In-code documentation for CVMix

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

January 13, 2014

Contents

1 Routine/Function Prologues

1.0.1 cvmix_driver

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

REVISION HISTORY:

```
SVN $Id: cvmix_driver.F90 212 2013-08-12 01:38:09Z mike.levy.work@gmail.com $
SVN $URL: https://cvmix.googlecode.com/svn/trunk/src/cvmix_driver.F90 $
```

INTERFACE:

```
Program cvmix_driver
```

USES:

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

1.0.2 cvmix_BL_pointer_driver

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

REVISION HISTORY:

```
SVN:$Id: cvmix_bgrnd_BL_pointer.F90 246 2013-09-21 21:14:09Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_bgrnd_BL_pointer.F90 $
```

INTERFACE:

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

USES:

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

  Implicit None
```

INPUT PARAMETERS:

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

1.0.3 cvmix_BL_memcopy_driver

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

REVISION HISTORY:

```
SVN:$Id: cvmix_bgrnd_BL_memcopy.F90 246 2013-09-21 21:14:09Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_bgrnd_BL_memcopy.F90 $
```

INTERFACE:

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

USES:

```
  use cvmix_kinds_and_types, only : cvmix_r8,                &
                                    cvmix_data_type,         &
                                    cvmix_global_params_type
  use cvmix_background,      only : cvmix_init_bkgnd,        &
                                    cvmix_coeffs_bkgnd,      &
                                    cvmix_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
  real(cvmix_r8), intent(in) :: ocn_depth   ! Depth of ocn
```

1.0.4 cvmix_shear_driver

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

REVISION HISTORY:

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

INTERFACE:

```
Subroutine cvmix_shear_driver(nlev)
```

USES:

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

  Implicit None
```

INPUT PARAMETERS:

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

1.0.5 cvmix_tidal_driver

A routine to test the Simmons implementation of tidal mixing.

REVISION HISTORY:

```
SVN:$Id: cvmix_tidal_Simmons.F90 235 2013-08-28 23:26:32Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_tidal_Simmons.F90 $
```

INTERFACE:

```
Subroutine cvmix_tidal_driver()
```

USES:

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

  Implicit None
```

1.0.6 cvmix_ddiff_driver

A routine to test the double diffusion mixing module.

REVISION HISTORY:

```
SVN:$Id: cvmix_ddiff_drv.F90 234 2013-08-23 18:07:44Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_ddiff_drv.F90 $
```

INTERFACE:

```
Subroutine cvmix_ddiff_driver(nlev)
```

USES:

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

  Implicit None
```

INPUT PARAMETERS:

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

1.0.7 cvmix_kpp_driver

A routine to test the KPP module.

REVISION HISTORY:

```
SVN:$Id: cvmix_kpp_drv.F90 265 2013-10-29 00:21:52Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/drivers/cvmix_kpp_drv.F90 $
```

INTERFACE:

```
Subroutine cvmix_kpp_driver()
```

USES:

```
  use cvmix_kinds_and_types, only : cvmix_r8,                 &
                                    cvmix_strlen,             &
                                    cvmix_data_type
  use cvmix_kpp,             only : cvmix_init_kpp,                          &
                                    cvmix_put_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,       &
                                    cvmix_output_write_att,   &
                                    cvmix_io_close

  Implicit None
```

1.1 Fortran: Module Interface cvmix_io

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

REVISION HISTORY:

```
SVN:$Id: cvmix_io.F90 247 2013-09-26 23:42:59Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/cvmix_io.F90 $
```

USES:

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

PUBLIC MEMBER FUNCTIONS:

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

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

  interface cvmix_output_write
    module procedure cvmix_output_write_single_col
    module procedure cvmix_output_write_multi_col
    module procedure cvmix_output_write_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(:)
```

1.1.1 cvmix_io_open

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.2 cvmix_input_read_1d_double

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.3 cvmix_input_read_2d_integer

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.4 cvmix_input_read_2d_double

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.5 cvmix_input_read_3d_double

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.6 cvmix_output_write_single_col

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.7 cvmix_output_write_multi_col

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.8 cvmix_write_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
```

1.1.9 cvmix_write_3d_double

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.10 cvmix_write_att_integer

INTERFACE:

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

DESCRIPTION:

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

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
```

1.1.11 cvmix_write_att_real

INTERFACE:

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

DESCRIPTION:

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

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
```

1.1.12 cvmix_write_att_string

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.13 cvmix_io_close

INTERFACE:

```
subroutine cvmix_io_close(file_id)
```

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.14 cvmix_io_close_all

INTERFACE:

```
subroutine cvmix_io_close_all
```

DESCRIPTION:

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

USES:

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

LOCAL VARIABLES:

```
integer :: fid
```

1.1.15 get_file_name

INTERFACE:

```
function get_file_name(file_id)
```

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.1.16 get_file_type

INTERFACE:

```
function get_file_type(file_id)
```

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
integer             :: get_file_type
```

LOCAL VARIABLES:

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

1.1.17 cvmix_input_get_netcdf_dim

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
integer                          :: cvmix_input_get_netcdf_dim
```

LOCAL VARIABLES:

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

1.1.18 get_netcdf_varid

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

LOCAL VARIABLES:

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

1.2 Fortran: Module Interface cvmix_kinds_and_types

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

REVISION HISTORY:

```
SVN:$Id: cvmix_kinds_and_types.F90 248 2013-09-30 21:47:48Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_kinds_and_types.F90 $
```

USES:

```
uses no other modules
```

DEFINED PARAMETERS:

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

! 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 :: zero = 0.0_cvmix_r8,                   &
                                     one  = 1.0_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 levels in column
                                 ! Setting default to -1 might be F95...

    ! Scalar quantities
    real(cvmix_r8) :: ocn_depth, & ! distance from sea level to ocean bottom
                                   ! (positive => below sea level)
```

```
                              ! units: m
                    OBL_depth, & ! distance from sea level to OBL bottom
                              ! (positive => below sea level)
                              ! units: m
                    surf_hgt,  & ! sea surface height
                              ! (positive => above sea level)
                              ! units: m
                    surf_fric, & ! turbulent friction velocity at surface
                              ! units: m/s
                    surf_buoy, & ! buoyancy forcing at surface
                              ! units: m^2 s^-3
                    lat,       & ! latitude of column (degrees north)
                              ! units: can be degrees or radians (there
                              !        are no internal computation based
                              !        on this term)
                    lon,       & ! longitude of column (degrees east)
                              ! units: can be degrees or radians (there
                              !        are no internal computation based
                              !        on this term)
                    Coriolis,  & ! Coriolis parameter
                              ! units: s^-1
                    kOBL_depth   ! index of cell containing OBL (fraction
                              ! > .5 => below cell center)
                              ! units: unitless

  ! Values on interfaces
  ! For KPP, need to store non-local transport term
  ! (:,1) = temperature tracer
  ! (:,2) = salinity / all non-temperature tracers
  ! (:,3) and (:,4) = momentum terms (x- and y-, respectively). Note that
  !                   currently both momentum terms are 0 everywhere
  ! 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.
  ! nlev+1, 4
  real(cvmix_r8), dimension(:,:), pointer :: kpp_transport_iface => NULL()
                                        ! units: unitless (see note above)
  ! nlev+1, 2
  ! diffusivity coefficients at interfaces (2 columns needed for double diff)
  real(cvmix_r8), dimension(:,:), pointer :: diff_iface => NULL()
                                        ! units: m^2/s
  ! nlev+1
  ! viscosity (momentum diffusivity) coefficients at interfaces
  real(cvmix_r8), dimension(:),   pointer :: visc_iface => NULL()
                                        ! units: m^2/s
  ! 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_iface  => NULL()
                                        ! units: m
    ! shear Richardson number at column interfaces
    real(cvmix_r8), dimension(:),   pointer :: Ri_iface   => NULL()
                                        ! units: unitless
    ! For tidal mixing, we need the squared buoyancy frequency
    real(cvmix_r8), dimension(:),   pointer :: buoy_iface => NULL()
                                        ! units: s^-2

    ! Values at tracer points
    ! nlev
    ! Two density values are stored: the actual density of water (dens) and
    ! the density of water after adiabatic displacement to the level below
    ! where the water actually is (dens_lwr)
    real(cvmix_r8), dimension(:),   pointer :: dens     => NULL()
                                        ! units: kg m^-3
    real(cvmix_r8), dimension(:),   pointer :: dens_lwr => NULL()
                                        ! units: kg m^-3
    ! height of cell centers in column (positive up => most are negative)
    real(cvmix_r8), dimension(:),   pointer :: zt       => NULL()
                                        ! units: m
    ! level thicknesses (positive semi-definite)
    real(cvmix_r8), dimension(:),   pointer :: dzt      => NULL()
                                        ! units: m
    ! bulk Richardson number
    real(cvmix_r8), dimension(:),   pointer :: Rib      => 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()
                                        ! units: unitless
    real(cvmix_r8), dimension(:),   pointer :: strat_param_denom => NULL()
                                        ! units: unitless
    ! For KPP we need buoyancy (as opposed to buoyancy frequency) and
    ! velocity (in both x direction and y direction)
    real(cvmix_r8), dimension(:),   pointer :: buoyancy          => NULL()
                                        ! units: m/(s^2)
    real(cvmix_r8), dimension(:),   pointer :: Vx                => NULL()
                                        ! units: m/s
    real(cvmix_r8), dimension(:),   pointer :: Vy                => 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
    integer                         :: max_nlev  ! maximum number of levels
                                       ! units: unitless
    real(cvmix_r8)                  :: prandtl   ! Prandtl number
                                       ! units: unitless
    ! For densities, user must keep track of units (kg/m^3 vs g/cm^3)
    real(cvmix_r8)                  :: fw_rho    ! fresh water density
                                       ! units: kg m^-3
    real(cvmix_r8)                  :: sw_rho    ! salt water density
                                       ! units: kg m^-3
end type cvmix_global_params_type
```

1.3 Fortran: Module Interface cvmix_background

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

REVISION HISTORY:

```
SVN:$Id: cvmix_background.F90 246 2013-09-21 21:14:09Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_background.F90 $
```

USES:

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

PUBLIC MEMBER FUNCTIONS:

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

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

interface cvmix_put_bkgnd
  module procedure cvmix_put_bkgnd_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_visc(:,:) ! ncol, 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_diff(:,:) ! ncol, nlev+1
                                                    ! units: m^2/s

    ! Flag for what to do with old values of CVmix_vars%diff and %visc
    integer :: handle_old_vals

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

1.3.1 cvmix_init_bkgnd_scalar

INTERFACE:

```
subroutine cvmix_init_bkgnd_scalar(bkgnd_diff, bkgnd_visc, 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_diff
real(cvmix_r8),                       intent(in) :: bkgnd_visc
character(len=cvmix_strlen), optional, intent(in) :: old_vals
```

OUTPUT PARAMETERS:

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

1.3.2 cvmix_init_bkgnd_1D

INTERFACE:

```
subroutine cvmix_init_bkgnd_1D(bkgnd_diff, bkgnd_visc, 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_diff
real(cvmix_r8), dimension(:),          intent(in) :: bkgnd_visc
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
```

1.3.3 cvmix_init_bkgnd_2D

INTERFACE:

```
subroutine cvmix_init_bkgnd_2D(bkgnd_diff, bkgnd_visc, 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 2D field.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.3.4 cvmix_init_bkgnd_BryanLewis

INTERFACE:

```
subroutine cvmix_init_bkgnd_BryanLewis(CVmix_vars, bl1, bl2, bl3, bl4,      &
                                       old_vals, CVmix_params_user,         &
                                       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:

```
! 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), target, optional, intent(in) ::          &
                                                 CVmix_params_user
```

OUTPUT PARAMETERS:

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

1.3.5 cvmix_coeffs_bkgnd

INTERFACE:

```
subroutine cvmix_coeffs_bkgnd(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_visc and _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
```

1.3.6 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
```

1.3.7 cvmix_bkgnd_static_diff

INTERFACE:

```
function cvmix_bkgnd_static_diff(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), intent(in) :: CVmix_bkgnd_params_user
integer, optional, intent(in) :: kw, colid
```

OUTPUT PARAMETERS:

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

1.3.8 cvmix_bkgnd_static_visc

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.3.9 cvmix_put_bkgnd_int

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_bkgnd_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.3.10 cvmix_put_bkgnd_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_bkgnd_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.3.11 cvmix_put_bkgnd_real_1D

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.3.12 cvmix_put_bkgnd_real_2D

INTERFACE:

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

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.3.13 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), optional, target, intent(in) ::            &
                                                 CVmix_bkgnd_params_user
```

OUTPUT PARAMETERS:

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

1.4 Fortran: Module Interface cvmix_shear

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

REVISION HISTORY:

```
SVN:$Id: cvmix_shear.F90 258 2013-10-21 15:17:26Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_shear.F90 $
```

USES:

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

PUBLIC MEMBER FUNCTIONS:

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

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

PUBLIC TYPES:

```
! cvmix_shear_params_type contains the necessary parameters for shear mixing
! (currently Pacanowski-Philander or Large et al)
type, public :: cvmix_shear_params_type
    private
    ! Type of shear mixing to run (PP => Pacanowski-Philander, KPP => LMD94)
    character(len=cvmix_strlen) :: mix_scheme
    ! numerator in viscosity term in PP81
    ! See Eqs. (1) and (2)
    real(cvmix_r8) :: PP_nu_zero  ! units: m^2/s
    ! coefficient of Richardson number in denominator of diff / visc terms
```

```
    real(cvmix_r8) :: PP_alpha    ! units: unitless
    ! exponent of denominator in viscosity term
    real(cvmix_r8) :: PP_exp      ! units: unitless
    ! leading coefficient of LMD94 shear mixing formula (max diff / visc)
    ! see Eq. (28b)
    real(cvmix_r8) :: KPP_nu_zero ! units: m^2/s
    ! critical Richardson number value (larger values result in 0 diffusivity
    ! and viscosity)
    real(cvmix_r8) :: KPP_Ri_zero ! units: unitless
    ! Exponent of unitless factor of diff / visc
    real(cvmix_r8) :: KPP_exp     ! units: unitless
end type cvmix_shear_params_type
```

1.4.1 cvmix_init_shear

INTERFACE:

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

DESCRIPTION:

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

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

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

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

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

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.4.2 cvmix_coeffs_shear

INTERFACE:

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

DESCRIPTION:

Computes vertical tracer and velocity mixing coefficients for shear-type mixing 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),           intent(in) :: CVmix_shear_params
! PP mixing requires CVmix_bkgnd_params
type(cvmix_bkgnd_params_type), optional, intent(in) :: CVmix_bkgnd_params
! colid is only needed if CVmix_bkgnd_params%lvary_horizontal is true
integer,                       optional, intent(in) :: colid
logical,                       optional, intent(in) :: no_diff
```

INPUT/OUTPUT PARAMETERS:

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

1.4.3 cvmix_put_shear_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_shear_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.4.4 cvmix_put_shear_str

INTERFACE:

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

DESCRIPTION:

Write a string into a cvmix_shear_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.4.5 cvmix_get_shear_real

INTERFACE:

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

DESCRIPTION:

Read the real value of a cvmix_shear_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.4.6 cvmix_get_shear_str

INTERFACE:

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

DESCRIPTION:

Read the string contents of a cvmix_shear_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.5 Fortran: Module Interface cvmix_tidal

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

REVISION HISTORY:

```
SVN:$Id: cvmix_tidal.F90 258 2013-10-21 15:17:26Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_tidal.F90 $
```

USES:

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

PUBLIC MEMBER FUNCTIONS:

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

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

PUBLIC TYPES:

```
! cvmix_tidal_params_type contains the necessary parameters for tidal mixing
! (currently just Simmons)
type, public :: cvmix_tidal_params_type
   private
   ! 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
   ! 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
  end type cvmix_tidal_params_type
```

1.5.1 cvmix_init_tidal

INTERFACE:

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

DESCRIPTION:

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

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.5.2 cvmix_coeffs_tidal

INTERFACE:

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

DESCRIPTION:

Computes vertical diffusion coefficients for tidal mixing parameterizations.

USES:

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

INPUT PARAMETERS:

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

INPUT/OUTPUT PARAMETERS:

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

1.5.3 cvmix_compute_vert_dep

INTERFACE:

```
function cvmix_compute_vert_dep(CVmix_vars, CVmix_tidal_params)
```

DESCRIPTION:

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

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.5.4 cvmix_put_tidal_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_tidal_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.5.5 cvmix_put_tidal_str

INTERFACE:

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

DESCRIPTION:

Write a string into a cvmix_tidal_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.5.6 cvmix_get_tidal_real

INTERFACE:

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

DESCRIPTION:

Returns the real value of a cvmix_tidal_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.5.7 cvmix_get_tidal_str

INTERFACE:

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

DESCRIPTION:

Returns the string value of a cvmix_tidal_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.6 Fortran: Module Interface cvmix_ddiff

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

REVISION HISTORY:

```
SVN:$Id: cvmix_ddiff.F90 258 2013-10-21 15:17:26Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_ddiff.F90 $
```

USES:

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

PUBLIC MEMBER FUNCTIONS:

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

interface cvmix_put_ddiff
  module procedure cvmix_put_ddiff_real
end interface cvmix_put_ddiff
```

PUBLIC TYPES:

```
! cvmix_ddiff_params_type contains the necessary parameters for double
! diffusion mixing
type, public :: cvmix_ddiff_params_type
  private
  ! 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
  ! 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
  ! leading coefficient in formula for salt-fingering regime for temperature
  ! diffusion (0.7*nu_f in LMD94)
  real(cvmix_r8) :: kappa_ddiff_t      ! 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
end type cvmix_ddiff_params_type
```

1.6.1 cvmix_init_ddiff

INTERFACE:

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

DESCRIPTION:

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

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

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

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

By default, $R_\rho^0 = 2.55$, but that can be changed by setting `strat_param_max` in the code. Similarly, by default $p_1 = 1$ (`ddiff_exp1`), $p_2 = 3$ (`ddiff_exp2`), and

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

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

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

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

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

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

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

USES:

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

INPUT PARAMETERS:

```
real(cvmix_r8),   optional, intent(in) :: strat_param_max, &
                                          kappa_ddiff_t, &
                                          kappa_ddiff_s, &
                                          ddiff_exp1, &
                                          ddiff_exp2, &
                                          mol_diff, &
                                          kappa_ddiff_param1, &
                                          kappa_ddiff_param2, &
                                          kappa_ddiff_param3
```

OUTPUT PARAMETERS:

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

1.6.2 cvmix_coeffs_ddiff

INTERFACE:

```
subroutine cvmix_coeffs_ddiff(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
```

LOCAL VARIABLES:

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

1.6.3 cvmix_put_ddiff_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_ddiff_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.6.4 cvmix_get_ddiff_real

INTERFACE:

```
function cvmix_get_ddiff_real(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
```

1.7 Fortran: Module Interface cvmix_kpp

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

REVISION HISTORY:

```
SVN:$Id: cvmix_kpp.F90 276 2013-12-17 05:48:39Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_kpp.F90 $
```

USES:

```
use cvmix_kinds_and_types, only : cvmix_r8,                     &
                                  zero,                         &
                                  one,                          &
                                  cvmix_data_type
use cvmix_put_get, only :         cvmix_put
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
```

DEFINED PARAMETERS:

```
integer, parameter :: CVMIX_KPP_INTERP_POP = -1
integer, parameter :: CVMIX_KPP_MATCH_BOTH = 1
integer, parameter :: CVMIX_KPP_MATCH_GRADIENT = 2
integer, parameter :: CVMIX_KPP_SIMPLE_SHAPES = 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
! 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

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
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) :: vonkarman      ! von Karman constant
  real(cvmix_r8) :: Cstar          ! coefficient for nonlinear transport
  ! 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
    integer :: MatchTechnique
    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        :: lavg_N_or_Nsqr ! True => N (or Nsqr) at cell center is
                                     !  average of values at interfaces above
                                     !  and below.
                                     ! False => N (or Nsqr) at cell center is
                                     !  set to value at interface below
                                     ! (only used in compute_unresolved_shear)
  end type cvmix_kpp_params_type
```

1.7.1 cvmix_init_kpp

INTERFACE:

```
  subroutine cvmix_init_kpp(ri_crit, vonkarman, Cstar, zeta_m, zeta_s,       &
                            surf_layer_ext, Cv, interp_type, interp_type2,   &
                            MatchTechnique, lEkman, lMonOb, lnoDGat1,        &
                            lavg_N_or_Nsqr, CVmix_kpp_params_user)
```

DESCRIPTION:

Initialization routine for KPP mixing.

USES:

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

INPUT PARAMETERS:

```
real(cvmix_r8),   optional :: ri_crit, &        ! units: unitless
                              vonkarman, &      ! units: unitless
                              Cstar, &          ! units: unitless
                              zeta_m, &         ! units: unitless
                              zeta_s, &         ! units: unitless
                              surf_layer_ext, & ! units: unitless
                              Cv                ! units: unitless
character(len=*), optional :: interp_type, interp_type2, MatchTechnique
logical,          optional :: lEkman, lMonOb, lnoDGat1, lavg_N_or_Nsqr
```

OUTPUT PARAMETERS:

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

1.7.2 cvmix_coeffs_kpp_wrap

INTERFACE:

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

DESCRIPTION:

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

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
```

1.7.3 cvmix_coeffs_kpp_low

INTERFACE:

```
subroutine cvmix_coeffs_kpp_low(diff, visc, zw_iface, zt_cntr, OBL_depth,   &
                                kOBL_depth, nonlocal, surf_fric, surf_buoy, &
                                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
real(cvmix_r8), dimension(:),   intent(in) :: zw_iface, zt_cntr
real(cvmix_r8),                 intent(in) :: OBL_depth, surf_fric,      &
                                              surf_buoy, kOBL_depth
```

INPUT/OUTPUT PARAMETERS:

```
real(cvmix_r8), dimension(:,:), intent(inout) :: diff, nonlocal
real(cvmix_r8), dimension(:),   intent(inout) :: visc
```

1.7.4 cvmix_put_kpp_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_kpp_params_type variable.

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
```

1.7.5 cvmix_put_kpp_int

INTERFACE:

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

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) :: CVmix_kpp_params
```

1.7.6 cvmix_put_kpp_logical

INTERFACE:

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

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) :: CVmix_kpp_params
```

1.7.7 cvmix_get_kpp_real

INTERFACE:

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

DESCRIPTION:

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

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
```

1.7.8 cvmix_kpp_compute_OBL_depth_low

INTERFACE:

```
subroutine cvmix_kpp_compute_OBL_depth_low(Ri_bulk, zw_iface, OBL_depth,    &
                                           kOBL_depth, zt_cntr, surf_fric,  &
                                           surf_buoy, Coriolis,             &
                                           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:

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

OUTPUT PARAMETERS:

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

1.7.9 cvmix_kpp_compute_kOBL_depth

INTERFACE:

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

DESCRIPTION:

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

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
```

1.7.10 cvmix_kpp_compute_enhanced_diff

INTERFACE:

```
subroutine cvmix_kpp_compute_enhanced_diff(diff_ktup, visc_ktup, diff, visc,&
                                           OBL_diff, OBL_visc, 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), dimension(2), intent(in) :: diff_ktup
real(cvmix_r8),               intent(in) :: visc_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), dimension(2), intent(inout) :: diff, OBL_diff
real(cvmix_r8),               intent(inout) :: visc, OBL_visc
```

1.7.11 cvmix_kpp_compute_OBL_depth_wrap

INTERFACE:

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

DESCRIPTION:

Computes the depth of the ocean boundary layer (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
```

1.7.12 cvmix_kpp_compute_bulk_Richardson

INTERFACE:

```
function cvmix_kpp_compute_bulk_Richardson(zt_cntr, delta_buoy_cntr,        &
                                           delta_Vsqr_cntr, Vt_sqr_cntr,    &
                                           ws_cntr, N_iface, Nsqr_iface,    &
                                           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
    real(cvmix_r8), dimension(:), intent(in), optional :: ws_cntr, N_iface,   &
                                                          Nsqr_iface,         &
                                                          Vt_sqr_cntr
    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
```

1.7.13 cvmix_kpp_compute_turbulent_scales_0d

INTERFACE:

```
  subroutine cvmix_kpp_compute_turbulent_scales_0d(sigma_coord, OBL_depth,    &
                                                   surf_buoy_force,           &
                                                   surf_fric_vel, 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
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
```

1.7.14 cvmix_kpp_compute_turbulent_scales_1d

INTERFACE:

```
subroutine cvmix_kpp_compute_turbulent_scales_1d(sigma_coord, OBL_depth,    &
                                                 surf_buoy_force,           &
                                                 surf_fric_vel, 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
type(cvmix_kpp_params_type), intent(in), optional, target ::              &
                                        CVmix_kpp_params_user
```

OUTPUT PARAMETERS:

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

1.7.15 cvmix_kpp_compute_unresolved_shear

INTERFACE:

```
function cvmix_kpp_compute_unresolved_shear(zt_cntr, ws_cntr, N_iface,      &
                                            Nsqr_iface, 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(:), intent(in), optional :: N_iface, Nsqr_iface
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
```

1.7.16 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
```

1.7.17 cvmix_compute_nu_at_OBL_depth

INTERFACE:

```
function compute_nu_at_OBL_depth(interp_type2, layer_depth, layer_nu,       &
                                 OBL_depth, depth_2above, nu_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:

```
integer,                      intent(in) :: interp_type2
! layer_depth = (/depth_above_OBL, depth_below_OBL/)
! layer_nu    = nu at these points
real(cvmix_r8), dimension(2), intent(in) :: layer_depth, layer_nu
real(cvmix_r8),               intent(in) :: OBL_depth
```

```
! nu at iface above the iface above OBL_depth (Not needed for linear
! interpolation or if OBL_depth is in top level
real(cvmix_r8), optional,     intent(in) :: depth_2above, nu_2above
```

OUTPUT PARAMETERS:

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

1.8 Fortran: Module Interface cvmix_convection

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

REVISION HISTORY:

```
SVN:$Id: cvmix_convection.F90 234 2013-08-23 18:07:44Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_convection.F90 $
```

USES:

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

PUBLIC MEMBER FUNCTIONS:

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

interface cvmix_put_conv
  module procedure cvmix_put_conv_real
  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
end type cvmix_conv_params_type
```

1.8.1 cvmix_init_conv

INTERFACE:

```
subroutine cvmix_init_conv(convect_diff, convect_visc, lBruntVaisala,       &
                           BVsqr_convect, 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, target, intent(out) ::           &
                                                    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
```

1.8.2 cvmix_coeffs_conv

INTERFACE:

```
subroutine cvmix_coeffs_conv(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
```

1.8.3 cvmix_put_conv_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_conv_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.8.4 cvmix_put_conv_logical

INTERFACE:

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

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), intent(inout) :: CVmix_conv_params_put
```

1.8.5 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
```

1.9 Fortran: Module Interface cvmix_math

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: cvmix_math.F90 243 2013-09-19 02:37:11Z mike.levy.work@gmail.com $
$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_math.F90 $
```

USES:

```
use cvmix_kinds_and_types, only : cvmix_r8
```

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
```

1.9.1 cvmix_math_poly_interp

INTERFACE:

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

DESCRIPTION:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.9.2 cvmix_math_evaluate_cubic

INTERFACE:

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

DESCRIPTION:

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

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
```

1.10 Fortran: Module Interface cvmix_put_get

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

REVISION HISTORY:

```
SVN:$Id: cvmix_put_get.F90 219 2013-08-13 20:17:17Z mike.levy.work@gmail.com $
SVN:$URL: https://cvmix.googlecode.com/svn/trunk/src/shared/cvmix_put_get.F90 $
```

USES:

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

PUBLIC MEMBER FUNCTIONS:

```
public :: cvmix_put

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

1.10.1 cvmix_put_int

INTERFACE:

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

DESCRIPTION:

Write an integer value into a cvmix_data_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.10.2 cvmix_put_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_data_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.10.3 cvmix_put_real_1D

INTERFACE:

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

DESCRIPTION:

Write an array of real values into a cvmix_data_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.10.4 cvmix_put_global_params_int

INTERFACE:

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

DESCRIPTION:

Write an integer value into a cvmix_global_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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

1.10.5 cvmix_put_global_params_real

INTERFACE:

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

DESCRIPTION:

Write a real value into a cvmix_global_params_type variable.

USES:

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

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

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