if (present(CVmix_kpp_params_user)) then
  CVmix_kpp_params_get => CVmix_kpp_params_user
end if

cvmix_get_kpp_real = cvmix_zero
select case (trim(varname))
  case ('Ri_crit')
    cvmix_get_kpp_real = CVmix_kpp_params_get%Ri_crit
  case ('vonkarman')
    cvmix_get_kpp_real = CVmix_kpp_params_get%vonkarman
  case ('Cstar')
    cvmix_get_kpp_real = CVmix_kpp_params_get%Cstar
  case ('zeta_m')
    cvmix_get_kpp_real = CVmix_kpp_params_get%zeta_m
  case ('zeta_s')
    cvmix_get_kpp_real = CVmix_kpp_params_get%zeta_s
  case ('a_m')
    cvmix_get_kpp_real = CVmix_kpp_params_get%a_m
  case ('a_s')
    cvmix_get_kpp_real = CVmix_kpp_params_get%a_s
  case ('c_m')
    cvmix_get_kpp_real = CVmix_kpp_params_get%c_m
  case ('c_s')
    cvmix_get_kpp_real = CVmix_kpp_params_get%c_s
  case ('surf_layer_ext')
    cvmix_get_kpp_real = CVmix_kpp_params_get%surf_layer_ext
  case ('Cv')
    cvmix_get_kpp_real = CVmix_kpp_params_get%Cv
  case ('c_CT')
    cvmix_get_kpp_real = CVmix_kpp_params_get%c_CT
  case ('c_ST')
    cvmix_get_kpp_real = CVmix_kpp_params_get%c_ST
  case ('c_LT')
    cvmix_get_kpp_real = CVmix_kpp_params_get%c_LT
  case ('p_LT')
    cvmix_get_kpp_real = CVmix_kpp_params_get%p_LT
  case ('RWHGK_ENTR_COEF')
     cvmix_get_kpp_real = CVmix_kpp_params_get%RWHGK_ENTR_COEF
  case ('RWHGK_ENTR_EXP')
     cvmix_get_kpp_real = CVmix_kpp_params_get%RWHGK_ENTR_EXP
  case DEFAULT
```

```
        print*, "ERROR: ", trim(varname), " not a valid choice!"
        stop 1
    end select
```

1.64 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) for a given column. Ri_bulk(h) = Ricr; h ¡ -zBottom, (stable+lMonOb) 0 ¡ h ¡ vonKaraman Lstar

USES:

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

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


    ! Local variables
```

```
real(kind=cvmix_r8), dimension(:), pointer :: depth
real(kind=cvmix_r8), dimension(4)          :: coeffs
real(kind=cvmix_r8) :: Ekman, MoninObukhov, OBL_Limit
integer             :: nlev, k, kRi
logical             :: lstable

type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_in

CVmix_kpp_params_in => CVmix_kpp_params_saved
if (present(CVmix_kpp_params_user)) then
  CVmix_kpp_params_in => CVmix_kpp_params_user
end if

! Error checks
! (1) if using Ekman length, need to pass surf_fric and Coriolis
if ((.not.(present(surf_fric).and.present(Coriolis))).and.                &
    CVmix_kpp_params_in%lEkman) then
  print*, "ERROR: must pass surf_fric and Coriolis if you want to ",      &
           "compute Ekman length"
  stop 1
end if

! (2) if using Monin-Obukhov length, need to pass surf_fric and surf_buoy
if ((.not.(present(surf_fric).and.present(surf_buoy))).and.               &
    CVmix_kpp_params_in%lMonOb) then
  print*, "ERROR: must pass surf_fric and surf_buoy if you want to ",     &
           "compute Monin-Obukhov length"
  stop 1
end if

! (3) zt_cntr must be length nlev and zw_iface must be nlev+1
nlev = size(zt_cntr)
if (size(zw_iface).ne.nlev+1) then
  print*, "ERROR: zt_cntr must have exactly one less element than zw_iface!"
  print*, "size(zt_cntr) = ", nlev, ", size(zw_iface) = ", size(zw_iface)
  stop 1
end if

! (4) Ri_bulk needs to be either the size of zw_iface or zt_cntr
if (size(Ri_bulk).eq.nlev) then
  depth => zt_cntr
else if (size(Ri_bulk).eq.nlev+1) then
  depth => zw_iface
else
  print*, "ERROR: Ri_bulk must have size nlev or nlev+1!"
  print*, "nlev = ", nlev, ", size(Ri_bulk) = ", size(Ri_bulk)
  stop 1
end if
```

```
  ! if lEkman = .true., OBL_depth must be between the surface and the Ekman
  ! depth. Similarly, if lMonOb = .true., OBL_depth must be between the
  ! surface and the Monin-Obukhov depth

if ( CVmix_kpp_params_in%lStokesMOST ) then
  ! OBL_depth must be at or above 1) zbottom, the effective ocean bottom,
  if ( present(zBottom) )  then
    OBL_limit = abs(zBottom)
  else
    OBL_limit = abs(zt_cntr(nlev))
  end if

  ! (1) Find k such that Ri_bulk at level k+1 > Ri_crit
  do k=0,size(Ri_bulk)-1
    kRi = k+1
    if (Ri_bulk(k+1).gt.CVmix_kpp_params_in%ri_crit) &
    exit
  end do

  if (k.eq.size(Ri_bulk)) then
    OBL_depth = OBL_limit
  elseif (k.eq.0) then
    OBL_depth = abs(zt_cntr(1))
  else

  ! (2) Interpolation
    if (k.eq.1) then
      call cvmix_math_poly_interp(coeffs, CVmix_kpp_params_in%interp_type,  &
                            depth(k:k+1), Ri_bulk(k:k+1))
    else
      call cvmix_math_poly_interp(coeffs, CVmix_kpp_params_in%interp_type,  &
                            depth(k:k+1), Ri_bulk(k:k+1), depth(k-1),      &
                            Ri_bulk(k-1))
    end if
    coeffs(1) = coeffs(1)-CVmix_kpp_params_in%ri_crit

    OBL_depth = -cvmix_math_cubic_root_find(coeffs, 0.5_cvmix_r8 *          &
                                                 (depth(k)+depth(k+1)))

  ! (3) OBL_depth needs to be at or below the center of the top level
    ! Note: OBL_depth can only be computed to be above this point if k=1,
    if (k.eq.1)  OBL_depth = max(OBL_depth, -zt_cntr(1))
  end if                  !  -zt_cntr(1)  <  OBL_depth < OBL_limit

    ! (5) the modified MoninObukhov limit (= vonk*Lstar) if stable and lMonOb=True
  if (CVmix_kpp_params_in%lMonOb ) then
    if ( present(Xi) .and. present(surf_buoy) ) then
```

```
      MoninObukhov = OBL_limit
      do k = 0, kRi-1
        if (surf_buoy(k+1) .gt. cvmix_zero) MoninObukhov =      &
              surf_fric**3 / (surf_buoy(k+1) * (cvmix_one-Xi(k+1)))
        if ( MoninObukhov .lt. abs(zt_cntr(k+1)) ) &
        exit
      end do
      if (k.eq.0) then
        OBL_limit  = abs(zt_cntr(1))
      elseif (k.lt.kRi) then
         OBL_limit = min( OBL_limit, abs(zw_iface(k)) )
      end if

    else
      print*, "ERROR: Stokes_XI and surf_buoy both must be present if lMonOb=true with Sto
      stop 1
    end if
  end if     ! lMonOb

    ! (4) OBL_depth must be at or above OBL_limit -zt_cntr(1)  <  OBL_depth < OBL_limit
  OBL_depth  = min(OBL_depth, OBL_limit )
  kOBL_depth = cvmix_kpp_compute_kOBL_depth(zw_iface, zt_cntr, OBL_depth)

else ! not Stokes_MOST

  OBL_limit  = abs(zt_cntr(nlev))

  ! Since depth gets more negative as you go deeper, that translates into
  ! OBL_depth = max(abs(computed depth), abs(Ekman depth), abs(M-O depth))
  if (CVmix_kpp_params_in%lEkman) then
    ! Column is stable if surf_buoy > 0
    lstable = (surf_buoy(nlev).gt.cvmix_zero)

    if (Coriolis.ne.cvmix_zero .and. lstable) then
      Ekman = 0.7_cvmix_r8*surf_fric/abs(Coriolis)
    else
      ! Rather than divide by zero (or if column is unstable), set Ekman depth to ocean bo
      Ekman = abs(zt_cntr(nlev))
    end if
    OBL_limit = min(OBL_limit, Ekman)
  end if

  if (CVmix_kpp_params_in%lMonOb) then
    ! Column is stable if surf_buoy > 0
    lstable = (surf_buoy(nlev).gt.cvmix_zero)

    if (lstable) then
      MoninObukhov = surf_fric**3/(surf_buoy(nlev)*CVmix_kpp_params_in%vonkarman)
```

```
  else
    MoninObukhov = abs(zt_cntr(nlev))
  end if
  OBL_limit = min(OBL_limit, MoninObukhov)
end if

! Interpolation Step
! (1) Find k such that Ri_bulk at level k+1 > Ri_crit
do k=0,size(Ri_bulk)-1
  if (Ri_bulk(k+1).gt.CVmix_kpp_params_in%ri_crit) &
    exit
end do

if (k.eq.size(Ri_bulk)) then
  OBL_depth = abs(OBL_limit)
elseif (k.eq.0) then
  OBL_depth = abs(zt_cntr(1))
else
  if (k.eq.1) then
    call cvmix_math_poly_interp(coeffs, CVmix_kpp_params_in%interp_type,  &
                            depth(k:k+1), Ri_bulk(k:k+1))
  else
    call cvmix_math_poly_interp(coeffs, CVmix_kpp_params_in%interp_type,  &
                            depth(k:k+1), Ri_bulk(k:k+1), depth(k-1),     &
                            Ri_bulk(k-1))
  end if
  coeffs(1) = coeffs(1)-CVmix_kpp_params_in%ri_crit

  OBL_depth = -cvmix_math_cubic_root_find(coeffs, 0.5_cvmix_r8 *          &
                                                (depth(k)+depth(k+1)))

  ! OBL_depth needs to be at or below the center of the top level
  ! Note: OBL_depth can only be computed to be above this point if k=1,
  !       depth => zw_iface instead of zt_cntr, and the interpolation
  !       results in Ri_bulk = Ri_crit at a depth above the center of the
  !       top level.
  if (k.eq.1) then
    OBL_depth = max(OBL_depth, -zt_cntr(1))
  end if

  ! OBL_depth needs to be at or above OBL_limit
  ! Note: maybe there are times when we don't need to do the interpolation
  !       because we know OBL_depth will equal OBL_limit?
  OBL_depth = min(OBL_depth, OBL_limit)
end if

OBL_depth = max(OBL_depth, CVmix_kpp_params_in%minOBLdepth)
if (CVmix_kpp_params_in%maxOBLdepth.gt.cvmix_zero)                        &
```

```
      OBL_depth = min(OBL_depth, CVmix_kpp_params_in%maxOBLdepth)
    kOBL_depth = cvmix_kpp_compute_kOBL_depth(zw_iface, zt_cntr, OBL_depth)

  end if    ! lStokesMOST
```

1.65 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)`

USES:

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

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
```

```
! Local variables
integer :: kw, nlev

nlev = size(zt_cntr)
if (size(zw_iface).ne.nlev+1) then
  print*, "ERROR: there should be one more interface z coordinate than ", &
          "cell center coordinate!"
  stop 1
```

```
end if

! Initial value = nlev + 0.75 => OBL_depth at center of bottom cell
cvmix_kpp_compute_kOBL_depth = real(nlev,cvmix_r8)+0.75_cvmix_r8
do kw=1,nlev
  if (OBL_depth.lt.abs(zw_iface(kw+1))) then
    if (OBL_depth.lt.abs(zt_cntr(kw))) then
      cvmix_kpp_compute_kOBL_depth = real(kw, cvmix_r8)+0.25_cvmix_r8
    else
      cvmix_kpp_compute_kOBL_depth = real(kw, cvmix_r8)+0.75_cvmix_r8
    end if
    exit
  end if
end do
```

1.66 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
! OBL_depth).
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
```

```
! Local variables

! enh_diff and enh_visc are the enhanced diffusivity and viscosity values
! at the interface nearest OBL_depth
real(cvmix_r8) :: enh_Mdiff, enh_Tdiff, enh_Sdiff
! Need to store original OBL_Tdiff and OBL_Sdiff for updating nonlocal
real(cvmix_r8) :: old_Tdiff, old_Sdiff

real(cvmix_r8) :: omd ! one minus delta

omd = cvmix_one - delta
old_Tdiff = OBL_Tdiff
old_Sdiff = OBL_Sdiff

if (lkteqkw) then
  ! => ktup = kwup
  ! Interface kw = ktup+1 is outside the OBL

  ! (a) compute enhanced diffs: get diffusivity values at kw = ktup+1
  !     from diff and visc rather than OBL_diff and OBL_visc
  enh_Mdiff = (omd**2)*Mdiff_ktup + (delta**2)*Mdiff
  enh_Tdiff = (omd**2)*Tdiff_ktup + (delta**2)*Tdiff
  enh_Sdiff = (omd**2)*Sdiff_ktup + (delta**2)*Sdiff

  ! (b) modify diffusivity values at kw = ktup+1 (again in diff and visc)
  Mdiff = omd*Mdiff + delta*enh_Mdiff
  Tdiff = omd*Tdiff + delta*enh_Tdiff
  Sdiff = omd*Sdiff + delta*enh_Sdiff

  ! (c) Update OBL_[MTS]diff
  OBL_Mdiff = Mdiff
  OBL_Tdiff = Tdiff
```

```
      OBL_Sdiff = Sdiff
    else
      ! => ktup = kwup - 1
      ! Interface kw = ktup+1 is in the OBL

      ! (a) compute enhanced diffs: get diffusivity values at kw = ktup+1
      !     from OBL_diff and OBL_visc rather than diff and visc
      enh_Mdiff = (omd**2)*Mdiff_ktup + (delta**2)*OBL_Mdiff
      enh_Tdiff = (omd**2)*Tdiff_ktup + (delta**2)*OBL_Tdiff
      enh_Sdiff = (omd**2)*Sdiff_ktup + (delta**2)*OBL_Sdiff

      ! (b) modify diffusivity values at kw = ktup+1 (again in OBL_diff and
      !     OBL_visc)
      OBL_Mdiff = omd*Mdiff + delta*enh_Mdiff
      OBL_Tdiff = omd*Tdiff + delta*enh_Tdiff
      OBL_Sdiff = omd*Sdiff + delta*enh_Sdiff

      ! (c) update nonlocal term
      if (old_Tdiff.ne.cvmix_zero) then
        Tnonlocal = Tnonlocal*OBL_Tdiff/old_Tdiff
      else
        Tnonlocal = cvmix_zero
      end if
      if (old_Sdiff.ne.cvmix_zero) then
        Snonlocal = Snonlocal*OBL_Sdiff/old_Sdiff
      else
        Snonlocal = cvmix_zero
      end if
    end if

! EOC

  end subroutine cvmix_kpp_compute_enhanced_diff
```

1.67 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 (OBL) for a given column.

USES:

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

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
```

```
    ! Local variables
!   real(cvmix_r8) :: lcl_obl_depth, lcl_kobl_depth
    real(cvmix_r8), dimension(1) :: lcl_Xi, lcl_sfcBuoy
    lcl_Xi(1) = CVmix_vars%StokesMostXi
    lcl_sfcBuoy(1) = CVmix_vars%SurfaceBuoyancyForcing

    call cvmix_kpp_compute_OBL_depth(CVmix_vars%BulkRichardson_cntr,          &
                                     CVmix_vars%zw_iface,                     &
                                     CVmix_vars%BoundaryLayerDepth,           &
                                     CVmix_vars%kOBL_depth,                   &
                                     CVmix_vars%zt_cntr,                      &
                                     CVmix_vars%SurfaceFriction,              &
                                     lcl_sfcBuoy,                             &
                                     CVmix_vars%Coriolis,                     &
                                     lcl_Xi   ,                               &
                                     CVmix_vars%zBottomOceanNumerics,         &
                                     CVmix_kpp_params_user)
```

1.68 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`.

USES:

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

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
```

```
! Local variables
! * unresolved_shear_cntr_sqr is the square of the unresolved level-center
!   velocity shear (Vt^2(d) in LMD94, units: m^2/s^2)
real(cvmix_r8), dimension(size(zt_cntr)) :: unresolved_shear_cntr_sqr
integer        :: kt
real(cvmix_r8) :: scaling, num, denom
type(cvmix_kpp_params_type),  pointer :: CVmix_kpp_params_in

CVmix_kpp_params_in => CVmix_kpp_params_saved
if (present(CVmix_kpp_params_user)) then
  CVmix_kpp_params_in => CVmix_kpp_params_user
end if

! Make sure all arguments are same size
if (any((/size(delta_buoy_cntr), size(delta_Vsqr_cntr)/).ne.              &
    size(zt_cntr))) then
  print*, "ERROR: delta_buoy, delta_vel_sqr, and zt_cntr must all be the",&
          "same size!"
  stop 1
end if
if (present(Vt_sqr_cntr)) then
  if (size(Vt_sqr_cntr).eq.size(zt_cntr)) then
    unresolved_shear_cntr_sqr = Vt_sqr_cntr
  else
    print*, "ERROR: Vt_sqr_cntr must be the same size as zt_cntr!"
    stop 1
  end if
else
  if (.not.present(ws_cntr)) then
    print*, "ERROR: you must pass in either Vt_sqr_cntr or ws_cntr!"
    stop 1
  end if
  unresolved_shear_cntr_sqr = cvmix_kpp_compute_unresolved_shear(         &
                                zt_cntr, ws_cntr, N_iface, Nsqr_iface,    &
                                EFactor, LaSL, bfsfc, ustar,              &
                                CVmix_kpp_params_user)
end if

! scaling because we want (d-dr) = (d-0.5*eps*d) = (1-0.5*eps)*d
scaling = cvmix_one - 0.5_cvmix_r8*CVmix_kpp_params_in%surf_layer_ext
do kt=1,size(zt_cntr)
  ! Negative sign because we use positive-up for height
  num   = -scaling*zt_cntr(kt)*delta_buoy_cntr(kt)
  denom = delta_Vsqr_cntr(kt) + unresolved_shear_cntr_sqr(kt)
  if (denom.ne.cvmix_zero) then
    cvmix_kpp_compute_bulk_Richardson(kt) = num/denom
  else
```

```
    ! Need a better fudge factor?
    cvmix_kpp_compute_bulk_Richardson(kt) = num*1e10_cvmix_r8
  end if
end do
```

1.69 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.

USES:

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

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
```

```
! Local variables
real(cvmix_r8), dimension(1) :: sigma, lcl_wm, lcl_ws
```

```
real(cvmix_r8)               :: lcl_XI
logical :: compute_wm, compute_ws

if( present( xi ) ) then
  lcl_XI = xi
else
  lcl_XI = cvmix_zero           ! NO WAVES
end if

compute_wm = present(w_m)
compute_ws = present(w_s)
sigma(1) = sigma_coord
if (compute_wm) &
  lcl_wm(1) = w_m
if (compute_ws) &
  lcl_ws(1) = w_s
if (compute_wm.and.compute_ws) then
  call cvmix_kpp_compute_turbulent_scales(sigma, OBL_depth,               &
                                          surf_buoy_force, surf_fric_vel, &
                                          xi  = lcl_XI,                   &
                                          w_m = lcl_wm, w_s = lcl_ws,     &
                                CVmix_kpp_params_user=CVmix_kpp_params_user)
else
  if (compute_wm) &
    call cvmix_kpp_compute_turbulent_scales(sigma, OBL_depth,             &
                                            surf_buoy_force,surf_fric_vel,&
                                            xi  = lcl_XI,                 &
                                            w_m = lcl_wm,                 &
                                CVmix_kpp_params_user=CVmix_kpp_params_user)
  if (compute_ws) &
    call cvmix_kpp_compute_turbulent_scales(sigma, OBL_depth,             &
                                            surf_buoy_force,surf_fric_vel,&
                                            xi  = lcl_XI,                 &
                                            w_s = lcl_ws,                 &
                                CVmix_kpp_params_user=CVmix_kpp_params_user)
end if

if (compute_wm) &
  w_m = lcl_wm(1)
if (compute_ws) &
  w_s = lcl_ws(1)
```

1.70 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.

USES:

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

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
```

```
! Local variables
integer :: n_sigma, kw
logical :: compute_wm, compute_ws, l_LMD_ws
real(cvmix_r8), dimension(size(sigma_coord)) :: zeta, sigma_loc
real(cvmix_r8) :: vonkar, surf_layer_ext
real(cvmix_r8) :: chi_m, chi_s, L_StokesL
type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_in
```

```
  n_sigma = size(sigma_coord)

  CVmix_kpp_params_in => CVmix_kpp_params_saved
  if (present(CVmix_kpp_params_user)) then
    CVmix_kpp_params_in => CVmix_kpp_params_user
  end if

  compute_wm = present(w_m)
  compute_ws = present(w_s)

  l_LMD_ws       = CVmix_kpp_params_in%l_LMD_ws
  vonkar         = CVmix_kpp_params_in%vonkarman
  surf_layer_ext = CVmix_kpp_params_in%surf_layer_ext

if ( CVmix_kpp_params_in%lStokesMOST ) then
  if (present(xi)) then
    L_StokesL = cvmix_one - xi

    if (surf_fric_vel.gt.cvmix_zero) then
      sigma_loc(:) = min(cvmix_one , sigma_coord(:))
      zeta(:) = sigma_loc(:) * OBL_depth * surf_buoy_force * vonkar / &
            (surf_fric_vel**3)
      if (compute_wm) then
        chi_m = compute_Stokes_chi( xi , lchi_m=.true. )
        do kw=1,n_sigma
          w_m(kw) = compute_phi_inv(zeta(kw),CVmix_kpp_params_in, L_StokesL, lphi_m=.true
                    vonkar*surf_fric_vel / chi_m
        end do
      end if
      if (compute_ws) then
        chi_s = compute_Stokes_chi( xi , lchi_s=.true. )
        do kw=1,n_sigma
          w_s(kw) = compute_phi_inv(zeta(kw),CVmix_kpp_params_in, L_StokesL, lphi_s=.true
                    vonkar*surf_fric_vel / chi_s
        end do
      end if

    else ! surf_fric_vel = 0
      if (compute_wm) then
        if (surf_buoy_force.ge.cvmix_zero) then  ! STABLE
          w_m = cvmix_zero
        else                                     ! convective limit
          chi_m = compute_Stokes_chi( xi , lchi_m=.true. )
          L_StokesL = cvmix_one - xi
          do kw=1,n_sigma
            w_m(kw) = -surf_buoy_force * real(14,cvmix_r8) * sigma_coord(kw) * &
                       OBL_depth * vonkar * L_StokesL
            w_m(kw) = vonkar*(w_m(kw)**(cvmix_one/real(3,cvmix_r8))) / chi_m
```

```
          end do
        end if
      end if   ! compute_wm

      if (compute_ws) then
        if (surf_buoy_force.ge.cvmix_zero) then  ! STABLE
          w_s = cvmix_zero
        else
          chi_s = compute_Stokes_chi( xi , lchi_s=.true. )
          L_StokesL = cvmix_one - xi
          do kw=1,n_sigma                    ! convective limit
            w_s(kw) = -surf_buoy_force * real(25,cvmix_r8) * sigma_coord(kw) * &
                      OBL_depth * vonkar * L_StokesL
            w_s(kw) = vonkar*(w_s(kw)**(cvmix_one/real(3,cvmix_r8))) / chi_s
          end do
        end if  ! surf_buoy_force >= 0
      end if    ! compute_ws

    end if ! surf_fric_vel != 0

  else
    print*, "ERROR: Similarity xi must be present in 1d_sigma to use Stokes_MOST package!"
    stop 1
  end if    ! lStokesMOST and xi not present

else    ! not lStokesMOST

  if (surf_fric_vel.ne.cvmix_zero) then
    if ((surf_buoy_force.ge.cvmix_zero) .and. l_LMD_ws) then
      sigma_loc(:) = sigma_coord(:)
    else
      sigma_loc(:) = min(surf_layer_ext, sigma_coord(:))
    end if
    ! compute scales at sigma if sigma < surf_layer_ext, otherwise compute
    ! at surf_layer_ext
    zeta(:) = sigma_loc(:) * OBL_depth * surf_buoy_force * vonkar /         &
              (surf_fric_vel**3)

    if (compute_wm) then
      w_m(1) = compute_phi_inv(zeta(1), CVmix_kpp_params_in, lphi_m=.true.)*&
               vonkar*surf_fric_vel
      do kw=2,n_sigma
        if (zeta(kw).eq.zeta(kw-1)) then
          w_m(kw) = w_m(kw-1)
        else
          w_m(kw) = vonkar*surf_fric_vel*compute_phi_inv(zeta(kw),          &
                                       CVmix_kpp_params_in, lphi_m=.true.)
        end if
```

```
    end do
  end if
  if (compute_ws) then
    w_s(1) = compute_phi_inv(zeta(1), CVmix_kpp_params_in, lphi_s=.true.)*&
             vonkar*surf_fric_vel
    do kw=2,n_sigma
      if (zeta(kw).eq.zeta(kw-1)) then
        w_s(kw) = w_s(kw-1)
      else
        w_s(kw) = vonkar*surf_fric_vel*compute_phi_inv(zeta(kw),          &
                                       CVmix_kpp_params_in, lphi_s=.true.)
      end if
    end do
  end if
else ! surf_fric_vel = 0
  if (compute_wm) then
    if (surf_buoy_force.ge.cvmix_zero) then
      ! Stable regime with surf_fric_vel = 0 => w_m = 0
      w_m = cvmix_zero
    else
      ! Unstable forcing, Eqs. (13) and (B1c) reduce to following
      do kw=1,n_sigma
        ! Compute (u*/phi_m)^3 [this is where the zeros in numerator and
        !                       denominator cancel when u* = 0]
        w_m(kw) = -CVmix_kpp_params_in%c_m *                              &
                  min(surf_layer_ext, sigma_coord(kw)) * OBL_depth *      &
                  vonkar * surf_buoy_force
        ! w_m = vonkar * u* / phi_m
        !     = vonkar * ((u*/phi_m)^3)^1/3
        w_m(kw) = vonkar*(w_m(kw)**(cvmix_one/real(3,cvmix_r8)))
      end do
    end if ! surf_buoy_force >= 0
  end if ! compute_wm

  if (compute_ws) then
    if (surf_buoy_force.ge.cvmix_zero) then
      ! Stable regime with surf_fric_vel = 0 => w_s = 0
      w_s = cvmix_zero
    else
      ! Unstable forcing, Eqs. (13) and (B1e) reduce to following
      do kw=1,n_sigma
        ! Compute (u*/phi_s)^3 [this is where the zeros in numerator and
        !                       denominator cancel when u* = 0]
        w_s(kw) = -CVmix_kpp_params_in%c_s *                              &
                  min(surf_layer_ext, sigma_coord(kw)) * OBL_depth *      &
                  vonkar * surf_buoy_force
        ! w_s = vonkar * u* / phi_s
        !     = vonkar * ((u*/phi_s)^3)^1/3
```

```
          w_s(kw) = vonkar*(w_s(kw)**(cvmix_one/real(3,cvmix_r8)))
        end do
      end if ! surf_buoy_force >= 0
    end if ! compute_ws
  end if ! surf_fric_vel != 0
end if    ! lStokesMOST
```

1.71 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)

USES:

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

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
```

```
  ! Local variables
  integer :: n_sigma, kw
  logical :: compute_wm, compute_ws, l_LMD_ws
  real(cvmix_r8), dimension(size(surf_buoy_force)) :: zeta, sigma_loc
  real(cvmix_r8) :: vonkar, surf_layer_ext
  real(cvmix_r8)  :: chi_m,chi_s,L_StokesL
  type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_in

  n_sigma = size(surf_buoy_force)

  CVmix_kpp_params_in => CVmix_kpp_params_saved
  if (present(CVmix_kpp_params_user)) then
    CVmix_kpp_params_in => CVmix_kpp_params_user
  end if

  compute_wm = present(w_m)
  compute_ws = present(w_s)

  l_LMD_ws       = CVmix_kpp_params_in%l_LMD_ws
  vonkar         = CVmix_kpp_params_in%vonkarman
  surf_layer_ext = CVmix_kpp_params_in%surf_layer_ext

if ( CVmix_kpp_params_in%lStokesMOST ) then
  if (present(xi)) then

    if (surf_fric_vel.ne.cvmix_zero) then
      zeta(:) = sigma_coord*OBL_depth(:)*surf_buoy_force(:)*vonkar / &
                  (surf_fric_vel**3)
      if (compute_wm) then
        do kw = 1,n_sigma
          chi_m = compute_Stokes_chi( xi(kw) , lchi_m=.true. )
          L_StokesL = cvmix_one - xi(kw)             !wgl - xi
          w_m(kw)=compute_phi_inv(zeta(kw),CVmix_kpp_params_in,L_StokesL,lphi_m=.true.) *
                    vonkar*surf_fric_vel / chi_m
        end do
      end if

      if (compute_ws) then
        do kw = 1,n_sigma
          chi_s = compute_Stokes_chi( xi(kw) , lchi_s=.true. )
          w_s(kw)=compute_phi_inv(zeta(kw),CVmix_kpp_params_in,L_StokesL,lphi_s=.true.) *
                    vonkar*surf_fric_vel  / chi_s
        end do
      end if

    else ! surf_fric_vel = 0
      if (compute_wm) then
```

```
        do kw = 1,n_sigma
          if (surf_buoy_force(kw).ge.cvmix_zero) then  ! STABLE
            w_m(kw) = cvmix_zero
          else                              ! convective limit
            L_StokesL = cvmix_one - xi(kw)
            w_m(kw) = -surf_buoy_force(kw) * real(14,cvmix_r8) * sigma_coord * &
                      OBL_depth(kw) * vonkar * L_StokesL
            w_m(kw) =vonkar*(w_m(kw)**(cvmix_one/real(3,cvmix_r8))) / compute_Stokes_chi(x
          end if
        end do
      end if
      if (compute_ws) then
        do kw = 1,n_sigma
          if (surf_buoy_force(kw).ge.cvmix_zero) then  ! STABLE
            w_s(kw) = cvmix_zero
          else
            L_StokesL = cvmix_one - xi(kw)
            w_s(kw) = -surf_buoy_force(kw) * real(25,cvmix_r8) * sigma_coord * &
                      OBL_depth(kw) * vonkar * L_StokesL
            w_s(kw) =vonkar*(w_s(kw)**(cvmix_one/real(3,cvmix_r8))) / compute_Stokes_chi(x
          end if  ! surf_buoy_force >= 0
        end do
      end if    ! compute_ws
    end if      ! surf_fric_vel = 0
  else
    print*, "ERROR: Similarity xi must be present in 1d_OBL to use Stokes_MOST package!"
    stop 1
  end if    ! lStokesMOST but xi not present

else    ! not lStokesMOST

  if (surf_fric_vel.ne.cvmix_zero) then
    sigma_loc = min(surf_layer_ext, sigma_coord)
    if (l_LMD_ws) then
      where (surf_buoy_force.ge.cvmix_zero)
        sigma_loc = sigma_coord
      end where
    end if
    zeta(:) = sigma_loc(:) * OBL_depth(:) * surf_buoy_force(:) * vonkar /   &
              (surf_fric_vel**3)

    if (compute_wm) then
      w_m(1) = compute_phi_inv(zeta(1), CVmix_kpp_params_in, lphi_m=.true.)*&
               vonkar*surf_fric_vel
      do kw=2,n_sigma
        if (zeta(kw).eq.zeta(kw-1)) then
          w_m(kw) = w_m(kw-1)
        else
```

```
        w_m(kw) = compute_phi_inv(zeta(kw), CVmix_kpp_params_in, lphi_m=.true.)*&
                  vonkar*surf_fric_vel
      end if
    end do
  end if

  if (compute_ws) then
    w_s(1) = compute_phi_inv(zeta(1), CVmix_kpp_params_in, lphi_s=.true.)*&
             vonkar*surf_fric_vel
    do kw=2,n_sigma
      if (zeta(kw).eq.zeta(kw-1)) then
        w_s(kw) = w_s(kw-1)
      else
        w_s(kw) = compute_phi_inv(zeta(kw), CVmix_kpp_params_in, lphi_s=.true.)*&
                  vonkar*surf_fric_vel
      end if
    end do
  end if

else ! surf_fric_vel = 0
  if (compute_wm) then
    ! Unstable forcing, Eqs. (13) and (B1c) reduce to following
    do kw=1,n_sigma
      if(surf_buoy_force(kw) .ge. cvmix_zero) then
        w_m(kw) = cvmix_zero
      else
        ! Compute (u*/phi_m)^3 [this is where the zeros in numerator and
        !                       denominator cancel when u* = 0]
        w_m(kw) = -CVmix_kpp_params_in%c_m *                                &
                  min(surf_layer_ext, sigma_coord) * OBL_depth(kw) *        &
                  vonkar * surf_buoy_force(kw)
        ! w_m = vonkar * u* / phi_m
        !     = vonkar * ((u*/phi_m)^3)^1/3
        w_m(kw) = vonkar*(w_m(kw)**(cvmix_one/real(3,cvmix_r8)))
    end if
    end do
  end if ! compute_wm

  if (compute_ws) then
      ! Unstable forcing, Eqs. (13) and (B1e) reduce to following
    do kw=1,n_sigma
      if (surf_buoy_force(kw) .ge. cvmix_zero) then
        ! Stable regime with surf_fric_vel = 0 => w_s = 0
        w_s(kw) = cvmix_zero
      else
        ! Unstable forcing, Eqs. (13) and (B1e) reduce to following
        ! Compute (u*/phi_s)^3 [this is where the zeros in numerator and
        !                       denominator cancel when u* = 0]
```

```
          w_s(kw) = -CVmix_kpp_params_in%c_s *                                &
                    min(surf_layer_ext, sigma_coord) * OBL_depth(kw) *        &
                    vonkar * surf_buoy_force(kw)
          ! w_s = vonkar * u* / phi_s
          !     = vonkar * ((u*/phi_s)^3)^1/3
          w_s(kw) = vonkar*(w_s(kw)**(cvmix_one/real(3,cvmix_r8)))
        end if ! surf_buoy_force >= 0
      end do
    end if ! compute_ws
  end if ! surf_fric_vel != 0
end if    ! lStokesMOST
```

1.72 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).

USES:

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

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
```

```
! Local variables
integer :: kt, nlev
real(cvmix_r8) :: Cv, Vtc
logical :: lwstar          ! use wstar rather than w_s
real(cvmix_r8) :: wstar    ! convective velocity scale
real(cvmix_r8) :: ws_wstar ! ratio in limit of pure convection
! N_cntr: buoyancy frequency at cell centers, derived from either N_iface
!        or Nsqr_iface (units: 1/s)
real(cvmix_r8), dimension(size(zt_cntr)) :: N_cntr
! c_CT, c_ST, c_LT, p_LT: parameters of Langmuir-enhanced entrainment
!                         in Li and Fox-Kemper, 2017, JPO
real(cvmix_r8) :: c_CT, c_ST, c_LT, p_LT
! RWHGK_ENTR_COEF, RWHGK_ENTR_EXP: parameters of Langmuir-enhanced
!                         entrainment in Reichl et al., 2016, JPO
real(cvmix_r8) :: RWHGK_ENTR_COEF, RWHGK_ENTR_EXP
! Vt2_Enhancement: enhancement factor for unresolved shear
real(cvmix_r8) :: Vt2_Enhancement
type(cvmix_kpp_params_type), pointer :: CVmix_kpp_params_in

nlev = size(zt_cntr)
if (size(ws_cntr).ne.nlev) then
  print*, "ERROR: zt_cntr and ws_cntr must be same size"
  stop 1
end if

if (present(N_iface).and.present(Nsqr_iface)) then
  print*, "ERROR: you must provide N_iface OR Nsqr_iface, can not send",  &
          "both!"
  stop 1
```

```
  end if

  CVmix_kpp_params_in => CVmix_kpp_params_saved
  if (present(CVmix_kpp_params_user)) then
    CVmix_kpp_params_in => CVmix_kpp_params_user
  end if

  if (present(N_iface)) then
    if (size(N_iface).ne.(nlev+1)) then
      print*, "ERROR: N_iface must have one more element than zt_cntr"
      stop 1
    end if
    do kt=1,nlev
      N_cntr(kt) = N_iface(kt+1)
    end do
  else
    if (present(Nsqr_iface)) then
      if (size(Nsqr_iface).ne.(nlev+1)) then
        print*, "ERROR: Nsqr_iface must have one more element than zt_cntr"
        stop 1
      end if
      do kt=1,nlev
        N_cntr(kt)=sqrt(max(Nsqr_iface(kt+1),cvmix_zero))
      end do
    else
      print*, "ERROR: you must provide N_iface OR Nsqr_iface"
      stop 1
    end if
  end if

if ( CVmix_kpp_params_in%lStokesMOST ) then
  if (present(N_iface)) then
    lwstar   = .false.   !  .true.

    ws_wstar = CVmix_kpp_params_in%vonkarman * real(25,cvmix_r8) ! *  &
    ws_wstar = CVmix_kpp_params_in%vonkarman * ws_wstar**(cvmix_one/real(3,cvmix_r8))


    Vtc = sqrt(0.2_cvmix_r8 *3.8409_cvmix_r8 /ws_wstar) /CVmix_kpp_params_in%Ri_crit

    Cv = 1.4_cvmix_r8

    do kt=1,nlev
      if (lwstar ) then
        wstar = (MAX(0.0 , zt_cntr(kt) * bfsfc(kt) ))**(cvmix_one/real(3,cvmix_r8))
        cvmix_kpp_compute_unresolved_shear(kt) = &
             -zt_cntr(kt) * N_iface(kt) * Cv * Vtc * wstar
      else
```

```
        cvmix_kpp_compute_unresolved_shear(kt) = &
             -zt_cntr(kt) * N_iface(kt) * Cv * Vtc * ws_cntr(kt) / ws_wstar
      end if

      if (cvmix_kpp_compute_unresolved_shear(kt).lt.                        &
            CVmix_kpp_params_in%minVtsqr) then
          cvmix_kpp_compute_unresolved_shear(kt) = CVmix_kpp_params_in%minVtsqr
      end if
    enddo
  else
   print*, "ERROR: StokesMOST package requires N_iface in cvmix_kpp_compute_unresolved_she
      stop 1
  end if
else    ! not lStokesMOST

  ! options for Langmuir enhanced entrainment
  select case (CVmix_kpp_params_in%Langmuir_Entrainment_Opt)

    case (LANGMUIR_ENTRAINMENT_LWF16)
      if (.not.(present(EFactor) )) then
         print*, "ERROR: you must pass in EFactor if ",&
              "Langmuir_entrainment_str .eq. 'LWF16'!"
         stop 1
      end if
      Vt2_Enhancement = EFactor

      ! From LMD 94, Vtc = sqrt(-beta_T/(c_s*eps))/kappa^2
      Vtc = sqrt(0.2_cvmix_r8/(cvmix_get_kpp_real('c_s', CVmix_kpp_params_in) * &
            cvmix_get_kpp_real('surf_layer_ext', CVmix_kpp_params_in))) / &
            (cvmix_get_kpp_real('vonkarman', CVmix_kpp_params_in)**2)

      do kt=1,nlev
        if (CVmix_kpp_params_in%lscalar_Cv) then
          Cv = cvmix_get_kpp_real('Cv', CVmix_kpp_params_in)
        else
          ! Cv computation comes from Danabasoglu et al., 2006
          if (N_cntr(kt).lt.0.002_cvmix_r8) then
            Cv = 2.1_cvmix_r8-real(200,cvmix_r8)*N_cntr(kt)
          else
            Cv = 1.7_cvmix_r8
          end if
        end if

        cvmix_kpp_compute_unresolved_shear(kt) = -Cv*Vtc*zt_cntr(kt)*          &
                              N_cntr(kt)*ws_cntr(kt)/                          &
                              CVmix_kpp_params_in%Ri_crit * Vt2_Enhancement
        if (cvmix_kpp_compute_unresolved_shear(kt).lt.                         &
            CVmix_kpp_params_in%minVtsqr) then
```

```
      cvmix_kpp_compute_unresolved_shear(kt) = CVmix_kpp_params_in%minVtsqr
    end if
  end do

case (LANGMUIR_ENTRAINMENT_LF17)

  if (.not.(present(LaSL) .and. present(bfsfc) .and. present(ustar))) then
    print*, "ERROR: you must pass in LaSL, bfsfc and ustar if ",&
          "Langmuir_entrainment_str == 'LF17'!"
    stop 1
  end if
  ! only apply Langmuir enhanced entrainment under unstable condition
  if (bfsfc(1)<cvmix_zero) then
    ! (26) of Li and Fox-Kemper, 2017, JPO
    c_CT =  cvmix_get_kpp_real('c_CT', CVmix_kpp_params_in)
    c_ST =  cvmix_get_kpp_real('c_ST', CVmix_kpp_params_in)
    c_LT =  cvmix_get_kpp_real('c_LT', CVmix_kpp_params_in)
    p_LT =  cvmix_get_kpp_real('p_LT', CVmix_kpp_params_in)
    do kt=1,nlev
      if (CVmix_kpp_params_in%lscalar_Cv) then
        Cv = cvmix_get_kpp_real('Cv', CVmix_kpp_params_in)
      else
      ! Cv computation comes from Danabasoglu et al., 2006
        if (N_cntr(kt).lt.0.002_cvmix_r8) then
          Cv = 2.1_cvmix_r8-real(200,cvmix_r8)*N_cntr(kt)
        else
          Cv = 1.7_cvmix_r8
        end if
      end if
      Vtc = sqrt((c_CT*bfsfc(kt)*zt_cntr(kt) + c_ST*ustar**3 +             &
               c_LT*ustar**3*LaSL**(-1.*p_LT))/ws_cntr(kt))
      cvmix_kpp_compute_unresolved_shear(kt) = -Cv*Vtc*zt_cntr(kt)*        &
                              N_cntr(kt)/CVmix_kpp_params_in%Ri_crit
      if (cvmix_kpp_compute_unresolved_shear(kt).lt.                       &
          CVmix_kpp_params_in%minVtsqr) then
        cvmix_kpp_compute_unresolved_shear(kt) = CVmix_kpp_params_in%minVtsqr
      end if
    end do
  else
    ! From LMD 94, Vtc = sqrt(-beta_T/(c_s*eps))/kappa^2
    Vtc = sqrt(0.2_cvmix_r8/(cvmix_get_kpp_real('c_s', CVmix_kpp_params_in) * &
          cvmix_get_kpp_real('surf_layer_ext', CVmix_kpp_params_in))) / &
          (cvmix_get_kpp_real('vonkarman', CVmix_kpp_params_in)**2)

    do kt=1,nlev
      if (CVmix_kpp_params_in%lscalar_Cv) then
        Cv = cvmix_get_kpp_real('Cv', CVmix_kpp_params_in)
      else
```

```
        ! Cv computation comes from Danabasoglu et al., 2006
        if (N_cntr(kt).lt.0.002_cvmix_r8) then
          Cv = 2.1_cvmix_r8-real(200,cvmix_r8)*N_cntr(kt)
        else
          Cv = 1.7_cvmix_r8
        end if
      end if

      cvmix_kpp_compute_unresolved_shear(kt) = -Cv*Vtc*zt_cntr(kt) *       &
                       N_cntr(kt)*ws_cntr(kt)/CVmix_kpp_params_in%Ri_crit
      if (cvmix_kpp_compute_unresolved_shear(kt).lt.                       &
          CVmix_kpp_params_in%minVtsqr) then
        cvmix_kpp_compute_unresolved_shear(kt) = CVmix_kpp_params_in%minVtsqr
      end if
    end do
  end if

case (LANGMUIR_ENTRAINMENT_RWHGK16)

  if (.not.(present(LaSL) )) then
     print*, "ERROR: you must pass in LaSL if ",&
          "Langmuir_entrainment_str == 'RWHGK16'!"
     stop 1
  end if
  RWHGK_ENTR_COEF =  cvmix_get_kpp_real('RWHGK_ENTR_COEF', &
       CVmix_kpp_params_in)
  RWHGK_ENTR_EXP =  cvmix_get_kpp_real('RWHGK_ENTR_EXP', &
       CVmix_kpp_params_in)
  Vt2_Enhancement = cvmix_one + RWHGK_ENTR_COEF * LASL**RWHGK_ENTR_EXP

  ! From LMD 94, Vtc = sqrt(-beta_T/(c_s*eps))/kappa^2
  Vtc = sqrt(0.2_cvmix_r8/(cvmix_get_kpp_real('c_s', CVmix_kpp_params_in) * &
        cvmix_get_kpp_real('surf_layer_ext', CVmix_kpp_params_in))) / &
        (cvmix_get_kpp_real('vonkarman', CVmix_kpp_params_in)**2)

  do kt=1,nlev
    if (CVmix_kpp_params_in%lscalar_Cv) then
      Cv = cvmix_get_kpp_real('Cv', CVmix_kpp_params_in)
    else
      ! Cv computation comes from Danabasoglu et al., 2006
      if (N_cntr(kt).lt.0.002_cvmix_r8) then
        Cv = 2.1_cvmix_r8-real(200,cvmix_r8)*N_cntr(kt)
      else
        Cv = 1.7_cvmix_r8
      end if
    end if

    cvmix_kpp_compute_unresolved_shear(kt) = -Cv*Vtc*zt_cntr(kt)*          &
```

```
                              N_cntr(kt)*ws_cntr(kt)/                          &
                              CVmix_kpp_params_in%Ri_crit * Vt2_Enhancement
          if (cvmix_kpp_compute_unresolved_shear(kt).lt.                       &
              CVmix_kpp_params_in%minVtsqr) then
            cvmix_kpp_compute_unresolved_shear(kt) = CVmix_kpp_params_in%minVtsqr
          end if
        end do

      case DEFAULT

        ! From LMD 94, Vtc = sqrt(-beta_T/(c_s*eps))/kappa^2
        Vtc = sqrt(0.2_cvmix_r8/(cvmix_get_kpp_real('c_s', CVmix_kpp_params_in) * &
              cvmix_get_kpp_real('surf_layer_ext', CVmix_kpp_params_in))) / &
              (cvmix_get_kpp_real('vonkarman', CVmix_kpp_params_in)**2)

        do kt=1,nlev
          if (CVmix_kpp_params_in%lscalar_Cv) then
            Cv = cvmix_get_kpp_real('Cv', CVmix_kpp_params_in)
          else
            ! Cv computation comes from Danabasoglu et al., 2006
            if (N_cntr(kt).lt.0.002_cvmix_r8) then
              Cv = 2.1_cvmix_r8-real(200,cvmix_r8)*N_cntr(kt)
            else
              Cv = 1.7_cvmix_r8
            end if
          end if

          cvmix_kpp_compute_unresolved_shear(kt) = -Cv*Vtc*zt_cntr(kt) *       &
                           N_cntr(kt)*ws_cntr(kt)/CVmix_kpp_params_in%Ri_crit
          if (cvmix_kpp_compute_unresolved_shear(kt).lt.                       &
              CVmix_kpp_params_in%minVtsqr) then
            cvmix_kpp_compute_unresolved_shear(kt) = CVmix_kpp_params_in%minVtsqr
          end if
        end do

    end select
  end if    ! lStokesMOST
```

1.73 compute_phi_inv

INTERFACE:

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

DESCRIPTION:

Computes 1/phi_m or 1/phi_s

USES:

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

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
```

```
logical :: lm, ls
real(cvmix_r8) :: zetastar

! If not specifying lphi_m or lphi_s, routine will error out, but
! initializing result to 0 removes warning about possibly returning an
! un-initialized value
compute_phi_inv = cvmix_zero

if (present(lphi_m)) then
  lm = lphi_m
else
  lm = .false.
end if

if (present(lphi_s)) then
  ls = lphi_s
else
  ls = .false.
end if

if (lm.eqv.ls) then
  print*, "ERROR: must compute phi_m or phi_s, can not compute both!"
  stop 1
end if
```

```
zetastar =  zeta
if ( present(L_Lstokes) ) zetastar = zeta * L_Lstokes

if ( CVmix_kpp_params_in%lStokesMOST ) then
  if (lm) then
    if (zeta.ge.cvmix_zero) then   ! STABLE
      compute_phi_inv = cvmix_one/(cvmix_one + real(14,cvmix_r8)*zetastar)
    else
      compute_phi_inv = &
      (cvmix_one - real(14,cvmix_r8)*zetastar)**(cvmix_one/real(3,cvmix_r8))
    end if
  end if
  if (ls) then
    if (zeta.ge.cvmix_zero) then   ! STABLE
      compute_phi_inv = cvmix_one/(cvmix_one + real( 5,cvmix_r8)*zetastar)
    else
      compute_phi_inv = &
      (cvmix_one - real(25,cvmix_r8)*zetastar)**(cvmix_one/real(3,cvmix_r8))
    end if
  end if
else    ! not lStokesMOST

  if (lm) then
    if (zeta.ge.cvmix_zero) then
      ! Stable region
      compute_phi_inv = cvmix_one/(cvmix_one + real(5,cvmix_r8)*zeta)
    else if (zeta.ge.CVmix_kpp_params_in%zeta_m) then
      compute_phi_inv = (cvmix_one - real(16,cvmix_r8)*zeta)**0.25_cvmix_r8
    else
      compute_phi_inv = (CVmix_kpp_params_in%a_m -                          &
                         CVmix_kpp_params_in%c_m*zeta)**                    &
                         (cvmix_one/real(3,cvmix_r8))
    end if
  end if

  if (ls) then
    if (zeta.ge.cvmix_zero) then
      ! Stable region
      compute_phi_inv = cvmix_one/(cvmix_one + real(5,cvmix_r8)*zeta)
    else if (zeta.ge.CVmix_kpp_params_in%zeta_s) then
      compute_phi_inv = (cvmix_one - real(16,cvmix_r8)*zeta)**0.5_cvmix_r8
    else
      compute_phi_inv = (CVmix_kpp_params_in%a_s -                          &
                         CVmix_kpp_params_in%c_s*zeta)**                    &
                         (cvmix_one/real(3,cvmix_r8))
    end if
  end if
end if
```

1.74 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)

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
logical :: lm, ls
real(cvmix_r8)  :: chi, xi_tmp

! If not specifying lchi_m or lchi_s, routine will error out, but
! initializing result to 1 (No Stokes value) removes warning about
! possibly returning an un-initialized value
compute_Stokes_chi = cvmix_one

if (present(lchi_m)) then
    lm = lchi_m
else
    lm = .false.
end if

if (present(lchi_s)) then
```

```
  ls = lchi_s
else
  ls = .false.
end if

if (lm.eqv.ls) then
  print*, "ERROR: must compute chi_m or chi_s, can not compute both!"
  stop 1
end if

if (lm) then
  xi_tmp = MAX( cvmix_zero , MIN( xi , 0.73_cvmix_r8 ) )
  chi = cvmix_one - 1.671_cvmix_r8 * xi_tmp
  if ( xi_tmp .ge. 0.35_cvmix_r8) &
    chi = 1.03_cvmix_r8 + xi_tmp * ( 1.58_cvmix_r8*xi_tmp - 2.31_cvmix_r8 )
  compute_Stokes_chi = chi
end if

if (ls) then
  xi_tmp = MAX( cvmix_zero , MIN( xi , 0.89_cvmix_r8 ) )
  chi = cvmix_one - 1.594_cvmix_r8 * xi_tmp
  if ( xi_tmp .ge. 0.35_cvmix_r8) &
    chi = 0.78_cvmix_r8 + xi_tmp * ( 0.67_cvmix_r8*xi_tmp - 1.20_cvmix_r8 )
  compute_Stokes_chi = chi
end if
```

1.75 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) = $(/a_0, a_1, a_2, a_3/)$

USES:

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

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
```

```
coeffs(1) =  cvmix_zero
coeffs(2) =  cvmix_one
coeffs(3) =  real(3,cvmix_r8)*GAT1 - DGAT1 - real(2,cvmix_r8)
coeffs(4) = -real(2,cvmix_r8)*GAT1 + DGAT1 + cvmix_one
```

1.76 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.

USES:

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

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
```

```
! Local variables
real(cvmix_r8), dimension(4) :: coeffs
real(cvmix_r8) :: dnu_dz_above, dnu_dz_below, dnu_dz_local, wgt
real(cvmix_r8) :: iface_depth

! (1) Compute derivatives of nu at layer centers (central difference)
!     Sign convention: dnu/dz is positive if nu increases as you
!                      move up in the column
if (present(diff_2above)) then
  dnu_dz_above = (diff_2above-diffs_iface(1))/layer_widths(1)
else
  ! Assume diffusivity goes to 0 at surface (z=0)
  dnu_dz_above = -diffs_iface(1)/layer_widths(1)
end if
dnu_dz_below = (diffs_iface(1)-diffs_iface(2))/layer_widths(2)
! Stability => require non-negative dnu_dz
if (dnu_dz_above.lt.0.0_cvmix_r8) dnu_dz_above = 0.0_cvmix_r8
if (dnu_dz_below.lt.0.0_cvmix_r8) dnu_dz_below = 0.0_cvmix_r8

! (2) Compute dnu/dz at OBL_depth by weighted average of values
!     computed above (see LMD94, Eq. (D5) for details)
iface_depth = depths_cntr(1) - 0.5_cvmix_r8*layer_widths(1)
wgt = (-iface_depth-OBL_depth) / layer_widths(1)
dnu_dz_local = wgt*dnu_dz_above + (cvmix_one-wgt)*dnu_dz_below

! (3) Linear interpolant: slope = value computed in (2) and the line goes
!     through the point (iface_depth, diffs_iface(1))
coeffs = cvmix_zero
coeffs(1) = diffs_iface(1) - dnu_dz_local*iface_depth
coeffs(2) = dnu_dz_local
if (present(dnu_dz)) then
```

```
  dnu_dz = dnu_dz_local
end if
cvmix_kpp_compute_nu_at_OBL_depth_LMD94=cvmix_math_evaluate_cubic(coeffs, &
                                                                  -OBL_depth)
```

1.77 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).

USES:

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

INPUT PARAMETERS:

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

```
real(cvmix_r8) :: us_sl, lasl_sqr_i
real(cvmix_r8) :: cvmix_kpp_EFactor_model

if (u10 .gt. cvmix_zero .and. ustar .gt. cvmix_zero) then
  ! surface layer averaged Stokes drift
  us_sl = cvmix_kpp_ustokes_SL_model(u10, hbl, CVmix_params_in)
  !
  ! LaSL^{-2}
  lasl_sqr_i = us_sl/ustar
  !
  ! enhancement factor (Li et al., 2016)
  cvmix_kpp_EFactor_model = sqrt(cvmix_one &
            +cvmix_one/1.5_cvmix_r8**2*lasl_sqr_i &
            +cvmix_one/5.4_cvmix_r8**4*lasl_sqr_i**2)
```

```
else
  ! otherwise set to one
  cvmix_kpp_EFactor_model = cvmix_one
end if
```

1.78 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
```

```
real(cvmix_r8), parameter :: &
    ! ratio of U19.5 to U10 (Holthuijsen, 2007)
    u19p5_to_u10 = 1.075_cvmix_r8, &
    ! ratio of mean frequency to peak frequency for
    ! Pierson-Moskowitz spectrum (Webb, 2011)
    fm_to_fp = 1.296_cvmix_r8, &
    ! ratio of surface Stokes drift to U10
    us_to_u10 = 0.0162_cvmix_r8, &
    ! loss ratio of Stokes transport
    r_loss = 0.667_cvmix_r8

real(cvmix_r8) :: us, hm0, fm, fp, vstokes, kphil, kstar
real(cvmix_r8) :: z0, z0i, r1, r2, r3, r4, tmp
real(cvmix_r8) :: cvmix_kpp_ustokes_SL_model

if (u10 .gt. cvmix_zero) then
  ! surface Stokes drift
  us = us_to_u10*u10
  !
  ! significant wave height from Pierson-Moskowitz
```

```
  ! spectrum (Bouws, 1998)
  hm0 = 0.0246_cvmix_r8*u10**2
  !
  ! peak frequency (PM, Bouws, 1998)
  tmp = 2.0_cvmix_r8*cvmix_PI*u19p5_to_u10*u10
  fp = 0.877_cvmix_r8*CVmix_params_in%Gravity/tmp
  !
  ! mean frequency
  fm = fm_to_fp*fp
  !
  ! total Stokes transport (a factor r_loss is applied to account
  !  for the effect of directional spreading, multidirectional waves
  !  and the use of PM peak frequency and PM significant wave height
  !  on estimating the Stokes transport)
  vstokes = 0.125_cvmix_r8*cvmix_PI*r_loss*fm*hm0**2
  !
  ! the general peak wavenumber for Phillips' spectrum
  ! (Breivik et al., 2016) with correction of directional spreading
  kphil = 0.176_cvmix_r8*us/vstokes
  !
  ! surface layer averaged Stokes dirft with Stokes drift profile
  ! estimated from Phillips' spectrum (Breivik et al., 2016)
  ! the directional spreading effect from Webb and Fox-Kemper, 2015
  ! is also included
  kstar = kphil*2.56_cvmix_r8
  ! surface layer
  z0 = 0.2_cvmix_r8*abs(hbl)
  z0i = cvmix_one/z0
  ! term 1 to 4
  r1 = (0.151_cvmix_r8/kphil*z0i-0.84_cvmix_r8) &
        *(cvmix_one-exp(-2.0_cvmix_r8*kphil*z0))
  r2 = -(0.84_cvmix_r8+0.0591_cvmix_r8/kphil*z0i) &
         *sqrt(2.0_cvmix_r8*cvmix_PI*kphil*z0) &
         *erfc(sqrt(2.0_cvmix_r8*kphil*z0))
  r3 = (0.0632_cvmix_r8/kstar*z0i+0.125_cvmix_r8) &
        *(cvmix_one-exp(-2.0_cvmix_r8*kstar*z0))
  r4 = (0.125_cvmix_r8+0.0946_cvmix_r8/kstar*z0i) &
         *sqrt(2.0_cvmix_r8*cvmix_PI*kstar*z0) &
         *erfc(sqrt(2.0_cvmix_r8*kstar*z0))
  cvmix_kpp_ustokes_SL_model = us*(0.715_cvmix_r8+r1+r2+r3+r4)
else
  cvmix_kpp_ustokes_SL_model = cvmix_zero
end if
```

1.79 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
```

```
%///////////////////////////////////////////////////////////
```

```
\mbox{}\hrulefill\
```

```
%///////////////////////////////////////////////////////////
```

```
\mbox{}\hrulefill\
```

```
 \subsection [] {cvmix\_kpp\_composite\_shape}
```

```
\bigskip{\sf INTERFACE:}
\begin{verbatim}
   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



real(cvmix_r8)  :: a2Gsig, a3Gsig, bGsig, sig_m, G_m, G_1, sig

a2Gsig = -2.1637_cvmix_r8
a3Gsig =  0.5831_cvmix_r8
sig_m  =  0.35_cvmix_r8
G_m    =  0.11_cvmix_r8     ! sig_m + sig_m * sig_m * (a2Gsig + a3Gsig * sig_m)
G_1    =  MAX( cvmix_zero  ,  MIN( Gat1 , G_m ) )

if (sigma .lT. sig_m)  then
  sig    = MAX( sigma , cvmix_zero)
  Gsig   = sig + sig * sig * (a2Gsig + a3Gsig * sig)
  dGdsig = cvmix_one + sig * (2.0_cvmix_r8 * a2Gsig + 3.0_cvmix_r8 * a3Gsig * sig)
else
  bGsig  =  (G_m-G_1) / (1.-sig_m)**2
  sig    = MIN( sigma , cvmix_one )
  Gsig   = G_1 + bGsig * (cvmix_one - sig) * (cvmix_one - sig)
  dGdsig = bGsig * 2.0_cvmix_r8 * (sig - cvmix_one)
end if
```

1.80 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 cente
 integer,        intent(in) :: kSL                           !< cell index of Surface La
 real(cvmix_r8), intent(in) :: SLDepth                       !< Surface Layer Depth Inte
```

```
real(cvmix_r8), intent(in) :: surf_buoy_force              !< Surface buoyancy flux fo
real(cvmix_r8), intent(in) :: surf_fric_vel, omega_w2x     !< Surface wind forcing fro
real(cvmix_r8), dimension(:), intent(in) :: uE, vE         !< Eulerian velocity at cer
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 cel

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 interfac
real(cvmix_r8), dimension(:), intent(inout) :: uSbar, vSbar !< Cell average Stokes dri
real(cvmix_r8), intent(inout) :: StokesXI                   !< Stokes similarity parame
```

```
type(cvmix_kpp_params_type),  pointer :: CVmix_kpp_params_in
real(cvmix_r8) :: PU, PS , PB                    ! surface layer TKE production terms
real(cvmix_r8) :: uS_TMP, vS_TMP, uSbar_TMP, vSbar_TMP                        ! Temporary
real(cvmix_r8) :: ustar, delH, delU, delV, omega_E2x, cosOmega, sinOmega
real(cvmix_r8) :: BLDepth, TauMAG, TauCG, TauDG, taux0, tauy0, Stk0 , Pinc
real(cvmix_r8) :: PBfact , CempCGm                                            ! Empirical
real(cvmix_r8) :: dtop, tauEtop, tauxtop, tauytop                             ! Cell top
real(cvmix_r8) :: dbot, tauEbot, tauxbot, tauybot, sigbot, Gbot               ! Cell bott
integer        :: ktmp                                                        ! vertical

CVmix_kpp_params_in => CVmix_kpp_params_saved
if (present(CVmix_kpp_params_user)) then
  CVmix_kpp_params_in => CVmix_kpp_params_user
end if

if ( CVmix_kpp_params_in%lStokesMOST ) then

! Move bottom of cell kSL up to Surface Layer Extent = SLDepth
uS_TMP = uS(kSL+1)
vS_TMP = vS(kSL+1)
uSbar_TMP = uSbar(kSL)
vSbar_TMP = vSbar(kSL)
uS(kSL+1)   = uS_SLD
vS(kSL+1)   = vS_SLD
uSbar(kSL)= uSbar_SLD
vSbar(kSL)= vSbar_SLD

CempCGm=  3.5_cvmix_r8

ustar   = MAX( surf_fric_vel , 1.e-4_cvmix_r8 )   ! > 0
taux0   = ustar**2 * cos(omega_w2x)
tauy0   = ustar**2 * sin(omega_w2x)
```

```
Stk0    = sqrt( uS(1)**2 + vS(1)**2 )
BLDepth = SLDepth / CVmix_kpp_params_in%surf_layer_ext

! Parameterized Buoyancy production of TKE
PBfact =  0.110_cvmix_r8
PB     = PBfact * MAX( -surf_buoy_force * BLdepth ,  cvmix_zero )

! Compute Both Shear Production Terms down from  Surface = initial top values
PU     = 0.0
PS     = 0.0
dtop   = 0.0
delU   = uE(1) - uE(2)
delV   = vE(1) - vE(2)
tauEtop = (taux0 * delU + tauy0 * delV ) / (zk(1) - zk(2) )
tauxtop = taux0
tauytop = tauy0

do ktmp = 1, kSL
  ! SLdepth can be between cell interfaces kSL and kSL+1
  delH = min( max(cvmix_zero, SLdepth - dtop), (zi(ktmp) - zi(ktmp+1) ) )
  dbot = MIN( dtop + delH ,  SLdepth)
  sigbot = dbot / BLdepth
  Gbot     = cvmix_kpp_composite_shape(sigbot)
  TauMAG   = ustar * ustar * Gbot / sigbot
  delU     = uE(ktmp) - uE(ktmp+1)
  delV     = vE(ktmp) - vE(ktmp+1)
  Omega_E2x= atan2( delV  , delU )
  cosOmega = cos(Omega_E2x)
  sinOmega = sin(Omega_E2x)
  tauCG    = CempCGm * Gbot *  (taux0 * cosOmega - tauy0 * sinOmega)
  ! tauDG    = sqrt( TauMAG**2 - tauCG**2 ) ! G
  tauDG    = TauMAG                        ! E
  tauxbot  = tauDG * cosOmega  -  tauCG * sinOmega
  tauybot  = tauDG * sinOmega  +  tauCG * cosOmega
  tauEbot  = (tauxbot * delU + tauybot * delV) / (zk(ktmp) - zk(ktmp+1) )

  ! Increment Eulerian Shear Production
  Pinc     = 0.5_cvmix_r8 * (tauEbot + tauEtop) * delH
  PU       = PU + MAX( Pinc , cvmix_zero )

  ! Increment Stokes Shear Production
  Pinc   = tauxtop*uS(ktmp) - tauxbot*uS(ktmp+1) + tauytop*vS(ktmp) - tauybot*vS(ktmp+1)
  Pinc   = Pinc - (tauxtop-tauxbot) * uSbar(ktmp)  - (tauytop-tauybot) * vSbar(ktmp)
  PS     = PS +  MAX( Pinc , cvmix_zero )

  ! Bottom becomes next top
  dtop    = dbot
  tauxtop = tauxbot
```

```
    tauytop = tauybot
    tauEtop = tauEbot
  enddo

  ! Compute Stokes similarity parameter
  StokesXI  = PS / MAX( PU + PS + PB , 1.e-12_cvmix_r8 )


  ! Restore bottom of cell kSL at zi(kSL+1) with stored Stokes Drift ; ditto average over
  uS(kSL+1)  = uS_TMP
  vS(kSL+1)  = vS_TMP
  uSbar(kSL) = uSbar_TMP
  vSbar(kSL) = vSbar_TMP

 else    ! not lStokesMOST
   StokesXI  = cvmix_zero
 end if
```

1.81 Fortran: Module Interface cvmix_convection (Source File: cvmix_convection.F90)

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
! mixing.
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
```

1.82 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.

USES:

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

OUTPUT PARAMETERS:

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

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
```

```
! Set convect_diff and convect_visc in conv_params_type
call cvmix_put_conv("convect_diff", convect_diff, CVmix_conv_params_user)
call cvmix_put_conv("convect_visc", convect_visc, CVmix_conv_params_user)

if (present(lBruntVaisala)) then
  call cvmix_put_conv("lBruntVaisala", lBruntVaisala,                     &
                      CVmix_conv_params_user)
else
  call cvmix_put_conv("lBruntVaisala", .false., CVmix_conv_params_user)
end if

if (present(BVsqr_convect)) then
  call cvmix_put_conv("BVsqr_convect", BVsqr_convect,                     &
                      CVmix_conv_params_user)
else
  call cvmix_put_conv("BVsqr_convect", cvmix_zero, CVmix_conv_params_user)
end if

if (present(lnoOBL)) then
  call cvmix_put_conv("lnoOBL", lnoOBL, CVmix_conv_params_user)
else
  call cvmix_put_conv("lnoOBL", .true., CVmix_conv_params_user)
end if

if (present(old_vals)) then
  select case (trim(old_vals))
    case ("overwrite")
      call cvmix_put_conv('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,     &
                          cvmix_conv_params_user)
    case ("sum")
      call cvmix_put_conv('handle_old_vals', CVMIX_SUM_OLD_AND_NEW_VALS,  &
                          cvmix_conv_params_user)
    case ("max")
```

```
        call cvmix_put_conv('handle_old_vals', CVMIX_MAX_OLD_AND_NEW_VALS,  &
                            cvmix_conv_params_user)
      case DEFAULT
        print*, "ERROR: ", trim(old_vals), " is not a valid option for ",   &
                "handling old values of diff and visc."
        stop 1
    end select
  else
    call cvmix_put_conv('handle_old_vals', CVMIX_OVERWRITE_OLD_VAL,         &
                            cvmix_conv_params_user)
  end if
```

1.83 cvmix_coeffs_conv_wrap

INTERFACE:

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

DESCRIPTION:

Computes vertical diffusion coefficients for convective mixing.

USES:

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

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
```

```
!-----------------------------------------------------------------------
!
!  local variables
!
!-----------------------------------------------------------------------
```

```
real(cvmix_r8), dimension(CVmix_vars%max_nlev+1) :: new_Mdiff, new_Tdiff
type (cvmix_conv_params_type), pointer :: CVmix_conv_params_in
integer :: nlev, max_nlev

if (present(CVmix_conv_params_user)) then
  CVmix_conv_params_in => CVmix_conv_params_user
else
  CVmix_conv_params_in => CVmix_conv_params_saved
end if

nlev = CVmix_vars%nlev
max_nlev = CVmix_vars%max_nlev

if (.not.associated(CVmix_vars%Mdiff_iface)) &
  call cvmix_put(CVmix_vars, "Mdiff", cvmix_zero, max_nlev)
if (.not.associated(CVmix_vars%Tdiff_iface)) &
  call cvmix_put(CVmix_vars, "Tdiff", cvmix_zero, max_nlev)

call cvmix_coeffs_conv(new_Mdiff, new_Tdiff,                              &
                       CVmix_vars%SqrBuoyancyFreq_iface,                  &
                       CVmix_vars%WaterDensity_cntr,                      &
                       CVmix_vars%AdiabWaterDensity_cntr,                 &
                       nlev, max_nlev, nint(CVMix_vars%kOBL_depth)+1,     &
                       CVmix_conv_params_user)
call cvmix_update_wrap(CVmix_conv_params_in%handle_old_vals, max_nlev,    &
                       Mdiff_out = CVmix_vars%Mdiff_iface,                &
                       new_Mdiff = new_Mdiff,                             &
                       Tdiff_out = CVmix_vars%Tdiff_iface,                &
                       new_Tdiff = new_Tdiff)
```

1.84 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.

USES:

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

INPUT PARAMETERS:

```
     integer,                                intent(in) :: nlev, max_nlev
     integer,                                intent(in) :: OBL_ind
     ! max_nlev+1
     real(cvmix_r8), dimension(max_nlev+1), intent(in) :: Nsqr
     ! max_nlev
     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:

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

```
!-----------------------------------------------------------------------
!
!  local variables
!
!-----------------------------------------------------------------------

    real(cvmix_r8) :: convect_mdiff, convect_tdiff, wgt
    integer        :: kw
    logical        :: lnoOBL
    type (cvmix_conv_params_type), pointer :: CVmix_conv_params_in

    if (present(CVmix_conv_params_user)) then
      CVmix_conv_params_in => CVmix_conv_params_user
    else
      CVmix_conv_params_in => CVmix_conv_params_saved
    end if
    lnoOBL = CVMix_conv_params_in%lnoOBL
    convect_mdiff = CVMix_conv_params_in%convect_visc
    convect_tdiff = CVMix_conv_params_in%convect_diff

!-----------------------------------------------------------------------
!
!  enhance the vertical mixing coefficients if gravitationally unstable
!
!-----------------------------------------------------------------------
```

```
if (CVmix_conv_params_in%lBruntVaisala) then
  ! Brunt-Vaisala mixing based on buoyancy
  ! Based on parameter BVsqr_convect
  ! diffusivity = convect_diff * wgt
  ! viscosity   = convect_visc * wgt

  ! For BVsqr_convect < 0:
  ! wgt = 0 for N^2 > 0
  ! wgt = 1 for N^2 < BVsqr_convect
  ! wgt = [1 - (1-N^2/BVsqr_convect)^2]^3 otherwise

  ! If BVsqr_convect >= 0:
  ! wgt = 0 for N^2 > 0
  ! wgt = 1 for N^2 <= 0

  ! Compute wgt
  if (CVmix_conv_params_in%BVsqr_convect.lt.0) then
    do kw=1, nlev
      wgt = cvmix_zero
      if (Nsqr(kw).le.0) then
        if (Nsqr(kw).gt.CVmix_conv_params_in%BVsqr_convect) then
          wgt = cvmix_one - Nsqr(kw) / CVmix_conv_params_in%BVsqr_convect
          wgt = (cvmix_one - wgt**2)**3
        else
          wgt = cvmix_one
        end if
      end if
      Mdiff_out(kw) = wgt*cvmix_get_conv_real('convect_visc',             &
                                              CVmix_conv_params_in)
      Tdiff_out(kw) = wgt*cvmix_get_conv_real('convect_diff',             &
                                              CVmix_conv_params_in)
    end do
  else ! BVsqr_convect >= 0 => step function
    do kw=1,nlev
      if ((Nsqr(kw).le.0).and.((kw.ge.OBL_ind).or.(.not.lnoOBL))) then
        Mdiff_out(kw) = cvmix_get_conv_real('convect_visc',               &
                                            CVmix_conv_params_in)
        Tdiff_out(kw) = cvmix_get_conv_real('convect_diff',               &
                                            CVmix_conv_params_in)
      else
        Mdiff_out(kw) = cvmix_zero
        Tdiff_out(kw) = cvmix_zero
      end if
    end do
  end if
  Mdiff_out(nlev+1) = cvmix_zero
  Tdiff_out(nlev+1) = cvmix_zero
else
```

```
    ! Default convection mixing based on density
    do kw=1,nlev-1
      if (dens(kw).gt.dens_lwr(kw)) then
        if (CVmix_conv_params_in%convect_visc.eq.cvmix_zero) then
          ! convection only affects tracers
          Mdiff_out(kw+1) = Mdiff_out(kw)
        else
          Mdiff_out(kw+1) = convect_mdiff
        end if
        Tdiff_out(kw+1) = convect_tdiff
      end if
    end do
  end if
```

1.85 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.

USES:

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

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


type (cvmix_conv_params_type), pointer :: CVmix_conv_params_out
```

```
if (present(CVmix_conv_params_user)) then
  CVmix_conv_params_out => CVmix_conv_params_user
else
  CVmix_conv_params_out => CVmix_conv_params_saved
end if

select case (trim(varname))
  case ("old_vals", "handle_old_vals")
    CVmix_conv_params_out%handle_old_vals = val
  case DEFAULT
    call cvmix_put_conv(varname, real(val, cvmix_r8),                    &
                        CVmix_conv_params_user)
end select
```

1.86 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
type (cvmix_conv_params_type), pointer :: CVmix_conv_params_out

if (present(CVmix_conv_params_user)) then
  CVmix_conv_params_out => CVmix_conv_params_user
else
  CVmix_conv_params_out => CVmix_conv_params_saved
end if

select case (trim(varname))
  case ('convect_diff')
    CVmix_conv_params_out%convect_diff = val
  case ('convect_visc')
    CVmix_conv_params_out%convect_visc = val
  case ('BVsqr_convect')
    CVmix_conv_params_out%BVsqr_convect = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.87 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.

USES:

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

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



type (cvmix_conv_params_type), pointer :: CVmix_conv_params_out

if (present(CVmix_conv_params_user)) then
  CVmix_conv_params_out => CVmix_conv_params_user
else
  CVmix_conv_params_out => CVmix_conv_params_saved
end if

select case (trim(varname))
  case ('lBruntVaisala')
    CVmix_conv_params_out%lBruntVaisala = val
  case ('lnoOBL')
    CVmix_conv_params_out%lnoOBL = val
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1
end select
```

1.88 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.

USES:

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

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
```

```
type(cvmix_conv_params_type), pointer :: CVmix_conv_params_get

if (present(CVmix_conv_params_user)) then
  CVmix_conv_params_get => CVmix_conv_params_user
else
  CVmix_conv_params_get => CVmix_conv_params_saved
end if

cvmix_get_conv_real = cvmix_zero
select case (trim(varname))
  case ('convect_diff')
    cvmix_get_conv_real = CVmix_conv_params_get%convect_diff
  case ('convect_visc')
    cvmix_get_conv_real = CVmix_conv_params_get%convect_visc
  case DEFAULT
    print*, "ERROR: ", trim(varname), " not a valid choice!"
    stop 1

end select
```

1.89 Fortran: Module Interface cvmix_math (Source File: cvmix_math.F90)

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:

Id
URL

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
```

1.90 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
```

```
    ! Local variables
    real(cvmix_r8) :: det
    integer        :: k, k2
    real(kind=cvmix_r8), dimension(4,4) :: Minv
    real(kind=cvmix_r8), dimension(4)   :: rhs

    ! All interpolation assumes form of
    ! y = dx^3 + cx^2 + bx + a
    ! linear => c = d = 0
    ! quad   => d = 0
    coeffs(1:4) = 0.0_cvmix_r8
    select case (interp_type)
      case (CVMIX_MATH_INTERP_LINEAR)
        ! Match y(1) and y(2)
!        print*, "Linear interpolation"
        coeffs(2) = (y(2)-y(1))/(x(2)-x(1))
        coeffs(1) = y(1)-coeffs(2)*x(1)
      case (CVMIX_MATH_INTERP_QUAD)
        ! Match y(1), y(2), and y'(1) [requires x(0)]
!        print*, "Quadratic interpolation"
        ! [ x2^2 x2 1 ][ c ]   [    y2 ]
        ! [ x1^2 x1 1 ][ b ] = [    y1 ]
        ! [  2x1  1 0 ][ a ]   [ slope ]
        !      ^^^
        !       M
        det = -((x(2)-x(1))**2)
```

```
        ! only using 3x3 block of Minv and first 3 elements of rhs
        rhs(1) = y(2)
        rhs(2) = y(1)
        if (present(x0).and.present(y0)) then
          rhs(3) = (y(1)-y0)/(x(1)-x0)
        else
          rhs(3) = 0.0_cvmix_r8
        end if

        Minv(1,1) = -cvmix_one/det
        Minv(1,2) = cvmix_one/det
        Minv(1,3) = -cvmix_one/(x(2)-x(1))
        Minv(2,1) = real(2, cvmix_r8)*x(1)/det
        Minv(2,2) = -real(2, cvmix_r8)*x(1)/det
        Minv(2,3) = (x(2)+x(1))/(x(2)-x(1))
        Minv(3,1) = -(x(1)**2)/det
        Minv(3,2) = x(2)*(real(2, cvmix_r8)*x(1)-x(2))/det
        Minv(3,3) = -x(2)*x(1)/(x(2)-x(1))

        do k=1,3
          do k2=1,3
            ! Note: weird "4-k2" term is used because I switched from
            ! y= 0x^3 + bx^2 + cx + d to
            ! y = a + bx + cx^2 + 0x^3
            coeffs(k2) = coeffs(k2)+Minv(4-k2,k)*rhs(k)
          end do
        end do
      case (CVMIX_MATH_INTERP_CUBE_SPLINE)
        ! Match y(1), y(2), y'(1), and y'(2)
!        print*, "Cubic spline interpolation"
        ! [ x2^3 x2^2 x2 1 ][ d ]   [     y2 ]
        ! [ x1^3 x1^2 x1 1 ][ c ] = [     y1 ]
        ! [  3x1  2x1  1 0 ][ b ]   [ slope1 ]
        ! [  3x2  2x2  1 0 ][ a ]   [ slope2 ]
        !      ^^^
        !       M
        det = -((x(2)-x(1))**3)
        rhs(1) = y(2)
        rhs(2) = y(1)
        if (present(x0).and.present(y0)) then
          rhs(3) = (y(1)-y0)/(x(1)-x0)
        else
          rhs(3) = 0.0_cvmix_r8
        end if
        rhs(4) = (y(2)-y(1))/(x(2)-x(1))

        Minv(1,1) = real(2, cvmix_r8)/det
        Minv(1,2) = -real(2, cvmix_r8)/det
```

```
      Minv(1,3) = (x(1)-x(2))/det
      Minv(1,4) = (x(1)-x(2))/det
      Minv(2,1) = -real(3, cvmix_r8)*(x(2)+x(1))/det
      Minv(2,2) = real(3, cvmix_r8)*(x(2)+x(1))/det
      Minv(2,3) = (x(2)-x(1))*(real(2, cvmix_r8)*x(2)+x(1))/det
      Minv(2,4) = (x(2)-x(1))*(real(2, cvmix_r8)*x(1)+x(2))/det
      Minv(3,1) = real(6, cvmix_r8)*x(2)*x(1)/det
      Minv(3,2) = -real(6, cvmix_r8)*x(2)*x(1)/det
      Minv(3,3) = -x(2)*(x(2)-x(1))*(real(2, cvmix_r8)*x(1)+x(2))/det
      Minv(3,4) = -x(1)*(x(2)-x(1))*(real(2, cvmix_r8)*x(2)+x(1))/det
      Minv(4,1) = -(x(1)**2)*(real(3, cvmix_r8)*x(2)-x(1))/det
      Minv(4,2) = -(x(2)**2)*(-real(3, cvmix_r8)*x(1)+x(2))/det
      Minv(4,3) = x(1)*(x(2)**2)*(x(2)-x(1))/det
      Minv(4,4) = x(2)*(x(1)**2)*(x(2)-x(1))/det

      do k=1,4
        do k2=1,4
          ! Note: weird "5-k2" term is used because I switched from
          ! y = a + bx + cx^2 + dx^3 to
          ! y= ax^3 + bx^2 + cx + d
          coeffs(k2) = coeffs(k2)+Minv(5-k2,k)*rhs(k)
        end do
      end do
    end select
```

1.91 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 $f'(x)$

USES:

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

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



    ! Local Variables
    integer :: i

    ! Initialize both the cubic and its derivative to its constant term and
    ! then add the powers of x_in via a do-loop. This both reduces the number
    ! of arithmetic steps in the algorithm and avoids possible compiler issues
    ! if x_in = 0 (because 0*0 is undefined in some compilers)
    cvmix_math_evaluate_cubic = coeffs(1)
    if (present(fprime)) &
      fprime = coeffs(2)
    do i=2,4
      cvmix_math_evaluate_cubic = cvmix_math_evaluate_cubic +                &
                                  coeffs(i)*(x_in**(i-1))
      if (present(fprime).and.(i.gt.2)) &
        fprime = fprime + coeffs(i)*real(i-1,cvmix_r8)*(x_in**(i-2))
    end do

  end function cvmix_math_evaluate_cubic

end module cvmix_math
module cvmix_put_get
```

1.92 Fortran: Module Interface cvmix_put_get (Source File: cvmix_put_get.F90)

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
```

1.93 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.

USES:

Only those used by entire module.

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
```

```
! Local variables
integer :: nlev

if (present(nlev_in)) then
  nlev = nlev_in
else
  nlev = CVmix_vars%max_nlev
end if

if ((trim(varname).ne.'nlev').and.(nlev.eq.-1)) then
  print*, "ERROR: you must specify the number of levels before ", &
          "you can pack data into a cvmix_data_type!"
  print*, "You tried to set ", trim(varname)
  stop 1
end if

select case (trim(cvmix_att_name(varname)))
  case ('nlev')
    CVmix_vars%nlev = val
    if (CVmix_vars%max_nlev.eq.-1) then
      CVmix_vars%max_nlev= val
    end if
  case ('max_nlev')
    CVmix_vars%max_nlev = val
    if (CVmix_vars%nlev.eq.-1) then
      CVmix_vars%nlev= val
    end if
  case default
    ! All other scalars are real(cvmix_r8)
    call cvmix_put_real(CVmix_vars, varname, real(val,cvmix_r8), nlev_in)
end select
```

1.94 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.

USES:

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

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
```

```
! Local variables
integer :: nlev

if (present(nlev_in)) then
  nlev = nlev_in
else
  nlev = CVmix_vars%max_nlev
end if

if (nlev.eq.-1) then
  print*, "ERROR: you must specify the number of levels before ", &
          "you can pack data into a cvmix_data_type!"
  print*, "You tried to set ", trim(varname)
  stop 1
end if

select case (trim(cvmix_att_name(varname)))
  case ('OceanDepth')
    CVmix_vars%OceanDepth = val
  case ('BoundaryLayerDepth')
```

```
        CVmix_vars%BoundaryLayerDepth = val
      case ('SeaSurfaceHeight')
        CVmix_vars%SeaSurfaceHeight = val
      case ('SurfaceFriction')
        CVmix_vars%SurfaceFriction = val
      case ("SurfaceBuoyancyForcing")
        CVmix_vars%SurfaceBuoyancyForcing = val
      case ("Latitude")
        CVmix_vars%lat = val
      case ("Longitude")
        CVmix_vars%lon = val
      case ("Coriolis")
        CVmix_vars%Coriolis = val
      case ("kOBL_depth")
        CVmix_vars%kOBL_depth = val
      case ("LangmuirEnhancementFactor")
        CVmix_vars%LangmuirEnhancementFactor = val
      case ("LangmuirNumber")
        CVmix_vars%LangmuirNumber = val
      case ('SimmonsCoeff')
        CVmix_vars%SimmonsCoeff = val

      case ("dzw")
!        print*, "WARNING: you are setting the cell midpoint to midpoint ",    &
!                "distance in all levels to a constant value"
        if (.not.associated(CVmix_vars%dzw)) then
          allocate(CVmix_vars%dzw(nlev+1))
        end if
        CVmix_vars%dzw(:) = val
      case ("Mdiff_iface")
        if (.not.associated(CVmix_vars%Mdiff_iface)) then
          allocate(CVmix_vars%Mdiff_iface(nlev+1))
        end if
        CVmix_vars%Mdiff_iface(:) = val
      case ("Tdiff_iface")
        if (.not.associated(CVmix_vars%Tdiff_iface)) then
          allocate(CVmix_vars%Tdiff_iface(nlev+1))
        end if
        CVmix_vars%Tdiff_iface(:) = val
      case ("Sdiff_iface")
        if (.not.associated(CVmix_vars%Sdiff_iface)) then
          allocate(CVmix_vars%Sdiff_iface(nlev+1))
        end if
        CVmix_vars%Sdiff_iface(:) = val
      case ("ShearRichardson_iface")
!        print*, "WARNING: you are setting the Richardson number in all ",     &
!                "levels to a constant value"
        if (.not.associated(CVmix_vars%ShearRichardson_iface)) then
```

```
          allocate(CVmix_vars%ShearRichardson_iface(nlev+1))
        end if
        CVmix_vars%ShearRichardson_iface(:) = val
      case ("SqrBuoyancyFreq_iface")
!        print*, "WARNING: you are setting the buoyancy in all levels to a ",  &
!                "constant value"
        if (.not.associated(CVmix_vars%SqrBuoyancyFreq_iface)) then
          allocate(CVmix_vars%SqrBuoyancyFreq_iface(nlev+1))
        end if
        CVmix_vars%SqrBuoyancyFreq_iface(:) = val
      case ("kpp_nonlocal_iface")
        if (.not.associated(CVmix_vars%kpp_Tnonlocal_iface)) then
          allocate(CVmix_vars%kpp_Tnonlocal_iface(nlev+1))
        end if
        if (.not.associated(CVmix_vars%kpp_Snonlocal_iface)) then
          allocate(CVmix_vars%kpp_Snonlocal_iface(nlev+1))
        end if
        CVmix_vars%kpp_Tnonlocal_iface(:) = val
        CVmix_vars%kpp_Snonlocal_iface(:) = val
      case ("kpp_Tnonlocal_iface")
        if (.not.associated(CVmix_vars%kpp_Tnonlocal_iface)) then
          allocate(CVmix_vars%kpp_Tnonlocal_iface(nlev+1))
        end if
        CVmix_vars%kpp_Tnonlocal_iface(:) = val
      case ("kpp_Snonlocal_iface")
        if (.not.associated(CVmix_vars%kpp_Snonlocal_iface)) then
          allocate(CVmix_vars%kpp_Snonlocal_iface(nlev+1))
        end if
        CVmix_vars%kpp_Snonlocal_iface(:) = val

      case ("dzt")
!        print*, "WARNING: you are setting the cell thickness in all levels ", &
!                "to a constant value"
        if (.not.associated(CVmix_vars%dzt)) then
          allocate(CVmix_vars%dzt(nlev))
        end if
        CVmix_vars%dzt(:) = val
      case ("WaterDensity_cntr")
!        print*, "WARNING: you are setting the density in all levels to a ",   &
!                "constant value"
        if (.not.associated(CVmix_vars%WaterDensity_cntr)) then
          allocate(CVmix_vars%WaterDensity_cntr(nlev))
        end if
        CVmix_vars%WaterDensity_cntr(:) = val
      case ("AdiabWaterDensity_cntr")
!        print*, "WARNING: you are setting the adiabatic density in all ",     &
!                "levels to a constant value"
        if (.not.associated(CVmix_vars%AdiabWaterDensity_cntr)) then
```

```
          allocate(CVmix_vars%AdiabWaterDensity_cntr(nlev))
        end if
        CVmix_vars%AdiabWaterDensity_cntr(:) = val
      case ("BulkRichardson_cntr")
!        print*, "WARNING: you are setting the bulk Richardson number in all", &
!                " levels to a constant value"
        if (.not.associated(CVmix_vars%BulkRichardson_cntr)) then
          allocate(CVmix_vars%BulkRichardson_cntr(nlev))
        end if
        CVmix_vars%BulkRichardson_cntr(:) = val
      case ('strat_param_num')
!        print*, "WARNING: you are setting the numerator of the ",             &
!                "stratification parameter in all levels to a constant value"
        if (.not.associated(CVmix_vars%strat_param_num)) then
          allocate(CVmix_vars%strat_param_num(nlev))
        end if
        CVmix_vars%strat_param_num(:) = val
      case ('strat_param_denom')
!        print*, "WARNING: you are setting the denominator of the ",           &
!                "stratification parameter in all levels to a constant value"
        if (.not.associated(CVmix_vars%strat_param_denom)) then
          allocate(CVmix_vars%strat_param_denom(nlev))
        end if
        CVmix_vars%strat_param_denom(:) = val
      case ("VertDep_iface")
        if (.not.associated(CVmix_vars%VertDep_iface)) then
          allocate(CVmix_vars%VertDep_iface(nlev+1))
        end if
        CVmix_vars%VertDep_iface(:) = val

      case default
        print*, "ERROR: ", trim(varname), " not a valid choice for cvmix_put_real!"
        stop 1

    end select
```

1.95 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
! Local variables
integer :: nlev

if (present(nlev_in)) then
  nlev = nlev_in
else
  nlev = CVmix_vars%max_nlev
end if

if (nlev.eq.-1) then
  print*, "ERROR: you must specify the number of levels before ", &
          "you can pack data into a cvmix_data_type!"
  print*, "You tried to set ", trim(varname)
  stop 1
end if

select case (trim(cvmix_att_name(varname)))
  case ("zw_iface")
    if (.not.associated(CVmix_vars%zw_iface)) then
      allocate(CVmix_vars%zw_iface(nlev+1))
    end if
    CVmix_vars%zw_iface(:) = val
  case ("dzw")
    if (.not.associated(CVmix_vars%dzw)) then
      allocate(CVmix_vars%dzw(nlev+1))
    end if
    CVmix_vars%dzw(:) = val
  case ("Mdiff_iface")
    if (.not.associated(CVmix_vars%Mdiff_iface)) then
```

```
    allocate(CVmix_vars%Mdiff_iface(nlev+1))
  end if
  CVmix_vars%Mdiff_iface(:) = val
case ("Tdiff_iface")
  if (.not.associated(CVmix_vars%Tdiff_iface)) then
    allocate(CVmix_vars%Tdiff_iface(nlev+1))
  end if
  CVmix_vars%Tdiff_iface(:) = val
case ("Sdiff_iface")
  if (.not.associated(CVmix_vars%Sdiff_iface)) then
    allocate(CVmix_vars%Sdiff_iface(nlev+1))
  end if
  CVmix_vars%Sdiff_iface(:) = val
case ("ShearRichardson_iface")
  if (.not.associated(CVmix_vars%ShearRichardson_iface)) then
    allocate(CVmix_vars%ShearRichardson_iface(nlev+1))
  end if
  CVmix_vars%ShearRichardson_iface(:) = val
case ("SqrBuoyancyFreq_iface")
  if (.not.associated(CVmix_vars%SqrBuoyancyFreq_iface)) then
    allocate(CVmix_vars%SqrBuoyancyFreq_iface(nlev+1))
  end if
  CVmix_vars%SqrBuoyancyFreq_iface(:) = val
case ("kpp_nonlocal_iface")
  if (.not.associated(CVmix_vars%kpp_Tnonlocal_iface)) then
    allocate(CVmix_vars%kpp_Tnonlocal_iface(nlev+1))
  end if
  if (.not.associated(CVmix_vars%kpp_Snonlocal_iface)) then
    allocate(CVmix_vars%kpp_Snonlocal_iface(nlev+1))
  end if
  CVmix_vars%kpp_Tnonlocal_iface(:) = val
  CVmix_vars%kpp_Snonlocal_iface(:) = val
case ("kpp_Tnonlocal_iface")
  if (.not.associated(CVmix_vars%kpp_Tnonlocal_iface)) then
    allocate(CVmix_vars%kpp_Tnonlocal_iface(nlev+1))
  end if
  CVmix_vars%kpp_Tnonlocal_iface(:) = val
case ("kpp_Snonlocal_iface")
  if (.not.associated(CVmix_vars%kpp_Snonlocal_iface)) then
    allocate(CVmix_vars%kpp_Snonlocal_iface(nlev+1))
  end if
  CVmix_vars%kpp_Snonlocal_iface(:) = val
case ("VertDep_iface")
  if (.not.associated(CVmix_vars%VertDep_iface)) then
    allocate(CVmix_vars%VertDep_iface(nlev+1))
  end if
  CVmix_vars%VertDep_iface(:) = val
case ("zt_cntr")
```

```
  if (.not.associated(CVmix_vars%zt_cntr)) then
    allocate(CVmix_vars%zt_cntr(nlev))
  end if
  CVmix_vars%zt_cntr(:) = val
case ("dzt")
  if (.not.associated(CVmix_vars%dzt)) then
    allocate(CVmix_vars%dzt(nlev))
  end if
  CVmix_vars%dzt(:) = val
case ("WaterDensity_cntr")
  if (.not.associated(CVmix_vars%WaterDensity_cntr)) then
    allocate(CVmix_vars%WaterDensity_cntr(nlev))
  end if
  CVmix_vars%WaterDensity_cntr(:) = val
case ("AdiabWaterDensity_cntr")
  if (.not.associated(CVmix_vars%AdiabWaterDensity_cntr)) then
    allocate(CVmix_vars%AdiabWaterDensity_cntr(nlev))
  end if
  CVmix_vars%AdiabWaterDensity_cntr(:) = val
case ("BulkRichardson_cntr")
  if (.not.associated(CVmix_vars%BulkRichardson_cntr)) then
    allocate(CVmix_vars%BulkRichardson_cntr(nlev))
  end if
  CVmix_vars%BulkRichardson_cntr(:) = val
case ('strat_param_num')
  if (.not.associated(CVmix_vars%strat_param_num)) then
    allocate(CVmix_vars%strat_param_num(nlev))
  end if
  CVmix_vars%strat_param_num(:) = val
case ('strat_param_denom')
  if (.not.associated(CVmix_vars%strat_param_denom)) then
    allocate(CVmix_vars%strat_param_denom(nlev))
  end if
  CVmix_vars%strat_param_denom(:) = val
case ("Vx_cntr")
  if (.not.associated(CVmix_vars%Vx_cntr)) then
    allocate(CVmix_vars%Vx_cntr(nlev))
  end if
  CVmix_vars%Vx_cntr(:) = val
case ("Vy_cntr")
  if (.not.associated(CVmix_vars%Vy_cntr)) then
    allocate(CVmix_vars%Vy_cntr(nlev))
  end if
  CVmix_vars%Vy_cntr(:) = val
case ("SchmittnerSouthernOcean")
  if (.not.associated(CVmix_vars%SchmittnerSouthernOcean)) then
    allocate(CVmix_vars%SchmittnerSouthernOcean(CVmix_vars%max_nlev+1))
  end if
```

```
      CVmix_vars%SchmittnerSouthernOcean(:) = val
    case ("SchmittnerCoeff")
      if (.not.associated(CVmix_vars%SchmittnerCoeff)) then
        allocate(CVmix_vars%SchmittnerCoeff(CVmix_vars%max_nlev+1))
      end if
      CVmix_vars%SchmittnerCoeff(:) = val

    case default
      print*, "ERROR: ", trim(varname), " not a valid choice for cvmix_put_real_1D!"
      stop 1

  end select
```

1.96 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
! Local variables
integer :: nlev
```

```
if (present(nlev_in)) then
  nlev = nlev_in
else
  nlev = CVmix_vars%max_nlev
end if

if (nlev.eq.-1) then
  print*, "ERROR: you must specify the number of levels before ", &
          "you can pack data into a cvmix_data_type!"
  print*, "You tried to set ", trim(varname)
  stop 1
end if

select case (trim(cvmix_att_name(varname)))
  case ("exp_hab_zetar")
    if (.not.associated(CVmix_vars%exp_hab_zetar)) then
      allocate(CVmix_vars%exp_hab_zetar(CVmix_vars%nlev+1,CVmix_vars%nlev+1))
    end if
    CVmix_vars%exp_hab_zetar = val


  case default
    print*, "ERROR: ", trim(varname), " not a valid choice for cvmix_put_real_2D!"
    stop 1

end select
```

1.97 cvmix_put_global_params_int

INTERFACE:

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

DESCRIPTION:

Write an integer value into a cvmix_global_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
select case (trim(varname))
  case ('max_nlev')
    CVmix_params%max_nlev = val

  case default
    print*, "ERROR: ", trim(varname), " not a valid choice for cvmix_put_global_params_
    stop 1

end select
```

1.98 cvmix_put_global_params_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_global_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
select case (trim(varname))
  case ('prandtl','Prandtl')
    CVmix_params%prandtl = val
  case ('fw_rho','FreshWaterDensity')
    CVmix_params%FreshWaterDensity = val
  case ('sw_rho','SaltWaterDensity')
    CVmix_params%SaltWaterDensity = val
  case ('g','Gravity')
    CVmix_params%Gravity = val
  case default
    print*, "ERROR: ", trim(varname), " not a valid choice for cvmix_put_global_params_r
    stop 1

end select
```

1.99 Fortran: Module Interface cvmix_utils (Source File: cvmix_utils.F90)

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
```

1.100 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)

USES:

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

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
```

```
integer :: kw

select case (old_vals)
  case (CVMIX_SUM_OLD_AND_NEW_VALS)
    if ((present(Mdiff_out)).and.(present(new_Mdiff))) &
      Mdiff_out = Mdiff_out + new_Mdiff
    if ((present(Tdiff_out)).and.(present(new_Tdiff))) &
      Tdiff_out = Tdiff_out + new_Tdiff
    if ((present(Sdiff_out)).and.(present(new_Sdiff))) &
      Sdiff_out = Sdiff_out + new_Sdiff
  case (CVMIX_MAX_OLD_AND_NEW_VALS)
    do kw=1,nlev+1
      if ((present(Mdiff_out)).and.(present(new_Mdiff))) &
        Mdiff_out(kw) = max(Mdiff_out(kw), new_Mdiff(kw))
      if ((present(Tdiff_out)).and.(present(new_Tdiff))) &
        Tdiff_out(kw) = max(Tdiff_out(kw), new_Tdiff(kw))
      if ((present(Sdiff_out)).and.(present(new_Sdiff))) &
        Sdiff_out(kw) = max(Sdiff_out(kw), new_Sdiff(kw))
    end do
  case (CVMIX_OVERWRITE_OLD_VAL)
    if ((present(Mdiff_out)).and.(present(new_Mdiff))) &
      Mdiff_out = new_Mdiff
    if ((present(Tdiff_out)).and.(present(new_Tdiff))) &
      Tdiff_out = new_Tdiff
    if ((present(Sdiff_out)).and.(present(new_Sdiff))) &
      Sdiff_out = new_Sdiff
  case DEFAULT
    print*, "ERROR: do not know how to handle old values!"
    stop 1
end select
```

1.101 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.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

```
select case(trim(varname))
  ! Scalars
  case ("nlev", "NumberLevels", "NumberOfLevels")
    cvmix_att_name = "nlev"
  case ("max_nlev", "MaxNumberLevels", "MaxNumberOfLevels")
    cvmix_att_name = "max_nlev"
  case ("depth", "ocn_depth", "OceanDepth", "DepthOfOcean")
    cvmix_att_name = "OceanDepth"
  case ('BoundaryLayerDepth','OBL_depth')
    cvmix_att_name = "BoundaryLayerDepth"
  case ("SSH", "surf_hgt", "SeaSurfaceHeight", "SurfaceHeight", "height")
    cvmix_att_name = "SeaSurfaceHeight"
  case ("surf_fric", "SurfaceFriction")
    cvmix_att_name = "SurfaceFriction"
  case ("surf_buoy", "SurfaceBuoyancy", "SurfaceBuoyancyForcing")
    cvmix_att_name = "SurfaceBuoyancyForcing"
  case ("lat", "latitude", "Latitude")
    cvmix_att_name = "Latitude"
  case ("lon", "longitude", "Longitude")
    cvmix_att_name = "Longitude"
```

```
case ("coriolis", "Coriolis", "CoriolisFreq", "CoriolisFrequency")
  cvmix_att_name = "Coriolis"
case ("kOBL_depth", "BoundaryLayerDepthIndex")
  cvmix_att_name = "kOBL_depth"
case ("LangmuirEnhancementFactor", "EnhancementFactor",   &
       "langmuir_Efactor")
  cvmix_att_name = "LangmuirEnhancementFactor"
case ("LangmuirNumber", "La")
  cvmix_att_name = "LangmuirNumber"
case ("ltidal_Schmittner_socn")
  cvmix_att_name = "UseSchmittnerSouthernOceanMods"
case ("ltidal_max")
  cvmix_att_name = "ApplyTidalMixingCap"

! Variables on level interfaces
case ("zw", "zw_iface")
  cvmix_att_name = "zw_iface"
case ("dzw", "dzw_iface")
  cvmix_att_name = "dzw"
case ("Mdiff", "Udiff", "MomentumDiff", "MomentumDiffusivity")
  cvmix_att_name = "Mdiff_iface"
case ("Tdiff", "TempDiff", "TemperatureDiff", "TemperatureDiffusivity")
  cvmix_att_name = "Tdiff_iface"
case ("Sdiff", "SaltDiff", "SalinityDiff", "SalinityDiffusivity")
  cvmix_att_name = "Sdiff_iface"
case ("Ri", "Ri_iface", "Richardson", "ShearRichardson",                &
      "RichardsonNumber", "ShearRichardsonNumber",                      &
      "ShearRichardson_iface")
  cvmix_att_name = "ShearRichardson_iface"
case ("buoy", "buoy_iface", "N", "Nsqr", "BuoyancyFreq", "SqrBuoyancy", &
      "SqrBuoyancyFreq", "SqrBuoyancyFreq_iface")
  cvmix_att_name = "SqrBuoyancyFreq_iface"
case ("kpp_transport", "kpp_nonlocal", "nonlocal_transport",            &
      "nonlocal", "kpp_nonlocal_iface")
  ! Note: this isn't an attribute in the data type, but put / get
  !       uses this as short hand for "both Tnonlocal and Snonlocal"
  cvmix_att_name = "kpp_nonlocal_iface"
case ("Tnonlocal", "KPP_T_Nonlocal", "kpp_Tnonlocal", "kpp_Ttransport", &
      "kpp_Tnonlocal_iface")
  cvmix_att_name = "kpp_Tnonlocal_iface"
case ("Snonlocal", "KPP_S_Nonlocal", "kpp_Snonlocal", "kpp_Stransport", &
      "kpp_Snonlocal_iface")
  cvmix_att_name = "kpp_Snonlocal_iface"

! Variables on level centers
case ("z","zt","zt_cntr")
  cvmix_att_name = "zt_cntr"
case ("dz", "dzt", "CellThickness")
```

```
      cvmix_att_name = "dzt"
    case ("rho", "dens", "WaterDensity", "WaterDensity_cntr")
      cvmix_att_name = "WaterDensity_cntr"
    case ("rho_lwr", "dens_lwr", "AdiabWaterDensity",                        &
          "AdiabWaterDensity_cntr")
      cvmix_att_name = "AdiabWaterDensity_cntr"
    case ("Rib", "Ri_bulk", "BulkRichardson", "BulkRichardsonNumber",        &
          "BulkRichardson_cntr")
      cvmix_att_name = "BulkRichardson_cntr"
    case ("Rrho", "strat_param")
      ! Note: this isn't an attribute in the data type, but the I/O routines
      !       use it to denote strat_param_num / strat_param_denom
      cvmix_att_name = "strat_param"
    case ("Rrho_num", "strat_param_num")
      cvmix_att_name = "strat_param_num"
    case ("Rrho_denom", "strat_param_denom")
      cvmix_att_name = "strat_param_denom"
    case ("Buoyancy","buoyancy","buoyancy_cntr")
      cvmix_att_name = "buoyancy_cntr"
    case ("U", "Vx", "Vx_cntr")
      cvmix_att_name = "Vx_cntr"
    case ("V", "Vy", "Vy_cntr")
      cvmix_att_name = "Vy_cntr"
    case ("SimmonsCoeff", "TidalCoeff")
      cvmix_att_name = "SimmonsCoeff"
    case ("SchmittnerCoeff")
      cvmix_att_name = "SchmittnerCoeff"
    case ("SchmittnerSouthernOcean")
      cvmix_att_name = "SchmittnerSouthernOcean"
    case ("exp_hab_zetar")
      cvmix_att_name = "exp_hab_zetar"
    case ("VertDep", "VertDep_iface", "vert_dep")
      cvmix_att_name = "VertDep_iface"
    case DEFAULT
      print*, "ERROR: ", trim(varname), " is not tied to an attribute of ", &
              "the cvmix_data_type structure."
      stop 1
  end select
```

1.102 Fortran: Module Interface Main Program (Stand-Alone) (Source File: cvmix_driver.F90)

1.103 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:

```
Program cvmix_driver
```

USES:

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


  integer :: nlev, max_nlev
  real(kind=cvmix_r8) :: ocn_depth
  character(len=cvmix_strlen) :: mix_type

  namelist/cvmix_nml/mix_type, nlev, max_nlev, ocn_depth

  mix_type = 'unset'
  nlev = 0
  max_nlev = 0
  ocn_depth = cvmix_zero
  read(*, nml=cvmix_nml)
  if (max_nlev.eq.0) then
    max_nlev = nlev
  end if

  select case (trim(mix_type))
    case ('BryanLewis')
      call cvmix_BL_driver(nlev, max_nlev, ocn_depth)
    case ('shear')
      call cvmix_shear_driver(nlev, max_nlev)
    case ('tidal')
      call cvmix_tidal_driver()
    case ('ddiff')
      call cvmix_ddiff_driver(2*nlev, 2*max_nlev)
    case ('kpp')
      call cvmix_kpp_driver()
    case DEFAULT
```

```
      print*, "WARNING: mix_type = '", trim(mix_type), "' is not supported by this driver."
  end select

End Program cvmix_driver
```

1.104 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



 ! Global parameter
 integer, parameter :: ncol = 2

 ! CVMix datatypes
 type(cvmix_data_type)         , dimension(ncol) :: CVmix_vars_pointer,      &
                                                    CVmix_vars_memcopy
 type(cvmix_global_params_type)                  :: CVmix_params
 ! Column 1 uses the params saved in module, Column 2 uses this one
```

```
  type(cvmix_bkgnd_params_type)                :: CVmix_BL_params

  ! Will use 2 columns, diffusivities be 2 x nlev+1
  ! iface_depth is the depth of each interface;  same in both columns
  real(cvmix_r8), dimension(:),   allocatable, target :: iface_depth
  real(cvmix_r8), dimension(:,:), allocatable, target :: Mdiff, Tdiff

  ! Namelist variables
  ! Bryan-Lewis mixing parameters for column 1
  real(cvmix_r8) :: col1_vdc1, col1_vdc2, col1_linv, col1_dpth
  ! Bryan-Lewis mixing parameters for column 2
  real(cvmix_r8) :: col2_vdc1, col2_vdc2, col2_linv, col2_dpth

  ! array indices
  integer :: i, icol,kw
  ! file indices
#ifdef _NETCDF
  integer, dimension(2) :: fid
#else
  integer, dimension(6) :: fid
#endif

  ! Namelists that may be read in, depending on desired mixing scheme
  namelist/BryanLewis1_nml/col1_vdc1, col1_vdc2, col1_linv, col1_dpth
  namelist/BryanLewis2_nml/col2_vdc1, col2_vdc2, col2_linv, col2_dpth

  print*, "Active levels: ", nlev
  print*, "Levels allocated in memory: ", max_nlev

  ! Calculate depth of cell interfaces based on number of levels and ocean
  ! depth (also allocate memory for diffusivity and viscosity)
  allocate(iface_depth(max_nlev+1))
  iface_depth(1) = cvmix_zero

  ! Depth is 0 at sea level and negative at ocean bottom in CVMix
  do kw = 2,max_nlev+1
    iface_depth(kw) = iface_depth(kw-1) - ocn_depth/real(nlev,cvmix_r8)
  end do

  ! Allocate memory to store viscosity and diffusivity values (for pointer)
  allocate(Mdiff(ncol,max_nlev+1), Tdiff(ncol,max_nlev+1))

  ! Initialization for CVMix data types
  call cvmix_put(CVmix_params,  'max_nlev', max_nlev)
  call cvmix_put(CVmix_params,  'prandtl',  cvmix_zero)
  do icol=1,ncol
    CVmix_vars_pointer(icol)%nlev=nlev
    CVmix_vars_pointer(icol)%max_nlev=max_nlev
```

```
    ! Point CVmix_vars_pointer values to memory allocated above
    CVmix_vars_pointer(icol)%Mdiff_iface => Mdiff(icol,:)
    CVmix_vars_pointer(icol)%Tdiff_iface => Tdiff(icol,:)
    CVmix_vars_pointer(icol)%zw_iface    => iface_depth

    ! Copy values into CVmix_vars_memcopy
    call cvmix_put(CVmix_vars_memcopy(icol), 'nlev',     nlev)
    call cvmix_put(CVmix_vars_memcopy(icol), 'max_nlev', max_nlev)
    call cvmix_put(CVmix_vars_memcopy(icol), 'Mdiff',    cvmix_zero)
    call cvmix_put(CVmix_vars_memcopy(icol), 'Tdiff',    cvmix_zero)
    call cvmix_put(CVmix_vars_memcopy(icol), 'zw_iface', iface_depth)
  end do

  ! Read B-L parameters for columns
  read(*, nml=BryanLewis1_nml)
  read(*, nml=BryanLewis2_nml)

  ! Two different columns are tested with each of two different memory options:
  ! Column 1 uses the internal background parameter dataset while column 2 uses
  ! CVmix_BL_params; for the memory copy, column 1 tests the routine
  ! cvmix_get_bkgnd_real_2D() rather than cvmix_coeffs_bkgnd.

  ! Pointer test
  call cvmix_init_bkgnd(CVmix_vars_pointer(1), col1_vdc1, col1_vdc2,          &
                        col1_linv, col1_dpth, CVmix_params)
  call cvmix_init_bkgnd(CVmix_vars_pointer(2), col2_vdc1, col2_vdc2,          &
                        col2_linv, col2_dpth, CVMix_params,                   &
                        CVmix_bkgnd_params_user=CVmix_BL_params)
  call cvmix_coeffs_bkgnd(CVmix_vars_pointer(1))
  call cvmix_coeffs_bkgnd(CVmix_vars_pointer(2),                              &
                          CVmix_bkgnd_params_user=CVmix_BL_params)

  ! Memcopy test
  call cvmix_init_bkgnd(CVmix_vars_memcopy(1), col1_vdc1, col1_vdc2,          &
                        col1_linv, col1_dpth, CVmix_params)
  call cvmix_init_bkgnd(CVmix_vars_memcopy(2), col2_vdc1, col2_vdc2,          &
                        col2_linv, col2_dpth, CVMix_params,                   &
                        CVmix_bkgnd_params_user=CVmix_BL_params)
  CVmix_vars_memcopy(1)%Mdiff_iface = reshape(                                &
                            cvmix_get_bkgnd_real_2D('static_Mdiff'),(/nlev+1/))
  CVmix_vars_memcopy(1)%Tdiff_iface = reshape(                                &
                            cvmix_get_bkgnd_real_2D('static_Tdiff'),(/nlev+1/))
  call cvmix_coeffs_bkgnd(CVmix_vars_memcopy(2),                              &
                          CVmix_bkgnd_params_user=CVmix_BL_params)

  ! Output
#ifdef _NETCDF
  ! data will have diffusivity from both columns (needed for NCL script)
```

```
  call cvmix_io_open(fid(1), "data_pointer.nc", "nc")
  call cvmix_io_open(fid(2), "data_memcopy.nc", "nc")
  ! Note: all entries in string of variables to output must be
  !       the same length... hence the space in "Tdiff"
  call cvmix_output_write(fid(1), CVmix_vars_pointer,                          &
                          (/"zw_iface", "Tdiff   "/))
  call cvmix_output_write(fid(2), CVmix_vars_memcopy,                          &
                          (/"zw_iface", "Tdiff   "/))
  do i=1,2
    call cvmix_output_write_att(fid(i), "long_name", "tracer diffusivity",     &
                                var_name="Tdiff")
    call cvmix_output_write_att(fid(i), "units", "m^2/s", var_name="Tdiff")
    call cvmix_output_write_att(fid(i), "long_name", "depth to interface",     &
                                var_name="zw")
    call cvmix_output_write_att(fid(i), "positive", "up", var_name="zw")
    call cvmix_output_write_att(fid(i), "units", "m", var_name="zw")
    call cvmix_io_close(fid(i))
  end do
#else
  ! data will have diffusivity from both columns (needed for NCL script)
  call cvmix_io_open(fid(1), "data_pointer.out", "ascii")
  call cvmix_io_open(fid(2), "data_memcopy.out", "ascii")
  ! col1 will just have diffusivity from low lat
  call cvmix_io_open(fid(3), "col1_pointer.out", "ascii")
  call cvmix_io_open(fid(4), "col1_memcopy.out", "ascii")
  ! col2 will just have diffusivity from high lat
  call cvmix_io_open(fid(5), "col2_pointer.out", "ascii")
  call cvmix_io_open(fid(6), "col2_memcopy.out", "ascii")

  ! Note: all entries in string of variables to output must be
  !       the same length... hence the space in "Tdiff"
  call cvmix_output_write(fid(1), CVmix_vars_pointer,                          &
                          (/"zw_iface", "Tdiff   "/))
  call cvmix_output_write(fid(2), CVmix_vars_memcopy,                          &
                          (/"zw_iface", "Tdiff   "/))
  call cvmix_output_write(fid(3), CVmix_vars_pointer(1),                       &
                          (/"zw_iface", "Tdiff   "/))
  call cvmix_output_write(fid(4), CVmix_vars_memcopy(1),                       &
                          (/"zw_iface", "Tdiff   "/))
  call cvmix_output_write(fid(5), CVmix_vars_pointer(2),                       &
                          (/"zw_iface", "Tdiff   "/))
  call cvmix_output_write(fid(6), CVmix_vars_memcopy(2),                       &
                          (/"zw_iface", "Tdiff   "/))

  do i=1,6
    call cvmix_io_close(fid(i))
  end do
#endif
```

1.105 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



 ! Global parameter
 integer, parameter :: ncol = 6

 ! CVMix datatypes
 type(cvmix_data_type)                  :: CVmix_vars_LMD_1D, CVmix_vars_PP_1D
 type(cvmix_data_type), dimension(ncol) :: CVmix_vars_PP_2D

 ! All regression tests look at Richardson numbers in [0,1]
 real(cvmix_r8), dimension(:), allocatable, target :: Ri_g
```

```
! "1D" variables will be 2 x nlev+1 (1 for LMD, 1 for PP)
! "2D" variables will be ncol x nlev+1 (for PP)
real(cvmix_r8), dimension(:,:), allocatable, target :: Mdiff_1D, Tdiff_1D
real(cvmix_r8), dimension(:,:), allocatable, target :: Mdiff_2D, Tdiff_2D

! Hard-code in parameters for each c in Table 1
real(cvmix_r8), dimension(ncol) :: PP_nu_zero_2D, PP_exp_2D, PP_alpha_2D
real(cvmix_r8)                  :: PP_nu_zero, PP_exp, PP_alpha

integer :: icol, kw, fid

! Namelist variables
! KPP mixing parameters for column
real(cvmix_r8) :: LMD_nu_zero, LMD_Ri_zero, LMD_exp

namelist/LMD_nml/LMD_nu_zero, LMD_Ri_zero, LMD_exp
namelist/PP_nml/PP_nu_zero, PP_alpha, PP_exp

print*, "Active levels: ", nlev
print*, "Levels allocated in memory: ", max_nlev

! Namelist (set defaults then read from file)
LMD_nu_zero = 5e-3_cvmix_r8
LMD_Ri_zero = 0.7_cvmix_r8
LMD_exp     = real(3,  cvmix_r8)

PP_nu_zero = 5e-3_cvmix_r8
PP_alpha   = real(5, cvmix_r8)
PP_exp     = real(2, cvmix_r8)

read(*, nml=LMD_nml)
read(*, nml=PP_nml)

print*, ""
print*, "Parameters Used in LMD test"
print*, "----"
print*, "KPP_nu_zero = ", LMD_nu_zero
print*, "KPP_Ri_zero = ", LMD_Ri_zero
print*, "KPP_exp = ", LMD_exp

print*, ""
print*, "Parameters Used in PP test"
print*, "----"
print*, "PP_nu_zero = ", PP_nu_zero
print*, "PP_alpha = ", PP_alpha
print*, "PP_exp = ", PP_exp
```

```
! Ri_g should increase from 0 to 1 in active portion of level
allocate(Ri_g(max_nlev+1))
Ri_g(1) = cvmix_zero
do kw = 2,max_nlev+1
  Ri_g(kw) = Ri_g(kw-1) + cvmix_one/real(nlev,cvmix_r8)
end do

! Allocate memory to store viscosity and diffusivity values
allocate(Mdiff_1D(2,max_nlev+1), Tdiff_1D(2,max_nlev+1))
allocate(Mdiff_2D(ncol,max_nlev+1), Tdiff_2D(ncol,max_nlev+1))

! Set Pacanowski-Philander coefficients
! (See Table 1 from paper, converted from cm^2/s to m^2/s)
PP_nu_zero_2D  = 0.001_cvmix_r8 * real((/2, 5, 10, 10, 15, 15/), cvmix_r8)
PP_exp_2D(:)   = real(2,cvmix_r8)
PP_alpha_2D(:) = real(5,cvmix_r8)
PP_exp_2D(4)   = real(1,cvmix_r8)
PP_alpha_2D(6) = real(10,cvmix_r8)

! Initialization for LMD test
call cvmix_put(CVmix_vars_LMD_1D, 'nlev',     nlev)
call cvmix_put(CVmix_vars_LMD_1D, 'max_nlev', max_nlev)
! Point CVmix_vars values to memory allocated above
CVmix_vars_LMD_1D%Mdiff_iface           => Mdiff_1D(1,:)
CVmix_vars_LMD_1D%Tdiff_iface           => Tdiff_1D(1,:)
CVmix_vars_LMD_1D%ShearRichardson_iface => Ri_g

! Initialization for 1D PP test
call cvmix_put(CVmix_vars_PP_1D, 'nlev',     nlev)
call cvmix_put(CVmix_vars_PP_1D, 'max_nlev', max_nlev)
! Point CVmix_vars values to memory allocated above
CVmix_vars_PP_1D%Mdiff_iface           => Mdiff_1D(2,:)
CVmix_vars_PP_1D%Tdiff_iface           => Tdiff_1D(2,:)
CVmix_vars_PP_1D%ShearRichardson_iface => Ri_g

! Initialization for 2D PP test
do icol=1,ncol
  call cvmix_put(CVmix_vars_PP_2D(icol), 'nlev',     nlev)
  call cvmix_put(CVmix_vars_PP_2D(icol), 'max_nlev', max_nlev)
  CVmix_vars_PP_2D(icol)%Mdiff_iface           => Mdiff_2D(icol,:)
  CVmix_vars_PP_2D(icol)%Tdiff_iface           => Tdiff_2D(icol,:)
  CVmix_vars_PP_2D(icol)%ShearRichardson_iface => Ri_g
end do

! Set LMD94 parameters
call cvmix_init_shear(mix_scheme='KPP', KPP_nu_zero=LMD_nu_zero,           &
                      KPP_Ri_zero=LMD_Ri_zero, KPP_exp=LMD_exp)
call cvmix_coeffs_shear(CVmix_vars_LMD_1D)
```

```
  ! Set PP81 single column parameters
  call cvmix_init_shear(mix_scheme='PP', PP_nu_zero=PP_nu_zero,               &
                        PP_alpha=PP_alpha, PP_exp=PP_exp)
  call cvmix_coeffs_shear(CVmix_vars_PP_1D)

  ! Set PP81 multiple column
  do icol=1,ncol
    call cvmix_init_shear(mix_scheme='PP', PP_nu_zero=PP_nu_zero_2D(icol),    &
                          PP_alpha=PP_alpha_2D(icol), PP_exp=PP_exp_2D(icol))
    call cvmix_coeffs_shear(CVmix_vars_PP_2D(icol))
  end do

  ! Output
  ! (1) LMD column
#ifdef _NETCDF
  call cvmix_io_open(fid, "data_LMD.nc", "nc")
#else
  call cvmix_io_open(fid, "data_LMD.out", "ascii")
#endif

  call cvmix_output_write(fid, CVmix_vars_LMD_1D, (/"Ri   ", "Tdiff"/))
#ifdef _NETCDF
  call cvmix_output_write_att(fid, "long_name", "Richardson number",          &
                              var_name="ShearRichardson")
  call cvmix_output_write_att(fid, "units", "unitless",                       &
                              var_name="ShearRichardson")
  call cvmix_output_write_att(fid, "long_name", "temperature diffusivity",    &
                              var_name="Tdiff")
  call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Tdiff")
#endif
  call cvmix_io_close(fid)

  ! (2) PP single column
#ifdef _NETCDF
  call cvmix_io_open(fid, "data_PP1d.nc", "nc")
#else
  call cvmix_io_open(fid, "data_PP1d.out", "ascii")
#endif

  call cvmix_output_write(fid, CVmix_vars_PP_1D, (/"Ri   ", "Mdiff"/))
#ifdef _NETCDF
  call cvmix_output_write_att(fid, "long_name", "Richardson number",          &
                              var_name="ShearRichardson")
  call cvmix_output_write_att(fid, "units", "unitless",                       &
                              var_name="ShearRichardson")
  call cvmix_output_write_att(fid, "long_name", "momentum diffusivity",       &
                              var_name="Mdiff")
```

```
  call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Mdiff")
#endif
  call cvmix_io_close(fid)

  ! (3) PP multiple columns
#ifdef _NETCDF
  call cvmix_io_open(fid, "data_PP2d.nc", "nc")
#else
  call cvmix_io_open(fid, "data_PP2d.out", "ascii")
#endif
  call cvmix_output_write(fid, CVmix_vars_PP_2D, (/"Ri   ", "Mdiff"/))
#ifdef _NETCDF
  call cvmix_output_write_att(fid, "long_name", "Richardson number",          &
                              var_name="ShearRichardson")
  call cvmix_output_write_att(fid, "units", "unitless",                       &
                              var_name="ShearRichardson")
  call cvmix_output_write_att(fid, "long_name", "momentum diffusivity",       &
                              var_name="Mdiff")
  call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Mdiff")
#endif
  call cvmix_io_close(fid)
```

1.106 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



 ! CVMix datatypes
 type(cvmix_data_type), dimension(:,:), allocatable :: CVmix_vars
 type(cvmix_global_params_type) :: CVmix_params
 type(cvmix_tidal_params_type)  :: CVmix_Simmons_params

 real(cvmix_r8), dimension(:,:,:), allocatable, target :: Mdiff, Tdiff

 ! file index
 integer :: fid

 ! Namelist variables
 character(len=cvmix_strlen) :: grid_file, physics_file, energy_flux_file,   &
```

```
                                energy_flux_var
  integer :: lon_out, lat_out

  ! Local variables
  real(cvmix_r8), dimension(:,:,:), allocatable :: buoy
  real(cvmix_r8), dimension(:,:),   allocatable :: ocn_depth, energy_flux,    &
                                                   lat, lon
  integer,        dimension(:,:),   allocatable :: ocn_levels
  real(cvmix_r8), dimension(:),     allocatable :: zw_iface, zt
  real(cvmix_r8)                                :: FillVal, this_lon, this_lat
  character(len=cvmix_strlen) :: lonstr, latstr
  integer :: i, j, k, nlon, nlat, nlev, max_nlev

  ! Namelists that may be read in, depending on desired mixing scheme
  namelist/Simmons_nml/grid_file, physics_file, energy_flux_file,             &
                       energy_flux_var, lon_out, lat_out

  ! Read namelist variables
  grid_file = "none"
  physics_file = "none"
  energy_flux_file = "none"
  energy_flux_var = "none"
  lon_out = 35
  lat_out = 345
  read(*, nml=Simmons_nml)

  ! Get dimensions from grid file
  ! Initialize all values to -1 before reading (avoid a warning when
  ! compiling without netCDF)
  nlon = -1
  nlat = -1
  max_nlev = -1
  call cvmix_io_open(fid, trim(grid_file), 'nc', read_only=.true.)
#ifdef _NETCDF
  nlon = cvmix_input_get_netcdf_dim(fid, 'lon')
  nlat = cvmix_input_get_netcdf_dim(fid, 'lat')
  max_nlev = cvmix_input_get_netcdf_dim(fid, 'nlev')
#endif
  call cvmix_io_close(fid)

  ! Print dimensions to screen
  write(*,*) "Grid dimensions"
  write(*,*) "---------------"
  write(*,"(1X,A,I0)") "nlon = ", nlon
  write(*,"(1X,A,I0)") "nlat = ", nlat
  write(*,"(1X,A,I0)") "max_nlev = ", max_nlev
  write(*,*) ""
```

```
! Make sure all dimensions are valid
if (any((/nlon, nlat, max_nlev/).eq.-1)) then
  print*, "Error reading dimensions!"
  stop 1
end if

! Allocate memory for CVmix columns
allocate(CVmix_vars(nlon, nlat))
allocate(lat(nlon, nlat),lon(nlon, nlat))

! Allocate memory for energy flux, ocean depth, number of ocean levels,
! depth of each level / interface, and buoyancy frequency
allocate(energy_flux(nlon, nlat), ocn_depth(nlon, nlat))
allocate(buoy(nlon, nlat,max_nlev+1))
allocate(ocn_levels(nlon, nlat))
allocate(zt(max_nlev), zw_iface(max_nlev+1))
! Set buoyancy frequency = 0 at top interface (POP doesn't store these
! zeroes and the input data set is coming from POP output)
buoy(:,:,1) = cvmix_zero

! Allocate memory to store diffusivity values
allocate(Mdiff(nlon, nlat, max_nlev+1))
allocate(Tdiff(nlon, nlat, max_nlev+1))
! Set diffusivity to _FillValue
FillVal = 1e5_cvmix_r8
Mdiff   = FillVal
Tdiff   = FillVal

! Read in global data from grid file, physics file, and energy flux file
call cvmix_io_open(fid, trim(grid_file), 'nc', read_only=.true.)
call cvmix_input_read(fid, 'lon', lon)
call cvmix_input_read(fid, 'lat', lat)
call cvmix_input_read(fid, 'zw', zw_iface)
call cvmix_input_read(fid, 'H', ocn_depth)
call cvmix_input_read(fid, 'H_index', ocn_levels)
call cvmix_io_close(fid)
call cvmix_io_open(fid, trim(physics_file), 'nc', read_only=.true.)
call cvmix_input_read(fid, 'Nsqr', buoy(:,:,2:max_nlev+1))
call cvmix_io_close(fid)
call cvmix_io_open(fid, trim(energy_flux_file), 'nc', read_only=.true.)
call cvmix_input_read(fid, trim(energy_flux_var), energy_flux)
call cvmix_io_close(fid)

! Compute center of each layer (maybe this should be stored in grid file?)
do k=1, max_nlev
  zt(k) = 0.5_cvmix_r8*(zw_iface(k)+zw_iface(k+1))
end do
```

```
! Initialize tidal mixing parameters
call cvmix_init_tidal(CVMix_tidal_params_user=CVmix_Simmons_params,         &
                      local_mixing_frac=0.33_cvmix_r8,                      &
                      max_coefficient=0.01_cvmix_r8)

! Print parameter values to screen
print*, "Namelist variables"
print*, "------------------"
print*, "mix_scheme = ",                                                    &
        trim(cvmix_get_tidal_str('mix_scheme', CVmix_Simmons_params))
print*, "efficiency = ",                                                    &
        cvmix_get_tidal_real('efficiency', CVmix_Simmons_params)
print*, "vertical_decay_scale = ",                                          &
        cvmix_get_tidal_real('vertical_decay_scale', CVmix_Simmons_params)
print*, "max_coefficient = ",                                               &
        cvmix_get_tidal_real('max_coefficient', CVmix_Simmons_params)
print*, "local_mixing_frac = ",                                             &
        cvmix_get_tidal_real('local_mixing_frac', CVmix_Simmons_params)
print*, "depth_cutoff = ",                                                  &
        cvmix_get_tidal_real('depth_cutoff', CVmix_Simmons_params)

! For starters, using column from 353.9634 E, 58.84838 N)
! That's i=35, j=345 (compare result to KVMIX(0, :, 344, 34) in NCL)
do i=1,nlon
  do j=1,nlat
    nlev = ocn_levels(i,j)

    ! Initialization for CVMix data types
    call cvmix_put(CVmix_vars(i,j), 'nlev', nlev)
    if (nlev.gt.0) then
      call cvmix_put(CVmix_vars(i,j),  'surf_hgt',         cvmix_zero)
      call cvmix_put(CVmix_vars(i,j),        'zw', zw_iface(1:nlev+1))
      call cvmix_put(CVmix_vars(i,j),        'zt',         zt(1:nlev))
      call cvmix_put(CVmix_vars(i,j),      'buoy', buoy(i,j,1:nlev+1))
      call cvmix_put(CVmix_vars(i,j), 'ocn_depth',     ocn_depth(i,j))
      call cvmix_put(CVmix_vars(i,j),       'lat',           lat(i,j))
      call cvmix_put(CVmix_vars(i,j),       'lon',           lon(i,j))

      call cvmix_put(CVmix_params, 'max_nlev',     max_nlev)
      call cvmix_put(CVmix_params,  'Prandtl', 10._cvmix_r8)
      call cvmix_put(CVmix_params,   'fw_rho', 1e3_cvmix_r8)
      call cvmix_compute_Simmons_invariant(CVmix_vars(i,j), CVmix_params,   &
                                           energy_flux(i,j),                &
                                           CVmix_Simmons_params)
      ! Point CVmix_vars values to memory allocated above
      CVmix_vars(i,j)%Mdiff_iface => Mdiff(i,j,1:nlev+1)
      CVmix_vars(i,j)%Tdiff_iface => Tdiff(i,j,1:nlev+1)
```

```
      call cvmix_coeffs_tidal(CVmix_vars(i,j), CVmix_params,                &
                              CVmix_Simmons_params)

    end if
  end do
end do

if (CVmix_vars(lon_out, lat_out)%nlev.gt.0) then
  this_lon = CVmix_vars(lon_out, lat_out)%lon
  ! Need this_lon between -180 and 180
  do while(this_lon.lt.-180_cvmix_r8)
    this_lon = this_lon + 360_cvmix_r8
  end do
  do while(this_lon.gt.180_cvmix_r8)
    this_lon = this_lon - 360_cvmix_r8
  end do
  this_lat = CVmix_vars(lon_out, lat_out)%lat
  call cvmix_io_open(fid, "single_col.nc", "nc")
  call cvmix_output_write(fid, CVmix_vars(lon_out, lat_out),                &
                          (/"zw_iface", "Mdiff   ", "Tdiff   "/))
  if (this_lon.ge.0) then
    write(lonstr,"(F6.2,1X,A)") this_lon, "E"
  else
    write(lonstr,"(F6.2,1X,A)") abs(this_lon), "W"
  end if
  if (this_lat.ge.0) then
    write(latstr,"(F6.2,1X,A)") this_lat, "N"
  else
    write(latstr,"(F6.2,1X,A)") abs(this_lat), "S"
  end if
  ! Global Attributes
  call cvmix_output_write_att(fid, "column_lon", lonstr)
  call cvmix_output_write_att(fid, "column_lat", latstr)

  ! Variable Attributes
  call cvmix_output_write_att(fid, "long_name", "momentum diffusivity",     &
                              var_name="Mdiff")
  call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Mdiff")
  call cvmix_output_write_att(fid, "long_name", "tracer diffusivity",       &
                              var_name="Tdiff")
  call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Tdiff")
  call cvmix_output_write_att(fid, "long_name", "height at interface",      &
                              var_name="zw")
  call cvmix_output_write_att(fid, "positive", "up", var_name="zw")
  call cvmix_output_write_att(fid, "units", "m", var_name="zw")
  call cvmix_io_close(fid)
else
  print*, "ERROR: column requested for output is a land cell."
```

```
  stop 1
end if

! Write diffusivity field to netcdf
call cvmix_io_open(fid, "Mdiff.nc", "nc")
call cvmix_output_write(fid, "Mdiff", (/"nlon  ", "nlat  ", "niface"/),     &
                        Mdiff, FillVal=FillVal)
call cvmix_output_write_att(fid, "long_name", "momentum diffusivity",       &
                            var_name="Mdiff")
call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Mdiff")
call cvmix_io_close(fid)

call cvmix_io_open(fid, "Tdiff.nc", "nc")
call cvmix_output_write(fid, "Tdiff", (/"nlon  ", "nlat  ", "niface"/),     &
                        Tdiff, FillVal=FillVal)
call cvmix_output_write_att(fid, "long_name", "tracer diffusivity",         &
                            var_name="Tdiff")
call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Tdiff")
call cvmix_io_close(fid)

! memory cleanup
deallocate(CVmix_vars)
deallocate(energy_flux, ocn_depth)
deallocate(buoy)
deallocate(ocn_levels)
deallocate(zt, zw_iface)
```

1.107 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



 integer, parameter :: ncol = 2

 ! CVMix datatypes
 type(cvmix_data_type), dimension(ncol) :: CVmix_vars

 real(cvmix_r8), dimension(:,:), allocatable, target :: Tdiff, Sdiff
 real(cvmix_r8), dimension(:),   allocatable, target :: Rrho_num, Rrho_denom

 ! column / file indices
 integer :: k, fid, ic

 ! Namelist variables
 real(cvmix_r8) :: ddiff_exp1, strat_param_max
```

```
  ! Namelists that may be read in, depending on desired mixing scheme
  namelist/ddiff_nml/ddiff_exp1, strat_param_max

  print*, "Active levels: ", nlev
  print*, "Levels allocated in memory: ", max_nlev

  ! Allocate memory to store diffusivity values
  ! Also store numerator / denominator for stratification parameter
  allocate(Tdiff(max_nlev+1,ncol), Sdiff(max_nlev+1,ncol))
  allocate(Rrho_num(max_nlev), Rrho_denom(max_nlev))
  do k=1,nlev/2
    ! For first column, Rrho varies from 1 to 2
    Rrho_num(k) = real(k-1,cvmix_r8)/real(nlev/2-1,cvmix_r8)+cvmix_one
    Rrho_denom(k) = cvmix_one
    ! For second column, 1/Rrho varies from 1 to 10
    ! (Note: last column has diff=0, hence only using nlev instead of nlev+1)
    Rrho_num(k+nlev/2)   = -cvmix_one
    Rrho_denom(k+nlev/2) = -real(9*(k-1),cvmix_r8)/real(nlev/2-1,cvmix_r8) -  &
                           cvmix_one
  end do

  ! Point CVmix_vars values to memory allocated above
  do ic=1,ncol
    call cvmix_put(CVmix_vars(ic), 'nlev', nlev)
    call cvmix_put(CVmix_vars(ic), 'max_nlev', nlev)
    CVmix_vars(ic)%Tdiff_iface => Tdiff(:,ic)
    CVmix_vars(ic)%Sdiff_iface => Sdiff(:,ic)
    CVmix_vars(ic)%strat_param_num => Rrho_num(:)
    CVmix_vars(ic)%strat_param_denom => Rrho_denom(:)
  end do

  ! Read / set double diffusion parameters
  read(*, nml=ddiff_nml)
  call cvmix_init_ddiff(ddiff_exp1=ddiff_exp1, diff_conv_type="MC76",         &
                        strat_param_max=strat_param_max)
  call cvmix_coeffs_ddiff(CVmix_vars(1))

  call cvmix_init_ddiff(ddiff_exp1=ddiff_exp1, diff_conv_type="K88",          &
                        strat_param_max=strat_param_max)
  call cvmix_coeffs_ddiff(CVmix_vars(2))

  ! Output
#ifdef _NETCDF
  call cvmix_io_open(fid, "data.nc", "nc")
#else
  call cvmix_io_open(fid, "data.out", "ascii")
#endif
```

```
  call cvmix_output_write(fid, CVmix_vars, (/"Rrho ", "Tdiff", "Sdiff"/))
#ifdef _NETCDF
  call cvmix_output_write_att(fid, "long_name", "double diffusion " //        &
                              "stratification parameter", var_name="Rrho")
  call cvmix_output_write_att(fid, "units", "unitless", var_name="Rrho")
  call cvmix_output_write_att(fid, "long_name", "temperature diffusivity",    &
                              var_name="Tdiff")
  call cvmix_output_write_att(fid, "long_name", "salinity diffusivity",       &
                              var_name="Sdiff")
  call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Tdiff")
  call cvmix_output_write_att(fid, "units", "m^2/s", var_name="Sdiff")
#endif
  call cvmix_io_close(fid)
```

1.108 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



 real(cvmix_r8), parameter :: third = cvmix_one / real(3,cvmix_r8)

 ! CVMix datatypes
 type(cvmix_data_type)       :: CVmix_vars1, CVmix_vars4, CVmix_vars5

 real(cvmix_r8), dimension(:),   allocatable, target :: Mdiff, Tdiff, Sdiff
 real(cvmix_r8), dimension(:),   allocatable, target :: zt, zw_iface,        &
                                                        Ri_bulk, Ri_bulk2
 real(cvmix_r8), dimension(:),   allocatable, target :: w_m, w_s, zeta
 real(cvmix_r8), dimension(:,:), allocatable, target :: TwoDArray
 real(cvmix_r8), dimension(:),   allocatable, target :: buoyancy, shear_sqr, &
```

```
                                                    delta_vel_sqr,       &
                                                    buoy_freq_iface
real(cvmix_r8), dimension(:,:), allocatable, target :: hor_vel
real(cvmix_r8), dimension(2)                        :: ref_vel
real(cvmix_r8), dimension(4) :: shape_coeffs
integer :: i, fid, kt, kw, nlev1, nlev3, nlev4, max_nlev4, OBL_levid4, nlev5
real(cvmix_r8) :: hmix1, hmix5, ri_crit, layer_thick1, layer_thick4,        &
                  layer_thick5, OBL_depth4, OBL_depth5, N, Nsqr
real(cvmix_r8) :: kOBL_depth, Bslope, Vslope
real(cvmix_r8) :: sigma6, OBL_depth6, surf_buoy_force6, surf_fric_vel6,     &
                  vonkarman6, tao, rho0, grav, alpha, Qnonpen, Cp0,         &
                  w_m6, w_s6, wm6_true, ws6_true
character(len=cvmix_strlen) :: interp_type_t1, interp_type_t4, interp_type_t5
character(len=cvmix_strlen) :: filename
! True => run specified test
logical :: ltest1, ltest2, ltest3, ltest4, ltest5, ltest6
logical :: lnoDGat1 ! True => G'(1) = 0 (in test 4)

namelist/kpp_col1_nml/ltest1, nlev1, layer_thick1, interp_type_t1, hmix1,   &
                      ri_crit
namelist/kpp_col2_nml/ltest2
namelist/kpp_col3_nml/ltest3, nlev3
namelist/kpp_col4_nml/ltest4, interp_type_t4, OBL_levid4, lnoDGat1
namelist/kpp_col5_nml/ltest5, nlev5, layer_thick5, hmix5, interp_type_t5
namelist/kpp_col6_nml/ltest6, vonkarman6, tao, rho0, grav, alpha, Qnonpen,  &
                      Cp0, OBL_depth6

! Read namelists

! Defaults for test 1
ltest1         = .false.
nlev1          = 4
layer_thick1   =  real(10,  cvmix_r8)
hmix1          = -real(15,  cvmix_r8)
ri_crit        =  0.3_cvmix_r8
interp_type_t1 = 'quadratic'

! Defaults for test 2
ltest2 = .false.

! Defaults for test 3
ltest3 = .false.
nlev3  = 220

! Defaults for test 4
ltest4         = .false.
OBL_levid4     = 3
interp_type_t4 = 'quadratic'
```

```
lnoDGat1       = .true.

! Defaults for test 5
ltest5 = .false.
nlev5 = 10
layer_thick5   = real(5,  cvmix_r8)
hmix5          = real(17, cvmix_r8)
interp_type_t5 = "linear"

! Defaults for test 6
ltest6     = .false.
vonkarman6 = 0.4_cvmix_r8
tao        = 0.2_cvmix_r8
grav       = 9.8_cvmix_r8
alpha      = 2.5e-4_cvmix_r8
rho0       =  real(1035, cvmix_r8)
Qnonpen    = -real(100,  cvmix_r8)
Cp0        =  real(3992, cvmix_r8)
OBL_depth6 =  real(6000, cvmix_r8)

read(*, nml=kpp_col1_nml)
read(*, nml=kpp_col2_nml)
read(*, nml=kpp_col3_nml)
read(*, nml=kpp_col4_nml)
read(*, nml=kpp_col5_nml)
read(*, nml=kpp_col6_nml)

! Test 1: user sets up levels via namelist (constant thickness) and specifies
!         critical Richardson number as well as depth parameter hmix1. The
!         bulk Richardson number is assumed to be 0 from surface to hmix1 and
!         then increases linearly at a rate of Ri_crit/2 (so bulk Richardson
!         number = Ri_crit at hmix1+2). For computation, though, the average
!         bulk Richardson number (integral over vertical layer divided by
!         layer thickness) is stored at cell centers and then interpolated
!         (user can specify linear, quadratic or cubic interpolant) between
!         cell centers. OBL_depth is set to depth where interpolated bulk
!         Richardson number = Ri_crit; level-center depth (zt) and bulk
!         Richardson numbers are written out to test1.nc or test1.out
if (ltest1) then
  print*, "Test 1: determining OBL depth"
  print*, "----------"

  ! Initialize parameter datatype and set up column
  call cvmix_init_kpp(ri_crit=ri_crit, interp_type=interp_type_t1)
  call cvmix_put(CVmix_vars1, 'nlev', nlev1)
  call cvmix_put(CVmix_vars1, 'ocn_depth', layer_thick1*real(nlev1,cvmix_r8))

  ! Set up vertical levels (centers and interfaces) and compute bulk
```

```
    ! Richardson number
    allocate(zt(nlev1), zw_iface(nlev1+1), Ri_bulk(nlev1))
    do kw=1,nlev1+1
      zw_iface(kw) = -layer_thick1*real(kw-1, cvmix_r8)
    end do
    do kt=1,nlev1
      zt(kt) = 0.5_cvmix_r8*(zw_iface(kt)+zw_iface(kt+1))
      if (zw_iface(kt+1).gt.hmix1) then
        Ri_bulk(kt) = cvmix_zero
      else
        if (Ri_bulk(kt-1).eq.0) then
          ! Exact integration for average value over first cell with non-zero
          ! Ri_bulk
          Ri_bulk(kt) = 0.25_cvmix_r8 * ri_crit *                          &
                        (zw_iface(kt+1)-hmix1)**2 / layer_thick1
        else
          Ri_bulk(kt) = 0.5_cvmix_r8*ri_crit*(hmix1-zt(kt))
        end if
      end if
    end do

    CVmix_vars1%zt_cntr             => zt(:)
    CVmix_vars1%zw_iface            => zw_iface(:)
    CVmix_vars1%BulkRichardson_cntr => Ri_bulk(:)

    ! Compute OBL depth
    call cvmix_kpp_compute_OBL_depth(CVmix_vars1)

    ! Output to screen and file
    print*, "OBL depth = ", CVmix_vars1%BoundaryLayerDepth
    print*, "kw of interface above OBL depth = ", floor(CVmix_vars1%kOBL_depth)
    print*, "kt of cell center above OBL depth = ", nint(CVmix_vars1%kOBL_depth)-1

#ifdef _NETCDF
    call cvmix_io_open(fid, "test1.nc", "nc")
#else
    call cvmix_io_open(fid, "test1.out", "ascii")
#endif

    call cvmix_output_write(fid, CVmix_vars1, (/"zt     ", "zw     ",         &
                                                "Ri_bulk"/))
#ifdef _NETCDF
    call cvmix_output_write_att(fid, "Interpolation", interp_type_t1)
    call cvmix_output_write_att(fid, "analytic_OBL_depth", -hmix1 +           &
                                                      real(2,cvmix_r8))
    call cvmix_output_write_att(fid, "computed_OBL_depth",                    &
                                CVmix_vars1%BoundaryLayerDepth)
    call cvmix_output_write_att(fid, "kOBL_depth", CVmix_vars1%kOBL_depth)
```

```
#endif

    call cvmix_io_close(fid)

    deallocate(zt, zw_iface, Ri_bulk)
  end if ! ltest for Test 1

  ! Test 2: Compute coefficients of shape function G(sigma) when G(1) = 0 and
  !         G'(1) = 0. Result should be G(sigma) = sigma - 2sigma^2 + sigma^3
  if (ltest2) then
    print*, ""
    print*, "Test 2: Computing G(sigma)"
    print*, "----------"

    call cvmix_init_kpp(MatchTechnique='MatchGradient')
    call cvmix_kpp_compute_shape_function_coeffs(cvmix_zero, cvmix_zero,      &
                                                 shape_coeffs)
    write(*,"(1X,A,4F7.3)") "Coefficients are: ", shape_coeffs
  end if ! ltest for test 2

  ! Test 3: Recreate Figure B1 in LMD94 (phi(zeta)). Note that von Karman,
  !         surface buoyancy forcing, and surface velocity are set such that
  !         Monin-Obukhov constant = 1 => zeta = sigma.
  if (ltest3) then
    print*, ""
    print*, "Test 3: determining phi_m and phi_s (inversely proportional to ",&
            "w_m and w_s, respectively)"
    print*, "----------"
    call cvmix_init_kpp(vonkarman=cvmix_one, surf_layer_ext=cvmix_one)
    print*, "Coefficients for computing phi_m and phi_s:"
    print*, "a_m = ", cvmix_get_kpp_real('a_m')
    print*, "c_m = ", cvmix_get_kpp_real('c_m')
    print*, "a_s = ", cvmix_get_kpp_real('a_s')
    print*, "c_s = ", cvmix_get_kpp_real('c_s')
    allocate(w_m(nlev3+1), w_s(nlev3+1), zeta(nlev3+1))
    ! Note: zeta = sigma*OBL_depth/MoninObukhov constant
    !       zeta < 0 => unstable flow
    !       zeta > 0 => stable flow
    zeta(1) = -real(2,cvmix_r8)
    do kw=2, nlev3+1
      zeta(kw) = zeta(kw-1) + 2.2_cvmix_r8/real(nlev3,cvmix_r8)
    end do
    ! Typically the first argument of compute_turbulent_scales is sigma, and then
    ! the routine calculates zeta based on the next three parameters. Setting
    ! OBL_depth = surf_buoy_force = surf_fric_vel = 1 (with von Karman = 1 as
    ! well) => sigma = zeta
    call cvmix_kpp_compute_turbulent_scales(zeta, cvmix_one, cvmix_one,       &
                                            cvmix_one, w_m=w_m, w_s=w_s)
```

```
    allocate(TwoDArray(nlev3+1,3))
    TwoDArray(:,1) = zeta
    TwoDArray(:,2) = cvmix_one/w_m ! phi_m
    TwoDArray(:,3) = cvmix_one/w_s ! phi_s
#ifdef _NETCDF
    call cvmix_io_open(fid, "test3.nc", "nc")
#else
    call cvmix_io_open(fid, "test3.out", "ascii")
#endif
    call cvmix_output_write(fid, "data", (/"nrow", "ncol"/), TwoDArray)
    call cvmix_io_close(fid)
#ifdef _NETCDF
    print*, "Done! Data is stored in test3.nc, run plot_flux_profiles.ncl ",  &
            "to see output."
#else
    print*, "Done! Data is stored in test3.out, run plot_flux_profiles.ncl ", &
            "to see output."
#endif
    deallocate(TwoDArray)
    deallocate(zeta, w_m, w_s)
  endif ! ltest3

  ! Test 4: Compute boundary layer diffusivity
  !         1) Boundary layer between top interface and cell center
  !         2) Boundary layer between cell center and bottom interface
  !         3,4) Same as above, without enhanced diffusivity
  if (ltest4) then
    print*, ""
    print*, "Test 4: Computing Diffusivity in boundary layer"
    print*, "        (2 cases - boundary layer above and below cell center)"
    print*, "        (Both cases run with and without enhanced diffusivity)"
    print*, "----------"

    nlev4 = 5
    max_nlev4 = 10
    if (OBL_levid4.gt.nlev4) &
      OBL_levid4 = nlev4
    layer_thick4 = real(5,cvmix_r8)

    ! Set up vertical levels (centers and interfaces) and compute bulk
    ! Richardson number
    allocate(zt(max_nlev4), zw_iface(max_nlev4+1))
    zt = cvmix_zero
    zw_iface = cvmix_zero
    do kw=1,nlev4+1
      zw_iface(kw) = -layer_thick4*real(kw-1, cvmix_r8)
    end do
```

```
do kt=1,nlev4
  zt(kt) = 0.5_cvmix_r8*(zw_iface(kt)+zw_iface(kt+1))
end do
CVmix_vars4%zt_cntr  => zt(:)
CVmix_vars4%zw_iface => zw_iface(:)

! Set up diffusivities
allocate(Mdiff(max_nlev4+1), Tdiff(max_nlev4+1), Sdiff(max_nlev4+1))
CVmix_vars4%Mdiff_iface => Mdiff
CVmix_vars4%Tdiff_iface => Tdiff
CVmix_vars4%Sdiff_iface => Sdiff

! Set physical properties of column for test 4
call cvmix_put(CVmix_vars4, 'nlev',      nlev4)
call cvmix_put(CVmix_vars4, 'max_nlev',  max_nlev4)
call cvmix_put(CVmix_vars4, 'ocn_depth', layer_thick4*real(nlev4,cvmix_r8))
call cvmix_put(CVmix_vars4, 'surf_fric', cvmix_one)
call cvmix_put(CVmix_vars4, 'surf_buoy', real(100, cvmix_r8))
call cvmix_put(CVmix_vars4, 'Coriolis',  1e-4_cvmix_r8)

do i=1,2
  ! Test 4a/c: Boundary layer above center of level containing it
  Tdiff    = cvmix_zero
  Tdiff(1) = cvmix_one
  Tdiff(2) = real(10,cvmix_r8)
  Tdiff(3) = real(5,cvmix_r8)
  Mdiff = Tdiff
  Sdiff = Tdiff

  OBL_depth4 = abs(zt(OBL_levid4))-layer_thick4/real(4,cvmix_r8)
  call cvmix_put(CVmix_vars4, 'OBL_depth', OBL_depth4)
  call cvmix_put(CVmix_vars4, 'kOBL_depth',                                  &
       cvmix_kpp_compute_kOBL_depth(zw_iface, zt, OBL_depth4))

  print*, "OBL_depth = ", CVmix_vars4%BoundaryLayerDepth
  print*, "kOBL_depth = ", CVmix_vars4%kOBL_depth

  call cvmix_init_kpp(ri_crit=ri_crit, vonkarman=0.4_cvmix_r8,               &
                      interp_type2=interp_type_t4, lnoDGat1=lnoDGat1,        &
                      lenhanced_diff=(i.eq.1))
  call cvmix_coeffs_kpp(CVmix_vars4)

  print*, "Height and Diffusivity throughout column: "
  do kw=1,nlev4+1
    write(*,"(1X, F6.2, 1X, F8.5)") zw_iface(kw), Mdiff(kw)
  end do
  print*, ""
```

```
      if (i.eq.1) then
        filename="test4a"
      else
        filename="test4c"
      endif
#ifdef _NETCDF
      call cvmix_io_open(fid, trim(filename) // ".nc", "nc")
#else
      call cvmix_io_open(fid, trim(filename) // ".out", "ascii")
#endif

      call cvmix_output_write(fid, CVmix_vars4, (/"zt   ", "zw   ", "Mdiff",  &
                                                  "Tdiff", "Sdiff"/))
#ifdef _NETCDF
      call cvmix_output_write_att(fid, "interp_type2", interp_type_t4)
      call cvmix_output_write_att(fid, "OBL_depth",                           &
                                  CVmix_vars4%BoundaryLayerDepth)
#endif

      call cvmix_io_close(fid)

    ! Test 4b/d: Boundary layer below center of level containing it
      Tdiff    = cvmix_zero
      Tdiff(1) = cvmix_one
      Tdiff(2) = real(10,cvmix_r8)
      Tdiff(3) = real(5,cvmix_r8)
      Mdiff = Tdiff
      Sdiff = Tdiff

      OBL_depth4 = abs(zt(OBL_levid4))+layer_thick4/real(4,cvmix_r8)
      call cvmix_put(CVmix_vars4, 'OBL_depth', OBL_depth4)
      call cvmix_put(CVmix_vars4, 'kOBL_depth',                               &
           cvmix_kpp_compute_kOBL_depth(zw_iface, zt, OBL_depth4))

      print*, "OBL_depth = ", CVmix_vars4%BoundaryLayerDepth
      print*, "kOBL_depth = ", CVmix_vars4%kOBL_depth

      call cvmix_init_kpp(ri_crit=ri_crit, vonkarman=0.4_cvmix_r8,            &
                          interp_type2=interp_type_t4, lnoDGat1=lnoDGat1,     &
                          lenhanced_diff=(i.eq.1))
      call cvmix_coeffs_kpp(CVmix_vars4)

      print*, "Height and Diffusivity throughout column: "
      do kw=1,nlev4+1
        write(*,"(1X, F6.2, 1X, F8.5)") zw_iface(kw), Mdiff(kw)
      end do

      if (i.eq.1) then
```

```
        print*, ""
        filename="test4b"
      else
        filename="test4d"
      endif
#ifdef _NETCDF
      call cvmix_io_open(fid, trim(filename) // ".nc", "nc")
#else
      call cvmix_io_open(fid, trim(filename) // ".out", "ascii")
#endif

      call cvmix_output_write(fid, CVmix_vars4, (/"zt   ", "zw   ", "Mdiff",  &
                                                  "Tdiff", "Sdiff"/))
#ifdef _NETCDF
      call cvmix_output_write_att(fid, "interp_type2", interp_type_t4)
      call cvmix_output_write_att(fid, "OBL_depth",                           &
                                  CVmix_vars4%BoundaryLayerDepth)
#endif

      call cvmix_io_close(fid)

    end do

    deallocate(zt, zw_iface)
    deallocate(Mdiff, Tdiff, Sdiff)
  end if ! ltest4

  ! Test 5: Recreate figure C1 from LMD94
  if (ltest5) then
    print*, ""
    print*, "Test 5: Computing Bulk Richardson number"
    print*, "----------"

    ! using linear interpolation, averaging Nsqr, and setting Cv = 1.5  to
    ! match LMD result
    call cvmix_init_kpp(Cv = 1.5_cvmix_r8, interp_type=interp_type_t5)

    ! Set up vertical levels (centers and interfaces) and compute bulk
    ! Richardson number
    allocate(zt(nlev5), zw_iface(nlev5+1))
    do kw=1,nlev5+1
      zw_iface(kw) = -layer_thick5*real(kw-1, cvmix_r8)
    end do
    do kt=1,nlev5
      zt(kt) = 0.5_cvmix_r8 * (zw_iface(kt)+zw_iface(kt+1))
    end do

    ! Compute Br-B(d), |Vr-V(d)|^2, and Vt^2
```

```
    allocate(buoyancy(nlev5), delta_vel_sqr(nlev5), hor_vel(nlev5,2),         &
             shear_sqr(nlev5), w_s(nlev5), Ri_bulk(nlev5), Ri_bulk2(nlev5),   &
             buoy_freq_iface(nlev5+1))

    ref_vel(1) = 0.1_cvmix_r8
    ref_vel(2) = cvmix_zero
    N          = 0.01_cvmix_r8
    Nsqr       = N*N
    Bslope     = -Nsqr
    Vslope     = -0.1_cvmix_r8 / (real(nlev5,cvmix_r8)*layer_thick5-hmix5)
    do kt=1,nlev5
      if ((zt(kt).ge.-hmix5).or.(kt.eq.1)) then
        buoyancy(kt)  = Nsqr
        hor_vel(kt,1) = 0.1_cvmix_r8
        buoy_freq_iface(kt) = cvmix_zero
      else
        if (zw_iface(kt).ge.-hmix5) then
          ! derivatives of buoyancy and horizontal velocity component are
          ! discontinuous in this layer (no change -> non-zero linear change)
          ! so we compute area-average of analytic function over layer
          buoyancy(kt)  = Bslope*(-zw_iface(kt+1)-real(hmix5,cvmix_r8))**2 /  &
                          (real(2,cvmix_r8)*layer_thick5) + Nsqr
          hor_vel(kt,1) = Vslope*(-zw_iface(kt+1)-real(hmix5,cvmix_r8))**2 /  &
                          (real(2,cvmix_r8)*layer_thick5) + 0.1_cvmix_r8
        else
          buoyancy(kt)  = Nsqr+Bslope*(-zt(kt)-real(hmix5,cvmix_r8))
          hor_vel(kt,1) = 0.1_cvmix_r8 + Vslope *                             &
                          (-zt(kt)-real(hmix5,cvmix_r8))
        end if
        buoy_freq_iface(kt) = sqrt(-(buoyancy(kt)-buoyancy(kt-1)) /           &
                                   layer_thick5)
      end if
      ! Compute w_s with zeta=0 per LMD page 393
      ! => w_s = von Karman * surf_fric_vel = 0.4*0.01 = 4e-3
      call cvmix_kpp_compute_turbulent_scales(cvmix_zero, -zt(kt),            &
                                              buoyancy(1), 0.01_cvmix_r8,     &
                                              w_s=w_s(kt))
      hor_vel(kt,2) = cvmix_zero
      delta_vel_sqr(kt) = (ref_vel(1)-hor_vel(kt,1))**2 +                     &
                          (ref_vel(2)-hor_vel(kt,2))**2
    end do
    buoy_freq_iface(nlev5+1) = N
!   MNL: test both uses of compute_bulk_Richardson
    Ri_bulk = cvmix_kpp_compute_bulk_Richardson(zt, (buoyancy(1)-buoyancy),   &
                                                delta_vel_sqr,                &
                                                Nsqr_iface = buoy_freq_iface**2, &
                                                ws_cntr = w_s)
```

```
    shear_sqr = cvmix_kpp_compute_unresolved_shear(zt, w_s,                    &
                                                   Nsqr_iface = buoy_freq_iface**2)
    ! Note that Vt_shear_sqr is the fourth argument in compute_bulk_Richardson
    ! so it does not need to declared explicitly (even though it is optional)
    Ri_bulk2  = cvmix_kpp_compute_bulk_Richardson(zt, (buoyancy(1)-buoyancy), &
                                                  delta_vel_sqr, shear_sqr)
    call cvmix_kpp_compute_OBL_depth(Ri_bulk, zw_iface, OBL_depth5,           &
                                     kOBL_depth, zt)
    do kt=1,nlev5
      if (abs(Ri_bulk(kt)-Ri_bulk2(kt)).gt.1e-12_cvmix_r8) then
        print*, "WARNING: two Ri_bulk computations did not match!"
        print*, zt(kt), Ri_bulk(kt), Ri_bulk2(kt)
      else
        print*, zt(kt), Ri_bulk(kt)
      end if
    end do
    print*, "OBL has depth of ", OBL_depth5
#ifdef _NETCDF
    print*, "Done! Data is stored in test5.nc, run plot_bulk_Rich.ncl ",      &
            "to see output."
#else
    print*, "Done! Data is stored in test5.out, run plot_bulk_Rich.ncl ",     &
            "to see output."
#endif

    CVmix_vars5%nlev                =  nlev5
    CVmix_vars5%BoundaryLayerDepth  =  OBL_depth5
    CVmix_vars5%zt_cntr             => zt
    CVmix_vars5%BulkRichardson_cntr => Ri_bulk
    CVmix_vars5%Vx_cntr             => hor_vel(:,1)
#ifdef _NETCDF
    call cvmix_io_open(fid, "test5.nc", "nc")
#else
    call cvmix_io_open(fid, "test5.out", "ascii")
#endif
    call cvmix_output_write(fid, CVmix_vars5, (/"zt      ",                   &
                                                "Ri_bulk ",                   &
                                                "Vx      ",                   &
                                                "buoyancy"/), buoyancy)
#ifdef _NETCDF
    call cvmix_output_write_att(fid, "OBL_depth",                             &
                                CVmix_vars5%BoundaryLayerDepth)
    call cvmix_output_write_att(fid, "longname", "ocean height (cell center)",&
                                "zt")
    call cvmix_output_write_att(fid, "units", "m", "zt")
    call cvmix_output_write_att(fid, "longname", "horizontal velocity", "U")
    call cvmix_output_write_att(fid, "units", "m/s", "U")
    call cvmix_output_write_att(fid, "units", "m/s^2", "buoyancy")
```

```
    call cvmix_output_write_att(fid, "longname", "Bulk Richardson number",    &
                                "BulkRichardson")
    call cvmix_output_write_att(fid, "units", "unitless", "BulkRichardson")
#endif
    call cvmix_io_close(fid)

    deallocate(zt, zw_iface)
    deallocate(buoyancy, delta_vel_sqr, hor_vel, shear_sqr, w_s, Ri_bulk,     &
               Ri_bulk2, buoy_freq_iface)
  end if ! ltest5

  ! Test 6: Recreate figure C1 from LMD94
  if (ltest6) then
    print*, ""
    print*, "Test 6: 2 simple tests for velocity scale"
    print*, "----------"

    call cvmix_init_kpp(vonkarman=vonkarman6)
    sigma6 = 0.1_cvmix_r8

    surf_buoy_force6 = cvmix_zero
    surf_fric_vel6   = sqrt(tao/rho0)
    print*, "6a: Bf = 0 m^2/s^3 and u* = sqrt(tao/rho0)"
    print*, "                           =", surf_fric_vel6
    wm6_true = cvmix_get_kpp_real("vonkarman")*surf_fric_vel6
    call cvmix_kpp_compute_turbulent_scales(sigma6, OBL_depth6,               &
                                            surf_buoy_force6, surf_fric_vel6, &
                                            w_m = w_m6, w_s = w_s6)

    print*, "    => w_m = w_s ~= vonkarman*u*"
    print*, "                   =", wm6_true
    print*, "w_m = ", w_m6
    print*, "w_s = ", w_s6
    print*, ""

    surf_buoy_force6 = grav*alpha*Qnonpen / (rho0*Cp0)
    surf_fric_vel6   = cvmix_zero
    print*, "6b: u* = 0 m/s and Bf = (grav*alpha/(rho0*Cp0))*Qnonpen"
    print*, "                      =", surf_buoy_force6
    wm6_true = cvmix_get_kpp_real("vonkarman")*                               &
               ((-surf_buoy_force6*OBL_depth6)**third)*                       &
   ((cvmix_get_kpp_real("c_m")*sigma6*cvmix_get_kpp_real("vonkarman"))**third)
    ws6_true = cvmix_get_kpp_real("vonkarman")*                               &
               ((-surf_buoy_force6*OBL_depth6)**third)*                       &
   ((cvmix_get_kpp_real("c_s")*sigma6*cvmix_get_kpp_real("vonkarman"))**third)
    call cvmix_kpp_compute_turbulent_scales(sigma6, OBL_depth6,               &
                                            surf_buoy_force6, surf_fric_vel6, &
                                            w_m = w_m6, w_s = w_s6)
```

```
  print*, "    => w_m = vonkarman * (-Bf * OBL_depth)^1/3 *",              &
                        " (c_m*0.1*vonkarman)^1/3 m/s"
  print*, "           = ", wm6_true
  print*, "    => w_s = vonkarman * (-Bf * OBL_depth)^1/3 *",              &
                        " (c_s*0.1*vonkarman)^1/3 m/s"
  print*, "           = ", ws6_true
  print*, "w_m = ", w_m6
  print*, "w_s = ", w_s6
  print*, ""

end if ! ltest6
```

1.109 Fortran: Module Interface cvmix_io (Source File: cvmix_io.F90)

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(:)
```

1.110 cvmix_io_open

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

LOCAL VARIABLES:

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


    if (present(read_only)) then
      readonly = read_only
    else
      readonly = .false.
    end if
    ! Need routine that will produce unique file_id
    ! Starting with 615 and incrementing by one for now...
    file_id = 615
    if (.not.(allocated(file_database))) then
      allocate(file_database(1))
      file_database(1)%file_id = file_id
      nullify(file_database(1)%prev)
      nullify(file_database(1)%next)
      file_index => file_database(1)
    else
      file_id = file_id+1
      file_index => file_database(1)
      do while(associated(file_index%next))
        file_id = file_id+1
        file_index => file_index%next
      end do
      allocate(file_index%next)
      file_index%next%file_id   = file_id
      file_index%next%prev     => file_index
      nullify(file_index%next%next)
      file_index => file_index%next
    end if
    file_index%file_name = trim(file_name)

    select case (trim(file_format))
      case ('nc')
#ifndef _NETCDF
        print*, "ERROR: you must compile -D_NETCDF to open a netCDF file"
```

```
        stop 1
#else
        file_index%file_type = NETCDF_FILE_TYPE
        ! Note: at this point, will either open file with NOWRITE for
        !       read-only, or will clobber file to write new data to it.
        !       Eventually we should add a check to see if the file exists
        !       and open it with NF90_WRITE for non-readonly files, but that
        !       will require checking to see if dims / variables already exist
        !       (and are correct dimension) before trying to define them.
        if (readonly) then
          call netcdf_check(nf90_open(file_name, NF90_NOWRITE, file_id))
        else
          call netcdf_check(nf90_create(file_name, NF90_CLOBBER, file_id))
        end if
        file_index%file_id = file_id
        ! For outputting params, want vertical dimension to be unlimited?
        ! (Will be looping through the levels)
#endif
      case ('ascii')
        file_index%file_type = ASCII_FILE_TYPE
        if (readonly) then
          open(file_id, file = file_name, status="old")
        else
          open(file_id, file = file_name, status="replace")
        end if
      case default
        print*, "ERROR: ", trim(file_format)," is not a valid file type"

    end select
```

1.111 cvmix_input_read_1d_double

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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


  local_copy = 0.0
  lerr_in_read = .false.
    select case (get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        varid = get_netcdf_varid(file_id, var_name, xtype, ndims)
        lerr_in_read = (varid.eq.-1)

        if (lerr_in_read) then
          write(*,"(A,A,1X,A,A)") "Could not find variable ", trim(var_name), &
                                  "in ", trim(get_file_name(file_id))
        else
          ! A couple more error checks
          if (xtype.ne.NF90_DOUBLE) then
            write(*, "(A,1X,A,1X,A)") "ERROR: variable", trim(var_name), &
                                      "is not a single-precision float!"
            lerr_in_read = .true.
          end if
          if (ndims.ne.1) then
            write(*,"(A,1X,I0,A)") "ERROR: you are trying to read a", ndims, &
                                   "-dimensional array into a 1D array."
            lerr_in_read = .true.
          end if
        end if

        if (.not.lerr_in_read) then
          call netcdf_check(nf90_inquire_variable(file_id, varid, dimids=dims))
          dims1 = dims(1)
```

```
          call netcdf_check(nf90_inquire_dimension(file_id, dims1, len=dims2))

          dims1 = size(local_copy)
          if (dims1.eq.dims2) then
            call netcdf_check(nf90_get_var(file_id, varid, local_copy))
          else
            write(*,"(A,1X,I0,A,1X,I0,A)") "ERROR: you are trying to read a", &
                   dims2, "-dimensional array into a local variable that is", &
                   dims1, "-dimensional."
            lerr_in_read = .true.
          end if
        end if
#endif
      case DEFAULT
        lerr_in_read = .true.
        write(*,"(A,1X,A,1X,A)") "ERROR: no read support for binary files,", &
                                 "use netCDF to read", trim(var_name)
    end select

    if (lerr_in_read) then
      call cvmix_io_close_all
      stop 1
    end if
```

1.112 cvmix_input_read_2d_integer

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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


  local_copy = 0
  lerr_in_read = .false.
    select case (get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        varid = get_netcdf_varid(file_id, var_name, xtype, ndims)
        lerr_in_read = (varid.eq.-1)

        if (lerr_in_read) then
          write(*,"(A,A,1X,A,A)") "Could not find variable ", trim(var_name), &
                                  "in ", trim(get_file_name(file_id))
        else
          ! A couple more error checks
          if (xtype.ne.NF90_INT) then
            write(*, "(A,1X,A,1X,A)") "ERROR: variable", trim(var_name), &
                                      "is not an integer!"
            lerr_in_read = .true.
          end if
          if (ndims.ne.2) then
            write(*,"(A,1X,I0,A)") "ERROR: you are trying to read a", ndims, &
                                   "-dimensional array into a 2D array."
            lerr_in_read = .true.
          end if
        end if

        if (.not.lerr_in_read) then
          call netcdf_check(nf90_inquire_variable(file_id, varid, dimids=dims1))
          do i=1,2
            call netcdf_check(nf90_inquire_dimension(file_id, dims1(i), &
                              len=dims2(i)))
          end do

          dims1 = shape(local_copy)
          if (all(dims1.eq.dims2)) then
            call netcdf_check(nf90_get_var(file_id, varid, local_copy))
```

```
          else
            write(*,"(A,1X,I0,1X,A,1X,I0,1X,A,1X,I0,1X,A,1X,I0)") &
                    "ERROR: you are trying to read a", dims2(1), "by", dims2(2), &
                    "array into a local variable that is", dims1(1), "by", dims1(2)
            lerr_in_read = .true.
          end if
        end if
#endif
      case DEFAULT
        lerr_in_read = .true.
        write(*,"(A,1X,A,1X,A)") "ERROR: no read support for binary files,", &
                                 "use netCDF to read", trim(var_name)
    end select

    if (lerr_in_read) then
      call cvmix_io_close_all
      stop 1
    end if
```

1.113 cvmix_input_read_2d_double

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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


  local_copy = cvmix_zero
  lerr_in_read = .false.
    select case (get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        varid = get_netcdf_varid(file_id, var_name, xtype, ndims)
        lerr_in_read = (varid.eq.-1)

        if (lerr_in_read) then
          write(*,"(A,A,1X,A,A)") "Could not find variable ", trim(var_name), &
                                  "in ", trim(get_file_name(file_id))
        else
          ! A couple more error checks
          if (xtype.ne.NF90_DOUBLE) then
            write(*, "(A,1X,A,1X,A)") "ERROR: variable", trim(var_name), &
                                      "is not a double-precision float!"
            lerr_in_read = .true.
          end if
          if (ndims.ne.2) then
            write(*,"(A,1X,I0,A)") "ERROR: you are trying to read a", ndims, &
                                   "-dimensional array into a 2D array."
            lerr_in_read = .true.
          end if
        end if

        if (.not.lerr_in_read) then
          call netcdf_check(nf90_inquire_variable(file_id, varid, dimids=dims1))
          do i=1,2
            call netcdf_check(nf90_inquire_dimension(file_id, dims1(i), &
                              len=dims2(i)))
          end do

          dims1 = shape(local_copy)
          if (all(dims1.eq.dims2)) then
            call netcdf_check(nf90_get_var(file_id, varid, local_copy))
          else
            write(*,"(A,1X,I0,1X,A,1X,I0,1X,A,1X,I0,1X,A,1X,I0)") &
                    "ERROR: you are trying to read a", dims2(1), "by", dims2(2), &
                    "array into a local variable that is", dims1(1), "by", dims1(2)
            lerr_in_read = .true.
```

```
          end if
        end if
#endif
      case DEFAULT
        lerr_in_read = .true.
        write(*,"(A,1X,A,1X,A)") "ERROR: no read support for binary files,", &
                                 "use netCDF to read", trim(var_name)
    end select

    if (lerr_in_read) then
      call cvmix_io_close_all
      stop 1
    end if
```

1.114 cvmix_input_read_3d_double

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

```
  local_copy = cvmix_zero
  lerr_in_read = .false.
    select case (get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        varid = get_netcdf_varid(file_id, var_name, xtype, ndims)
        lerr_in_read = (varid.eq.-1)

        if (lerr_in_read) then
          write(*,"(A,A,1X,A,A)") "Could not find variable ", trim(var_name), &
                                  "in ", trim(get_file_name(file_id))
        else
          ! A couple more error checks
          if (xtype.ne.NF90_DOUBLE) then
            write(*, "(A,1X,A,1X,A)") "ERROR: variable", trim(var_name), &
                                      "is not a double-precision float!"
            lerr_in_read = .true.
          end if
          if (ndims.ne.3) then
            write(*,"(A,1X,I0,A)") "ERROR: you are trying to read a", ndims, &
                                   "-dimensional array into a 2D array."
            lerr_in_read = .true.
          end if
        end if

        if (.not.lerr_in_read) then
          call netcdf_check(nf90_inquire_variable(file_id, varid, dimids=dims1))
          do i=1,3
            call netcdf_check(nf90_inquire_dimension(file_id, dims1(i), &
                              len=dims2(i)))
          end do

          dims1 = shape(local_copy)
          if (all(dims1.eq.dims2)) then
            call netcdf_check(nf90_get_var(file_id, varid, local_copy))
          else
            write(*,"(A,1X,I0,1X,A,1X,I0,1X,A,1X,I0,1X,A,1X,I0,1X,A,1X,I0,1X,A,1X,I0)") &
                    "ERROR: you are trying to read a", dims2(1), "by", dims2(2), &
                    "by", dims2(3), "array into a local variable that is", &
                    dims1(1), "by", dims1(2), "by", dims1(3)
            lerr_in_read = .true.
          end if
        end if
#endif
      case DEFAULT
        lerr_in_read = .true.
        write(*,"(A,1X,A,1X,A)") "ERROR: no read support for binary files,", &
```

```
                                "use netCDF to read", trim(var_name)
    end select

    if (lerr_in_read) then
      call cvmix_io_close_all
      stop 1
    end if
```

1.115 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).

USES:

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

INPUT PARAMETERS:

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

```
    nlev = CVmix_vars%nlev
    select case (get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        nt = nlev
        nw = nlev+1
        call netcdf_check(nf90_def_dim(file_id, "nt", nt, nt_id))
        call netcdf_check(nf90_def_dim(file_id, "nw", nw, nw_id))
        allocate(var_id(size(var_names)))
        do var=1,size(var_names)
          select case(trim(cvmix_att_name(var_names(var))))
            case ("zt_cntr")
              call netcdf_check(nf90_def_var(file_id, "zt", NF90_DOUBLE,      &
                                             (/nt_id/), var_id(var)))
            case ("zw_iface")
              call netcdf_check(nf90_def_var(file_id, "zw", NF90_DOUBLE,      &
                                             (/nw_id/), var_id(var)))
            case ("ShearRichardson_iface")
              call netcdf_check(nf90_def_var(file_id, "ShearRichardson",      &
                                             NF90_DOUBLE, (/nw_id/),          &
                                             var_id(var)))
            case ("BulkRichardson_cntr")
              call netcdf_check(nf90_def_var(file_id, "BulkRichardson",       &
                                             NF90_DOUBLE, (/nt_id/),          &
                                             var_id(var)))
            case ("Mdiff_iface")
              call netcdf_check(nf90_def_var(file_id, "Mdiff", NF90_DOUBLE,   &
                                             (/nw_id/), var_id(var)))
            case ("Tdiff_iface")
              call netcdf_check(nf90_def_var(file_id, "Tdiff", NF90_DOUBLE,   &
                                             (/nw_id/), var_id(var)))
            case ("Sdiff_iface")
              call netcdf_check(nf90_def_var(file_id, "Sdiff", NF90_DOUBLE,   &
                                             (/nw_id/), var_id(var)))
            case ("strat_param")
              call netcdf_check(nf90_def_var(file_id, "Rrho", NF90_DOUBLE,    &
                                             (/nt_id/), var_id(var)))
            case ("buoyancy_cntr")
              call netcdf_check(nf90_def_var(file_id, "buoyancy", NF90_DOUBLE,&
                                             (/nt_id/), var_id(var)))
            case ("SqrBuoyancyFreq_iface")
              call netcdf_check(nf90_def_var(file_id, "SqrBuoyancyFreq",      &
                                             NF90_DOUBLE, (/nw_id/),          &
                                             var_id(var)))
            case ("Vx_cntr")
              call netcdf_check(nf90_def_var(file_id, "U", NF90_DOUBLE,       &
                                             (/nt_id/), var_id(var)))
```

```
    case ("Vy_cntr")
      call netcdf_check(nf90_def_var(file_id, "V", NF90_DOUBLE,       &
                                     (/nt_id/), var_id(var)))
    case DEFAULT
      print*, "ERROR: unable to write variable ", var_names(var)
      stop 1
  end select
end do
call netcdf_check(nf90_enddef(file_id))
do var=1,size(var_names)
  select case(trim(cvmix_att_name(var_names(var))))
    case ("zt_cntr")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                                     CVmix_vars%zt_cntr(1:nlev)))
    case ("zw_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                                     CVmix_vars%zw_iface(1:nlev+1)))
    case ("ShearRichardson_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                          CVmix_vars%ShearRichardson_iface(1:nlev+1)))
    case ("BulkRichardson_cntr")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                             CVmix_vars%BulkRichardson_cntr(1:nlev)))
    case ("Mdiff_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                                     CVmix_vars%Mdiff_iface(1:nlev+1)))
    case ("Tdiff_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                                     CVmix_vars%Tdiff_iface(1:nlev+1)))
    case ("Sdiff_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                                     CVmix_vars%Sdiff_iface(1:nlev+1)))
    case ("strat_param")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                                CVmix_vars%strat_param_num(1:nlev) /  &
                                CVmix_vars%strat_param_denom(1:nlev)))
    case ("buoyancy_cntr")
      if (present(buoyancy_cntr)) then
        call netcdf_check(nf90_put_var(file_id, var_id(var),          &
                                       buoyancy_cntr(1:nlev)))
      else
        print*, "ERROR: to write buoyancy at cell center in ",        &
                "cvmix_io, you need to provide the optional ",        &
                "buoyancy_cntr argument!"
        stop
      end if
    case ("SqrBuoyancyFreq_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
```

```
                                  CVmix_vars%SqrBuoyancyFreq_iface(1:nlev+1)))
            case ("Vx_cntr")
              call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                                             CVmix_vars%Vx_cntr(1:nlev)))
            case ("Vy_cntr")
              call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                                             CVmix_vars%Vy_cntr(1:nlev)))
            case DEFAULT
              print*, "ERROR: unable to write variable ", var_names(var)
              stop 1
          end select
        end do
#endif
      case (ASCII_FILE_TYPE)
        do kw=1,nlev+1
          do var=1,size(var_names)
            select case(trim(cvmix_att_name(var_names(var))))
              case ("zt_cntr")
                if (kw.gt.1) then
                  write(file_id,"(E24.17E2)",advance='no')                    &
                        CVmix_vars%zt_cntr(kw-1)
                else
                  write(file_id,"(A)",advance='no') "--- Cell Center Vals ---"
                end if
              case ("zw_iface")
                write(file_id,"(E24.17E2)",advance='no')                      &
                      CVmix_vars%zw_iface(kw)
              case ("ShearRichardson_iface")
                write(file_id,"(E24.17E2)",advance='no')                      &
                      CVmix_vars%ShearRichardson_iface(kw)
              case ("BulkRichardson_cntr")
                if (kw.gt.1) then
                  write(file_id,"(E24.17E2)",advance='no')                    &
                        CVmix_vars%BulkRichardson_cntr(kw-1)
                else
                  write(file_id,"(A)",advance='no') "--- Cell Center Vals ---"
                end if
              case ("Mdiff_iface")
                write(file_id,"(E24.17E2)",advance='no')                      &
                      CVmix_vars%Mdiff_iface(kw)
              case ("Tdiff_iface")
                write(file_id,"(E24.17E2)",advance='no')                      &
                      CVmix_vars%Tdiff_iface(kw)
              case ("Sdiff_iface")
                write(file_id,"(E24.17E2)",advance='no')                      &
                      CVmix_vars%Sdiff_iface(kw)
              case ("strat_param")
                if (kw.lt.nlev+1) then
```

```
            write(file_id,"(E24.17E2)",advance='no')                    &
                  CVmix_vars%strat_param_num(kw) /                      &
                  CVmix_vars%strat_param_denom(kw)
          else
            write(file_id,"(E24.17E2)",advance='no') 0.0
          end if
        case ("buoyancy_cntr")
          if (present(buoyancy_cntr)) then
            if (kw.gt.1) then
              write(file_id,"(E24.17E2)",advance='no')                 &
                    buoyancy_cntr(kw-1)
            else
              write(file_id,"(A)",advance='no')                        &
                    "--- Cell Center Vals ---"
            end if
          else
            print*, "ERROR: to write buoyancy at cell center in ",     &
                    "cvmix_io, you need to provide the optional ",     &
                    "buoyancy_cntr argument!"
            stop
          end if
        case ("SqrBuoyancyFreq_iface")
          write(file_id,"(E24.17E2)",advance='no')                     &
                CVmix_vars%SqrBuoyancyFreq_iface(kw)
        case ("Vx_cntr")
          if (kw.gt.1) then
            write(file_id,"(E24.17E2)",advance='no')                   &
                  CVmix_vars%Vx_cntr(kw-1)
          else
            write(file_id,"(A)",advance='no') "--- Cell Center Vals ---"
          end if
        case ("Vy_cntr")
          if (kw.gt.1) then
            write(file_id,"(E24.17E2)",advance='no')                   &
                  CVmix_vars%Vy_cntr(kw-1)
          else
            write(file_id,"(A)",advance='no') "--- Cell Center Vals ---"
          end if
        case DEFAULT
          print*, "ERROR: unable to write variable ", var_names(var)
          stop 1
      end select
      if (var.ne.size(var_names)) write(file_id, "(1X)", advance='no')
    end do
    write(file_id, *)
  end do
case DEFAULT
  print*, "ERROR: Invalid file type"
```

```
        stop 1
    end select
```

1.116 cvmix_output_write_multi_col

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

```
     integer :: 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


    z_err = .false.
    ncol = size(CVmix_vars)
    nlev = CVmix_vars(1)%nlev
    ! Make sure all levels are the same
```

```
    do icol=2,ncol
      if (CVmix_vars(icol)%nlev+1.ne.nlev+1) then
        z_err = .true.
      else
        ! Make sure z_iface lines up for Bryan-Lewis case
        if (associated(CVmix_vars(1)%zw_iface)) then
          if (any(CVmix_vars(icol)%zw_iface.ne.CVmix_vars(icol-1)%zw_iface)) then
            z_err = .true.
          end if
        end if
      end if
    end do

    if (z_err) then
      print*, "ERROR: z-coordinates are not the same in every column!"
      stop 1
    end if

    select case (get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        call netcdf_check(nf90_def_dim(file_id, "nt",   nlev,   nt_id))
        call netcdf_check(nf90_def_dim(file_id, "nw",   nlev+1, nw_id))
        call netcdf_check(nf90_def_dim(file_id, "ncol", ncol,   ncol_id))
        allocate(var_id(size(var_names)))
        do var=1,size(var_names)
          var_name = cvmix_att_name(var_names(var))
          select case(var_name)
            case("zw_iface")
              call netcdf_check(nf90_def_var(file_id, "zw", NF90_DOUBLE,      &
                                             (/nw_id/), var_id(var)))
            case("strat_param")
              call netcdf_check(nf90_def_var(file_id, "Rrho", NF90_DOUBLE,    &
                                             (/nt_id/), var_id(var)))
            case("ShearRichardson_iface")
              call netcdf_check(nf90_def_var(file_id, "ShearRichardson",      &
                                              NF90_DOUBLE, (/nw_id/),         &
                                              var_id(var)))
            case("Mdiff_iface")
              call netcdf_check(nf90_def_var(file_id, "Mdiff", NF90_DOUBLE,   &
                                             (/ncol_id,nw_id/), var_id(var)))
            case("Tdiff_iface")
              call netcdf_check(nf90_def_var(file_id, "Tdiff", NF90_DOUBLE,   &
                                             (/ncol_id,nw_id/), var_id(var)))
            case("Sdiff_iface")
              call netcdf_check(nf90_def_var(file_id, "Sdiff", NF90_DOUBLE,   &
                                             (/ncol_id,nw_id/), var_id(var)))
          end select
```

```
  ! Before writing netcdf file, we gather data from all the columns
  ! into a local array
  if (trim(var_name).eq."Mdiff_iface") then
    allocate(lcl_Mdiff(ncol,nlev+1))
    do icol=1,ncol
      lcl_Mdiff(icol,:) = CVmix_vars(icol)%Mdiff_iface(1:nlev+1)
    end do
  end if
  if (trim(var_name).eq."Tdiff_iface") then
    allocate(lcl_Tdiff(ncol,nlev+1))
    do icol=1,ncol
      lcl_Tdiff(icol,:) = CVmix_vars(icol)%Tdiff_iface(1:nlev+1)
    end do
  end if
  if (trim(var_name).eq."Sdiff_iface") then
    allocate(lcl_Sdiff(ncol,nlev+1))
    do icol=1,ncol
      lcl_Sdiff(icol,:) = CVmix_vars(icol)%Sdiff_iface(1:nlev+1)
    end do
  end if

end do
call netcdf_check(nf90_enddef(file_id))

! Write data to netCDF file
do var=1,size(var_names)
  select case(trim(cvmix_att_name(var_names(var))))
    case ("zw_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                        CVmix_vars(1)%zw_iface(1:nlev+1)))
    case("strat_param")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                        CVmix_vars(1)%strat_param_num(1:nlev) /       &
                        CVmix_vars(1)%strat_param_denom(1:nlev)))
    case ("ShearRichardson_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                       CVmix_vars(1)%ShearRichardson_iface(1:nlev+1)))
    case("Mdiff_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                        lcl_Mdiff))
      deallocate(lcl_Mdiff)
    case("Tdiff_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
                        lcl_Tdiff))
      deallocate(lcl_Tdiff)
    case("Sdiff_iface")
      call netcdf_check(nf90_put_var(file_id, var_id(var),            &
```

```
                                lcl_Sdiff))
              deallocate(lcl_Sdiff)
            case DEFAULT
              print*, "ERROR: unable to write variable ", var_names(var)
              stop 1
          end select
        end do
#endif
      case (ASCII_FILE_TYPE)
        do kw=1,nlev+1
          do var=1,size(var_names)
            select case(trim(cvmix_att_name(var_names(var))))
              case ("zw_iface")
                write(file_id,"(E24.17E2)",advance='no') &
                      CVmix_vars(1)%zw_iface(kw)
              case ("strat_param")
                if (kw.ne.nlev+1) then
                  write(file_id,"(E24.17E2)",advance='no')                    &
                        CVmix_vars(1)%strat_param_num(kw) /                   &
                        CVmix_vars(1)%strat_param_denom(kw)
                else
                  write(file_id,"(E24.17E2)",advance='no') 0.0
                end if
              case ("ShearRichardson_iface")
                write(file_id,"(E24.17E2)",advance='no') &
                      CVmix_vars(1)%ShearRichardson_iface(kw)
              case ("Mdiff_iface")
                do icol=1,ncol
                  write(file_id,"(E24.17E2)",advance='no') &
                        CVmix_vars(icol)%Mdiff_iface(kw)
                  if (icol.ne.ncol) write(file_id, "(1X)", advance='no')
                end do
              case ("Tdiff_iface")
                do icol=1,ncol
                  write(file_id,"(E24.17E2)",advance='no') &
                        CVmix_vars(icol)%Tdiff_iface(kw)
                  if (icol.ne.ncol) write(file_id, "(1X)", advance='no')
                end do
              case ("Sdiff_iface")
                do icol=1,ncol
                  write(file_id,"(E24.17E2)",advance='no') &
                        CVmix_vars(icol)%Sdiff_iface(kw)
                  if (icol.ne.ncol) write(file_id, "(1X)", advance='no')
                end do
              case DEFAULT
                print*, "ERROR: unable to write variable ", var_names(var)
                stop 1
            end select
```

```
          if (var.ne.size(var_names)) write(file_id, "(1X)", advance='no')
        end do
        write(file_id, *)
      end do
    case DEFAULT
      print*, "ERROR: Invalid file type"
      stop 1
  end select
```

1.117 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).

USES:

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

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
```

```
    dims = shape(field)
    add_fill = present(FillVal)
    select case(get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        do i=1,2
          call netcdf_check(nf90_def_dim(file_id, trim(dim_names(i)), dims(i), &
                                         dimids(i)))
        end do
        call netcdf_check(nf90_def_var(file_id, trim(var_name), NF90_DOUBLE,   &
                                       dimids, varid))
        if (add_fill) &
          call netcdf_check(nf90_put_att(file_id, varid, "_FillValue", &
                                         FillVal))
        call netcdf_check(nf90_enddef(file_id))
        call netcdf_check(nf90_put_var(file_id, varid, field))
#endif

      case (ASCII_FILE_TYPE)
        do i=1,dims(1)
          do j=1,dims(2)
            write(file_id, "(E24.17E2)",advance='no') field(i,j)
            if (j.ne.dims(2)) write(file_id, "(1X)", advance='no')
          end do
          write(file_id, *)
        end do
      case DEFAULT
        print*, "ERROR: cvmix_output_write_2d_double only writes to netcdf"
        print*, "(attempt to write ", trim(var_name), " with dimensions ", &
                trim(dim_names(1)), " and ", trim(dim_names(2))
        call cvmix_io_close_all
        stop 1
        ! Dummy code to supress unused variable warnings
        if (add_fill) &
          dims(1) = dims(2)
    end select
```

1.118 cvmix_write_3d_double

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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


    dims = shape(field)
    add_fill = present(FillVal)
    select case(get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        do i=1,3
          call netcdf_check(nf90_def_dim(file_id, trim(dim_names(i)), dims(i), &
                                         dimids(i)))
        end do
        call netcdf_check(nf90_def_var(file_id, trim(var_name), NF90_DOUBLE,   &
                                       dimids, varid))
        if (add_fill) &
          call netcdf_check(nf90_put_att(file_id, varid, "_FillValue", &
                                         FillVal))
        call netcdf_check(nf90_enddef(file_id))
        call netcdf_check(nf90_put_var(file_id, varid, field))
#endif

      case DEFAULT
```

```
        print*, "ERROR: cvmix_output_write_3d_double only writes to netcdf"
        print*, "(attempt to write ", trim(var_name), " with dimensions ", &
                trim(dim_names(1)), ", ", trim(dim_names(2)), ", and ",    &
                trim(dim_names(3))
        call cvmix_io_close_all
        stop 1
        ! Dummy code to supress unused variable warnings
        if (add_fill) &
          dims(1) = dims(2)
    end select
```

1.119 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).

USES:

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

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
```

```
    select case(get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        var_found = .true.
        if (present(var_name)) then
          varid = get_netcdf_varid(file_id, var_name)
          if (varid.eq.-1) then
            print*, "WARNING: can not find variable ", trim(var_name), " in ", &
                    trim(get_file_name(file_id)), "... can not add attribute."
            var_found = .false.
          end if
        else
          varid=NF90_GLOBAL
        end if
        if (var_found) then
          call netcdf_check(nf90_redef(file_id))
          call netcdf_check(nf90_put_att(file_id, varid, trim(att_name), &
                            att_val))
          call netcdf_check(nf90_enddef(file_id))
        end if
#endif
      case DEFAULT
        print*, "ERROR: cvmix_output_write_att_integer only writes to netcdf"
        print*, "(attempted to set attribute ", trim(att_name), " to ", &
                att_val
        if (present(var_name)) &
          print*, "(for variable ", trim(var_name), ")"
        call cvmix_io_close_all
        stop 1
    end select
```

1.120 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).

USES:

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

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
```

```
    select case(get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        var_found = .true.
        if (present(var_name)) then
          varid = get_netcdf_varid(file_id, var_name)
          if (varid.eq.-1) then
            print*, "WARNING: can not find variable ", trim(var_name), " in ", &
                    trim(get_file_name(file_id)), "... can not add attribute."
            var_found = .false.
          end if
        else
          varid=NF90_GLOBAL
        end if
        if (var_found) then
          call netcdf_check(nf90_redef(file_id))
          call netcdf_check(nf90_put_att(file_id, varid, trim(att_name), &
                            att_val))
          call netcdf_check(nf90_enddef(file_id))
        end if
#endif
      case DEFAULT
        print*, "ERROR: cvmix_output_write_att_real only writes to netcdf"
        print*, "(attempted to set attribute ", trim(att_name), " to ", &
                att_val
        if (present(var_name)) &
          print*, "(for variable ", trim(var_name), ")"
        call cvmix_io_close_all
        stop 1
```

```
    end select
```

1.121 cvmix_write_att_string

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

```
    select case(get_file_type(file_id))
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        var_found = .true.
        if (present(var_name)) then
          varid = get_netcdf_varid(file_id, var_name)
          if (varid.eq.-1) then
            print*, "WARNING: can not find variable ", trim(var_name), " in ", &
                    trim(get_file_name(file_id)), "... can not add attribute."
            var_found = .false.
```

```
          end if
        else
          varid=NF90_GLOBAL
        end if
        if (var_found) then
          call netcdf_check(nf90_redef(file_id))
          call netcdf_check(nf90_put_att(file_id, varid, trim(att_name), &
                            trim(adjustl(att_val))))
          call netcdf_check(nf90_enddef(file_id))
        end if
#endif
      case DEFAULT
        print*, "ERROR: cvmix_output_write_att_string only writes to netcdf"
        print*, "(attempted to set attribute ", trim(att_name), " to ", &
                trim(att_val)
        if (present(var_name)) &
          print*, "(for variable ", trim(var_name), ")"
        call cvmix_io_close_all
        stop 1
    end select
```

1.122 cvmix_io_close

INTERFACE:

```
   subroutine cvmix_io_close(file_id)
```

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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


! Is fid in the file database?
nullify(file_to_close)
if (allocated(file_database)) then
  ifile => file_database(1)
  do while(associated(ifile%next))
    if (ifile%file_id.eq.file_id) then
      file_to_close => ifile
    end if
    ifile => ifile%next
  end do
  if (ifile%file_id.eq.file_id) then
     file_to_close => ifile
  end if
end if
file_found = associated(file_to_close)

if (.not.file_found) then
  write(*,"(A,IO,A)") "Warning: file id ", file_id, " is not an open file!"
  return
end if
file_type = file_to_close%file_type

if (associated(file_to_close%prev)) then
  ifile => file_to_close%prev
  if (associated(file_to_close%next)) then
    ifile%next => file_to_close%next
    ifile%next%prev => ifile
  else
    nullify(ifile%next)
  end if
  deallocate(file_to_close)
else
  ! file_id is stored in the first entry
  if (associated(file_database(1)%next)) then
    ! Database has more than one entry, so copy last entry into first
    file_to_close => file_database(1)
    do while(associated(file_to_close%next))
      file_to_close => file_to_close%next
    end do
    ifile => file_to_close%prev
    file_database(1)%file_id   = file_to_close%file_id
    file_database(1)%file_type = file_to_close%file_type
    file_database(1)%file_name = file_to_close%file_name
```

```
        nullify(ifile%next)
        deallocate(file_to_close)
      else
        ! file_id is only entry in database
        deallocate(file_database)
      end if
    end if

    select case (file_type)
#ifdef _NETCDF
      case (NETCDF_FILE_TYPE)
        call netcdf_check(nf90_close(file_id))
#endif
      case (ASCII_FILE_TYPE)
        close(file_id)
      case (BIN_FILE_TYPE)
        close(file_id)
    end select
```

1.123 cvmix_io_close_all

INTERFACE:

```
   subroutine cvmix_io_close_all
```

DESCRIPTION:

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

USES:

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

LOCAL VARIABLES:

```
     integer :: fid
```

```
    write(*,"(A)") "Closing all open files..."
    do while (allocated(file_database))
      fid = file_database(1)%file_id
      write(*, "(A,1X,A)") "...", trim(get_file_name(fid))
```

```
  call cvmix_io_close(fid)
end do
write(*,"(A)") "All files closed."
```

1.124 get_file_name

INTERFACE:

```
function get_file_name(file_id)
```

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

LOCAL VARIABLES:

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

```
ifile => file_database(1)
if (ifile%file_id.eq.file_id) then
  get_file_name = ifile%file_name
  return
end if
do while(associated(ifile%next))
  ifile => ifile%next
  if (ifile%file_id.eq.file_id) then
    get_file_name = ifile%file_name
    return
```

```
  end if
end do
get_file_name = "FILE_NOT_FOUND"
```

1.125 get_file_type

INTERFACE:

```
function get_file_type(file_id)
```

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
integer             :: get_file_type
```

LOCAL VARIABLES:

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

```
ifile => file_database(1)
if (ifile%file_id.eq.file_id) then
  get_file_type = ifile%file_type
  return
end if
do while(associated(ifile%next))
  ifile => ifile%next
  if (ifile%file_id.eq.file_id) then
    get_file_type = ifile%file_type
    return
```

```
  end if
end do
get_file_type = FILE_NOT_FOUND
```

1.126 cvmix_input_get_netcdf_dim

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
integer                      :: cvmix_input_get_netcdf_dim
```

LOCAL VARIABLES:

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


cvmix_input_get_netcdf_dim = -1
if (get_file_type(file_id).ne.NETCDF_FILE_TYPE) then
  print*, "WARNING: can not find dimid, ", trim(get_file_name(file_id)), &
          " is not a netcdf file."
  return
end if

dimid = -1
```

```
! Find number of variables in file
call netcdf_check(nf90_inquire(file_id, nDimensions=ndim))
i = 1
do while((i.le.ndim).and.(dimid.eq.-1))
  ! Loop to figure out if var_name is a valid variable in the file
  call netcdf_check(nf90_inquire_dimension(file_id, i, name=tmp_name))
  if (trim(dim_name).eq.trim(tmp_name)) then
    dimid = i
  else
    i = i+1
  end if
end do
if (dimid.ne.-1) &
  call netcdf_check(nf90_inquire_dimension(file_id, dimid, &
                    len=cvmix_input_get_netcdf_dim))
```

1.127 get_netcdf_varid

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

LOCAL VARIABLES:

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


get_netcdf_varid = -1
if (get_file_type(file_id).ne.NETCDF_FILE_TYPE) then
  print*, "WARNING: can not find varid, ", trim(get_file_name(file_id)), &
          " is not a netcdf file."
  return
end if

! Find number of variables in file
call netcdf_check(nf90_inquire(file_id, nVariables=nvar))
i = 1
do while((i.le.nvar).and.(get_netcdf_varid.eq.-1))
  ! Loop to figure out if var_name is a valid variable in the file
  call netcdf_check(nf90_inquire_variable(file_id, i, name=tmp_name, &
                                          xtype=xtype, ndims=ndims))
  if (trim(var_name).eq.trim(tmp_name)) then
    get_netcdf_varid = i
  else
    i = i+1
  end if
end do
```