GEOS-Chem Reference
4. The Harvard-NASA Emissions Component (HEMCO)

GEOS-Chem Support Team

10 Jul 2018

Contents

1 HEMCO "Core" modules

These contain routines that perform core HEMCO functions, such as establishing the data structures used to store the emissions data, etc.

1.1 Fortran: Module Interface hco_interp_mod.F90

Module HCO_INTERP_MOD contains routines to interpolate input data onto the HEMCO grid. This module contains routine for horizontal regridding between regular grids (MAP_A2A), as well as vertical interpolation amongst GEOS model levels (full ¡–¿ reduced).

Regridding is supported for concentration quantities (default) and index-based values. For the latter, the values in the regridded grid boxes correspond to the value of the original grid that contrbutes most to the given box.

INTERFACE:

```
MODULE HCO_Interp_Mod
```

USES:

```
USE HCO_Types_Mod
USE HCO_Error_Mod
USE HCO_State_Mod,       ONLY : Hco_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: ModelLev_Check
PUBLIC  :: ModelLev_Interpolate
PUBLIC  :: REGRID_MAPA2A
```

PUBLIC MEMBER FUNCTIONS:

```
PRIVATE :: GEOS5_TO_GEOS4_LOWLEV
PRIVATE :: COLLAPSE
PRIVATE :: INFLATE
```

REVISION HISTORY:

```
 30 Dec 2014 - C. Keller - Initialization
 03 Feb 2015 - C. Keller   - Added REGRID_MAPA2A (from hcoio_dataread_mod.F90).
```

1.1.1 Regrid_MAPA2A

Subroutine Regrid_MAPA2A regrids input array NcArr onto the simulation grid and stores the data in list container Lct. Horizontal regridding is performed using MAP_A2A algorithm. Vertical interpolation between GEOS levels (full vs. reduced, GEOS-5 vs. GEOS-4),

is also supported.

This routine can remap concentrations and index-based quantities.

INTERFACE:

```
SUBROUTINE REGRID_MAPA2A ( am_I_Root, HcoState, NcArr, LonE, LatE, Lct, RC )
```

USES:

```
USE REGRID_A2A_Mod,     ONLY : MAP_A2A
USE HCO_FileData_Mod,   ONLY : FileData_ArrCheck
USE HCO_UNIT_MOD,       ONLY : HCO_IsIndexData
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root         ! Are we on the root CPU?
TYPE(HCO_State),  POINTER        :: HcoState          ! HEMCO state object
REAL(sp),         POINTER        :: NcArr(:,:,:,:)    ! 4D input data
REAL(hp),         POINTER        :: LonE(:)           ! Input grid longitude edges
REAL(hp),         POINTER        :: LatE(:)           ! Input grid latitude edges
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ListCont),   POINTER        :: Lct               ! HEMCO list container
INTEGER,          INTENT(INOUT)  :: RC                ! Success or failure?
```

REVISION HISTORY:

```
03 Feb 2015 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.1.2 ModelLev_Check

Subroutine ModelLev_Check checks if the passed number of vertical levels indicates that these are model levels or not.

INTERFACE:

```
SUBROUTINE ModelLev_Check ( am_I_Root, HcoState, nLev, IsModelLev, RC )
```

USES:

```
USE HCO_FileData_Mod,   ONLY : FileData_ArrCheck
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root         ! Are we on the root CPU?
TYPE(HCO_State),  POINTER        :: HcoState          ! HEMCO state object
INTEGER,          INTENT(IN   )  :: nlev              ! number of levels
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(INOUT)  :: IsModelLev        ! Are these model levels?
INTEGER,          INTENT(INOUT)  :: RC                ! Success or failure?
```

REVISION HISTORY:

```
29 Sep 2015 - C. Keller   - Initial version
22 May 2017 - R. Yantosca - Bug fix: Add MERRA2 to the #elif statement
```

1.1.3 ModelLev_Interpolate

Subroutine ModelLev_Interpolate puts 3D data from an arbitrary number of model levels onto the vertical levels of the simulation grid. Since the input data is already on model levels, this is only to inflate/collapse fields between native/reduced vertical levels, e.g. from 72 native GEOS-5 levels onto the reduced 47 levels. The vertical interpolation scheme depends on compiler switches. If none of the compiler switches listed below is used, no vertical interpolation is performed, e.g. the vertical levels of the input grid are retained.

The input data (REGR_4D) is expected to be already regridded horizontally. The 4th dimension of REGR_4D denotes time.

The 3rd dimension of REGR_3D holds the vertical levels. It is assumed that these are model levels, starting at the surface (level 1). If the input data holds 72 input levels, this is interpreted as native data and will be collapsed onto the reduced grid. If the input data holds X ¡=47 levels, these levels are interpreted as levels 1-X of the reduced grid. In other words, input data with 33 levels will be interpreted as 33 levels on the reduced grid, and the data is accordingly mapped onto the simulation grid. If data becomes inflated or collapsed, the output data will always extent over all vertical levels of the simulation grid. If necessary, the unused upper levels will be filled with zeros. If no data interpolation is needed, the vertical extent of the output data is limited to the number of used levels. For instance, if the input data has 5 vertical levels, the output array will only extent over those 5 (bottom) levels.

Currently, this routine can remap the following combinations:

- Native GEOS-5 onto reduced GEOS-5 (72 –¿ 47 levels)
- Reduced GEOS-5 onto native GEOS-5 (47 –¿ 72 levels)
- Native GEOS-4 onto reduced GEOS-4 (55 –¿ 30 levels)
- Reduced GEOS-4 onto native GEOS-4 (30 –¿ 55 levels)
- Native GEOS-5 onto native GEOS-4 (72 –¿ 55 levels)
- Reduced GEOS-5 onto native GEOS-4 (47 –¿ 55 levels)
- Native GEOS-5 onto reduced GEOS-4 (72 –¿ 30 levels)
- Reduced GEOS-5 onto reduced GEOS-4 (47 –¿ 30 levels)

Interpolation from GEOS-5 onto GEOS-4 levels is currently not supported.

INTERFACE:

```
SUBROUTINE ModelLev_Interpolate ( am_I_Root, HcoState, REGR_4D, Lct, RC )
```

USES:

```
USE HCO_FileData_Mod,   ONLY : FileData_ArrCheck
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root         ! Are we on the root CPU?
TYPE(HCO_State),  POINTER        :: HcoState          ! HEMCO state object
REAL(sp),         POINTER        :: REGR_4D(:,:,:,:)  ! 4D input data
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ListCont),   POINTER        :: Lct               ! HEMCO list container
INTEGER,          INTENT(INOUT)  :: RC                ! Success or failure?
```

REVISION HISTORY:

```
30 Dec 2014 - C. Keller   - Initial version
24 Feb 2015 - R. Yantosca - Now exit if vertical interpolation isn't needed
12 Aug 2015 - R. Yantosca - Vertically remap MERRA2 as we do for GEOS-FP
24 Sep 2015 - C. Keller   - Added interpolation on edges.
06 Dec 2015 - C. Keller   - Pass # of GEOS-5 levels to be mapped onto GEOS-4
```

1.1.4 GEOS5_TO_GEOS4_LOWLEV

Helper routine to map the lowest 28 GEOS-5 levels onto the lowest 11 GEOS-4 levels. The individual level weights were calculated offline and are hard-coded here. These are the edge pressure values on the lowest 28 GEOS-5 levels: 1013.25, 998.05, 982.76, 967.47, 952.19, 936.91 921.62, 906.34, 891.05, 875.77, 860.49, 845.21, 829.92, 809.55, 784.08, 758.62, 733.15, 707.69, 682.23, 644.05, 605.87, 567.70, 529.54, 491.40, 453.26, 415.15, 377.07, 339.00, 288.92
And these are the edge pressure values on the lowest 12 GEOS-4 levels: 1013.25, 998.16, 968.49, 914.79, 841.15, 752.89, 655.96, 556.85, 472.64, 401.14, 340.43, 288.92
The value at every given GEOS-4 level is determined from the GEOS-5 values by multiplying the (GEOS-5) input data by the normalized level weights. For instance, the first GEOS-5 level is the only level contributing to the 1st GEOS-4 level. For the 2nd GEOS-4 level, contributions from GEOS-5 levels 1-3 are used. Of GEOS-5 level 1, only 0.7is in GEOS-4 level 1), whereas 100level 3 contribute to GEOS-4 level 2. The corresponding normalized weights become 0.00378,0.515, and 0.481, respectively.

The weights don't always add up to exactly 1.00 due to rounding errors.

INTERFACE:

```
SUBROUTINE GEOS5_TO_GEOS4_LOWLEV( HcoState, Lct, REGR_4D, NZ, T, RC )
```

INPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState          ! HEMCO state object
REAL(sp),         POINTER        :: REGR_4D(:,:,:,:)  ! 4D input data
INTEGER,          INTENT(IN)     :: T                 ! Time index
INTEGER,          INTENT(IN)     :: NZ                ! # of vertical levels to remap.
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ListCont),   POINTER        :: Lct               ! HEMCO list container
INTEGER,          INTENT(INOUT)  :: RC                ! Return code
```

REVISION HISTORY:

```
07 Jan 2015 - C. Keller   - Initial version.
24 Sep 2015 - C. Keller   - Added option to interpolate edges.
06 Dec 2015 - C. Keller   - Added input argument NZ
```

1.1.5 COLLAPSE

Helper routine to collapse input levels onto the output grid. The input data is weighted by the grid box thicknesses defined on top of this module. The input parameter T determines the time slice to be considered, and MET denotes the met field type of the input data (4 = GEOS-4 levels, GEOS-5 otherwise).

INTERFACE:

```
SUBROUTINE COLLAPSE ( Lct, REGR_4D, OutLev, InLev1, NLEV, T, MET )
```

INPUT PARAMETERS:

```
REAL(sp),         POINTER        :: REGR_4D(:,:,:,:)  ! 4D input data
INTEGER,          INTENT(IN)     :: OutLev
INTEGER,          INTENT(IN)     :: InLev1
INTEGER,          INTENT(IN)     :: NLEV
INTEGER,          INTENT(IN)     :: T
INTEGER,          INTENT(IN)     :: MET               ! 4=GEOS-4, else GEOS-5
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ListCont),   POINTER        :: Lct               ! HEMCO list container
```

REVISION HISTORY:

```
30 Dec 2014 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.1.6 INFLATE

Helper routine to inflate input levels onto the output grid. The values on the input data are evenly distributed amongst all output levels.

INTERFACE:

```
SUBROUTINE INFLATE ( Lct, REGR_4D, InLev, OutLev1, NLEV, T )
```

INPUT PARAMETERS:

```
REAL(sp),         POINTER        :: REGR_4D(:,:,:,:)  ! 4D input data
INTEGER,          INTENT(IN)     :: InLev
INTEGER,          INTENT(IN)     :: OutLev1
INTEGER,          INTENT(IN)     :: NLEV
INTEGER,          INTENT(IN)     :: T
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ListCont),   POINTER        :: Lct               ! HEMCO list container
```

REVISION HISTORY:

```
30 Dec 2014 - C. Keller   - Initial version
```

1.2 Fortran: Module Interface hcoio_messy_mod.F90

Module HCOIO_MESSY_MOD interfaces HEMCO with the regridding tool NCREGRID of the Modular Earth Submodel System (MESSy). This regridding scheme is used for vertical regridding and/or for index data, i.e. data with discrete values (e.g. land type integers). This code currently only works for rectilinear (regular lon-lat) grids but can be extended to support curvilinear grids.
REFERENCES:

- Joeckel, P. Technical note: Recursive rediscretisation of geo- scientific data in the Modular Earth Submodel System (MESSy), ACP, 6, 3557–3562, 2006.

INTERFACE:

```
MODULE HCOIO_MESSY_MOD
```

USES:

```
USE HCO_ERROR_MOD
USE HCO_TYPES_MOD,         ONLY : ListCont
USE HCO_STATE_MOD,         ONLY : Hco_State
USE MESSY_NCREGRID_BASE,   ONLY : NARRAY, AXIS

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCO_MESSY_REGRID
```

PRIVATE MEMBER FUNCTIONS:

```
! Set this value to TRUE if you want to reduce the output array
! to the minimum required number of vertical levels.
LOGICAL, PARAMETER            :: ReduceVert = .FALSE.
!MODULE INTERFACES:
```

REVISION HISTORY:

```
 24 Jun 2014 - C. Keller   - Initial version
```

1.2.1 Hco_Messy_Regrid

This is the wrapper routine to regrid a 4D input array NcArr (x,y,z,t) onto the HEMCO emissions grid (defined in HcoState) using the regridding tool NCREGRID. LonEdge, LatEdge and LevEdge are the grid point edges of the input grid. The data is written into list container Lct.

If the input grid is 2D (horizontal only), LevEdge must not be specified (null pointer) and the data is regridded in the horizontal only. If the input grid has only one vertical level, it is assumed that this is the surface level and the output data is 3D but with only one vertical level.

For input data with more than one vertical level, the data is mapped onto the entire 3D grid. The module parameter ReduceVert can be used to cap the output data at the lowest possible level. For example, if the input grid only covers three surface levels with a minimum sigma value of 0.75, vertical regridding is performed within this sigma range (1-0.75) and the output grid is reduced accordingly. This option is not used in the standard HEMCO setup because problems can arise if the data array of a given container suddenly changes its size (i.e. when updated data covers more/less vertical levels than the data beforehand).

The input argument IsModelLev denotes whether or not the vertical coordinates of the input data are on model levels. If set to yes and LevEdge is not provided (i.e. a nullified pointer), the MESSy regridding routines are only used for the horizontal remapping and subroutine ModelLev_Interpolate (module hco_interp_mod.F90) is used for the vertical remapping.

INTERFACE:

```
SUBROUTINE HCO_MESSY_REGRID ( am_I_Root, HcoState,   NcArr,   &
                              LonEdge,   LatEdge,    LevEdge, &
                              Lct,       IsModelLev, RC        )
```

USES:

```
USE HCO_FILEDATA_MOD,     ONLY : FileData_ArrCheck
USE HCO_UNIT_MOD,         ONLY : HCO_IsIndexData
```

```
USE HCO_INTERP_MOD,       ONLY : ModelLev_Interpolate
USE MESSY_NCREGRID_BASE,  ONLY : RG_INT, RG_IDX
USE MESSY_NCREGRID_BASE,  ONLY : NREGRID
USE MESSY_NCREGRID_BASE,  ONLY : INIT_NARRAY
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root       ! Root CPU?
TYPE(HCO_State),  POINTER        :: HcoState        ! HEMCO obj.
REAL(sp),         POINTER        :: ncArr(:,:,:,:)  ! Input array(x,y,z,t)
REAL(hp),         POINTER        :: LonEdge(:)      ! lon edges
REAL(hp),         POINTER        :: LatEdge(:)      ! lat edges
REAL(hp),         POINTER        :: LevEdge(:,:,:)  ! sigma level edges
TYPE(ListCont),   POINTER        :: Lct             ! Target list container
LOGICAL,          INTENT(IN   )  :: IsModelLev      ! Are these model levels?
INTEGER,          INTENT(INOUT)  :: RC              ! Return code
```

REVISION HISTORY:

```
27 Jun 2014 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.2.2 Axis_Create

Subroutine AXIS_CREATE creates a MESSy axis type from the grid defined by mid points Lon, Lat, Lev. Lev must be in (unitless) sigma coordinates: sigma(i,j,l) = p(i,j,l) / p_surface(i,j)

INTERFACE:

```
SUBROUTINE AXIS_CREATE ( am_I_Root, HcoState, lon, lat, lev, ax, RC )
```

USES:

```
USE MESSY_NCREGRID_BASE,  ONLY : INIT_AXIS
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )          :: am_I_Root
TYPE(HCO_State),  POINTER                :: HcoState
TYPE(ListCont),   POINTER                :: Lct
REAL(hp),         POINTER                :: Lon(:)
REAL(hp),         POINTER                :: Lat(:)
REAL(hp),         POINTER                :: Lev(:,:,:)
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(axis),       POINTER                :: ax(:)
INTEGER,          INTENT(INOUT)          :: RC
```

REVISION HISTORY:

```
22 Jun 2014 - C. Keller - Initial version (from messy_ncregrid_geohyb.f90)
```

1.2.3 Axis_Delete

Subroutine AXIS_DELETE deletes the specified MESSy axis.

INTERFACE:

```
SUBROUTINE AXIS_DELETE ( ax1, ax2, RC )
```

USES:

```
USE MESSY_NCREGRID_BASE,  ONLY : INIT_AXIS
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(axis),       POINTER           :: ax1(:)
TYPE(axis),       POINTER           :: ax2(:)
INTEGER,          INTENT(INOUT)     :: RC
```

REVISION HISTORY:

```
28 Aug 2013 - C. Keller - Initial version
```

1.2.4 Hco2Messy

Subroutine HCO2MESSY converts a HEMCO data array into a messy array-structure.

INTERFACE:

```
SUBROUTINE HCO2MESSY( am_I_Root, HcoState, InArr, narr, ax, RC )
```

USES:

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
TYPE(HCO_State),  POINTER        :: HcoState
REAL(sp),         POINTER        :: InArr(:,:,:,:)
TYPE(narray),     POINTER        :: narr(:)
TYPE(axis),       POINTER        :: ax(:)
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
27 Jun 2014 - C. Keller - Initial version
```

1.2.5 Messy2Hco

Subroutine MESSY2HCO converts a MESSy array structure into a HEMCO data array. This is basically the reverse function of HCO2MESSY.

INTERFACE:

```
SUBROUTINE MESSY2HCO( am_I_Root, HcoState, narr, Lct, LEV, Ptr4D, RC )
```

USES:

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )          :: am_I_Root
TYPE(HCO_State),  POINTER                :: HcoState
TYPE(narray),     POINTER                :: narr(:)
TYPE(ListCont),   POINTER                :: Lct
INTEGER,          INTENT(IN   )          :: LEV
REAL(sp),         POINTER                :: Ptr4D(:,:,:,:)
INTEGER,          INTENT(INOUT)          :: RC
```

REVISION HISTORY:

```
27 Jun 2014 - C. Keller - Initial version
```

1.3 Fortran: Module Interface hcoio_read_esmf_mod.F90

Module HCOIO_Read_ESMF_mod is the HEMCO interface for data reading within the ESMF framework.

INTERFACE:

```
MODULE HCOIO_Read_ESMF_mod
```

USES:

```
USE HCO_Types_Mod
USE HCO_Error_Mod
USE HCO_State_Mod,       ONLY : Hco_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
#if defined(ESMF_)
PUBLIC  :: HCOIO_Read_ESMF
```

REVISION HISTORY:

```
22 Aug 2013 - C. Keller   - Initial version
01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
22 Feb 2016 - C. Keller   - Split off from hcoio_dataread_mod.F90
```

1.3.1 HCOIO_DataRead (ESMF/MAPL version)

Interface routine between ESMF and HEMCO to obtain the data array for a given HEMCO data container. The data is obtained through the ExtData interface. The HEMCO source file attribute is taken to identify the ExtData pointer name.

INTERFACE:

```
SUBROUTINE HCOIO_Read_ESMF( am_I_Root, HcoState, Lct, RC )
```

USES:

```
USE ESMF
USE MAPL_mod
USE HCO_FILEDATA_MOD, ONLY : FileData_ArrInit

# include "MAPL_Generic.h"
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
TYPE(HCO_State),  POINTER        :: HcoState
TYPE(ListCont),   POINTER        :: Lct
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
28 Aug 2013 - C. Keller   - Initial version
27 Aug 2014 - R. Yantosca - Err msg now displays hcoio_dataread_mod.F90
```

1.4 Fortran: Module Interface hco_timeshift_mod.F90

Module hco_timeshift_mod.F90 contains routines to shift the file reference time by a given value. Time stamps shifts can be provided as optional fifth element to the time stamp attribute in the HEMCO configuration file.

For instance, consider the case where 3-hourly averages are provided in individual files with centered time stamps, e.g.: file.yyyymmdd_0130z.nc, file.yyyymmdd_0430z.nc, ..., file.yyymmdd_2230z.nc To read these files *at the beginning* of their time intervals, the time stamp can be shifted by 90 minutes, e.g. the file name, variable, and time attribute section reads: ... file.$yyyy*mm*dd_$hh$mnz.nc VARNAME 2000-2016/1-12/1-31/0-23/+90minutes ...

At time 00z, HEMCO will then read file 0130z and keep using this file until 03z, when it switches to file 0430z. Similarly, it is possible to shift the file reference time by any number of years, months, days, or hours. Time shifts can be forward or backward in time (use - sign to shift backwards).

This module contains subroutines to determine the time to be shifted (stored in filedata variable tshift) and shifts the desired reference time as needed.

INTERFACE:

```
MODULE HCO_TIMESHIFT_MOD
```

USES:

```
USE HCO_Error_Mod
USE HCO_State_Mod,  ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: TimeShift_Set
PUBLIC :: TimeShift_Apply
```

REMARKS:

REVISION HISTORY:

```
29 Feb 2016 - C. Keller   - Initial version
```

1.4.1 TimeShift_Set

Subroutine TimeShift_Set sets the time shift values. The time shift attribute tshift contains two entries: the first entry denotes the number of months to be shifted (integer value), while the second entry denotes then number of seconds to be shifted.

INTERFACE:

```
SUBROUTINE TimeShift_Set( HcoConfig, Dta, shift, RC )
```

USES:

```
USE HCO_TYPES_MOD,    ONLY : ListCont
USE HCO_TYPES_MOD,    ONLY : FileData
USE HCO_TYPES_MOD,    ONLY : ConfigObj
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER        :: HcoConfig    ! HEMCO config
TYPE(FileData),   POINTER        :: Dta       ! file container
CHARACTER(LEN=*), INTENT(IN   )  :: shift     ! time shift
INTEGER,          INTENT(INOUT)  :: RC        ! Return code
```

REVISION HISTORY:

```
29 Feb 2016 - C. Keller - Initial version
```

1.4.2 TimeShift_Apply

Subroutine TimeShift_Apply shifts the reference time (provided through arguments yr, mt, dy, hr, and mn, by the time shift specified in the HEMCO configuration file (if specified at all).

INTERFACE:

```
SUBROUTINE TimeShift_Apply( am_I_Root, HcoState, Lct, &
                            yr, mt, dy, hr, mn,  RC )
```

USES:

```
USE Julday_Mod
USE HCO_TYPES_MOD,    ONLY : ListCont
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root ! Root CPU
TYPE(HCO_State),  POINTER        :: HcoState  ! Hemco state
TYPE(ListCont),   POINTER        :: Lct       ! List container
INTEGER,          INTENT(INOUT)  :: yr        ! year
INTEGER,          INTENT(INOUT)  :: mt        ! month
INTEGER,          INTENT(INOUT)  :: dy        ! day
INTEGER,          INTENT(INOUT)  :: hr        ! hour
INTEGER,          INTENT(INOUT)  :: mn        ! minute
INTEGER,          INTENT(INOUT)  :: RC        ! Return code
```

REVISION HISTORY:

```
29 Feb 2016 - C. Keller - Initial version
```

1.5 Fortran: Module Interface hco_scale_mod.F90

Module hco_scale_mod contains a collection of routines to uniformly scale emissions by species-specific scale scale factors.

INTERFACE:

```
MODULE HCO_Scale_Mod
```

USES:

```
USE HCO_Error_Mod

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCO_ScaleInit
PUBLIC :: HCO_ScaleGet
PUBLIC :: HCO_ScaleArr
PUBLIC :: HCO_ScaleFinal
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: HCO_ScaleArr3D_sp
PRIVATE :: HCO_ScaleArr3D_dp
PRIVATE :: HCO_ScaleArr2D_sp
PRIVATE :: HCO_ScaleArr2D_dp
PRIVATE :: HCO_ScaleArr1D_sp
PRIVATE :: HCO_ScaleArr1D_dp
!PRIVATE VARIABLES:
REAL(hp), ALLOCATABLE :: SpcScal(:)
```

REVISION HISTORY:

```
11 May 2017 - C. Keller   - Initial version
```

1.5.1 HCO_ScaleInit

Function HCO_ScaleInit initialized the uniform scale factors for every HEMCO species.

INTERFACE:

```
SUBROUTINE HCO_ScaleInit ( am_I_Root, HcoState, RC )
!USES
  USE HCO_STATE_MOD,    ONLY : HCO_STATE
  USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
  LOGICAL,         INTENT(IN   )  :: am_I_Root      ! Root CPU?
  TYPE(HCO_State), POINTER        :: HcoState       ! HEMCO state object
```

OUTPUT PARAMETERS:

```
  INTEGER,         INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
11 May 2017 - C. Keller - Initial version
```

1.5.2 HCO_ScaleGet

Function HCO_ScaleGet returns the scale factor for the given species ID.

INTERFACE:

```
FUNCTION HCO_ScaleGet ( HcoID ) RESULT ( ScalFact )
!USES
```

INPUT PARAMETERS:

```
  INTEGER,         INTENT(IN   )  :: HcoID          ! HEMCO species ID
!RESULT:
  REAL(hp)                        :: ScalFact
```

REMARKS:

REVISION HISTORY:

```
11 May 2017 - C. Keller - Initial version
```

1.5.3 HCO_ScaleArr3D_sp

Function HCO_ScaleArr3D scales the 3D array.

INTERFACE:

```
SUBROUTINE HCO_ScaleArr3D_sp ( am_I_Root, HcoState, HcoID, Arr3D, RC )
!USES
  USE HCO_STATE_MOD,    ONLY : HCO_STATE
  USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
  LOGICAL,         INTENT(IN   )  :: am_I_Root      ! Root CPU?
  TYPE(HCO_State), POINTER        :: HcoState       ! HEMCO state object
  INTEGER,         INTENT(IN   )  :: HcoID          ! Species ID
```

INPUT/OUTPUT PARAMETERS:

```
  REAL(sp),        INTENT(INOUT)  :: Arr3D( HcoState%NX, &
                                            HcoState%NY, &
                                            HcoState%NZ )
  INTEGER ,        INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
11 May 2017 - C. Keller - Initial version
```

1.5.4 HCO_ScaleArr3D_dp

Function HCO_ScaleArr3D scales the 3D array.

INTERFACE:

```
SUBROUTINE HCO_ScaleArr3D_dp ( am_I_Root, HcoState, HcoID, Arr3D, RC )
!USES
  USE HCO_STATE_MOD,    ONLY : HCO_STATE
  USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
  LOGICAL,          INTENT(IN   )  :: am_I_Root      ! Root CPU?
  TYPE(HCO_State), POINTER         :: HcoState       ! HEMCO state object
  INTEGER,          INTENT(IN   )  :: HcoID          ! Species ID
```

INPUT/OUTPUT PARAMETERS:

```
  REAL(dp),         INTENT(INOUT)  :: Arr3D( HcoState%NX, &
                                             HcoState%NY, &
                                             HcoState%NZ )
  INTEGER ,         INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
 11 May 2017 - C. Keller - Initial version
```

1.5.5 HCO_ScaleArr2D_sp

Function HCO_ScaleArr2D scales the 2D array.

INTERFACE:

```
SUBROUTINE HCO_ScaleArr2D_sp ( am_I_Root, HcoState, HcoID, Arr2D, RC )
!USES
  USE HCO_STATE_MOD,    ONLY : HCO_STATE
  USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
  LOGICAL,          INTENT(IN   )  :: am_I_Root      ! Root CPU?
  TYPE(HCO_State), POINTER         :: HcoState       ! HEMCO state object
  INTEGER,          INTENT(IN   )  :: HcoID          ! Species ID
```

INPUT/OUTPUT PARAMETERS:

```
  REAL(sp),         INTENT(INOUT)  :: Arr2D( HcoState%NX, &
                                             HcoState%NY )
  INTEGER ,         INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
11 May 2017 - C. Keller - Initial version
```

1.5.6 HCO_ScaleArr2D_dp

Function HCO_ScaleArr2D scales the 2D array.

INTERFACE:

```
SUBROUTINE HCO_ScaleArr2D_dp ( am_I_Root, HcoState, HcoID, Arr2D, RC )
!USES
  USE HCO_STATE_MOD,    ONLY : HCO_STATE
  USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
  LOGICAL,          INTENT(IN   )  :: am_I_Root      ! Root CPU?
  TYPE(HCO_State), POINTER         :: HcoState       ! HEMCO state object
  INTEGER,          INTENT(IN   )  :: HcoID          ! Species ID
```

INPUT/OUTPUT PARAMETERS:

```
  REAL(dp),         INTENT(INOUT)  :: Arr2D( HcoState%NX, &
                                             HcoState%NY )
  INTEGER ,         INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
11 May 2017 - C. Keller - Initial version
```

1.5.7 HCO_ScaleArr1D_sp

Function HCO_ScaleArr1D scales a single value.

INTERFACE:

```
SUBROUTINE HCO_ScaleArr1D_sp ( am_I_Root, HcoState, HcoID, Arr1D, RC )
!USES
  USE HCO_STATE_MOD,    ONLY : HCO_STATE
  USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )  :: am_I_Root      ! Root CPU?
    TYPE(HCO_State), POINTER         :: HcoState       ! HEMCO state object
    INTEGER,          INTENT(IN   )  :: HcoID          ! Species ID
```

INPUT/OUTPUT PARAMETERS:

```
    REAL(sp),         INTENT(INOUT)  :: Arr1D
    INTEGER ,         INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
    11 May 2017 - C. Keller - Initial version
```

1.5.8 HCO_ScaleArr1D_dp

Function HCO_ScaleArr1D scales a single value.

INTERFACE:

```
  SUBROUTINE HCO_ScaleArr1D_dp ( am_I_Root, HcoState, HcoID, Arr1D, RC )
  !USES
    USE HCO_STATE_MOD,    ONLY : HCO_STATE
    USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )  :: am_I_Root      ! Root CPU?
    TYPE(HCO_State), POINTER         :: HcoState       ! HEMCO state object
    INTEGER,          INTENT(IN   )  :: HcoID          ! Species ID
```

INPUT/OUTPUT PARAMETERS:

```
    REAL(dp),         INTENT(INOUT)  :: Arr1D
    INTEGER ,         INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
    11 May 2017 - C. Keller - Initial version
```

1.5.9 HCO_ScaleFinal

Function HCO_ScaleFinal finalizes the module.

INTERFACE:

```
SUBROUTINE HCO_ScaleFinal()
!USES
```

INPUT PARAMETERS:

REMARKS:

REVISION HISTORY:

```
11 May 2017 - C. Keller - Initial version
```

1.6 Fortran: Module Interface hcoio_write_std_mod.F90

Module HCOIO_write_std_mod.F90 is the HEMCO data output interface for the 'standard' model environment. It contains routines to write out diagnostics into a netCDF file.

INTERFACE:

```
MODULE HCOIO_WRITE_STD_MOD
```

USES:

```
USE HCO_ERROR_MOD
USE HCO_DIAGN_MOD

IMPLICIT NONE
PRIVATE
#if !defined(ESMF_)
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOIO_WRITE_STD
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: ConstructTimeStamp
```

REMARKS:

```
HEMCO diagnostics are still in testing mode. We will fully activate them
at a later time.  They will be turned on when debugging & unit testing.
```

REVISION HISTORY:

```
04 May 2014 - C. Keller   - Initial version.
11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
11 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
28 Jul 2014 - C. Keller   - Removed GC specific initialization calls and
                            moved to HEMCO core.
05 Aug 2014 - C. Keller   - Added dummy interface for ESMF.
03 Apr 2015 - C. Keller   - Added HcoDiagn_Write
22 Feb 2016 - C. Keller   - Split off from hcoio_diagn_mod.F90
```

1.6.1 HCOIO_write_std

Subroutine HCOIO_write_std writes diagnostics to netCDF file. If the ForceWrite flag is set to TRUE, all diagnostics are written out except they have already been written out during this time step. This option is usually only used at the end of a simulation run. If ForceWrite is False, only the diagnostics that are at the end of their time averaging interval are written. For example, if the current month is different from the previous (emissions) month, all diagnostics with hourly, daily and monthly time averaging intervals are written out. If the optional argument OnlyIfFirst is set to TRUE, diagnostics will only be written out if its nnGetCalls is 1. This can be used to avoid that diagnostics will be written out twice. The nnGetCalls is reset to zero the first time a diagnostics is updated. For diagnostics that point to data stored somewhere else (i.e. that simply contain a data pointer, nnGetCalls is never reset and keeps counting.

INTERFACE:

```
SUBROUTINE HCOIO_write_std( am_I_Root,   HcoState, ForceWrite,  &
                            RC,          PREFIX,   UsePrevTime, &
                            OnlyIfFirst, COL                    )
```

USES:

```
USE m_netCDF_io_define
USE Ncdf_Mod,            ONLY : NC_Create
USE Ncdf_Mod,            ONLY : NC_Close
USE Ncdf_Mod,            ONLY : NC_Var_Def
USE Ncdf_Mod,            ONLY : NC_Var_Write
USE Ncdf_Mod,            ONLY : NC_Get_RefDateTime
USE CHARPAK_Mod,         ONLY : TRANLC
USE HCO_State_Mod,       ONLY : HCO_State
USE JulDay_Mod,          ONLY : JulDay
USE HCO_EXTLIST_MOD,     ONLY : GetExtOpt, CoreNr
USE HCO_Types_Mod,       ONLY : DiagnCont
USE HCO_Clock_Mod

! Parameters for netCDF routines
include "netcdf.inc"
```

INPUT PARAMETERS:

```
LOGICAL,                    INTENT(IN   ) :: am_I_Root   ! root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState    ! HEMCO state object
LOGICAL,                    INTENT(IN   ) :: ForceWrite  ! Write all diagnostics?
CHARACTER(LEN=*), OPTIONAL, INTENT(IN   ) :: PREFIX      ! File prefix
LOGICAL,          OPTIONAL, INTENT(IN   ) :: UsePrevTime ! Use previous time
LOGICAL,          OPTIONAL, INTENT(IN   ) :: OnlyIfFirst ! Only write if nnDiagn is 1
INTEGER,          OPTIONAL, INTENT(IN   ) :: COL         ! Collection Nr.
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC          ! Failure or success
```

REVISION HISTORY:

```
12 Sep 2013 - C. Keller   - Initial version
11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
11 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
19 Feb 2015 - C. Keller   - Added optional argument OnlyIfFirst
23 Feb 2015 - R. Yantosca - Now make Arr1D REAL(sp) so that we can write
                            out lon & lat as float instead of double
06 Nov 2015 - C. Keller   - Output time stamp is now determined from
                            variable OutTimeStamp.
14 Jan 2016 - E. Lundgren - Create netcdf title out of filename prefix
20 Jan 2016 - C. Keller   - Added options DiagnRefTime and DiagnNoLevDim.
03 Mar 2016 - M. Sulprizio- Change netCDF format to netCDF-4
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
21 Jan 2017 - C. Holmes   - Write all variable metadata in define mode, then
                            switch to data mode just once. Much faster
                            writing.
17 Feb 2017 - C. Holmes   - Enable netCDF-4 compression
08 Mar 2017 - R. Yantosca - Use unlimited time dimensions for netCDF files
03 Jan 2018 - R. Yantosca - Added more metadata for COARDS compliance.
                            Also make TIME a 8-byte var to avoid roundoffs
05 Jan 2018 - R. Yantosca - Now print out all index variables as REAL*8
```

1.6.2 ConstructTimeStamp

Subroutine ConstructTimeStamp is a helper routine to construct the time stamp of a given diagnostics collection.

INTERFACE:

```
SUBROUTINE ConstructTimeStamp ( am_I_Root, HcoState, PS, PrevTime, Yr, Mt, Dy, hr, mn, R
```

USES:

```
USE HCO_State_Mod,       ONLY : HCO_State
USE HCO_Clock_Mod
USE JULDAY_MOD
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )    :: am_I_Root    ! Root CPU?
TYPE(HCO_State), POINTER          :: HcoState     ! HEMCO state obj
INTEGER,         INTENT(IN   )    :: PS           ! collecion ID
LOGICAL,         INTENT(IN   )    :: PrevTime     ! Use previous time?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)    :: RC           ! Return code
```

OUTPUT PARAMETERS:

```
INTEGER,         INTENT(  OUT)    :: Yr
INTEGER,         INTENT(  OUT)    :: Mt
INTEGER,         INTENT(  OUT)    :: Dy
INTEGER,         INTENT(  OUT)    :: hr
INTEGER,         INTENT(  OUT)    :: mn
```

REVISION HISTORY:

```
06 Nov 2015 - C. Keller   - Initial version
```

1.7 Fortran: Module Interface hco_logfile_mod

Module HCO_LOGFILE_MOD contains some wrapper routines to write data into the HEMCO logfile.

INTERFACE:

```
MODULE HCO_LOGFILE_MOD
```

USES:

```
USE HCO_ERROR_MOD

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCO_Spec2Log
PUBLIC  :: HCO_PrintList
PUBLIC  :: HCO_PrintDataCont
```

REVISION HISTORY:

```
27 May 2014 - C. Keller   - Initialization
```

1.7.1 hco_spec2log

Subroutine HCO_Spec2Log writes information of a species to the logfile.

INTERFACE:

```
SUBROUTINE HCO_Spec2Log( am_I_Root, HcoState, ID )
```

USES:

```
  USE HCO_STATE_MOD, ONLY : HCO_State
!INPUT PARAMETER
  LOGICAL,          INTENT(IN)     :: am_I_Root  ! Root CPU
  TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
  INTEGER,          INTENT(IN)     :: ID         ! HEMCO species ID
```

REVISION HISTORY:

```
27 May 2014 - C. Keller   - Initialization
```

1.7.2 HCO_PrintList

Subroutine HCO_PrintList displays the content of List.

INTERFACE:

```
    SUBROUTINE HCO_PrintList ( HcoState, List, Verbose )
```

USES:

```
    USE HCO_STATE_MOD,     ONLY : HCO_State
    USE HCO_TYPES_MOD,     ONLY : ListCont
!INPUT ARGUMENTS:
    TYPE(HCO_STATE),POINTER    :: HcoState
    TYPE(ListCont), POINTER    :: List
    INTEGER,        INTENT(IN) :: Verbose
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.7.3 HCO_PrintDataCont

Subroutine HCO_PrintDataCont displays the content of the data container Dct.

INTERFACE:

```
    SUBROUTINE HCO_PrintDataCont ( HcoState, Dct, Verbose )
!USES
    USE HCO_STATE_MOD,     ONLY : HCO_State
    USE HCO_TYPES_MOD,     ONLY : DataCont, HCO_DCTTYPE_BASE
!INPUT ARGUMENTS:
    TYPE(HCO_STATE),POINTER    :: HcoState
    TYPE(DataCont), POINTER    :: Dct
    INTEGER,        INTENT(IN) :: Verbose
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
16 Mar 2015 - M. Sulprizio- Now print min and max values for debugging
```

1.8 Fortran: Module Interface hco_filedata_mod.F90

Module HCO_Filedata_Mod contains routines and variables to handle the HEMCO file data object FileData. FileData holds all information of source file data, such as file name, update frequency, temporal resolution, the data array itself, etc. These values are specified in the HEMCO configuration file. Many of these attributes are primarily used for reading/updating the data from file using the HEMCO generic file reading routines. Within an ESMF environment, these attributes may be obsolete.

FileData consists of the following elements:

- ncFile: path and filename to the source file, as specified in the configuration file.
- ncPara: file parameter (variable) of interest, as specified in the configuration file.
- ncYrs: range of years in the source file, as specified in the configuration file through the timestamp attribute.
- ncMts: range of months in the source file, as specified in the configuration file through the timestamp attribute.
- ncDys: range of days in the source file, as specified in the configuration file through the timestamp attribute.
- ncHrs: range of hours in the source file, as specified in the configuration file through the timestamp attribute.
- CycleFlag: determines how to deal with time stamps that do not correspond to one of the source file time slices. If set to 1, the closest available time slice (in the past) is used (or the first available time slice if model time is before first available time slice). If set to 2, the file data is ignored if model time is outside of the source file range. If CycleFlag is set to 3, an error is returned if none of the file time slices matches the model time.
- MustFind: if yes, the code returns with an error if no field can be found for the given simulation time (and according to the cycle flag and time attribute settings). Only of relevance for cycle flags range and exact.

- UpdtFlag: determines the update frequency of the data. This is currently only used to distinguish containers that are updated on every time step (always) or according to the frequency provided in the HEMCO configuration file via attribute 'srcTime'.
- ncRead: logical denoting whether or not we need to read this data container. ncRead is set to false for containers whose data is directly specified in the configuration file. For internal use only.
- OrigUnit: original unit of data.
- ArbDimName: name of additional (arbitrary) file dimension.
- ArbDimVal : desired value of arbitrary dimension.
- IsConc: Set to true if data is concentration. Concentration data will be added to the concentration array instead of the emission array.
- IsLocTime: Set to true if data is in local time. Defaults to false and becomes only true if data is scalar (e.g. uniform diurnal scale factors), country-specific data (read from ASCII), or weekdaily data.
- V3: vector of 3D fields. For 3D-data, this vector will hold the 3D arrays of all time slices kept in memory (e.g. 24 elements for hourly data).
- V2: vector of 2D fields. For 2D-data, this vector will hold the 2D arrays of all time slices kept in memory (e.g. 24 elements for hourly data).
- tIDx: derived type used for proper indexing of the time slices in memory. Internal use only.
- Cover: data coverage on this CPU: 0=no overlap; 1=full overlap; -1=partial overlap. As determined from the mask regions specified in the configuration file.
- SpaceDim: spatial dimension of data array: 1 = spatially uniform (x=y=z=1); 2 = 2D data (x,y); 3 = 3D data (x,y,z).
- Levels: handling of vertical levels (3D data only). For internal use only.
- nt: time dimension. length of vector V3 or V2. For internal use only.
- DeltaT: time interval between time slices. For internal use only. ID i (e.g. cIDList(3) points to data-container w/ cID = 3).
- DoShare: will be set to True if this file data object is shared by multiple data containers. For internal use only.
- IsInList: will be set to True if this file data object is part of the emissions list EmisList. For internal use only.
- IsTouched: will be set to True as soon as the container becomes touched for the first time. For internal use only.

INTERFACE:

```
MODULE HCO_FileData_Mod
```

USES:

```
USE HCO_TYPES_MOD
USE HCO_ERROR_MOD
USE HCO_ARR_MOD

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: FileData_Init
PUBLIC  :: FileData_Cleanup
PUBLIC  :: FileData_ArrCheck
PUBLIC  :: FileData_ArrIsDefined
PUBLIC  :: FileData_ArrIsTouched
PUBLIC  :: FileData_ArrInit
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: FileData_ArrCheck2D
PRIVATE :: FileData_ArrCheck3D
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller   - Initialization
01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
21 Aug 2014 - C. Keller   - Added concentration
23 Dec 2014 - C. Keller   - Added argument IsInList
06 Oct 2015 - C. Keller   - Added argument MustFind
```

1.8.1 FileData_Init

Subroutine FileData_Init initializes a new (blank) file data object.

INTERFACE:

```
SUBROUTINE FileData_Init( FileDta )
```

INPUT PARAMETERS:

```
TYPE(FileData), POINTER    :: FileDta
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller   - Initialization
21 Aug 2014 - C. Keller   - Added concentration
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.8.2 FileData_Cleanup

Subroutine FileData_Cleanup cleans up the file data object FileDta. If DeepClean is set to False, only the data arrays will be removed.

INTERFACE:

```
SUBROUTINE FileData_Cleanup( FileDta, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(FileData), POINTER    :: FileDta
LOGICAL,        INTENT(IN) :: DeepClean
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
```

1.8.3 FileData_ArrCheck2D

Subroutine FileData_ArrCheck2D allocates the 2D data array vector of the given file data object if it is not yet allocated. If already allocated, it compares the array dimensions against the passed dimensions.

INTERFACE:

```
SUBROUTINE FileData_ArrCheck2D( HcoConfig, FileDta, nx, ny, nt, RC )
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),POINTER       :: HcoConfig ! HEMCO config object
TYPE(FileData), POINTER       :: FileDta   ! file data object
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
INTEGER,        INTENT(IN)    :: nt        ! time dim => vector length
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
```

1.8.4 FileData_ArrCheck3D

Subroutine FileData_ArrCheck3D allocates the 3D data array vector of the given file data object if it is not yet allocated. If already allocated, it compares the array dimensions against the passed dimensions.

INTERFACE:

```
SUBROUTINE FileData_ArrCheck3D( HcoConfig, FileDta, nx, ny, nz, nt, RC )
```

USES:

INPUT PARAMETERS:

```
TYPE(ConfigObj),POINTER       :: HcoConfig ! HEMCO config object
TYPE(FileData), POINTER       :: FileDta  ! Container
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
INTEGER,        INTENT(IN)    :: nz        ! z-dim
INTEGER,        INTENT(IN)    :: nt        ! Time dim => vector length
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
```

1.8.5 FileData_ArrIsDefined

Function FileData_ArrIsDefined returns true if the data array of the given file data object is defined.

INTERFACE:

```
FUNCTION FileData_ArrIsDefined( FileDta ) RESULT( IsDefined )
```

INPUT PARAMETERS:

```
TYPE(FileData), POINTER :: FileDta  ! Container
```

RETURN VALUE:

```
LOGICAL                 :: IsDefined
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
```

1.8.6 FileData_ArrIsTouched

Function FileData_ArrIsTouched returns true if the data array of the given file data object has already been touched, e.g. if the data has already been read (or at least attempted to being read). This information is mostly important for file data objects that are shared by multiple data containers. See ReadList_Fill in hco_readlist_mod.F90 for more details.

INTERFACE:

```
FUNCTION FileData_ArrIsTouched( FileDta ) RESULT( IsTouched )
```

INPUT PARAMETERS:

```
TYPE(FileData), POINTER :: FileDta  ! Container
```

RETURN VALUE:

```
LOGICAL                 :: IsTouched
```

REVISION HISTORY:

```
17 Mar 2015 - C. Keller - Initial version
```

1.8.7 FileData_ArrInit2D

Subroutine FileData_ArrInit2D is a wrapper routine to initialize 2D data arrays of a file data object. To ensure proper functioning of the file data object and related routines, this routine should always be used to initialize file data arrays (and NOT HCO_ArrInit directly!).

INTERFACE:

```
SUBROUTINE FileData_ArrInit2D( FileDta, nt, nx, ny, RC )
```

USES:

INPUT PARAMETERS:

```
TYPE(FileData), POINTER       :: FileDta  ! Container
INTEGER,        INTENT(IN)    :: nt        ! Time dim => vector length
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
01 Oct 2014 - C. Keller - Initial version
```

1.8.8 FileData_ArrInit3D

Subroutine FileData_ArrInit3D is a wrapper routine to initialize 3D data arrays of a file data object. To ensure proper functioning of the file data object and related routines, this routine should always be used to initialize file data arrays (and NOT HCO_ArrInit directly!).

INTERFACE:

```
SUBROUTINE FileData_ArrInit3D( FileDta, nt, nx, ny, nz, RC )
```

USES:

INPUT PARAMETERS:

```
TYPE(FileData), POINTER       :: FileDta  ! Container
INTEGER,        INTENT(IN)    :: nt        ! Time dim => vector length
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
INTEGER,        INTENT(IN)    :: nz        ! z-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
```

1.9 Fortran: Module Interface hcoio_dataread_mod.F90

Module HCOIO_DataRead_Mod controls data processing for HEMCO. Depending on the model environment (standard, ESMF, etc.), it invokes the corresponding routines to read the data from file, convert units as required, and interpolate the data onto the model grid.

Currently, HEMCO can read data from the following data sources:

- Gridded data from netCDF file. The netCDF file should adhere to the COARDS conventions and can hold data on resolutions different than then simulation grid. Regridding is performed as part of the data reading. Routine HCOIO_DataRead is the driver routine to read data from netCDF file. In an ESMF environment, this routine simply calls down to the MAPL/ESMF - generic I/O routines. In a non-ESMF environment, the HEMCO generic reading and remapping algorithms are used. Those support vertical regridding, unit conversion, and more (see below).
- Scalar data directly specified in the HEMCO configuration file. If multiple values - separated by the separator sign (/) - are provided, they are interpreted as temporally changing values: 7 values = Sun, Mon, ..., Sat; 12 values = Jan, Feb, ..., Dec; 24 values = 0am, 1am, ..., 23pm (local time!). For masks, exactly four values must be provided, interpreted as lower left and upper right mask box corners (lon1/lat1/lon2/lat2).
- Country-specific data specified in a separate ASCII file. This file must end with the suffix '.txt' and hold the country specific values listed by country ID. The IDs must correspond to the IDs of a corresponding (netCDF) mask file. The container name of this mask file must be given in the first line of the file, and must be listed HEMCO configuration file. ID 0 is reserved for the default values, applied to all countries with no specific values listed. The .txt file must be structured as follows:

Country mask file name CountryMask

CountryName CountryID CountryValues DEFAULT 0 1.0/1.0/1.0/1.0/1.0/1.0/1/0
USA 840 0.8/0.9/1.0/1.1/1.2/1.1/0.9

The CountryValues are interpreted the same way as scalar values, except that they are applied to all grid boxes with the given country ID.

Outside of an ESMF environment, the GEOS-Chem netCDF reading utilities are used to read netCDF data from disk. The selection of the time slice to be read depends on the current simulation time and the datetime settings assigned to a given data container (set in the configuration file). These settings include:

- datetime range (srcTime), given as YYYY/MM/DD/hh. These can be given as fixed date (e.g. 2010/1/1/0), ranges (e.g. 1990-2010/1/1/0 or 2000-2100/1-12/0/0-23), or using tokens (e.g. $YYYY$/MM/1/0). Data is automatically updated if a 'dynamic' time attribute is given. For instance, for attribute $YYYY/1/1/0 the file will be updated every year, attribute$ every day, etc. The date time tokens are replaced with the current simulation datetime. If a range is provided, only time stamps within the given range are being used. If there is no exact match between the preferred datetime (determined from srcTime) and the time slices in the netCDF file, the cycle flag determines what time slice index is selected.

- Cycling behavior. This determines what to do if there is no exact match between preferred datetime and available datetimes of a file. The options are cycling (C, default), range (R), exact (E), and interpolation (I). If cycling is used, data is recycled if the simulation time is outside of the available data time interval. If cycling is set to range, a data container is ignored if the *simulation* time is outside the given range. For example, if the range is set to 2010-2015/1-12/1/0, this data container is used for simulation dates between 2010 and 2015. If the actual netCDF file data is outside that range, the closest available time slice is selected using the algorithm described below. If cycling is set to exact, HEMCO returns w/ an error if no time slice can be found in the netCDF file that exactly matches the preferred time slices. Finally, if interpolation is selected, data will be interpolated between two time slices if the current preferred datetime cannot be found in the input data. If the preferred datetime does not match with any of the ncdf datetimes, the following method is used to select the time slice to be used: If the preferred datetime is within the range of the available dates, the closest available time stamp in the past is used in most cases. For example, assume a file contains January data between 2005 and 2010, and a simulation starts on July 2007. In this case, the data from Jan 2007 will be used and replaced with Jan 2008 as soon as the simulation date changes to 2008. If the datetimes of the netCDF file contain discontinuities (e.g. don't have the same time interval between all time stamps), an attempt is made to maintain the highest cycling frequency. For instance, if a file contains monthly data for years 2005 and 2020 and the srcTime attribute is set to $YYYY/1-12/1/0. For July 2008, this will use the data from July 2005, and not December 2005 (which would be the closest$ $0 TEST file$YYYY.nc VAL 2005-2010/1/1/0 I ...

INTERFACE:

```
MODULE HCOIO_DataRead_Mod
```

USES:

```
USE HCO_Types_Mod
USE HCO_Error_Mod
USE HCO_CharTools_Mod
USE HCO_State_Mod,       ONLY : Hco_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCOIO_DataRead
```

REVISION HISTORY:

```
 22 Aug 2013 - C. Keller   - Initial version
 01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 22 Feb 2016 - C. Keller   - Environment specific routines are now
                             in respective modules.
```

1.9.1 HCOIO_DataRead

Routine HCOIO_DataRead invokes the appropriate data reading routines for the given model environment.

INTERFACE:

```
SUBROUTINE HCOIO_DataRead( am_I_Root, HcoState, Lct, RC )
```

USES:

```
#if defined(ESMF_)
    USE HCOIO_READ_ESMF_MOD,  ONLY : HCOIO_READ_ESMF
#else
    USE HCOIO_READ_STD_MOD,   ONLY : HCOIO_READ_STD
#endif
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )  :: am_I_Root
    TYPE(HCO_State),  POINTER        :: HcoState
    TYPE(ListCont),   POINTER        :: Lct
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
 28 Aug 2013 - C. Keller   - Initial version
 27 Aug 2014 - R. Yantosca - Err msg now displays hcoio_dataread_mod.F90
 22 Feb 2016 - C. Keller   - Now calls down to model-specific routines.
 24 Mar 2016 - C. Keller   - Removed LUN and CloseFile. Not needed any more.
```

1.9.2 HCO_CharSplit_R8

Subroutine HCO_CharSplit_R8 splits the passed character string into N real8 values, using character SEP as separator. Wildcard values (WC) are set to -999.

INTERFACE:

```
SUBROUTINE HCO_CharSplit_R8( CharStr, SEP, WC, Reals, N, RC )
```

USES:

```
USE CharPak_Mod,  ONLY : StrSplit
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN   ) :: CharStr   ! Character string
CHARACTER(LEN=1), INTENT(IN   ) :: SEP       ! Separator
CHARACTER(LEN=1), INTENT(IN   ) :: WC        ! Wildcard character
```

OUTPUT PARAMETERS:

```
REAL(dp),         INTENT(  OUT) :: Reals(:)  ! Output values
INTEGER,          INTENT(  OUT) :: N         ! # of valid values
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC         ! Return code
```

REVISION HISTORY:

```
18 Sep 2013 - C. Keller - Initial version (update)
```

1.9.3 HCO_CharSplit_R4

Subroutine HCO_CharSplit_R4 splits the passed character string into N real4 values, using character SEP as separator. Wildcard values (WC) are set to -999.

INTERFACE:

```
SUBROUTINE HCO_CharSplit_R4( CharStr, SEP, WC, Reals, N, RC )
```

USES:

```
USE CharPak_Mod,  ONLY : StrSplit
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN   ) :: CharStr    ! Character string
CHARACTER(LEN=1), INTENT(IN   ) :: SEP        ! Separator
CHARACTER(LEN=1), INTENT(IN   ) :: WC         ! Wildcard character
```

OUTPUT PARAMETERS:

```
REAL(sp),         INTENT(  OUT) :: Reals(:)   ! Output values
INTEGER,          INTENT(  OUT) :: N          ! # of valid values
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC         ! Return code
```

REVISION HISTORY:

```
18 Sep 2013 - C. Keller - Initial version (update)
```

1.9.4 HCO_CharSplit_Int

Subroutine HCO_CharSplit_Int splits the passed character string into N integers, using character SEP as separator. Wildcard values (WC) are set to -999.

INTERFACE:

```
SUBROUTINE HCO_CharSplit_INT( CharStr, SEP, WC, Ints, N, RC )
```

USES:

```
USE CharPak_Mod, ONLY : StrSplit
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN   ) :: CharStr    ! Character string
CHARACTER(LEN=1), INTENT(IN   ) :: SEP        ! Separator
CHARACTER(LEN=1), INTENT(IN   ) :: WC         ! Wildcard character
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT) :: Ints(:)    ! Output values
INTEGER,          INTENT(  OUT) :: N          ! # of valid values
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC         ! Return code
```

REVISION HISTORY:

```
18 Sep 2013 - C. Keller - Initial version (update)
```

1.9.5 HCO_CharMatch

Subroutine HCO_CharMatch returns the index of each vector element of vec1 in vec2. nnmatch denotes the number of vec1 elements which have a matching counterpart in vec2. For example, if vec1 is (/ 'NO', 'CO', 'ALK4', 'HBr' /), and vec2 is (/ 'CO', 'NO', 'CH3Br' /), then matchidx becomes (/ 2, 1, -1, -1 /) and nnmatch is 2.

INTERFACE:

```
SUBROUTINE HCO_CharMatch( vec1, n1, vec2, n2, matchidx, nnmatch )
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN   ) :: vec1(n1)     ! char. vector 1
INTEGER,          INTENT(IN   ) :: n1           ! len of vec1
CHARACTER(LEN=*), INTENT(IN   ) :: vec2(n2)     ! char. vector 2
INTEGER,          INTENT(IN   ) :: n2           ! len of vec2
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT) :: matchidx(n1) ! index of vec2 in vec1
INTEGER,          INTENT(  OUT) :: nnmatch      ! # of matches
```

REVISION HISTORY:

```
18 Sep 2013 - C. Keller - Initial version (update)
```

1.9.6 HCO_CharParse

Routine HCO_CharParse parses the provided character string by searching for tokens such as $ROOT, $YYYY, etc., within the string and replacing those values by the intendend characters.

The following list shows the 'default' HEMCO tokens. These are available in any HEMCO simulation. Tokens $ROOT, $MET, and $RES are internally stored as a HEMCO option in module hco_extlist_mod.F90 (see subroutine HCO_SetDefaultToken).

- $ROOT: will be replaced by the root path specified in the settings section of the configuration file.
- $MET: will be replaced by the met-field token.
- $RES: will be replaced by the resolution token.
- $YYYY: will be replaced by the (4-digit) year according to the source time settings set in the configuration file.
- $MM: will be replaced by the (2-digit) month according to the source time settings set in the configuration file.
- $DD: will be replaced by the (2-digit) day according to the source time settings set in the configuration file.
- $HH: will be replaced by the (2-digit) hour according to the source time settings set in the configuration file.
- $MN: will be replaced by the (2-digit) minute.

INTERFACE:

```
SUBROUTINE HCO_CharParse ( HcoConfig, str, yyyy, mm, dd, hh, mn, RC )
```

USES:

```
USE HCO_ExtList_Mod, ONLY  : HCO_GetOpt, HCO_Root
USE HCO_Types_Mod,   ONLY  : ConfigObj
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER        :: HcoConfig
INTEGER,          INTENT(IN   )  :: yyyy  ! replace $YYYY with this value
INTEGER,          INTENT(IN   )  :: mm    ! replace $MM with this value
INTEGER,          INTENT(IN   )  :: dd    ! replace $DD with this value
INTEGER,          INTENT(IN   )  :: hh    ! replace $HH with this value
INTEGER,          INTENT(IN   )  :: mn    ! replace $MN with this value
```

OUTPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(  OUT)  :: str   ! string to be parsed
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC         ! return code
```

REVISION HISTORY:

```
01 Oct 2014 - C. Keller - Initial version
20 Sep 2015 - C. Keller - Tokens can now be any option setting set in the
                          HEMCO configuration file.
07 Jul 2017 - C. Keller - Extended list of token delimiters.
```

1.9.7 HCO_GetBase

Routine HCO_GetBase returns the base location of the given file. This is the entire file path up to the last forward slash, e.g. for file '/home/dir/Config.rc', the base is '/home/dir/'

INTERFACE:

```
SUBROUTINE HCO_GetBase ( str, base, RC )
```

USES:

```
USE CharPak_Mod, ONLY : StrSplit
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN   )  :: str   ! string to be checked
```

OUTPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(  OUT)  :: base  ! base
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC         ! return code
```

REVISION HISTORY:

```
16 Mar 2015 - C. Keller - Initial version
```

1.9.8 IsInWord

Function IsInWord checks if the word InString contains the sequence of SearchString.
INTERFACE:

```
FUNCTION IsInWord( InString, SearchString ) RESULT ( Cnt )
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN   ) :: InString
CHARACTER(LEN=*), INTENT(IN   ) :: SearchString
```

RETURN VALUE:

```
LOGICAL                         :: Cnt
```

REVISION HISTORY:

```
23 Oct 2012 - C. Keller - Initial Version
```

1.9.9 NextCharPos

Function NextCharPos returns the position of the next occurrence of character CHR in word WORD, starting from position START. Returns -1 if the word does not contain CHR at all (after position START).

INTERFACE:

```
FUNCTION NextCharPos ( WORD, CHR, START ) RESULT ( POS )
```

USES:

```
!INPUT ARGUMENTS:
  CHARACTER(LEN=*),           INTENT(IN)  :: WORD
  CHARACTER(LEN=1),           INTENT(IN)  :: CHR
  INTEGER,          OPTIONAL, INTENT(IN)  :: START
!RETURN ARGUMENT:
  INTEGER                                 :: POS
```

REVISION HISTORY:

```
09 Jul 2014 - C. Keller - Initial Version
```

1.9.10 GetNextLine

Subroutine GetNextLine returns the next line.

INTERFACE:

```
SUBROUTINE GetNextLine( am_I_Root, LUN, LINE, EOF, RC )
```

USES:

INPUT PARAMETERS:

```
  LOGICAL,          INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
  INTEGER,          INTENT(IN   ) :: LUN         ! Stream to read from
!OUTPUT PARAMETERS
  CHARACTER(LEN=*), INTENT(  OUT) :: LINE        ! Next (valid) line in stream
!INPUT/OUTPUT PARAMETERS
  LOGICAL,          INTENT(INOUT) :: EOF         ! End of file encountered?
  INTEGER,          INTENT(INOUT) :: RC          ! Success or failure?
```

REVISION HISTORY:

```
10 Apr 2015 - C. Keller - Initial Version
```

1.9.11 HCO_ReadLine

Subroutine HCO_Line reads a line from the provided stream.

INTERFACE:

```
SUBROUTINE HCO_ReadLine( LUN, LINE, EOF, RC )
```

INPUT PARAMETERS:

```
  INTEGER,          INTENT(IN   ) :: LUN      ! Stream LUN
```

OUTPUT PARAMETERS:

```
  CHARACTER(LEN=*), INTENT(INOUT) :: LINE     ! Line
  LOGICAL,          INTENT(INOUT) :: EOF      ! End of file?
  INTEGER,          INTENT(INOUT) :: RC       ! Return code
```

REVISION HISTORY:

```
18 Sep 2013 - C. Keller - Initial version (adapted from B. Yantosca's code)
15 Jul 2014 - R. Yantosca - Remove dependency on routine IOERROR
```

1.10 Fortran: Module Interface hco_restart_mod.F90

Module HCO_RESTART_MOD contains wrapper routines to define, get and write restart fields.

Restart variables are required by some of the HEMCO extensions. The HEMCO restart variables can be organized through the HEMCO restart diagnostics collection. At the end of a simulation, all diagnostic fields ('containers') of the restart collection are written to the

HEMCO restart file.

All fields from the HEMCO restart file can be easily read back into HEMCO via the HEMCO I/O infrastructure, i.e. by listing them in the HEMCO configuration file.

In an ESMF/MAPL environment, restart variables should be organized through the ESMF internal state object. This is particularly important for simulation that rely on checkpoint files (e.g. replay simulations). In this cases, the restart variables are obtained from / written to the ESMF internal state object, rather than the HEMCO restart file.

This module contains wrapper routines to define, obtain and write restart fields for the two aforementioned restart types. The routines work both for 'traditional' and ESMF restart variables. In an ESMF application, the first check is always performed within the internal state, e.g. it is first checked if the given field variable exists in the internal state of the gridded component that HEMCO sits in. If so, the field is obtained from / written to the internal state. If no internal state object exist, an attempt is made to obtain the field through the HEMCO data list, i.e. it is checked if the restart variable is specified in the HEMCO configuration file. A HEMCO diagnostics container is created in the diagnostics restart collection in both the traditional and the ESMF environment.

Routine HCO_RestartDefine should be called during the initialization stage. Routine HCO_RestartGet should be called on the first run call and after each rewinding of the clock. HEMCO routines HcoClock_First and HcoClock_Rewind can be used to determine if it's time to read/update the restart variable. HCO_RestartWrite should be called *on every time step*. This is important in ESMF applications that rely on checkpoint files (e.g. replay simulations) that are written out by ESMF/MAPL throughout the simulation. In a non-ESMF environment, the HCO_RestartWrite call is basically void but it should be called nevertheless.

INTERFACE:

```
MODULE HCO_RESTART_MOD
```

USES:

```
  USE HCO_ERROR_MOD

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  ! defined in all environment
  PUBLIC :: HCO_RestartDefine
  PUBLIC :: HCO_RestartGet
  PUBLIC :: HCO_RestartWrite
```

PRIVATE MEMBER FUNCTIONS:

```
#if defined(ESMF_)
  PRIVATE :: HCO_CopyFromIntnal_ESMF
#endif
```

```
INTERFACE HCO_RestartDefine
   MODULE PROCEDURE HCO_RestartDefine_3D
   MODULE PROCEDURE HCO_RestartDefine_2D
END INTERFACE HCO_RestartDefine

INTERFACE HCO_RestartGet
   MODULE PROCEDURE HCO_RestartGet_3D
   MODULE PROCEDURE HCO_RestartGet_2D
END INTERFACE HCO_RestartGet

INTERFACE HCO_RestartWrite
   MODULE PROCEDURE HCO_RestartWrite_3D
   MODULE PROCEDURE HCO_RestartWrite_2D
END INTERFACE HCO_RestartWrite
```

REVISION HISTORY:

```
 10 Mar 2015 - C. Keller   - Initial version
```

1.10.1 HCO_RestartDefine_3D

Subroutine HCO_RestartDefine_3D defines a restart diagnostics. This adds a diagnostics with output frequency 'End' to the HEMCO diagnostics list. Arr3D is the 3D field of interest. The diagnostics will not copy the current content of Arr3D but establish a 'link' (e.g. pointer) to it. This way, any updates to Arr3D will automatically be seen by the diagnostics and there is no need to explicitly update the content of the diagnostics.

INTERFACE:

```
SUBROUTINE HCO_RestartDefine_3D( am_I_Root, HcoState, Name, Arr3D, &
                                 Unit,      RC                     )
```

USES:

```
  USE HCO_DIAGN_MOD,    ONLY : Diagn_Create
  USE HCO_STATE_MOD,    ONLY : HCO_State
!INPUT ARGUMENTS:
  LOGICAL,             INTENT(IN   )         :: am_I_Root ! Root CPU?
  TYPE(HCO_State),     POINTER               :: HcoState  ! HEMCO state obj.
  CHARACTER(LEN=*),    INTENT(IN   )         :: Name      ! Name of restart variable
  ! Array with data of interest
  REAL(sp),            INTENT(IN   ), TARGET :: Arr3D(HcoState%NX,HcoState%NY,HcoState%NZ
  CHARACTER(LEN=*),    INTENT(IN   )         :: Unit      ! Units of Arr3D
!INPUT/OUTPUT ARGUMENTS:
  INTEGER,             INTENT(INOUT)         :: RC        ! Return code
```

REVISION HISTORY:

```
 11 Mar 2015 - C. Keller - Initial version
```

1.10.2 HCO_RestartDefine_2D

Subroutine HCO_RestartDefine_2D defines a restart diagnostics. This adds a diagnostics with output frequency 'End' to the HEMCO diagnostics list. Arr2D is the 2D field of interest. The diagnostics will not copy the current content of Arr2D but establish a 'link' (e.g. pointer) to it. This way, any updates to Arr2D will automatically be seen by the diagnostics and there is no need to explicitly update the content of the diagnostics.

INTERFACE:

```
SUBROUTINE HCO_RestartDefine_2D( am_I_Root, HcoState, Name, Arr2D, &
                                 Unit,      RC                     )
```

USES:

```
  USE HCO_DIAGN_MOD,    ONLY : Diagn_Create
  USE HCO_STATE_MOD,    ONLY : HCO_State
!INPUT ARGUMENTS:
  LOGICAL,             INTENT(IN   )         :: am_I_Root ! Root CPU?
  TYPE(HCO_State),     POINTER               :: HcoState  ! HEMCO state obj.
  CHARACTER(LEN=*),    INTENT(IN   )         :: Name      ! Name of restart variable
  ! Array with data of interest
  REAL(sp),            INTENT(IN   ), TARGET :: Arr2D(HcoState%NX,HcoState%NY)
  CHARACTER(LEN=*),    INTENT(IN   )         :: Unit      ! Units of Arr2D
!INPUT/OUTPUT ARGUMENTS:
  INTEGER,             INTENT(INOUT)         :: RC        ! Return code
```

REVISION HISTORY:

```
 11 Mar 2015 - C. Keller - Initial version
```

1.10.3 HCO_RestartGet_3D

Subroutine HCO_RestartGet_3D attempts to read a restart field. In an ESMF environment, it first checks if the given field (name) is included in the internal state object, in which case the data object is filled with these values. If not found or if not in an ESMF environment, the HEMCO data list (specified in the HEMCO configuration file) is searched. A default value can be specified in case that no field could be imported via ESMF and/or the HEMCO interface.

INTERFACE:

```
SUBROUTINE HCO_RestartGet_3D( am_I_Root, HcoState, Name,   Arr3D, &
                              RC,        FILLED,   Def3D,  DefVal  )
```

USES:

```
  USE HCO_STATE_MOD,    ONLY : HCO_State
  USE HCO_EMISLIST_MOD, ONLY : HCO_GetPtr
!INPUT ARGUMENTS:
```

```
  LOGICAL,             INTENT(IN   )           :: am_I_Root ! Root CPU?
  TYPE(HCO_State),     POINTER                 :: HcoState  ! HEMCO state object
  CHARACTER(LEN=*),    INTENT(IN   )           :: Name      ! Name of restart variable
  ! Default value to be used if restart variable could not be found
  REAL(sp),            INTENT(IN   ), OPTIONAL :: Def3D(HcoState%NX,HcoState%NY,HcoState%
  ! Default uniform value to be used if restart variable could not be found and
  ! Def2D is not defined.
  REAL(sp),            INTENT(IN   ), OPTIONAL :: DefVal
!OUTPUT ARGUMENTS:
  LOGICAL,             INTENT(  OUT), OPTIONAL :: FILLED    ! Was the restart variable fo
!INPUT/OUTPUT ARGUMENTS:
  ! Data field with restart variable
  REAL(sp),            INTENT(INOUT)           :: Arr3D(HcoState%NX,HcoState%NY,HcoState%
  INTEGER,             INTENT(INOUT)           :: RC       ! Return code
```

REVISION HISTORY:

```
11 Mar 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.10.4 HCO_RestartGet_2D

Subroutine HCO_RestartGet_2D attempts to read a restart field. In an ESMF environment, it first checks if the given field (name) is included in the internal state object, in which case the data object is filled with these values. If not found or if not in an ESMF environment, the HEMCO data list (specified in the HEMCO configuration file) is searched. A default value can be specified in case that no field could be imported via ESMF and/or the HEMCO interface.

INTERFACE:

```
SUBROUTINE HCO_RestartGet_2D( am_I_Root, HcoState, Name,   Arr2D, &
                              RC,        FILLED,   Def2D,  DefVal  )
```

USES:

```
  USE HCO_STATE_MOD,    ONLY : HCO_State
  USE HCO_EMISLIST_MOD, ONLY : HCO_GetPtr
!INPUT ARGUMENTS:
  LOGICAL,             INTENT(IN   )           :: am_I_Root ! Root CPU?
  TYPE(HCO_State),     POINTER                 :: HcoState  ! HEMCO state object
  CHARACTER(LEN=*),    INTENT(IN   )           :: Name      ! Name of restart variable
  ! Default value to be used if restart variable could not be found
  REAL(sp),            INTENT(IN   ), OPTIONAL :: Def2D(HcoState%NX,HcoState%NY)
  ! Default uniform value to be used if restart variable could not be found and
  ! Def2D is not defined.
  REAL(sp),            INTENT(IN   ), OPTIONAL :: DefVal
!OUTPUT ARGUMENTS:
```

```
  LOGICAL,             INTENT(  OUT), OPTIONAL :: FILLED    ! Was the restart variable fo
!INPUT/OUTPUT ARGUMENTS:
  ! Data field with restart variable
  REAL(sp),            INTENT(INOUT)           :: Arr2D(HcoState%NX,HcoState%NY)
  INTEGER,             INTENT(INOUT)           :: RC       ! Return code
```

REVISION HISTORY:

```
11 Mar 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.10.5 HCO_RestartWrite_3D

Subroutine HCO_RestartWrite_3D writes a restart variable to the ESMF internal state. This is only of relevance in an ESMF environment. The 'regular' HEMCO diagnostics created in HCO_RestartDefine becomes automatically written to disk.

INTERFACE:

```
SUBROUTINE HCO_RestartWrite_3D( am_I_Root, HcoState, Name, Arr3D, RC, FOUND )
```

USES:

```
  USE HCO_STATE_MOD,    ONLY : HCO_State
!INPUT ARGUMENTS:
  LOGICAL,             INTENT(IN   )           :: am_I_Root
  TYPE(HCO_State),     POINTER                 :: HcoState
  CHARACTER(LEN=*),    INTENT(IN   )           :: Name
!OUTPUT ARGUMENTS:
  LOGICAL,             INTENT(  OUT), OPTIONAL :: FOUND
!INPUT/OUTPUT ARGUMENTS:
  REAL(sp),            INTENT(INOUT)           :: Arr3D(HcoState%NX,HcoState%NY,HcoState%
  INTEGER,             INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
11 Mar 2015 - C. Keller - Initial version
```

1.10.6 HCO_RestartWrite_2D

Subroutine HCO_RestartWrite_2D writes a restart variable to the ESMF internal state. This is only of relevance in an ESMF environment. The 'regular' HEMCO diagnostics created in HCO_RestartDefine becomes automatically written to disk.

INTERFACE:

```
SUBROUTINE HCO_RestartWrite_2D( am_I_Root, HcoState, Name, Arr2D, RC, FOUND )
```

USES:

```
    USE HCO_STATE_MOD,    ONLY : HCO_State
  !INPUT ARGUMENTS:
    LOGICAL,             INTENT(IN   )           :: am_I_Root
    TYPE(HCO_State),     POINTER                 :: HcoState
    CHARACTER(LEN=*),    INTENT(IN   )           :: Name
  !OUTPUT ARGUMENTS:
    LOGICAL,             INTENT(  OUT), OPTIONAL :: FOUND
  !INPUT/OUTPUT ARGUMENTS:
    REAL(sp),            INTENT(INOUT)           :: Arr2D(HcoState%NX,HcoState%NY)
    INTEGER,             INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
11 Mar 2015 - C. Keller - Initial version
```

1.10.7 HCO_CopyFromIntnal_ESMF

Subroutine HCO_CopyFromIntnal_ESMF attempts to transfer data to and from the ESMF/MAPL internal state.

INTERFACE:

```
  SUBROUTINE HCO_CopyFromIntnal_ESMF ( am_I_Root, HcoState,  Name,    &
                                       Direction, Found, RC, Arr2D, Arr3D )
```

USES:

```
#include "MAPL_Generic.h"
    USE ESMF
    USE MAPL_Mod
    USE HCO_STATE_MOD,   ONLY : Hco_State
```

ARGUMENTS:

```
    LOGICAL,             INTENT(IN   )           :: am_I_Root
    TYPE(HCO_State),     POINTER                 :: HcoState
    CHARACTER(LEN=*),    INTENT(IN   )           :: Name
    INTEGER,             INTENT(IN   )           :: Direction    ! 1: internal to Arr2D; -1
    LOGICAL,             INTENT(  OUT)           :: Found
    INTEGER,             INTENT(INOUT)           :: RC
    REAL(sp),            INTENT(INOUT), OPTIONAL :: Arr2D(HcoState%NX,HcoState%NY)
    REAL(sp),            INTENT(INOUT), OPTIONAL :: Arr3D(HcoState%NX,HcoState%NY,HcoState%
```

REVISION HISTORY:

```
10 Mar 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11 Fortran: Module Interface hco_diagn_mod.F90

Module HCO_Diagn_mod contains routines and variables to handle the HEMCO diagnostics. The HEMCO diagnostics consist of a flexible suite of diagnostics container organized in list DiagnList. Each diagnostics container contains information about the diagnostics type (extension number, emission category / hierarchy, species ID), data structure (Scalar, 2D, 3D), and output units (mass, area, time).

The HEMCO diagnostics module can store multiple, independent diagnostics 'collections', identifiable through the assigned collection number. Each collection has an output frequency assigned to it, as well as an output file name (prefix). All containers of the same collection will have the same output frequency. Currently, the following output frequencies are defined: 'Hourly', 'Daily', 'Monthly', 'Annually', 'End', 'Manual'.

HEMCO has three default built-in diagnostic collections: default, manual, and restart. These three collections become automatically defined during initialization of HEMCO, and diagnostic containers can be added to them anytime afterwards. The output frequency of the default collection can be specified in the HEMCO configuration file through argument 'DiagnFreq'. This can be a character indicating the output frequency (valid entries are 'Always', 'Hourly', 'Daily', 'Monthly', 'Annually', 'Manual', and 'End') or by two integer strings of format '00000000 000000' denoting the year-month-day and hour-minute- second output interval, respectively. For example, setting DiagnFreq to '00000001 000000' would be equivalent to setting it to 'Daily'. A value of '00000000 030000' indicates that the diagnostics shall be written out every 3 hours.

The restart collection always gets an output frequency of 'End', but writing its content to disk can be forced at any given time using routine HcoDiagn_Write (see below). The manual diagnostics has an output frequency of 'Manual', which means that its content is never written to disk. Instead, its fields need to be fetched explicitly from other routines via routine Diagn_Get.

The public module variables HcoDiagnIDDefault, HcoDiagnIDManual, and HcoDiagnRestart can be used to refer to these collections. The user can also define its own collections. It is recommended to do this outside of this module, e.g. at the model - HEMCO interface.

Diagnostic collections are written to disk using the routines in module hcoio_diagn_mod.F90. Routine HcoDiagn_Write will write out the three built-in HEMCO collections. Other collections need be written out explicitly using routine HCOIO_Diagn_WriteOut. The HEMCO option 'HcoWritesDiagn' determines if the three HEMCO collections are automatically written out by the HEMCO driver routines (hco_driver_mod.F90). If HcoWritesDiagn is set to FALSE, the user can freely decide when to write out the diagnostics. This is useful if the HEMCO diagnostics contain fields that are used/filled outside of HEMCO.

Diagnostics container are created at the beginning of a simulation using subroutine Diagn_Create. During the simulation, content is added to the individual containers via Diagn_Update. Diagnostics data is fetched using Diagn_Get. All emissions are stored in units of [kg/m2] and only converted to desired output units when returning the data. The container variable IsOutFormat denotes whether data is currently stored in output units

or internal units. Variable nnGetCalls counts the number of times a diagnostics is called through Diagn_Get without updating its content. This is useful if you want to make sure that data is only written once per time step.

There are two types of emission diagnostics: automatic ('AutoFill') and manual diagnostics. AutoFill diagnostics become automatically filled during execution of HEMCO. AutoFill diagnostics can be at species level (level 1), ExtNr level (level 2), emission category level (level 3), or hierarchy level (level 4). Level 1 diagnostics write out the collected emissions of the specified species, level 2 diagnostics write out emissions for the given ExtNr only (ignoring emissions from all other ExtNr's), etc. Manual diagnostics can represent any content. They never become filled automatically and all update calls (Diagn_Update) have to be set manually.

!
Individual diagnostics are identified by its name and/or container ID. Both are specified when creating the diagnostics (Diagn_Create).

Before adding diagnostics to a collection, the collection needs to be created using subroutine DiagnCollection_Create. The collection number argument (COL) should always be specified when creating, editing or obtaining a diagnostics. If this argument is omitted, the default HEMCO collection (HcoDiagnIDDefault) is taken.

INTERFACE:

```
MODULE HCO_Diagn_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Types_Mod
USE HCO_Arr_Mod
USE HCO_Clock_Mod
USE HCO_State_Mod, ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HcoDiagn_AutoUpdate
PUBLIC  :: HcoDiagn_Init
PUBLIC  :: Diagn_Create
PUBLIC  :: Diagn_Update
PUBLIC  :: Diagn_Get
PUBLIC  :: Diagn_TotalGet
PUBLIC  :: Diagn_AutoFillLevelDefined
PUBLIC  :: Diagn_Print
PUBLIC  :: Diagn_DefineFromConfig
PUBLIC  :: DiagnCont_Find
PUBLIC  :: DiagnCollection_Create
```

```
PUBLIC  :: DiagnCollection_Cleanup
PUBLIC  :: DiagnCollection_Get
PUBLIC  :: DiagnCollection_Set
PUBLIC  :: DiagnCollection_GetDefaultDelta
PUBLIC  :: DiagnCollection_IsTimeToWrite
PUBLIC  :: DiagnCollection_LastTimesSet
PUBLIC  :: DiagnFileOpen
PUBLIC  :: DiagnFileGetNext
PUBLIC  :: DiagnFileClose
PUBLIC  :: DiagnBundle_Cleanup
PUBLIC  :: DiagnBundle_Init
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: DiagnList_Cleanup
PRIVATE :: DiagnCont_Init
PRIVATE :: DiagnCont_PrepareOutput
PRIVATE :: DiagnCont_Link_2D
PRIVATE :: DiagnCont_Link_3D
PRIVATE :: DiagnCont_Cleanup
PRIVATE :: DiagnCollection_DefineID
PRIVATE :: DiagnCollection_Find
PRIVATE :: Diagn_UpdateDriver
PRIVATE :: Diagn_UpdateSp0d
PRIVATE :: Diagn_UpdateSp2d
PRIVATE :: Diagn_UpdateSp3d
PRIVATE :: Diagn_UpdateDp0d
PRIVATE :: Diagn_UpdateDp2d
PRIVATE :: Diagn_UpdateDp3d

INTERFACE Diagn_Update
   MODULE PROCEDURE Diagn_UpdateSp0d
   MODULE PROCEDURE Diagn_UpdateSp2d
   MODULE PROCEDURE Diagn_UpdateSp3d
   MODULE PROCEDURE Diagn_UpdateDp0d
   MODULE PROCEDURE Diagn_UpdateDp2d
   MODULE PROCEDURE Diagn_UpdateDp3d
END INTERFACE
```

REVISION HISTORY:

```
 19 Dec 2013 - C. Keller   - Initialization
 08 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 08 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 01 Aug 2014 - C. Keller   - Added manual output frequency
 12 Aug 2014 - C. Keller   - Added cumulative sum option
 09 Jan 2015 - C. Keller   - Added diagnostics collections
 03 Apr 2015 - C. Keller   - Now tie output frequency to collection instead
                             of individual diagnostic containers.
```

```
    06 Nov 2015 - C. Keller   - Added argument OutTimeStamp to collection to
                                control the file output time stamp (beginning,
                                middle, end of diagnostics interval).
    25 Jan 2016 - R. Yantosca - Added bug fixes for pgfortran compiler
    19 Sep 2016 - R. Yantosca - Add extra overloaded functions to the
                                Diagn_Update interface to avoid Gfortran errors
```

1.11.1 HcoDiagn_autoupdate

Subroutine HCODIAGN_AUTOUPDATE updates the AutoFill diagnostics at species level. This routine should be called after running HEMCO core and all extensions.

INTERFACE:

```
SUBROUTINE HcoDiagn_AutoUpdate( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_STATE_MOD, ONLY : HCO_GetHcoID
USE HCO_STATE_MOD, ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root  ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
INTEGER,          INTENT(INOUT)  :: RC         ! Failure or success
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller   - Initial version
11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
11 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.2 HcoDiagn_Init

Subroutine HCODIAGN_INIT initializes the three built-in HEMCO diagnostic collections: default, restart, and manual. The identification ID of each collection is written into public variable HcoDiagnIDDefault, HcoDiagnIDRestart, and HcoDiagnIDManual, respectively. Those are used to easily refer to one of the diagnostics when adding fields ('containers') to a collection or fetching it's content.

INTERFACE:

```
SUBROUTINE HcoDiagn_Init( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : HCO_GetHcoID
USE HCO_STATE_MOD,   ONLY : HCO_State
USE HCO_ExtList_Mod, ONLY : GetExtOpt
USE HCO_ExtList_Mod, ONLY : CoreNr
USE CHARPAK_MOD,     ONLY : TRANLC
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root  ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
INTEGER,          INTENT(INOUT)  :: RC         ! Failure or success
```

REVISION HISTORY:

```
03 Apr 2015 - C. Keller   - Initial version
10 Apr 2015 - C. Keller   - Now create diagnostics based on entries
                            in the HEMCO diagnostics definition file.
06 Nov 2015 - C. Keller   - Added OutTimeStamp.
01 Nov 2017 - E. Lundgren - Change default OutTimeStamp from end to start
                            for diagnostics collection
29 Dec 2017 - C. Keller   - Added datetime tokens to file prefixes.
```

1.11.3 Diagn_DefineFromConfig

Subroutine Diagn_DefineFromConfig defines HEMCO diagnostic containers as specified in the diagnostics input file.

This routine reads information from a HEMCO diagnostics definition file (specified in the main HEMCO configuration file) and creates HEMCO diagnostic containers for each entry of the diagnostics definition file. Each line of the diagnostics definition file represents a diagnostics container and is expected to consist of 7 entries: container name (character), HEMCO species (character), extension number (integer), emission category (integer), emission hierarchy (integer), space dimension (2 or 3), output unit (character).

The HEMCO setting 'DiagnFile' can be used to specify a diagnostics file. This setting should be placed in the settings section of the HEMCO configuration file.

If argument 'Add2MaplExp' is set to true, the diagnostics field defined in the diagnostics definition file are not added to the HEMCO diagnostics collection (yet), but rather added to the MAPL export state. This is useful in an ESMF environment to automate the coupling of HEMCO diagnostics, e.g. subroutine Diagn_DefineFromConfig can be called during SetServices to make sure that all diagnostic fields defined in DiagnFile have a corresponding Export state object (and can thus be written out via the MAPL History component).

INTERFACE:

```
SUBROUTINE Diagn_DefineFromConfig( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_CharTools_Mod
USE CHARPAK_Mod,       ONLY : STRREPL, STRSPLIT
USE inquireMod,        ONLY : findFreeLUN
USE HCO_STATE_MOD,     ONLY : HCO_GetHcoID
USE HCO_STATE_MOD,     ONLY : HCO_State
USE HCO_EXTLIST_MOD,   ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root   ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER                 :: HcoState    ! HEMCO state object
INTEGER,          INTENT(INOUT)           :: RC          ! Failure or success
```

REVISION HISTORY:

```
10 Apr 2015 - C. Keller   - Initial version
21 Feb 2016 - C. Keller   - Added default diagnostics (optional)
```

1.11.4 Diagn_Create

Subroutine Diagn_Create creates a new diagnostics. This routine takes the following input arguments:

- am_I_Root: is this the root CPU?
- cName: distinct diagnostics (container) name.
- long_name: long_name attribute used for netCDF output.
- ExtNr: emissions extension number.
- Cat: emissions category.
- Hier: emissions hierarchy.
- HcoID: HEMCO species ID of diagnostics species.
- SpaceDim: spatial dimension: 1 (scalar), 2 (lon-lat), or 3 (lon-lat-lev).
- OutUnit: output unit. Emissions will be converted to this unit. Conversion factors will be determined using the HEMCO unit module (see HCO_UNITS_Mod.F90). No unit conversions will be performed if the argument OutOper is set (see below).
- HcoState: HEMCO state object. Used to determine the species properties if any of arguments MW_g, EmMW_g or MolecRatio is missing.
- OutOper: output operation for non-standard units. If this argument is used, the specified operation is performed and all unit specifications are ignored. Can be one of 'Mean', 'Sum', 'CumulSum', or 'Instantaneous'.

- AutoFill: containers with an AutoFill flag of 1 will be auto- matically updated by the HEMCO standard diagnostics calls (e.g. in hco_calc_mod.F90). If set to 0, the diagnostics updates have to be set manually.
- Trgt2D: 2D target array. If specified, the diagnostics array will point to this data. This disables all time averaging, unit conversions, etc., and the data will be written to disk as is.
- Trgt3D: as Trgt2D, but for 3D data.
- MW_g: species molecular weight. Used to determine unit conversion factors. Not needed for target containers or if argument OutOper is specified. Can be omitted if HcoState is given.
- EmMW_g: Molecular weight of emitted species. Used to determine unit conversion factors. Not needed for target containers or if argument OutOper is specified. Can be omitted if HcoState is given.
- MolecRatio: Molecules of species per emitted molecule. Used to determine unit conversion factors. Not needed for target containers or if argument OutOper is specified. Can be omitted if HcoState is given.
- ScaleFact: constant scale factor. If provided, the diagnostics are scaled uniformly by this value before outputting. Will be applied on top of any other unit conversions. Does not work on data pointers.
- cID: assigned container ID. Useful for later reference to this diagnostics container.
- RC: HEMCO return code.

INTERFACE:

```
SUBROUTINE Diagn_Create( am_I_Root, HcoState,   cName,      &
                         ExtNr,     Cat,        Hier,       &
                         HcoID,     SpaceDim,   OutUnit,    &
                         OutOper,   LevIdx,     AutoFill,   &
                         Trgt2D,    Trgt3D,     MW_g,       &
                         EmMW_g,    MolecRatio, ScaleFact,  &
                         cID,       RC,    COL, OkIfExist,  &
                         long_name                           )
```

USES:

```
USE HCO_State_Mod, ONLY : HCO_State
USE HCO_Unit_Mod,  ONLY : HCO_Unit_GetMassScal
USE HCO_Unit_Mod,  ONLY : HCO_Unit_GetAreaScal
USE HCO_Unit_Mod,  ONLY : HCO_Unit_GetTimeScal
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root     ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState      ! HEMCO state obj.
CHARACTER(LEN=*), INTENT(IN   )           :: cName         ! Diagnostics name
```

```
CHARACTER(LEN=*), INTENT(IN   )           :: OutUnit       ! Output units
INTEGER,          INTENT(IN   ), OPTIONAL :: SpaceDim      ! Spatial dimension
INTEGER,          INTENT(IN   ), OPTIONAL :: ExtNr         ! Extension #
INTEGER,          INTENT(IN   ), OPTIONAL :: Cat           ! Category
INTEGER,          INTENT(IN   ), OPTIONAL :: Hier          ! Hierarchy
INTEGER,          INTENT(IN   ), OPTIONAL :: HcoID         ! HEMCO species ID
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: OutOper       ! Output operation
INTEGER,          INTENT(IN   ), OPTIONAL :: LevIdx        ! Level index to use
INTEGER,          INTENT(IN   ), OPTIONAL :: AutoFill      ! 1=fill auto.;0=don't
REAL(sp),         INTENT(IN   ), OPTIONAL :: Trgt2D(:,:)   ! 2D target data
REAL(sp),         INTENT(IN   ), OPTIONAL :: Trgt3D(:,:,:) ! 3D target data
REAL(hp),         INTENT(IN   ), OPTIONAL :: MW_g          ! species MW (g/mol)
REAL(hp),         INTENT(IN   ), OPTIONAL :: EmMW_g        ! emission MW (g/mol)
REAL(hp),         INTENT(IN   ), OPTIONAL :: MolecRatio    ! molec. emission ratio
REAL(hp),         INTENT(IN   ), OPTIONAL :: ScaleFact     ! uniform scale factor
INTEGER,          INTENT(IN   ), OPTIONAL :: COL           ! Collection number
INTEGER,          INTENT(IN   ), OPTIONAL :: cID           ! Container ID
LOGICAL,          INTENT(IN   ), OPTIONAL :: OkIfExist     ! Is it ok if already exists
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: long_name     ! long name attribute
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC            ! Return code
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller - Initialization
05 Mar 2015 - C. Keller - container ID can now be set by the user
31 Mar 2015 - C. Keller - added argument OkIfExist
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.5 Diagn_UpdateSp0d

Subroutine Diagn_UpdateSp0d is the wrapper routine to update the diagnostics for single precision scalar values. It invokes the main diagnostics update routine with the appropriate arguments.

INTERFACE:

```
SUBROUTINE Diagn_UpdateSp0d( am_I_Root,   HcoState, cID,   cName,   ExtNr, &
                             Cat,         Hier,   HcoID,   AutoFill,       &
                             Scalar,      Total,  PosOnly, COL,            &
                             MinDiagnLev, RC                       )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root      ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState       ! HEMCO state obj
INTEGER,          INTENT(IN   ), OPTIONAL :: cID            ! Assigned
```

```
                                                                !  container ID
    CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: cName          ! Diagnostics
                                                                !  name
    INTEGER,          INTENT(IN   ), OPTIONAL :: ExtNr          ! Extension #
    INTEGER,          INTENT(IN   ), OPTIONAL :: Cat            ! Category
    INTEGER,          INTENT(IN   ), OPTIONAL :: Hier           ! Hierarchy
    INTEGER,          INTENT(IN   ), OPTIONAL :: HcoID          ! HEMCO species
                                                                !  ID number
    INTEGER,          INTENT(IN   ), OPTIONAL :: AutoFill       ! 1=yes; 0=no;
                                                                ! -1=either
    REAL(sp),         INTENT(IN   )           :: Scalar         ! 0D scalar
    REAL(sp),         INTENT(IN   ), OPTIONAL :: Total          ! Total
    LOGICAL,          INTENT(IN   ), OPTIONAL :: PosOnly        ! Use only vals
                                                                !  >= 0?
    INTEGER,          INTENT(IN   ), OPTIONAL :: COL            ! Collection Nr.
    INTEGER,          INTENT(IN   ), OPTIONAL :: MinDiagnLev    ! minimum diagn level
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)           :: RC             ! Return code
```

REVISION HISTORY:

```
    20 Apr 2015 - C. Keller - Initialization
    19 Sep 2016 - R. Yantosca - Rewritten for Gfortran: remove Scalar and
                                Array3d (put those in other overloaded methods)
```

1.11.6 Diagn_UpdateSp2d

Subroutine Diagn_UpdateSp2d is the wrapper routine to update the diagnostics for single precision 2-D arrays. It invokes the main diagnostics update routine with the appropriate arguments.

INTERFACE:

```
  SUBROUTINE Diagn_UpdateSp2d( am_I_Root,   HcoState, cID,   cName,   ExtNr,     &
                               Cat,         Hier,  HcoID,   AutoFill,  &
                               Array2D,     Total, PosOnly, COL,       &
                               MinDiagnLev, RC                        )
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )           :: am_I_Root      ! Root CPU?
    TYPE(HCO_State),  POINTER                 :: HcoState       ! HEMCO state obj
    INTEGER,          INTENT(IN   ), OPTIONAL :: cID            ! Assigned
                                                                !  container ID
    CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: cName          ! Diagnostics
                                                                !  name
    INTEGER,          INTENT(IN   ), OPTIONAL :: ExtNr          ! Extension #
```

```
INTEGER,          INTENT(IN   ), OPTIONAL :: Cat           ! Category
INTEGER,          INTENT(IN   ), OPTIONAL :: Hier          ! Hierarchy
INTEGER,          INTENT(IN   ), OPTIONAL :: HcoID         ! HEMCO species
                                                           !  ID number
INTEGER,          INTENT(IN   ), OPTIONAL :: AutoFill      ! 1=yes; 0=no;
                                                           ! -1=either
REAL(sp),         INTENT(IN   )           :: Array2D(:,:)  ! 2D array
REAL(sp),         INTENT(IN   ), OPTIONAL :: Total         ! Total
LOGICAL,          INTENT(IN   ), OPTIONAL :: PosOnly       ! Use only vals
                                                           !  >= 0?
INTEGER,          INTENT(IN   ), OPTIONAL :: COL           ! Collection Nr.
INTEGER,          INTENT(IN   ), OPTIONAL :: MinDiagnLev   ! minimum diagn level
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC            ! Return code
```

REVISION HISTORY:

```
20 Apr 2015 - C. Keller - Initialization
19 Sep 2016 - R. Yantosca - Rewritten for Gfortran: remove Scalar and
                            Array3d (put those in other overloaded methods)
```

1.11.7 Diagn_UpdateSp3d

Subroutine Diagn_UpdateSp is the wrapper routine to update the diagnostics for single precision 3-D arrays. It invokes the main diagnostics update routine with the appropriate arguments.

INTERFACE:

```
SUBROUTINE Diagn_UpdateSp3d( am_I_Root,   HcoState, cID,   cName,   ExtNr,     &
                             Cat,         Hier,  HcoID,   AutoFill,  &
                             Array3D,     Total, PosOnly, COL,       &
                             MinDiagnLev, RC                         )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root     ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState      ! HEMCO state obj
INTEGER,          INTENT(IN   ), OPTIONAL :: cID           ! Assigned
                                                           !  container ID
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: cName         ! Diagnostics
                                                           !  name
INTEGER,          INTENT(IN   ), OPTIONAL :: ExtNr         ! Extension #
INTEGER,          INTENT(IN   ), OPTIONAL :: Cat           ! Category
INTEGER,          INTENT(IN   ), OPTIONAL :: Hier          ! Hierarchy
INTEGER,          INTENT(IN   ), OPTIONAL :: HcoID         ! HEMCO species
                                                           !  ID number
```

```
INTEGER,          INTENT(IN   ), OPTIONAL :: AutoFill       ! 1=yes; 0=no;
                                                            ! -1=either
REAL(sp),         INTENT(IN   )           :: Array3D(:,:,:) ! 3D array
REAL(sp),         INTENT(IN   ), OPTIONAL :: Total          ! Total
LOGICAL,          INTENT(IN   ), OPTIONAL :: PosOnly        ! Use only vals
                                                            !  >= 0?
INTEGER,          INTENT(IN   ), OPTIONAL :: COL            ! Collection Nr.
INTEGER,          INTENT(IN   ), OPTIONAL :: MinDiagnLev    ! minimum diagn level
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC             ! Return code
```

REVISION HISTORY:

```
20 Apr 2015 - C. Keller - Initialization
19 Sep 2016 - R. Yantosca - Rewritten for Gfortran: remove Scalar and
                            Array2d (put those in other overloaded methods)
```

1.11.8 Diagn_UpdateDp0d

Subroutine Diagn_UpdateSp0d is the wrapper routine to update the diagnostics for double-precision scalar values. It invokes the main diagnostics update routine with the appropriate arguments.

INTERFACE:

```
SUBROUTINE Diagn_UpdateDp0d( am_I_Root,   HcoState, cID,   cName,   ExtNr,     &
                             Cat,         Hier,  HcoID,   AutoFill,  &
                             Scalar,      Total, PosOnly, COL,       &
                             MinDiagnLev, RC                         )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root      ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState       ! HEMCO state obj
INTEGER,          INTENT(IN   ), OPTIONAL :: cID            ! Assigned
                                                            !  container ID
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: cName          ! Diagnostics
                                                            !  name
INTEGER,          INTENT(IN   ), OPTIONAL :: ExtNr          ! Extension #
INTEGER,          INTENT(IN   ), OPTIONAL :: Cat            ! Category
INTEGER,          INTENT(IN   ), OPTIONAL :: Hier           ! Hierarchy
INTEGER,          INTENT(IN   ), OPTIONAL :: HcoID          ! HEMCO species
                                                            !  ID number
INTEGER,          INTENT(IN   ), OPTIONAL :: AutoFill       ! 1=yes; 0=no;
                                                            ! -1=either
REAL(dp),         INTENT(IN   )           :: Scalar         ! 1D scalar
REAL(dp),         INTENT(IN   ), OPTIONAL :: Total          ! Total
```

```
    LOGICAL,          INTENT(IN   ), OPTIONAL :: PosOnly        ! Use only vals
                                                                !  >= 0?
    INTEGER,          INTENT(IN   ), OPTIONAL :: COL            ! Collection Nr.
    INTEGER,          INTENT(IN   ), OPTIONAL :: MinDiagnLev    ! minimum diagn level
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)           :: RC             ! Return code
```

REVISION HISTORY:

```
    20 Apr 2015 - C. Keller   - Initialization
    19 Sep 2016 - R. Yantosca - Rewritten for Gfortran: remove Array2d and
                                Array3d (put those in other overloaded methods)
```

1.11.9 Diagn_UpdateDp2d

Subroutine Diagn_UpdateSp2d is the wrapper routine to update the diagnostics for single precision 2D arrays. It invokes the main diagnostics update routine with the appropriate arguments.

INTERFACE:

```
  SUBROUTINE Diagn_UpdateDp2d( am_I_Root,   HcoState, cID,   cName,   ExtNr,     &
                               Cat,         Hier,  HcoID,   AutoFill,  &
                               Array2D,     Total, PosOnly, COL,       &
                               MinDiagnLev, RC                        )
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )           :: am_I_Root      ! Root CPU?
    TYPE(HCO_State),  POINTER                 :: HcoState       ! HEMCO state obj
    INTEGER,          INTENT(IN   ), OPTIONAL :: cID            ! Assigned
                                                                !  container ID
    CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: cName          ! Diagnostics
                                                                !  name
    INTEGER,          INTENT(IN   ), OPTIONAL :: ExtNr          ! Extension #
    INTEGER,          INTENT(IN   ), OPTIONAL :: Cat            ! Category
    INTEGER,          INTENT(IN   ), OPTIONAL :: Hier           ! Hierarchy
    INTEGER,          INTENT(IN   ), OPTIONAL :: HcoID          ! HEMCO species
                                                                !  ID number
    INTEGER,          INTENT(IN   ), OPTIONAL :: AutoFill       ! 1=yes; 0=no;
                                                                ! -1=either
    REAL(dp),         INTENT(IN   )           :: Array2D(:,:)   ! 2D array
    REAL(dp),         INTENT(IN   ), OPTIONAL :: Total          ! Total
    LOGICAL,          INTENT(IN   ), OPTIONAL :: PosOnly        ! Use only vals
                                                                !  >= 0?
    INTEGER,          INTENT(IN   ), OPTIONAL :: COL            ! Collection Nr.
    INTEGER,          INTENT(IN   ), OPTIONAL :: MinDiagnLev    ! minimum diagn level
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)          :: RC             ! Return code
```

REVISION HISTORY:

```
20 Apr 2015 - C. Keller   - Initialization
19 Sep 2016 - R. Yantosca - Rewritten for Gfortran: remove Scalar and
                            Array3d (put those in other overloaded methods)
```

1.11.10 Diagn_UpdateDp3d

Subroutine Diagn_UpdateSp3d is the wrapper routine to update the diagnostics for single precision arrays. It invokes the main diagnostics update routine with the appropriate arguments.

INTERFACE:

```
SUBROUTINE Diagn_UpdateDp3d( am_I_Root,   HcoState, cID,   cName,   ExtNr, &
                             Cat,         Hier,  HcoID,   AutoFill,        &
                             Array3D,     Total, PosOnly, COL,             &
                             MinDiagnLev, RC                       )
```

USES:

```
USE HCO_State_Mod, ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root      ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState       ! HEMCO state obj
INTEGER,          INTENT(IN   ), OPTIONAL :: cID            ! Assigned
                                                            !  container ID
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: cName          ! Diagnostics
                                                            !  name
INTEGER,          INTENT(IN   ), OPTIONAL :: ExtNr          ! Extension #
INTEGER,          INTENT(IN   ), OPTIONAL :: Cat            ! Category
INTEGER,          INTENT(IN   ), OPTIONAL :: Hier           ! Hierarchy
INTEGER,          INTENT(IN   ), OPTIONAL :: HcoID          ! HEMCO species
                                                            !  ID number
INTEGER,          INTENT(IN   ), OPTIONAL :: AutoFill       ! 1=yes; 0=no;
                                                            ! -1=either
REAL(dp),         INTENT(IN   )           :: Array3D(:,:,:) ! 3D array
REAL(dp),         INTENT(IN   ), OPTIONAL :: Total          ! Total
LOGICAL,          INTENT(IN   ), OPTIONAL :: PosOnly        ! Use only vals
                                                            !  >= 0?
INTEGER,          INTENT(IN   ), OPTIONAL :: COL            ! Collection Nr.
INTEGER,          INTENT(IN   ), OPTIONAL :: MinDiagnLev    ! minimum diagn level
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)          :: RC             ! Return code
```

REVISION HISTORY:

```
20 Apr 2015 - C. Keller   - Initialization
19 Sep 2016 - R. Yantosca - Rewritten for Gfortran: remove Scalar and
                            Array2d (put those in other overloaded methods)
```

1.11.11 Diagn_UpdateDriver

Subroutine Diagn_UpdateDriver updates the content of a diagnostics container. The container to be updated is determined from the passed variables. If a valid (i.e. positive) container ID is provided, this container is used. Otherwise, if a valid HEMCO species ID (HcoID) is provided, all containers with the same combination of HcoID, extension number (ExtNr), emission category (Cat) and hierarchy (Hier) are updated. If no valid HcoID and no valid cID is given, the container name has to be provided. The passed data array (Scalar, Array2D, or Array3D) needs to match the spatial dimension of the given container. For 2D diagnostics, a 3D array can be passed, in which case the level index specified during initialization ('LevIdx') is used. If LevIdx is set to -1, the column sum is used (default).

If no matching container is found, the subroutine leaves with no error. This allows automatic diagnostics generation, e.g. of intermediate emission fields created in HCO_CALC_Mod.F90.

The optional input argument 'MinDiagnLev' determines how 'deep' this routine will search for diagnostics with matching HcoID, ExtNr, etc. For example, if a HcoID, an ExtNr, and a category is provided, HEMCO by default will only update diagnostics containers with exactly the same HcoID, ExtNr, and category - but not diagnostics of 'lower level', e.g. with the same HcoID and ExtNr but no assigned category. This behavior can be changed by explicitly setting MinDiagnLev to the minimum diagnostics level. In the given example, setting MinDiagnLev to 1 would also update level 1 and level 2 diagnostics of the same HcoID (e.g. diagnostics with the same HcoID and no assigned ExtNr and category; as well as diagnostics with the same HcoID and ExtNr and no assigned category).

Notes:

- For a given time step, the same diagnostics container can be updated multiple times. The field average is always defined as temporal average, e.g. multiple updates on the same time step will not increase the averaging weight of that time step.
- If the passed array is empty (i.e. not associated), it is treated as empty values (i.e. zeros).
- The collection number can be set to -1 to scan trough all existing diagnostic collections.

INTERFACE:

```
SUBROUTINE Diagn_UpdateDriver( am_I_Root, HcoState,   cID,        cName,      &
                               ExtNr,     Cat,        Hier,       HcoID,      &
                               AutoFill,  Scalar,     Array2D,    Array3D,    &
```

```
                                Total,     Scalar_SP,  Array2D_SP, Array3D_SP,  &
                                Total_SP,  Scalar_HP,  Array2D_HP, Array3D_HP, &
                                Total_HP,  PosOnly,    COL,        MinDiagnLev, &
                                RC                                              )
```

USES:

```
USE HCO_Arr_Mod,   ONLY : HCO_ArrAssert
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )                  :: am_I_Root         ! Root CPU?
TYPE(HCO_State),  POINTER                        :: HcoState          ! HEMCO state ob
INTEGER,          INTENT(IN   ), OPTIONAL        :: cID               ! container ID
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL        :: cName             ! Dgn name
INTEGER,          INTENT(IN   ), OPTIONAL        :: ExtNr             ! Extension #
INTEGER,          INTENT(IN   ), OPTIONAL        :: Cat               ! Category
INTEGER,          INTENT(IN   ), OPTIONAL        :: Hier              ! Hierarchy
INTEGER,          INTENT(IN   ), OPTIONAL        :: HcoID             ! HEMCO species
INTEGER,          INTENT(IN   ), OPTIONAL        :: AutoFill          ! 1=yes; 0=no;
                                                                      ! -1=either
REAL(dp),         INTENT(IN   ), OPTIONAL        :: Scalar            ! 1D scalar
REAL(dp),         INTENT(IN   ), OPTIONAL, TARGET :: Array2D   (:,:)   ! 2D array
REAL(dp),         INTENT(IN   ), OPTIONAL, TARGET :: Array3D   (:,:,:) ! 3D array
REAL(dp),         INTENT(IN   ), OPTIONAL        :: Total             ! Total
REAL(sp),         INTENT(IN   ), OPTIONAL        :: Scalar_SP         ! 1D scalar
REAL(sp),         INTENT(IN   ), OPTIONAL, TARGET :: Array2D_SP(:,:)   ! 2D array
REAL(sp),         INTENT(IN   ), OPTIONAL, TARGET :: Array3D_SP(:,:,:) ! 3D array
REAL(sp),         INTENT(IN   ), OPTIONAL        :: Total_SP          ! Total
REAL(hp),         INTENT(IN   ), OPTIONAL        :: Scalar_HP         ! 1D scalar
REAL(hp),         INTENT(IN   ), OPTIONAL, TARGET :: Array2D_HP(:,:)   ! 2D array
REAL(hp),         INTENT(IN   ), OPTIONAL, TARGET :: Array3D_HP(:,:,:) ! 3D array
REAL(hp),         INTENT(IN   ), OPTIONAL        :: Total_HP          ! Total
LOGICAL,          INTENT(IN   ), OPTIONAL        :: PosOnly           ! Use only vals
                                                                      ! >= 0?
INTEGER,          INTENT(IN   ), OPTIONAL        :: COL               ! Collection Nr
INTEGER,          INTENT(IN   ), OPTIONAL        :: MinDiagnLev       ! Collection Nr
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)                  :: RC                ! Return code
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller - Initialization
25 Sep 2014 - C. Keller - Now allow updating multiple diagnostics
11 Mar 2015 - C. Keller - Now allow scanning of all diagnostic collections
13 Mar 2015 - C. Keller - Bug fix: only prompt warning if it's a new timestep
17 Jun 2015 - C. Keller - Added argument MinDiagnLev
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.12 Diagn_Get

Subroutine Diagn_Get returns a diagnostics container from the diagnostics list, with the data converted to the output unit specified during initialization. Only diagnostics that contain data, i.e. with an update counter higher than zero, are returned. If EndOfIntvOnly is set to TRUE, only containers at the end of their time averaging interval are returned. The current HEMCO time will be used to determine which containers are at the end of their interval. The IsOutFormat flag of the container is set to true, making sure that the currently saved data will be erased during the next update (Diagn_Update).

If DgnCont is already associated, the search continues from the container next to DgnCont. If DgnCont is empty (null), the search starts from the first container of the diagnostics list ListDiagn. If the optional attribute cName or cID is provided, this particular container is searched (through the entire diagnostics list), but is only returned if it is at the end of it's interval or if EndOfIntvOnly is disabled.

The optional argument InclManual denotes whether or not containers with a manual update frequency shall be considered. This argument is only valid if EndOfIntvOnly is set to FALSE.

The return flag FLAG is set to HCO_SUCCESS if a container is found, and to HCO_FAIL otherwise.

INTERFACE:

```
SUBROUTINE Diagn_Get( am_I_Root, HcoState,              &
                      EndOfIntvOnly,           DgnCont, &
                      FLAG,      RC,           cName,   &
                      cID,       AutoFill,     COL,     &
                      SkipZeroCount                     )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root       ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState        ! HEMCO state obj
LOGICAL,          INTENT(IN   )           :: EndOfIntvOnly   ! End of
                                                             ! interval
                                                             ! only?
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: cName           ! container name
INTEGER,          INTENT(IN   ), OPTIONAL :: cID             ! container ID
INTEGER,          INTENT(IN   ), OPTIONAL :: AutoFill        ! 0=no; 1=yes;
                                                             ! -1=either
INTEGER,          INTENT(IN   ), OPTIONAL :: COL             ! Collection Nr.
LOGICAL,          INTENT(IN   ), OPTIONAL :: SkipZeroCount   ! Skip if counter
                                                             ! is zero
```

OUTPUT PARAMETERS:

```
TYPE(DiagnCont),  POINTER                 :: DgnCont         ! Return
                                                             ! container
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: FLAG            ! Return flag
INTEGER,          INTENT(INOUT)           :: RC              ! Return code
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.13 Diagn_TotalGet

Subroutine Diagn_TotalGet returns the total of a given diagnostics container.

INTERFACE:

```
SUBROUTINE Diagn_TotalGet( am_I_Root, Diagn, cName, cID,   COL, &
                           FOUND,     Total, Reset, RC    )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root      ! Root CPU?
TYPE(DiagnBundle),POINTER                 :: Diagn          ! Diagn bundle obj
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: cName          ! container name
INTEGER,          INTENT(IN   ), OPTIONAL :: cID            ! container ID
INTEGER,          INTENT(IN   ), OPTIONAL :: COL            ! Collection Nr.
LOGICAL,          INTENT(IN   ), OPTIONAL :: Reset          ! Reset total?
```

OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(  OUT), OPTIONAL :: FOUND          ! Container found
REAL(sp),         INTENT(  OUT)           :: Total          ! Container total
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC             ! Return code
```

REVISION HISTORY:

```
15 Mar 2015 - C. Keller: Initialization
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.14 DiagnList_Cleanup

Subroutine DiagnList_Cleanup cleans up all the diagnostics containers of the given diagnostics list.

INTERFACE:

```
SUBROUTINE DiagnList_Cleanup ( DiagnList )
```

INPUT PARAMETERS:

```
TYPE(DiagnCont), POINTER  :: DiagnList   ! List to be removed
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller   - Initialization
25 Jan 2016 - R. Yantosca - Bug fix for pgfortran compiler: Test if the
                            TMPCONT object is associated before deallocating
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.15 Diagn_AutoFillLevelDefined

Function Diagn_AutoFillLevelDefined returns .TRUE. if there is at least one AutoFill diagnostics container defined for the given level (1=Species level, 2=ExtNr level, 3=Category level, 4=Hierarchy level).

INTERFACE:

```
FUNCTION Diagn_AutoFillLevelDefined( Diagn, Level, COL ) RESULT ( IsDefined )
```

INPUT PARAMETERS:

```
TYPE(DiagnBundle),POINTER     :: Diagn     ! Diagn bundle obj
INTEGER, INTENT(IN)           :: Level     ! Level of interest
INTEGER, INTENT(IN), OPTIONAL :: COL       ! Collection Nr.
```

RETURN VALUE:

```
LOGICAL                       :: IsDefined ! Return argument
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.16 DiagnCollection_Get

Subroutine DiagnCollection_Get returns variables assigned to a given diagnostics collection.

INTERFACE:

```
SUBROUTINE DiagnCollection_Get( Diagn,        COL,               &
                                InUse,        Prefix,            &
                                nnDiagn,      DeltaYMD,          &
                                LastYMD,      DeltaHMS, LastHMS, &
                                OutTimeStamp, RC                  )
!INPUT ARGUMENTS:

 INTEGER,          INTENT(IN), OPTIONAL :: COL       ! Collection Nr.
```

OUTPUT PARAMETERS:

```
TYPE(DiagnBundle),POINTER              :: Diagn
LOGICAL,          INTENT(OUT), OPTIONAL :: InUse
CHARACTER(LEN=*), INTENT(OUT), OPTIONAL :: Prefix
INTEGER,          INTENT(OUT), OPTIONAL :: nnDiagn
INTEGER,          INTENT(OUT), OPTIONAL :: DeltaYMD
INTEGER,          INTENT(OUT), OPTIONAL :: LastYMD
INTEGER,          INTENT(OUT), OPTIONAL :: DeltaHMS
INTEGER,          INTENT(OUT), OPTIONAL :: LastHMS
INTEGER,          INTENT(OUT), OPTIONAL :: OutTimeStamp
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.17 DiagnCollection_Set

Subroutine DiagnCollection_Set sets variables assigned to a given diagnostics collection.

INTERFACE:

```
SUBROUTINE DiagnCollection_Set( Diagn, COL, InUse, LastYMD, LastHMS, RC )
!INPUT ARGUMENTS:
  TYPE(DiagnBundle),POINTER              :: Diagn     ! Diagn bundle
  INTEGER,          INTENT(IN), OPTIONAL :: COL       ! Collection Nr.
```

OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(OUT), OPTIONAL :: InUse
INTEGER,          INTENT(IN ), OPTIONAL :: LastYMD
INTEGER,          INTENT(IN ), OPTIONAL :: LastHMS
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.18 DiagnCont_Init

Subroutine DiagnCont_Init initializes a new (blank) diagnostics container DgnCont.

INTERFACE:

```
SUBROUTINE DiagnCont_Init( OutCont )
```

OUTPUT PARAMETERS:

```
TYPE(DiagnCont),       POINTER      :: OutCont    ! Created container
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
```

1.11.19 DiagnCont_Cleanup

Subroutine DiagnCont_Cleanup cleans up diagnostics container DgnCont.

INTERFACE:

```
SUBROUTINE DiagnCont_Cleanup( DgnCont )
```

USES:

```
USE HCO_ARR_Mod, ONLY : HCO_ArrCleanup
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(DiagnCont), POINTER :: DgnCont  ! Container to be cleaned
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
```

1.11.20 DiagnCont_PrepareOutput

Subroutine DiagnCont_PrepareOutput converts the data of the given diagnostics container to proper output units.

INTERFACE:

```
SUBROUTINE DiagnCont_PrepareOutput ( am_I_Root, HcoState, DgnCont, RC )
```

USES:

```
USE HCO_State_Mod, ONLY : HCO_State
USE HCO_Arr_Mod,   ONLY : HCO_ArrAssert
```

INPUT PARAMETERS:

```
LOGICAL,           INTENT(IN   ) :: am_I_Root ! Root CPU?
TYPE(HCO_State),   POINTER       :: HcoState  ! HEMCO state obj
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(DiagnCont),   POINTER       :: DgnCont   ! diagnostics container
INTEGER,           INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.21 DiagnCont_Find

Subroutine DiagnCont_Find searches for a diagnostics container in ListDiagn. If a valid container ID (¿0) is given, the container with this ID is searched. Otherwise, if a valid HEMCO specied ID (¿0) is given, the container with the same combination of HcoID, extension number (ExtNr), and emission category (Cat) and hierarchy (Hier), is searched. If no valid HcoID and no valid cID is provided, the container with the given container name is searched.

If the optional resume flag is set to TRUE, search will resume after OutCnt. If OutCnt is not associated or resume flag is FALSE, search starts at the beginning of the diagnostics list.

This subroutine does return the diagnostics as is, i.e. in the internal units. It should NOT be used to access the content of a diagnostics but is rather intended to be used in the background, e.g. to check if a diagnostics exists at all. To get the values of a diagnostics, use routine Diagn_Get. **INTERFACE:**

```
SUBROUTINE DiagnCont_Find ( Diagn, cID,      ExtNr, Cat,    Hier,   HcoID, &
                            cName, AutoFill, FOUND, OutCnt, Resume, COL )
```

INPUT PARAMETERS:

```
TYPE(DiagnBundle), POINTEr      :: Diagn    ! diagn bundle
INTEGER,           INTENT(IN)   :: cID      ! wanted cont. ID
INTEGER,           INTENT(IN)   :: ExtNr    ! wanted ExtNr
INTEGER,           INTENT(IN)   :: Cat      ! wanted category
INTEGER,           INTENT(IN)   :: Hier     ! wanted hierarchy
INTEGER,           INTENT(IN)   :: HcoID    ! wanted spec. ID
CHARACTER(LEN=*),  INTENT(IN)   :: cName    ! wanted name
INTEGER,           INTENT(IN)   :: AutoFill ! 0=no; 1=yes; -1=either
LOGICAL, OPTIONAL, INTENT(IN)   :: Resume   ! Resume at OutCnt?
INTEGER, OPTIONAL, INTENT(IN)   :: COL      ! Collection number
```

OUTPUT PARAMETERS:

```
LOGICAL,           INTENT(OUT)  :: FOUND    ! container found?
```

INPUT/OUTPUT PARAMETERS:

```
    TYPE(DiagnCont),   POINTER      :: OutCnt   ! data container
```

REVISION HISTORY:

```
    19 Dec 2013 - C. Keller: Initialization
    25 Sep 2014 - C. Keller: Added Resume flag
    09 Apr 2015 - C. Keller: Can now search all collections
    26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.22 DiagnCont_Link_2D

Subroutine DiagnCont_Link_2D links the data of container DgnCont to the 2D array Tgt2D. This will disable all time averaging, unit conversion, etc., i.e. the data will be returned as is.

INTERFACE:

```
  SUBROUTINE DiagnCont_Link_2D( am_I_Root, DgnCont, ThisColl, Trgt2D, RC, HcoState )
```

USES:

```
    USE HCO_State_Mod, ONLY : HCO_State
  !INPUT ARGUMENTS:
    LOGICAL,               INTENT(IN   )          :: am_I_Root   ! Root CPU?
    TYPE(HCO_STATE),       POINTER, OPTIONAL      :: HcoState    ! HEMCO state obj
    REAL(sp),              INTENT(IN   ), TARGET  :: Trgt2D(:,:) ! 2D target data
    TYPE(DiagnCollection), POINTER                :: ThisColl    ! Collection
```

INPUT/OUTPUT PARAMETERS:

```
    TYPE(DiagnCont),       POINTER                :: DgnCont     ! diagnostics container
    INTEGER,               INTENT(INOUT)          :: RC          ! Return code
```

REVISION HISTORY:

```
    19 Dec 2013 - C. Keller: Initialization
```

1.11.23 DiagnCont_Link_3D

Subroutine DiagnCont_Link_3D links the data of container DgnCont to the 3D array Tgt3D. This will disable all time averaging, unit conversion, etc., i.e. the data will be returned as is.

INTERFACE:

```
  SUBROUTINE DiagnCont_Link_3D( am_I_Root, DgnCont, ThisColl, Trgt3D, RC, HcoState )
```

USES:

```
    USE HCO_State_Mod, ONLY : HCO_State
  !INPUT PARAEMTERS:
    LOGICAL,               INTENT(IN   )          :: am_I_Root     ! Root CPU?
    TYPE(HCO_STATE),       POINTER, OPTIONAL      :: HcoState      ! HEMCO state obj
    REAL(sp),              INTENT(IN   ), TARGET  :: Trgt3D(:,:,:) ! 3D target data
    TYPE(DiagnCollection), POINTER                :: ThisColl      ! Collection
```

INPUT/OUTPUT PARAMETERS:

```
  TYPE(DiagnCont),       POINTER              :: DgnCont       ! diagnostics
                                                               !  container
  INTEGER,               INTENT(INOUT)        :: RC            ! Return code
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
```

1.11.24 Diagn_Print

Subroutine Diagn_Print displays the content of the passed diagnostics container.

INTERFACE:

```
SUBROUTINE Diagn_Print ( HcoState, Dgn, VerbNr )
```

USES:

```
  USE HCO_STATE_MOD,  ONLY : HCO_STATE
!INPUT ARGUMENTS:
  TYPE(HCO_STATE),       POINTER    :: HcoState
  TYPE(DiagnCont),       POINTER    :: Dgn
  INTEGER,               INTENT(IN) :: VerbNr
```

REVISION HISTORY:

```
01 Aug 2014 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.25 DiagnCollection_Create

Subroutine DiagnCollection_Create creates a new diagnostics collection at position COL. The class arguments are set as specified by the input arguments.

If the given position is already occupied, the routine returns an error if the input argument do not match with the corresponding arguments of the diagnostics class at that position.

INTERFACE:

```
SUBROUTINE DiagnCollection_Create ( am_I_Root, Diagn,  NX, NY, NZ,    &
                                    TS,   AM2, PREFIX,                &
                                    deltaYMD, deltaHMS, OutTimeStamp, &
                                    RC,       COL, HcoState )
```

USES:

```
  USE HCO_STATE_MOD,  ONLY : HCO_STATE
!INPUT ARGUMENTS:
  LOGICAL,            INTENT(IN)           :: am_I_Root    ! Root CPU?
  TYPE(DiagnBundle),  POINTER              :: Diagn        ! Diagn bundle
  INTEGER,            INTENT(IN)           :: NX           ! # of lons
  INTEGER,            INTENT(IN)           :: NY           ! # of lats
  INTEGER,            INTENT(IN)           :: NZ           ! # of levels
  REAL(sp),           INTENT(IN)           :: TS           ! timestep [s]
  REAL(hp),           POINTER              :: AM2(:,:)     ! grid box areas [m2]
  CHARACTER(LEN=*),   INTENT(IN)           :: PREFIX       ! Output prefix
  INTEGER,            INTENT(IN), OPTIONAL :: deltaYMD     ! Output frequency
  INTEGER,            INTENT(IN), OPTIONAL :: deltaHMS     ! Output frequency
  INTEGER,            INTENT(IN), OPTIONAL :: OutTimeStamp ! Output time stamp
  TYPE(HCO_State),    POINTER,    OPTIONAL :: HcoState     ! HEMCO state obj
!OUTPUT ARGUMENTS:
  INTEGER,            INTENT(  OUT)        :: COL        ! Collection Nr.
!INPUT/OUTPUT ARGUMENTS:
  INTEGER,            INTENT(INOUT)        :: RC         ! Return code
```

REVISION HISTORY:

```
08 Jan 2015 - C. Keller   - Initial version
06 Nov 2015 - C. Keller   - Added OutTimeStamp.
```

1.11.26 DiagnCollection_Cleanup

Subroutine DiagnCollection_Cleanup cleans up the diagnostics collection for collection COL.

INTERFACE:

```
SUBROUTINE DiagnCollection_Cleanup ( Diagn )
!INPUT ARGUMENTS:
  TYPE(DiagnBundle), POINTER       :: Diagn
```

REVISION HISTORY:

```
08 Jan 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.27 DiagnCollection_DefineID

Subroutine DiagnCollection_DefineID is a helper routine to return the collection ID.

INTERFACE:

```
SUBROUTINE DiagnCollection_DefineID ( Diagn, PS, RC, COL, DEF, OkIfAll, &
                                      InUse, ThisColl, HcoState )
```

USES:

```
  USE HCO_STATE_MOD,     ONLY : HCO_STATE
!INPUT ARGUMENTS:
  INTEGER,               INTENT(IN   ), OPTIONAL :: COL        ! desired collection numbe
  INTEGER,               INTENT(IN   ), OPTIONAL :: DEF        ! default collection numbe
  LOGICAL,               INTENT(IN   ), OPTIONAL :: OkIfAll    ! Ok if all (PS=-1)
!INPUT/OUTPUT ARGUMENTS:
  TYPE(DiagnBundle),     POINTER                 :: Diagn      ! Diagn bundle obj
  TYPE(HCO_STATE),       POINTER,       OPTIONAL :: HcoState   ! HEMCO state obj
  INTEGER,               INTENT(INOUT)           :: PS         ! Assigned collection numb
  INTEGER,               INTENT(INOUT)           :: RC         ! Return code
  LOGICAL,               INTENT(  OUT), OPTIONAL :: InUse      ! Is this in use?
!OUTPUT ARGUMENTS:
  TYPE(DiagnCollection), POINTER,      OPTIONAL :: ThisColl    ! Pointer to collection
```

REVISION HISTORY:

```
01 Apr 2015 - C. Keller - Initial version
```

1.11.28 DiagnCollection_Find

Subroutine DiagnCollection_Find searches the collection linked list for the collection with the given collection ID.

INTERFACE:

```
SUBROUTINE DiagnCollection_Find ( Diagn, PS, FOUND, RC, ThisColl )
```

INPUT PARAMETERS:

```
  INTEGER,               INTENT(IN   )           :: PS       ! desired collection number
```

INPUT/OUTPUT PARAMETERS:

```
  TYPE(DiagnBundle),     POINTER                 :: Diagn    ! Diagn bundle obj
  LOGICAL,               INTENT(  OUT)           :: FOUND    ! Collection exists?
  INTEGER,               INTENT(INOUT)           :: RC       ! Return code
  TYPE(DiagnCollection), POINTER,      OPTIONAL :: ThisColl ! Pointer to collection
```

REVISION HISTORY:

```
01 Apr 2015 - C. Keller   - Initial version
10 Jul 2015 - R. Yantosca - Fixed minor issues in ProTeX header
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.11.29 DiagnCollection_GetDefaultDelta returns the default diagnostics

output intervals based on the 'DiagnFreq' entry of the HEMCO configuration file. This can be one of the following character values: 'Hourly', 'Daily', 'Monthly', 'Annually', 'Always', or 'End'; or two integer explicitly denoting the year-month-day and hour-minute-second interval, respectively (format 00000000 000000). For example, setting DiagnFreq to '00000000 010000' would be the same as setting it to 'Hourly'.

INTERFACE:

```
SUBROUTINE DiagnCollection_GetDefaultDelta ( am_I_Root, HcoState, &
                                             deltaYMD,  deltaHMS, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : HCO_State
USE HCO_ExtList_Mod, ONLY : GetExtOpt
USE HCO_ExtList_Mod, ONLY : CoreNr
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )          :: am_I_Root ! Root CPU?
TYPE(HCO_STATE),  POINTER                :: HcoState  ! HEMCO state obj
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT)          :: deltaYMD  ! delta YYYYMMDD
INTEGER,          INTENT(  OUT)          :: deltaHMS  ! delta HHMMSS
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)          :: RC        ! Return code
```

REVISION HISTORY:

```
06 Aug 2015 - C. Keller   - Initial version
```

1.11.30 Function DiagnCollection_IsTimeToWrite returns true if it is time

to write the provided diagnostics collection (identified by the collection number) to output. Whether it is time to write the diagnostics is based upon the current simulation time, the diagnostics output frequency, and the time span since the last output datetime.

INTERFACE:

```
FUNCTION DiagnCollection_IsTimeToWrite( am_I_Root, HcoState, PS ) &
   RESULT ( TimeToWrite )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN   )     :: am_I_Root   ! Root CPU?
INTEGER, INTENT(IN   )     :: PS          ! Diagnostics collection
TYPE(HCO_State), POINTER   :: HcoState    ! HEMCO state obj
```

OUTPUT PARAMETERS:

```
LOGICAL                    :: TimeToWrite ! Is it time to write?
```

REVISION HISTORY:

```
06 Aug 2015 - C. Keller   - Initial version
30 Sep 2015 - C. Keller   - Bug fix: now set current hour from 0 to 24 to
                            make sure that it will be greater than previous
                            hour.
```

1.11.31 Function DiagnCollection_LastTimesSet returns true if there

exists a valid entry for the last datetime that collection PS has been written to disk. This is primarily important to check if the last output date needs be initialized (to non-default values).

INTERFACE:

```
FUNCTION DiagnCollection_LastTimesSet( Diagn, PS ) Result ( LastTimesSet )
```

USES:

INPUT PARAMETERS:

```
TYPE(DiagnBundle), POINTER :: Diagn
INTEGER, INTENT(IN   )     :: PS   ! Diagnostics collection
```

OUTPUT PARAMETERS:

```
LOGICAL                    :: LastTimesSet ! Are last times defined or not?
```

REVISION HISTORY:

```
09 Sep 2015 - C. Keller   - Initial version
```

Opens a diagnostic configuration file. This is where you tell HEMCO which diagnostics you would like to send directly to netCDF output.

INTERFACE:

```
SUBROUTINE DiagnFileOpen( am_I_Root, HcoConfig, LUN, RC )
```

USES:

```
USE inquireMod,        ONLY : findFreeLUN
USE HCO_ExtList_Mod,   ONLY : CoreNr, GetExtOpt
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root   ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER                 :: HcoConfig   ! HEMCO config obj
INTEGER,          INTENT(INOUT)           :: RC          ! Failure or success
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT)           :: LUN         ! File LUN
```

REVISION HISTORY:

```
10 Apr 2015 - C. Keller   - Initial version
```

1.11.32 DiagnFileGetNext returns the diagnostics entries of the next

line of the diagnostics list file. Gets information from the next line of the diagnostic configuration file.

INTERFACE:

```
SUBROUTINE DiagnFileGetNext( am_I_Root, HcoConfig, LUN, cName, &
                             SpcName,   ExtNr,     Cat,        &
                             Hier,      SpaceDim,  OutUnit,    &
                             EOF,       RC,        lName,      &
                             UnitName                          )
```

USES:

```
USE HCO_CharTools_Mod
USE CHARPAK_Mod,       ONLY : STRREPL, STRSPLIT
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root   ! root CPU?
INTEGER,          INTENT(IN   )           :: LUN         ! file LUN
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER                 :: HcoConfig
LOGICAL,          INTENT(INOUT)           :: EOF
INTEGER,          INTENT(INOUT)           :: RC          ! Failure or success
```

OUTPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(  OUT)           :: cName
CHARACTER(LEN=*), INTENT(  OUT)           :: SpcName
INTEGER,          INTENT(  OUT)           :: ExtNr
INTEGER,          INTENT(  OUT)           :: Cat
INTEGER,          INTENT(  OUT)           :: Hier
INTEGER,          INTENT(  OUT)           :: SpaceDim
CHARACTER(LEN=*), INTENT(  OUT)           :: OutUnit
CHARACTER(LEN=*), INTENT(  OUT), OPTIONAL :: lName
CHARACTER(LEN=*), INTENT(  OUT), OPTIONAL :: UnitName
```

REVISION HISTORY:

```
10 Apr 2015 - C. Keller   - Initial version
23 Feb 2016 - C. Keller   - Added lName and UnitName arguments
```

1.11.33 DiagnFileClose

Closes the diagnostic configuration file.

INTERFACE:

```
SUBROUTINE DiagnFileClose ( LUN )
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)          :: LUN         ! File LUN
```

REVISION HISTORY:

```
10 Apr 2015 - C. Keller   - Initial version
```

1.11.34 DiagnBundle_Init

Creates an empty diagnostics bundle

INTERFACE:

```
SUBROUTINE DiagnBundle_Init ( Diagn )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(DiagnBundle), POINTER                :: Diagn
```

REVISION HISTORY:

```
17 Feb 2016 - C. Keller   - Initial version
```

1.11.35 DiagnBundle_Cleanup

Cleans up a diagnostics bundle

INTERFACE:

```
SUBROUTINE DiagnBundle_Cleanup ( Diagn )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(DiagnBundle), POINTER                :: Diagn
```

REVISION HISTORY:

```
17 Feb 2016 - C. Keller   - Initial version
```

1.12 Fortran: Module Interface hcoio_write_esmf_mod.F90

Module HCOIO_Write_ESMF_Mod.F90 is the HEMCO output interface for the ESMF environment. In an ESMF/MAPL environment, the HEMCO diagnostics are not directly written to disk but passed to the gridded component export state, where they can be picked up by the MAPL HISTORY component.

INTERFACE:

```
MODULE HCOIO_WRITE_ESMF_MOD
```

USES:

```
  USE HCO_ERROR_MOD
  USE HCO_DIAGN_MOD

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
#if defined(ESMF_)
  PUBLIC :: HCOIO_WRITE_ESMF
```

REMARKS:

```
   HEMCO diagnostics are still in testing mode. We will fully activate them
   at a later time.  They will be turned on when debugging & unit testing.
```

REVISION HISTORY:

```
   04 May 2014 - C. Keller   - Initial version.
   11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
   11 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
   28 Jul 2014 - C. Keller   - Removed GC specific initialization calls and
                               moved to HEMCO core.
   05 Aug 2014 - C. Keller   - Added dummy interface for ESMF.
   03 Apr 2015 - C. Keller   - Added HcoDiagn_Write
   22 Feb 2016 - C. Keller   - Split off from hcoio_diagn_mod.F90
```

1.12.1 HCOIO_Diagn_WriteOut

Subroutine HCOIO_Diagn_WriteOut is the interface routine to link the HEMCO diagnostics arrays to the corresponding data pointers of the MAPL/ESMF history component.

Since the history component internally organizes many diagnostics tasks such as output scheduling, file writing, and data averaging, all HEMCO diagnostics are made available to the history component on every time step, e.g. the entire content of the HEMCO diagnostics list is 'flushed' every time this subroutine is called.

For now, all diagnostics data is copied to the corresponding MAPL data pointer so that this routine works for cases where the HEMCO precision is not equal to the ESMF precision.

Once the HEMCO precision is pegged to the ESMF precision, we can just establish pointers between the export arrays and the diagnostics the first time this routine is called.

INTERFACE:

```
SUBROUTINE HCOIO_WRITE_ESMF ( am_I_Root, HcoState, RC, OnlyIfFirst, COL )
```

USES:

```
USE ESMF
USE MAPL_MOD
USE HCO_Types_Mod, ONLY : DiagnCont
USE HCO_State_Mod, ONLY : HCO_State

# include "MAPL_Generic.h"
```

INPUT PARAMETERS:

```
LOGICAL,                    INTENT(IN   ) :: am_I_Root   ! root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState    ! HEMCO state object
LOGICAL,          OPTIONAL, INTENT(IN   ) :: OnlyIfFirst !
INTEGER,          OPTIONAL, INTENT(IN   ) :: COL         ! Collection Nr.
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,                    INTENT(INOUT) :: RC          ! Failure or success
```

REVISION HISTORY:

```
05 Aug 2014 - C. Keller    - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13 Fortran: Module Interface hco_extlist_mod

Module HCO_EXTLIST_MOD contains routines and variables to organize HEMCO extensions and the corresponding settings (options). This is done through the ExtList object, which is a simple list containing all enabled HEMCO extensions (name and ext. ID) and the corresponding options, as defined in the HEMCO configuration file. The general HEMCO settings are stored as options of the HEMCO core extension (Extension number = 0). The CORE extension is activated in every HEMCO run, while all other extensions are only activated if enabled in the configuration file.

Extension number -999 is used as 'wildcard' value, e.g. data containers with extension number -999 will always be read by HEMCO but will be ignored for emission calculation. This is particularly useful for data fields that shall be used outside of HEMCO, e.g. stratospheric chemistry prod/loss rates, etc.

Extension options are 'flexible' in a sense that any option name/value pair can be assigned to an extension. The value of any of these options can be queried using subroutine GetExtOpt or function HCO_GetOpt. In fact, the HEMCO filename parser (in hco_chartools_mod.F90) will attempt to find an option value for any HEMCO 'token' (a character starting with the HEMCO token sign (which is, the dollar sign '$'). This allows the user to specify as many individual tokens as HEMCO settings as needed.

INTERFACE:

```
MODULE HCO_ExtList_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Types_Mod

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: AddExt
PUBLIC  :: AddExtOpt
PUBLIC  :: GetExtOpt
PUBLIC  :: GetExtNr
PUBLIC  :: GetExtSpcStr
PUBLIC  :: GetExtSpcVal
PUBLIC  :: SetExtNr
PUBLIC  :: ExtNrInUse
PUBLIC  :: ExtFinal
PUBLIC  :: HCO_GetOpt
PUBLIC  :: HCO_SetDefaultToken
PUBLIC  :: HCO_ROOT

PRIVATE :: HCO_AddOpt
PRIVATE :: HCO_CleanupOpt

! Core extension number
INTEGER, PARAMETER, PUBLIC  :: CoreNr = -1
```

REVISION HISTORY:

```
 02 Oct 2013 - C. Keller   - Initial version
 01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 30 Sep 2014 - R. Yantosca - ThisExt%Spcs now has 2047 chars for extensions
                             having many individual species
 20 Sep 2015 - C. Keller   - Reorganize options in linked lists. Tokens are
                             now the same as options and can be flexibly
                             set by the user.
 24 Aug 2017 - M. Sulprizio- Remove support for GCAP, GEOS-4, GEOS-5 and MERRA
```

1.13.1 AddExt

Subroutine AddExt adds a new extension to the extensions list. The extension name, number and species (multiple species separated by the HEMCO separator sign) need to be provided. Extension options are left blank but can be added lateron using AddExtOpt.

INTERFACE:

```
SUBROUTINE AddExt( am_I_Root, HcoConfig, ExtName, ExtNr, InUse, Spcs, RC )
```

USES:

```
USE CHARPAK_MOD,  ONLY : TRANLC
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root
TYPE(ConfigObj),  POINTER       :: HcoConfig
CHARACTER(LEN=*), INTENT(IN   ) :: ExtName
INTEGER,          INTENT(IN   ) :: ExtNr
LOGICAL,          INTENT(IN   ) :: InUse
CHARACTER(LEN=*), INTENT(IN   ) :: Spcs
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
03 Oct 2013 - C. Keller - Initial version
20 Sep 2015 - C. Keller - Options are now linked list
12 Dec 2015 - C. Keller - Added argument InUse
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.2 AddExtOpt

Function AddExtOpt appends the given string to the options character of the desired extension (identified by its extension number). The options string is expected to contain an option name and value, separated by a colon (:). Function GetExtOpt can be used to extract the option value at a later point.

INTERFACE:

```
SUBROUTINE AddExtOpt( am_I_Root, HcoConfig, Opt, ExtNr, RC, IgnoreIfExist )
```

USES:

```
USE CHARPAK_MOD,       ONLY : STRSPLIT, TRANLC
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )           :: am_I_Root      ! Root CPU?
    TYPE(ConfigObj),  POINTER                 :: HcoConfig      ! Configuration object
    CHARACTER(LEN=*), INTENT(IN   )           :: Opt            ! Option name & value
    INTEGER,          INTENT(IN   )           :: ExtNr          ! Add to this extension
    LOGICAL,          INTENT(IN   ), OPTIONAL :: IgnoreIfExist  ! Ignore this entry if it e
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
03 Oct 2013 - C. Keller - Initial version
20 Sep 2015 - C. Keller - Options are now linked list
12 Dec 2015 - C. Keller - Added argument IgnoreIfExist
```

1.13.3 GetExtOpt

Function GetExtOpt returns the option value for a given extension and option name. The type of the return value depends on the provided argument (real, boolean, character). The optional output argument FOUND returns TRUE if the given option name was found, and FALSE otherwise. If the FOUND argument is provided, no error is returned if the option name is not found! If the ExtNr is set to -999, the settings of all extensions are searched.

INTERFACE:

```
  SUBROUTINE GetExtOpt ( HcoConfig,  ExtNr,      OptName,   OptValHp, &
                         OptValSp,   OptValDp,   OptValInt,           &
                         OptValBool, OptValChar, Found,     RC )
```

USES:

```
  USE CHARPAK_MOD,       ONLY : STRSPLIT, TRANLC
```

INPUT PARAMETERS:

```
    TYPE(ConfigObj),  POINTER                 :: HcoConfig
    INTEGER,          INTENT(IN   )           :: ExtNr
    CHARACTER(LEN=*), INTENT(IN   )           :: OptName
```

OUTPUT PARAMETERS:

```
    REAL(hp),         INTENT(  OUT), OPTIONAL  :: OptValHp
    REAL(sp),         INTENT(  OUT), OPTIONAL  :: OptValSp
    REAL(dp),         INTENT(  OUT), OPTIONAL  :: OptValDp
    INTEGER,          INTENT(  OUT), OPTIONAL  :: OptValInt
    LOGICAL,          INTENT(  OUT), OPTIONAL  :: OptValBool
    CHARACTER(LEN=*), INTENT(  OUT), OPTIONAL  :: OptValChar
    LOGICAL,          INTENT(  OUT), OPTIONAL  :: Found
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
03 Oct 2013 - C. Keller   - Initial version
13 Jan 2015 - R. Yantosca - Add optional variable of flex precision (hp)
14 Feb 2015 - C. Keller   - Add option to search all extensions (ExtNr=-999).
17 Apr 2015 - C. Keller   - Passed option OptName must now exactly match the
                            stored option name to avoid ambiguity.
20 Sep 2015 - C. Keller   - Options are now linked list.
20 Jan 2016 - C. Keller   - Bug fix: boolean options are now case insensitive.
```

1.13.4 GetExtNr

Function GetExtNr returns the extension number of extension ExtName. Returns -999 if no extension with the given name is found.

INTERFACE:

```
FUNCTION GetExtNr( ExtList, ExtName ) Result ( ExtNr )
```

USES:

```
USE CHARPAK_MOD,  ONLY : TRANLC
```

INPUT PARAMETERS:

```
TYPE(Ext),        POINTER       :: ExtList
CHARACTER(LEN=*), INTENT(IN   ) :: ExtName
```

RETURN VALUE:

```
INTEGER                         :: ExtNr
```

REVISION HISTORY:

```
03 Oct 2013 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.5 GetExtSpcStr

Subroutine GetExtSpcStr returns the HEMCO species names string of all species assigned to the given extension (identified by its extension number).

INTERFACE:

```
SUBROUTINE GetExtSpcStr( HcoConfig, ExtNr, SpcStr, RC )
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER       :: HcoConfig
INTEGER,          INTENT(IN   ) :: ExtNr    ! Extension Nr.
```

OUTPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(  OUT) :: SpcStr   ! Species string
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC       ! Success or failure?
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.6 GetExtSpcVal_Sp

Subroutine GetExtSpcVal_Sp returns single precision values associated with the species for a given extension. Specifically, this routine searches for extension setting '¡Prefix¿_SpecName' for every species passed through input argument SpcNames and writes those into output argument SpcScal. The default value DefValue is assigned to all elements of SpcScal with no corresponding extension setting.

INTERFACE:

```
SUBROUTINE GetExtSpcVal_Sp( HcoConfig, ExtNr,  NSPC,    SpcNames, &
                            Prefix,  DefValue, SpcScal, RC         )
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),       POINTER       :: HcoConfig
INTEGER,               INTENT(IN   ) :: ExtNr          ! Extension Nr.
INTEGER,               INTENT(IN   ) :: NSPC           ! # of species
CHARACTER(LEN=*),      INTENT(IN   ) :: SpcNames(NSPC) ! Species string
CHARACTER(LEN=*),      INTENT(IN   ) :: Prefix         ! search prefix
REAL(sp),              INTENT(IN   ) :: DefValue       ! default value
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp), ALLOCATABLE, INTENT(INOUT) :: SpcScal(:)     ! Species scale factors
INTEGER,               INTENT(INOUT) :: RC             ! Success or failure?
```

REVISION HISTORY:

```
10 Jun 2015 - C. Keller - Initial version
20 Sep 2015 - C. Keller - Now allocate output array in this routine.
```

1.13.7 GetExtSpcVal_Int

Subroutine GetExtSpcVal_Int returns integer values associated with the species for a given extension. Specifically, this routine searches for extension setting '¡Prefix¿_SpecName' for every species passed through input argument SpcNames and writes those into output argument SpcScal. The default value DefValue is assigned to all elements of SpcScal with no corresponding extension setting.

INTERFACE:

```
SUBROUTINE GetExtSpcVal_Int( HcoConfig, ExtNr,    NSPC,    SpcNames, &
                             Prefix,    DefValue, SpcScal, RC         )
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),       POINTER       :: HcoConfig
INTEGER,               INTENT(IN   ) :: ExtNr          ! Extension Nr.
INTEGER,               INTENT(IN   ) :: NSPC           ! # of species
CHARACTER(LEN=*),      INTENT(IN   ) :: SpcNames(NSPC) ! Species string
CHARACTER(LEN=*),      INTENT(IN   ) :: Prefix         ! search prefix
INTEGER,               INTENT(IN   ) :: DefValue       ! default value
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,  ALLOCATABLE, INTENT(INOUT) :: SpcScal(:)     ! Species scale factors
INTEGER,               INTENT(INOUT) :: RC             ! Success or failure?
```

REVISION HISTORY:

```
10 Jun 2015 - C. Keller - Initial version
20 Sep 2015 - C. Keller - Now allocate output array in this routine.
```

1.13.8 GetExtSpcVal_Char

Subroutine GetExtSpcVal_Char returns character values associated with the species for a given extension. Specifically, this routine searches for extension setting '¡Prefix¿_SpecName' for every species passed through input argument SpcNames and writes those into output argument SpcScal. The default value DefValue is assigned to all elements of SpcScal with no corresponding extension setting.

INTERFACE:

```
SUBROUTINE GetExtSpcVal_Char( HcoConfig, ExtNr,    NSPC,    SpcNames, &
                              Prefix,    DefValue, SpcScal, RC )
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),               POINTER       :: HcoConfig
INTEGER,                       INTENT(IN   ) :: ExtNr          ! Extension Nr.
INTEGER,                       INTENT(IN   ) :: NSPC           ! # of species
CHARACTER(LEN=*),              INTENT(IN   ) :: SpcNames(NSPC) ! Species string
CHARACTER(LEN=*),              INTENT(IN   ) :: Prefix         ! search prefix
CHARACTER(LEN=*),              INTENT(IN   ) :: DefValue       ! default value
```

INPUT/OUTPUT PARAMETERS:

```
CHARACTER(LEN=*), ALLOCATABLE, INTENT(INOUT) :: SpcScal(:)     ! Species scale factors
INTEGER,                       INTENT(INOUT) :: RC             ! Success or failure?
```

REVISION HISTORY:

```
10 Jun 2015 - C. Keller - Initial version
20 Sep 2015 - C. Keller - Now allocate output array in this routine.
```

1.13.9 GetExtSpcVal_Dr

Subroutine GetExtSpcVal_Dr is the GetExtSpcVal driver routine.

INTERFACE:

```
SUBROUTINE GetExtSpcVal_Dr( HcoConfig, ExtNr, NSPC,    &
                            SpcNames,  Prefix, RC,     &
                            DefVal_SP, SpcScal_SP,     &
                            DefVal_Char, SpcScal_Char, &
                            DefVal_IN, SpcScal_IN       )
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),               POINTER                 :: HcoConfig
INTEGER,                       INTENT(IN   )           :: ExtNr          ! Extension N
INTEGER,                       INTENT(IN   )           :: NSPC           ! # of species
CHARACTER(LEN=*),              INTENT(IN   )           :: SpcNames(NSPC) ! Species str
CHARACTER(LEN=*),              INTENT(IN   )           :: Prefix         ! search pref
REAL(sp),                      INTENT(IN   ), OPTIONAL :: DefVal_SP      ! default valu
INTEGER,                       INTENT(IN   ), OPTIONAL :: DefVal_IN      ! default valu
CHARACTER(LEN=*),              INTENT(IN   ), OPTIONAL :: DefVal_Char    ! default valu
```

OUTPUT PARAMETERS:

```
REAL(sp),                      INTENT(  OUT), OPTIONAL :: SpcScal_SP(NSPC)   ! Species
INTEGER,                       INTENT(  OUT), OPTIONAL :: SpcScal_IN(NSPC)   ! Species
CHARACTER(LEN=*),              INTENT(  OUT), OPTIONAL :: SpcScal_Char(NSPC) ! Species
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,               INTENT(INOUT)           :: RC       ! Success or failure?
```

REVISION HISTORY:

```
10 Jun 2015 - C. Keller   - Initial version
20 Sep 2015 - C. Keller   - Options are now linked list.
```

1.13.10 SetExtNr

Subroutine SetExtNr overwrites the extension number of a given extension. The extension of interest is provided in argument ExtName. If this argument is omitted, the extension numbers of all extensions currently listed in ExtList will be set to the provided number. This is useful to disable all extensions by setting the ExtNr to a negative value.

INTERFACE:

```
SUBROUTINE SetExtNr( am_I_Root, HcoConfig, ExtNr, ExtName, RC )
```

USES:

```
USE CHARPAK_MOD,  ONLY : TRANLC
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root
TYPE(ConfigObj),  POINTER                 :: HcoConfig
INTEGER,          INTENT(IN   )           :: ExtNr
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL :: ExtName
!INPUT/OUTPUT PARAMETER:
INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
12 Jan 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.11 ExtNrInUse

Function ExtNrInUse checks if extension number ExtNr is in the list of used extensions or not.

INTERFACE:

```
FUNCTION ExtNrInUse( ExtList, ExtNr ) Result ( InUse )
```

INPUT PARAMETERS:

```
TYPE(Ext), POINTER       :: ExtList
INTEGER,   INTENT(IN   ) :: ExtNr
```

RETURN VALUE:

```
LOGICAL                  :: InUse
```

REVISION HISTORY:

```
03 Oct 2013 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.12 ExtFinal

Function ExtFinal finalizes the extensions list.

INTERFACE:

```
SUBROUTINE ExtFinal( ExtList )
!INPUT/OUTPUT ARGUMENT:
  TYPE(Ext), POINTER       :: ExtList
```

REVISION HISTORY:

```
03 Oct 2013 - C. Keller - Initial version
20 Sep 2015 - C. Keller - Options are now linked list.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.13 HCO_AddOpt

Subroutine HCO_AddOpt adds a option name/value pair to the list of options.

INTERFACE:

```
SUBROUTINE HCO_AddOpt ( am_I_Root, HcoConfig, OptName, OptValue, ExtNr, RC, &
                        VERB,      IgnoreIfExist )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root      ! Root CPU?
TYPE(ConfigObj),  POINTER                 :: HcoConfig      ! HEMCO config obj
CHARACTER(LEN=*), INTENT(IN   )           :: OptName        ! OptName
CHARACTER(LEN=*), INTENT(IN   )           :: OptValue       ! OptValue
INTEGER,          INTENT(IN   )           :: ExtNr          ! Extension Nr.
LOGICAL,          INTENT(IN   ), OPTIONAL :: VERB           ! Verbose on
LOGICAL,          INTENT(IN   ), OPTIONAL :: IgnoreIfExist  ! Ignore if already exists
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC             ! Return code
```

REVISION HISTORY:

```
18 Sep 2015 - C. Keller - Initial version
12 Dec 2015 - C. Keller - Added argument IgnoreIfExist
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.14 HCO_GetOpt

Subroutine HCO_GetOpt returns a option value for the given option name.

INTERFACE:

```
FUNCTION HCO_GetOpt ( ExtList, OptName, ExtNr ) RESULT ( OptValue )
```

INPUT PARAMETERS:

```
TYPE(Ext),        POINTER                 :: ExtList        ! Extension list
CHARACTER(LEN=*), INTENT(IN   )           :: OptName  ! OptName
INTEGER,          INTENT(IN   ), OPTIONAL :: ExtNr      ! Extension Nr.
```

OUTPUT PARAMETERS:

```
CHARACTER(LEN=OPTLEN)                     :: OptValue ! OptValue
```

REVISION HISTORY:

```
18 Sep 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.15 HCO_ROOT

Function HCO_ROOT returns the root character string. This is a wrapper routine equivalent to HCO_GetOpt('ROOT'). Since the ROOT character is called very frequently, it is recommended to use this routine instead.

INTERFACE:

```
FUNCTION HCO_ROOT ( HcoConfig ) RESULT ( OutRoot )
```

INPUT PARAMETERS:

OUTPUT PARAMETERS:

```
TYPE(ConfigObj), POINTER                  :: HcoConfig
CHARACTER(LEN=OPTLEN)                     :: OutRoot ! Root output
```

REVISION HISTORY:

```
18 Sep 2015 - C. Keller - Initial version
```

1.13.16 HCO_CleanupOpt

Subroutine HCO_CleanupOpt cleans up the given options linked list.

INTERFACE:

```
SUBROUTINE HCO_CleanupOpt ( OptList )
```

INPUT PARAMETERS:

OUTPUT PARAMETERS:

```
TYPE(Opt), POINTER        :: OptList
```

REVISION HISTORY:

```
18 Sep 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.13.17 HCO_SetDefaultToken

Subroutine HCO_SetDefaultToken is a wrapper routine to initialize the default set of HEMCO tokens. These can be obtained at any place in the HEMCO code via subroutine HCO_GetOpt, e.g. HCO_GetOpt('RES') will return the 'RES' token.

INTERFACE:

```
SUBROUTINE HCO_SetDefaultToken ( am_I_Root, CF, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root  ! Root CPU?
TYPE(ConfigObj),  POINTER                 :: CF
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC         ! Return code
```

REVISION HISTORY:

```
18 Sep 2015 - C. Keller - Initial version
```

1.14 Fortran: Module Interface hco_geotools_mod.F90

Module HCO_GeoTools_Mod contains a collection of helper routines for extracting geographical information. These routines are based upon GEOS-5 data and may need to be revised for other met. fields!

INTERFACE:

```
MODULE HCO_GeoTools_Mod
```

USES:

```
USE HCO_Error_Mod

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCO_LandType
PUBLIC :: HCO_ValidateLon
PUBLIC :: HCO_GetSUNCOS
PUBLIC :: HCO_GetHorzIJIndex
PUBLIC :: HCO_CalcVertGrid
PUBLIC :: HCO_SetPBLm
 PUBLIC :: HCO_CalcPBLlev

INTERFACE HCO_LandType
   MODULE PROCEDURE HCO_LandType_Dp
   MODULE PROCEDURE HCO_LandType_Sp
END INTERFACE HCO_LandType

INTERFACE HCO_ValidateLon
   MODULE PROCEDURE HCO_ValidateLon_Dp
   MODULE PROCEDURE HCO_ValidateLon_Sp
END INTERFACE HCO_ValidateLon

 INTERFACE HCO_CalcPBLlev
    MODULE PROCEDURE HCO_CalcPBLlev2D
    MODULE PROCEDURE HCO_CalcPBLlev3D
 END INTERFACE HCO_CalcPBLlev
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE:: HCO_LandType_Dp
PRIVATE:: HCO_LandType_Sp
PRIVATE:: HCO_ValidateLon_Dp
PRIVATE:: HCO_ValidateLon_Sp
```

REVISION HISTORY:

```
18 Dec 2013 - C. Keller   - Initialization
01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
16 Jul 2014 - C. Keller   - Added HCO_ValidateLon
```

1.14.1 HCO_LandType_Sp

Function HCO_LANDTYPE returns the land type based upon the land water index (0=water,1=land,2=ice) and the surface albedo. Inputs are in single precision.

INTERFACE:

```
FUNCTION HCO_LandType_Sp( WLI, Albedo ) Result ( LandType )
```

INPUT PARAMETERS:

```
 REAL(sp), INTENT(IN) :: WLI       ! Land type: 0=water,1=land,2=ice
 REAL(sp), INTENT(IN) :: Albedo    ! Surface albedo
!RETURN VALUE
 INTEGER              :: LandType  ! Land type: 0=water,1=land,2=ice
```

REMARKS:

```
This function is largely based on the GEOS-Chem functions in dao_mod.F.
```

REVISION HISTORY:

```
18 Dec 2013 - C. Keller - Initialization!
```

1.14.2 HCO_LandType_Dp

Function HCO_LandType_Dp returns the land type based upon the land water index (0=water,1=land,2=ice) and the surface albedo. Inputs are in double precision.

INTERFACE:

```
FUNCTION HCO_LandType_Dp( WLI, Albedo ) Result ( LandType )
```

INPUT PARAMETERS:

```
 REAL(dp), INTENT(IN) :: WLI       ! Land type: 0=water,1=land,2=ice
 REAL(dp), INTENT(IN) :: Albedo    ! Surface albedo
```

RETURN VALUE:

```
 INTEGER              :: LandType  ! Land type: 0=water,1=land,2=ice
```

REMARKS:

```
This function is largely based on the GEOS-Chem functions in dao_mod.F.
```

REVISION HISTORY:

```
18 Dec 2013 - C. Keller - Initialization
```

1.14.3 HCO_ValidateLon_Sp

Subroutine HCO_ValidateLon_Sp ensures that the passed single precision longitude axis LON is steadily increasing.

INTERFACE:

```
SUBROUTINE HCO_ValidateLon_Sp ( HcoState, NLON, LON, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : HCO_STATE
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState       ! HEMCO state object
INTEGER,         INTENT(IN   ) :: NLON        ! # of lons
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp), INTENT(INOUT) :: LON(NLON)   ! longitude axis
INTEGER,  INTENT(INOUT) :: RC          ! Return code
```

REVISION HISTORY:

```
16 Jul 2014 - C. Keller - Initialization
```

1.14.4 HCO_ValidateLon_Dp

Subroutine HCO_ValidateLon_Sp ensures that the passed double precision longitude axis LON is steadily increasing.

INTERFACE:

```
SUBROUTINE HCO_ValidateLon_Dp ( HcoState, NLON, LON, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : HCO_STATE
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState       ! HEMCO state object
INTEGER,         INTENT(IN   ) :: NLON        ! # of lons
```

INPUT/OUTPUT PARAMETERS:

```
REAL(dp), INTENT(INOUT) :: LON(NLON)   ! longitude axis
INTEGER,  INTENT(INOUT) :: RC          ! Return code
```

REVISION HISTORY:

```
16 Jul 2014 - C. Keller - Initialization
```

1.14.5 HCO_GetSUNCOS

Subroutine HCO_GetSUNCOS calculates the solar zenith angle for the given date.

INTERFACE:

```
SUBROUTINE HCO_GetSUNCOS( am_I_Root, HcoState, SUNCOS, DT, RC )
!USES
  USE HCO_STATE_MOD,   ONLY : HCO_STATE
  USE HCO_CLOCK_MOD,   ONLY : HcoClock_Get
  USE HCO_CLOCK_MOD,   ONLY : HcoClock_GetLocal
```

INPUT/OUTPUT PARAMETERS:

```
  LOGICAL,         INTENT(IN   )  :: am_I_Root      ! Root CPU?
  TYPE(HCO_State), POINTER        :: HcoState       ! HEMCO state object
  INTEGER,         INTENT(IN   )  :: DT             ! Time shift relative to current date
```

OUTPUT PARAMETERS:

```
  REAL(hp),        INTENT(  OUT)  :: SUNCOS(HcoState%NX,HcoState%NY)
```

INPUT/OUTPUT PARAMETERS:

```
  INTEGER,         INTENT(INOUT)  :: RC             ! Return code
```

REVISION HISTORY:

```
  22 May 2015 - C. Keller   - Initial version, based on GEOS-Chem's dao_mod.F.
  10 Jul 2015 - R. Yantosca - Corrected issues in ProTeX header
  02 Mar 2017 - R. Yantosca - Now compute local time as UTC + Longitude/15,
                              so as to avoid using Voronoi TZ's for SUNCOS
```

1.14.6 HCO_GetHorzIJIndex

Function HCO_GetHorzIJIndex returns the grid box index for the given longitude (deg E, -180...180), and latitude (deg N, -90...90).

INTERFACE:

```
SUBROUTINE HCO_GetHorzIJIndex( am_I_Root, HcoState, N, Lon, Lat, idx, jdx, RC )
!USES
#include "MAPL_Generic.h"
  USE ESMF
  USE MAPL_Mod
  USE HCO_STATE_MOD,   ONLY : HCO_STATE
```

INPUT PARAMETERS:

```
  LOGICAL,         INTENT(IN   )  :: am_I_Root      ! Root CPU?
  TYPE(HCO_State), POINTER        :: HcoState       ! HEMCO state object
  INTEGER,         INTENT(IN   )  :: N
  REAL(hp),        INTENT(IN   )  :: Lon(N)
  REAL(hp),        INTENT(IN   )  :: Lat(N)
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)  :: RC
```

OUTPUT PARAMETERS:

```
INTEGER,         INTENT(  OUT)  :: IDX(N), JDX(N)
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller - Initial version
10 Jul 2015 - R. Yantosca - Corrected issues in ProTeX header
```

1.14.7 HCO_GetHorzIJIndex

Function HCO_GetHorzIJIndex returns the grid box index for the given longitude (deg E, -180...180), and latitude (deg N, -90...90).

INTERFACE:

```
SUBROUTINE HCO_GetHorzIJIndex( am_I_Root, HcoState, N, Lon, Lat, idx, jdx, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : HCO_STATE
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root      ! Root CPU?
TYPE(HCO_State), POINTER        :: HcoState       ! HEMCO state object
INTEGER,         INTENT(IN   )  :: N
REAL(hp),        INTENT(IN   )  :: Lon(N)
REAL(hp),        INTENT(IN   )  :: Lat(N)
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)  :: RC
```

OUTPUT PARAMETERS:

```
INTEGER,         INTENT(  OUT)  :: IDX(N), JDX(N)
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller - Initial version
10 Jul 2015 - R. Yantosca - Corrected issues in ProTeX header
```

1.14.8 HCO_CalcVertGrid

Function HCO_CalcVertGrid calculates the vertical grid quantities surface pressure PSFC [Pa], surface geopotential height ZSFC [m], grid box height BXHEIGHT [m], and pressure edges PEDGE [Pa]. Any of these fields can be passed explicitly to the routine, in which case these fields are being used. If not passed through the routine (i.e. if the corresponding input argument pointer is nullified), the field is searched in the HEMCO configuration file. If not found in the configuration file, the field is approximated from other quantities (if possible). For example, if surface pressures are provided (either passed as argument or in the HEMCO configuration file as field PSFC), pressure edges are calculated from PSFC and the vertical grid coordinates (Ap and Bp for a hybrid sigma coordinate system). The temperature field TK [K] is needed to approximate box heights and/or geopotential height (via the hydrostatic equation).

INTERFACE:

```
SUBROUTINE HCO_CalcVertGrid ( am_I_Root, HcoState, PSFC,    &
                              ZSFC,  TK, BXHEIGHT, PEDGE, RC )
!USES
  USE HCO_Arr_Mod,      ONLY : HCO_ArrAssert
  USE HCO_STATE_MOD,    ONLY : HCO_STATE
  USE HCO_CALC_MOD,     ONLY : HCO_EvalFld
```

INPUT PARAMETERS:

```
  LOGICAL,         INTENT(IN   )  :: am_I_Root       ! Root CPU?
  TYPE(HCO_State), POINTER        :: HcoState        ! HEMCO state object
  REAL(hp),        POINTER        :: PSFC(:,:)       ! surface pressure (Pa)
  REAL(hp),        POINTER        :: ZSFC(:,:)       ! surface geopotential height (m)
  REAL(hp),        POINTER        :: TK  (:,:,:)     ! air temperature (K)
  REAL(hp),        POINTER        :: BXHEIGHT(:,:,:) ! grid box height (m)
  REAL(hp),        POINTER        :: PEDGE(:,:,:)    ! pressure edges (Pa)
```

INPUT/OUTPUT PARAMETERS:

```
  INTEGER,         INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
28 Sep 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.14.9 HCO_SetPBLm

Subroutine HCO_SetPBLm sets the HEMCO PBL mixing height in meters. It first tries to read it from field 'FldName' (from the HEMCO data list), then to fill it from field 'PBLM', and then assigns the default value 'DefVal' to it.

INTERFACE:

```
SUBROUTINE HCO_SetPBLm ( am_I_Root, HcoState, FldName, PBLM, DefVal, RC )
!USES
  USE HCO_Arr_Mod,      ONLY : HCO_ArrAssert
  USE HCO_STATE_MOD,    ONLY : HCO_STATE
  USE HCO_CALC_MOD,     ONLY : HCO_EvalFld
```

INPUT PARAMETERS:

```
  LOGICAL,          INTENT(IN   )           :: am_I_Root   ! Root CPU?
  TYPE(HCO_State),  POINTER                 :: HcoState    ! HEMCO state object
  CHARACTER(LEN=*), OPTIONAL, INTENT(IN   ) :: FldName     ! field name
  REAL(hp),         OPTIONAL, POINTER       :: PBLM(:,:)   ! pbl mixing height
  REAL(hp),         OPTIONAL, INTENT(IN   ) :: DefVal      ! default value
```

INPUT/OUTPUT PARAMETERS:

```
  INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
28 Sep 2015 - C. Keller - Initial version
```

1.15 Fortran: Module Interface hco_clock_mod.F90

Module HCO_Clock_Mod contains routines and variables to handle the HEMCO time and calendar settings through the HEMCO clock object. The HEMCO clock carries information of the current UTC/local time as well as the UTC times from the previous time step. These values should be updated on every time step (–¿ HcoClock_Set). It also contains separate variables for the current and previous emission datetime (year, month, day, hour, min, sec). This allows us to keep track of emission time steps even if the emission time steps are less frequent than the regular time step.

Subroutine HcoClock_EmissionsDone indicates that emisisons have been completely calculated for the current time step. Any calls to HcoClock_Get will return IsEmisTime FALSE until the clock has been advanced to the next emission time step (via HcoClock_Set).

The HEMCO clock object HcoClock is a private object and cannot be accessed directly from outside of this module. The HcoClock_Get routine should be used instead. There are also some wrapper routines for frequently used checks, i.e. if this is a new year, month, etc.

Local times are calculated for 26 time zones, ranging from UTC-12 hours to UTC+13 hours. The time zone to be used at a given grid point is based on its geographical position. By default, the time zone is picked according to the longitude, with each time zone spanning 15 degrees. More detailed time zones can be provided through an external input file, specified in the HEMCO configuration file. The field name must be 'TIMEZONES', and the file must contain UTC offsets in hours. If such a file is provided, the time zones are determined based on these values. Minute offsets are ignored, e.g. UTC+9hr30min is treated as UTC+9hr. If the input field contains any invalid values (e.g. outside the range

of UTC-12 - UTC+13 hours), the default algorithm is applied.

The HEMCO clock object also controls cases where the emission dates shall be held constant, e.g. for simulations where emission year 2000 shall be used irrespective of the simulation date. Fixed simulation dates can be set in the settings section of the HEMCO configuration file via settings 'Emission year', 'Emission month', 'Emission day', and 'Emission hour'. Only a subset of those settings can be provided, in which case all other time attributes will be taken from the simulation datetime.

INTERFACE:

```
MODULE HCO_CLOCK_MOD
```

USES:

```
USE HCO_Error_Mod
USE Julday_Mod
USE HCO_TYPES_MOD, ONLY : HcoClock

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
! HEMCO Clock object:
PUBLIC :: HcoClock_Init
PUBLIC :: HcoClock_InitTzPtr
PUBLIC :: HcoClock_Set
PUBLIC :: HcoClock_Get
PUBLIC :: HcoClock_GetLocal
PUBLIC :: HcoClock_Cleanup
PUBLIC :: HcoClock_NewYear
PUBLIC :: HcoClock_NewMonth
PUBLIC :: HcoClock_NewDay
PUBLIC :: HcoClock_NewHour
PUBLIC :: HcoClock_First
PUBLIC :: HcoClock_Rewind
PUBLIC :: HcoClock_CalcDOY
PUBLIC :: HcoClock_Increase
PUBLIC :: HcoClock_EmissionsDone
PUBLIC :: HcoClock_SetLast
PUBLIC :: Get_LastDayOfMonth
```

REMARKS:

```
The current local time implementation assumes a regular grid,
i.e. local time does not change with latitude
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller   - Initialization
```

```
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
08 Oct 2014 - C. Keller   - Added mid-month day calculation
03 Dec 2014 - C. Keller   - Now use fixed number of time zones (24)
12 Jan 2015 - C. Keller   - Added emission time variables.
02 Feb 2015 - C. Keller   - Added option to get time zones from file
23 Feb 2015 - R. Yantosca - Added routine HcoClock_InitTzPtr
11 Jun 2015 - C. Keller   - Added simulation times and option to fix
                            emission year, month, day, and/or hour.
```

1.15.1 HcoClock_Init

Subroutine HcoClock_Init initializes the HEMCO clock.

INTERFACE:

```
SUBROUTINE HcoClock_Init ( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_ARR_MOD,     ONLY : HCO_ArrInit
USE HCO_STATE_MOD,   ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root ! Root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState  ! HEMCO state obj
INTEGER,         INTENT(INOUT)  :: RC        ! Success or failure?
```

REVISION HISTORY:

```
10 Sep 2013 - C. Keller - Initialization
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
```

1.15.2 HcoClock_InitTzPtr

Subroutine HcoClock_InitTzPtr initializes the TIMEZONES module variable. TIMEZONES points to the timezones data (i.e. offsets from UTC in hours) as read from disk. If the timezones data file is not being used, then the TIMEZONES pointer will be left unassociated.

INTERFACE:

```
SUBROUTINE HcoClock_InitTzPtr( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
USE HCO_EMISLIST_MOD, ONLY : HCO_GetPtr
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root  ! Root CPU?
TYPE(HCO_State), POINTER        :: HcoState  ! HcoState object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT)          :: RC         ! Success or failure?
```

REMARKS:

```
This routine has to be called in the HCO_Run routine, immediately after
the call to ReadList_Read.   The HEMCO configuration file has to be read
first in order to determine if we are getting our timezone information from
a file, or if we are computing it just based on longitude in the default
manner.
```

REVISION HISTORY:

```
23 Feb 2015 - R. Yantosca - Initial version
10 Mar 2015 - C. Keller   - Packed message into am_I_Root statement
```

1.15.3 HcoClock_Set

Subroutine HcoClock_Set updates the HEMCO clock. These routine should be called at the beginning of every emission time step! If the current day of year (cDoy) is not provided, it is automatically calculated from the current date.

INTERFACE:

```
SUBROUTINE HcoClock_Set ( am_I_Root, HcoState, cYr,  cMt,   cDy, cHr, &
                          cMin,      cSec,     cDOY, IsEmisTime, RC    )
```

USES:

```
USE HCO_TYPES_MOD,   ONLY : ConfigObj, Ext
USE HCO_STATE_MOD,   ONLY : HCO_State
USE HCO_EXTLIST_MOD, ONLY : GetExtOpt, CoreNr
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )           :: am_I_Root ! Root CPU?
INTEGER,         INTENT(IN   )           :: cYr       ! Current year
INTEGER,         INTENT(IN   )           :: cMt       ! Current month
INTEGER,         INTENT(IN   )           :: cDy       ! Current day
INTEGER,         INTENT(IN   )           :: cHr       ! Current hour
INTEGER,         INTENT(IN   )           :: cMin      ! Current minute
INTEGER,         INTENT(IN   )           :: cSec      ! Current second
INTEGER,         INTENT(IN   ), OPTIONAL :: cDoy      ! Current day of year
LOGICAL,         INTENT(IN   ), OPTIONAL :: IsEmisTime! Is it time for emissions?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER                :: HcoState  ! HcoState object
INTEGER,         INTENT(INOUT)          :: RC        ! Success or failure?
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller   - Initialization
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
08 Jul 2014 - C. Keller   - Now calculate DOY if not provided
12 Jan 2015 - C. Keller   - Added IsEmisTime
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.15.4 HcoClock_Get

Subroutine HcoClock_Get returns the selected UTC variables from the HEMCO clock object.

INTERFACE:

```
SUBROUTINE HcoClock_Get ( am_I_Root, Clock,             &
                          cYYYY,   cMM, cDD,  cH,       &
                          cM,      cS,  cDOY, cWEEKDAY, &
                          pYYYY,   pMM, pDD,  pH,       &
                          pM,      pS,  pDOY, pWEEKDAY, &
                          sYYYY,   sMM, sDD,  sH,       &
                          sM,      sS,                  &
                          LMD,     nSteps,    cMidMon,  &
                          dslmm,   dbtwmm,    IsEmisTime, &
                          IsLast,  RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN   )              :: am_I_Root  ! Root CPU
TYPE(HcoClock), POINTER             :: Clock   ! HEMCO clock obj
```

OUTPUT PARAMETERS:

```
INTEGER, INTENT(  OUT), OPTIONAL    :: cYYYY      ! Current year
INTEGER, INTENT(  OUT), OPTIONAL    :: cMM        ! Current month
INTEGER, INTENT(  OUT), OPTIONAL    :: cDD        ! Current day
INTEGER, INTENT(  OUT), OPTIONAL    :: cH         ! Current hour
INTEGER, INTENT(  OUT), OPTIONAL    :: cM         ! Current minute
INTEGER, INTENT(  OUT), OPTIONAL    :: cS         ! Current second
INTEGER, INTENT(  OUT), OPTIONAL    :: cDOY       ! Current day of year
```

```
INTEGER, INTENT(  OUT), OPTIONAL     :: cWEEKDAY   ! Current weekday
INTEGER, INTENT(  OUT), OPTIONAL     :: pYYYY      ! Previous year
INTEGER, INTENT(  OUT), OPTIONAL     :: pMM        ! Previous month
INTEGER, INTENT(  OUT), OPTIONAL     :: pDD        ! Previous day
INTEGER, INTENT(  OUT), OPTIONAL     :: pH         ! Previous hour
INTEGER, INTENT(  OUT), OPTIONAL     :: pM         ! Previous minute
INTEGER, INTENT(  OUT), OPTIONAL     :: pS         ! Previous second
INTEGER, INTENT(  OUT), OPTIONAL     :: sYYYY      ! Simulation year
INTEGER, INTENT(  OUT), OPTIONAL     :: sMM        ! Simulation month
INTEGER, INTENT(  OUT), OPTIONAL     :: sDD        ! Simulation day
INTEGER, INTENT(  OUT), OPTIONAL     :: sH         ! Simulation hour
INTEGER, INTENT(  OUT), OPTIONAL     :: sM         ! Simulation minute
INTEGER, INTENT(  OUT), OPTIONAL     :: sS         ! Simulation second
INTEGER, INTENT(  OUT), OPTIONAL     :: pDOY       ! Previous day of year
INTEGER, INTENT(  OUT), OPTIONAL     :: pWEEKDAY   ! Previous weekday
INTEGER, INTENT(  OUT), OPTIONAL     :: LMD        ! Last day of month
INTEGER, INTENT(  OUT), OPTIONAL     :: nSteps     ! # of passed steps
INTEGER, INTENT(  OUT), OPTIONAL     :: cMidMon    ! Mid-month day of curr. month
INTEGER, INTENT(  OUT), OPTIONAL     :: dslmm      ! days since last mid-month
INTEGER, INTENT(  OUT), OPTIONAL     :: dbtwmm     ! days between mid-month
LOGICAL, INTENT(  OUT), OPTIONAL     :: IsEmisTime ! days between mid-month
LOGICAL, INTENT(  OUT), OPTIONAL     :: IsLast     ! last time step?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT)               :: RC        ! Success or failure?
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller - Initialization
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
08 Oct 2014 - C. Keller   - Added mid-month day arguments
12 Jan 2015 - C. Keller   - Added IsEmisTime
```

1.15.5 HcoClock_GetLocal

Subroutine HcoClock_GetLocal returns the selected local time variables from the HEMCO clock object for the given longitude and latitude. At the moment, the time zone is selected purely on the given longitude and the passed latitude is not evaluated.

INTERFACE:

```
SUBROUTINE HcoClock_GetLocal ( HcoState,   I,     J, cYYYY, cMM, &
                               cDD, cH,    CWEEKDAY, RC, verb     )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
```

INPUT PARAMETERS:

```
TYPE(HCO_State), POINTER          :: HcoState  ! Hemco state
INTEGER,  INTENT(IN   )           :: I         ! Longitude index
INTEGER,  INTENT(IN   )           :: J         ! Latitude  index
```

OUTPUT PARAMETERS:

```
INTEGER,  INTENT(  OUT), OPTIONAL :: cYYYY     ! Current year
INTEGER,  INTENT(  OUT), OPTIONAL :: cMM       ! Current month
INTEGER,  INTENT(  OUT), OPTIONAL :: cDD       ! Current day
REAL(hp), INTENT(  OUT), OPTIONAL :: cH        ! Current hour
INTEGER,  INTENT(  OUT), OPTIONAL :: cWEEKDAY  ! Current weekday
INTEGER,  INTENT(IN   ), OPTIONAL :: verb      ! verbose
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,  INTENT(INOUT)           :: RC        ! Success or failure?
```

REMARKS:

```
Module variable TIMEZONES points to the timezone data (i.e. offsets in
hours from UTC) as read from disk.  The data file containing UTC offsets
is specified in the "NON-EMISSIONS DATA" section of the HEMCO configuraiton
file, under the container name "TIMEZONES".
The TIMEZONES module variable is initialized by calling HcoClock_InitTzPtr.
in the HEMCO run method HCO_Run (in module hco_driver_mod.F90).  The call
to HcoClock_InitTzPtr immediately follows the call to ReadList_Read, and
is only done on the very first emissions timestep.  The reason we have to
initialize the TIMEZONES module variable in the run method (instead of in
the init method) is because the HEMCO configuration file has to be read
before the timezones  data can be loaded into a HEMCO data container.
If we are not reading timezone data from a file, then the TIMEZONES
module variable will remain unassociated.
This fix was necessary in order to avoid segmentation faults when running
with OpenMP parallelization turned on.
   -- Bob Yantosca (23 Feb 2015)
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller   - Initialization
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Feb 2015 - R. Yantosca - Remove call to Hco_GetPtr: this was causing
                            errors on runs with OpenMP parallelization
```

1.15.6 HcoClock_First

Function HcoClock_First returns TRUE on the first HEMCO time step, FALSE otherwise.

INTERFACE:

```
FUNCTION HcoClock_First( Clock, EmisTime ) RESULT ( First )
!INPUT ARGUMENTS:
  TYPE(HcoClock),  POINTER    :: Clock
  LOGICAL,         INTENT(IN) :: EmisTime
```

RETURN VALUE:

```
  LOGICAL :: First
```

REVISION HISTORY:

```
 29 Dec 2012 - C. Keller   - Initialization
 12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
 06 Apr 2015 - C. Keller   - Now use nEmisSteps and nSteps instead of
                             previous years.
```

1.15.7 HcoClock_Rewind

Function HcoClock_Rewind returns TRUE if the last archived HEMCO time step is not in the past.

INTERFACE:

```
FUNCTION HcoClock_Rewind( Clock, EmisTime ) RESULT ( Rwnd )
!INPUT ARGUMENTS:
  TYPE(HcoClock),  POINTER    :: Clock
  LOGICAL,         INTENT(IN) :: EmisTime
```

RETURN VALUE:

```
  LOGICAL :: Rwnd
```

REVISION HISTORY:

```
 08 May 2015 - C. Keller   - Initial version
```

1.15.8 HcoClock_NewYear

Function HcoClock_NewYear returns TRUE if this is a new year (compared to the previous emission time step), FALSE otherwise.

INTERFACE:

```
FUNCTION HcoClock_NewYear( Clock, EmisTime ) RESULT ( NewYear )
!INPUT ARGUMENTS:
  TYPE(HcoClock),  POINTER    :: Clock
  LOGICAL,         INTENT(IN) :: EmisTime
```

RETURN VALUE:

```
LOGICAL  :: NewYear
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller - Initialization
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
```

1.15.9 HcoClock_NewMonth

Function HcoClock_NewMonth returns TRUE if this is a new month (compared to the previous emission time step), FALSE otherwise.

INTERFACE:

```
FUNCTION HcoClock_NewMonth( Clock, EmisTime ) RESULT ( NewMonth )
!INPUT ARGUMENTS:
  TYPE(HcoClock),  POINTER    :: Clock
  LOGICAL,         INTENT(IN) :: EmisTime
```

RETURN VALUE:

```
LOGICAL  :: NewMonth
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller - Initialization
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
```

1.15.10 HcoClock_NewDay

Function HcoClock_NewDay returns TRUE if this is a new day (compared to the previous emission time step), FALSE otherwise.

INTERFACE:

```
FUNCTION HcoClock_NewDay( Clock, EmisTime ) RESULT ( NewDay )
!INPUT ARGUMENTS:
  TYPE(HcoClock),  POINTER    :: Clock
  LOGICAL,         INTENT(IN) :: EmisTime
```

RETURN VALUE:

```
LOGICAL  :: NewDay
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller - Initialization
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
```

1.15.11 HcoClock_NewHour

Function HcoClock_NewHour returns TRUE if this is a new hour (compared to the previous emission time step), FALSE otherwise.

INTERFACE:

```
FUNCTION HcoClock_NewHour( Clock, EmisTime ) RESULT ( NewHour )
!INPUT ARGUMENTS:
  TYPE(HcoClock),  POINTER    :: Clock
  LOGICAL,         INTENT(IN) :: EmisTime
```

RETURN VALUE:

```
  LOGICAL  :: NewHour
```

REVISION HISTORY:

```
 29 Dec 2012 - C. Keller - Initialization
 12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
```

1.15.12 HcoClock_Cleanup

Subroutine HcoClock_Cleanup removes the given HcoHcoClock type.

INTERFACE:

```
SUBROUTINE HcoClock_Cleanup ( Clock )
```

USES:

```
  USE HCO_ARR_MOD,  ONLY : HCO_ArrCleanup
!INPUT ARGUMENTS:
  TYPE(HcoClock), POINTER    :: Clock
```

REVISION HISTORY:

```
 29 Dec 2012 - C. Keller - Initialization
 12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
```

1.15.13 HCO_GetWeekday

Function HCO_GetWeekday returns the weekday for the given date (year, month, day). 0 = Sunday, 1 = Monday, ..., 6 = Saturday.

INTERFACE:

```
FUNCTION HCO_GetWeekday( year, month, day, gmt ) RESULT ( weekday )
```

INPUT PARAMETERS:

```
INTEGER,  INTENT(IN)   :: year
INTEGER,  INTENT(IN)   :: month
INTEGER,  INTENT(IN)   :: day
REAL(sp), INTENT(IN)   :: gmt
```

RETURN VALUE:

```
INTEGER               :: weekday
! NOTES: This function is largely based on the GEOS-Chem functions
in time_mod.F.
```

REVISION HISTORY:

```
18 Dec 2013 - C. Keller - Initialization
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
```

1.15.14 get_lastdayofmonth

Function GET_LASTDAYOFMONTH returns the last day of MONTH.

INTERFACE:

```
FUNCTION Get_LastDayOfMonth( Month, Year ) RESULT ( LastDay )
```

INPUT PARAMETERS:

```
INTEGER, INTENT(IN) :: Month
INTEGER, INTENT(IN) :: Year
```

RETURN VALUE:

```
INTEGER             :: LastDay
```

REVISION HISTORY:

```
13 Jan 2014 - C. Keller - Initial version
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
```

1.15.15 Set_LocalTime

Subroutine Set_LocalTime sets the local time vectors in the HEMCO clock object. Local time is calculated for each of the 24 defined time zones.

INTERFACE:

```
SUBROUTINE Set_LocalTime ( am_I_Root, HcoState, Clock, UTC, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root ! Root CPU?
TYPE(HCO_STATE), POINTER       :: HcoState
TYPE(HcoClock),  POINTER       :: Clock  ! Clock object
REAL(sp),        INTENT(IN   ) :: UTC       ! UTC time
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC        ! Success or failure?
```

REVISION HISTORY:

```
13 Jan 2014 - C. Keller   - Initial version
12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
03 Dec 2014 - C. Keller   - Now use fixed number of time zones (24)
```

1.15.16 HcoClock_CalcDOY

FUNCTION HcoClock_CalcDOY calculates the day of year for the given year, month, and day.

INTERFACE:

```
FUNCTION HcoClock_CalcDOY( YYYY, MM, DD ) RESULT ( DOY )
!INPUT ARGUMENTS:
  INTEGER, INTENT(IN) :: YYYY  ! Year
  INTEGER, INTENT(IN) :: MM    ! Month
  INTEGER, INTENT(IN) :: DD    ! Day
```

RETURN VALUE:

```
INTEGER             :: DOY   ! Day of year
```

REVISION HISTORY:

```
08 Jul 2014 - C. Keller - Initial version
```

1.15.17 HcoClock_Increase

Subroutine HcoClock_Increase increases the HEMCO clock by the specified time.

INTERFACE:

```
SUBROUTINE HcoClock_Increase ( am_I_Root, HcoState, TimeStep, EmisTime, RC )
```

USES:

```
USE HCO_STATE_MOD, ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root ! Root CPU
TYPE(HCO_State), POINTER       :: HcoState  ! Hemco state
REAL(sp),        INTENT(IN   ) :: TimeStep  ! Time step increase [s]
LOGICAL,         INTENT(IN   ) :: EmisTime  ! Is new time step emission time?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC        ! Success or failure?
```

REVISION HISTORY:

```
29 Jul 2014 - C. Keller - Initial version
08 Sep 2014 - C. Keller - Bug fix: now calculate UTC as fraction of day.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.15.18 HcoClock_EmissionsDone

Subroutine HcoClock_EmissionsDone marks the current (emission) time step as having emissions completed. This is useful if the HEMCO core routines are called multiple times on the same time step, e.g. if there are two run phases.

INTERFACE:

```
SUBROUTINE HcoClock_EmissionsDone( am_I_Root, Clock, RC )
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root ! Root CPU
TYPE(HcoClock),  POINTER       :: Clock     ! HEMCO clock obj
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC        ! Success or failure?
```

REVISION HISTORY:

```
13 Jan 2015 - C. Keller - Initial version
```

1.15.19 HcoClock_SetLast

Subroutine HcoClock_SetLast sets the IsLast flag.

INTERFACE:

```
SUBROUTINE HcoClock_SetLast( am_I_Root, Clock, IsLast, RC )
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root ! Root CPU
TYPE(HcoClock),  POINTER       :: Clock     ! HEMCO clock obj
LOGICAL,         INTENT(IN   ) :: IsLast    ! Is last time step?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC        ! Success or failure?
```

REVISION HISTORY:

```
01 Nov 2016 - C. Keller - Initial version
```

1.16 Fortran: Module Interface hco_vertgrid_mod.F90

Module HCO_VERTGRID_Mod contains routines and variables for the definition of the vertical grid and related calculations.

INTERFACE:

```
MODULE HCO_VertGrid_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Arr_Mod
USE HCO_Types_Mod, ONLY : VertGrid

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCO_VertGrid_Init
PUBLIC :: HCO_VertGrid_Define
PUBLIC :: HCO_VertGrid_Cleanup
```

PRIVATE MEMBER FUNCTIONS:

PARAMETERS:

```
   INTEGER,  PARAMETER, PUBLIC  :: HCO_ZTYPE_HYBSIG = 1

   ! Ap [Pa] for 47 levels (48 edges)
   REAL(hp), PARAMETER          :: Ap47(48) = (/                        &
               0.000000e+00_hp, 4.804826e-00_hp, 6.593752e+02_hp, 1.313480e+03_hp, &
               1.961311e+03_hp, 2.609201e+03_hp, 3.257081e+03_hp, 3.898201e+03_hp, &
               4.533901e+03_hp, 5.169611e+03_hp, 5.805321e+03_hp, 6.436264e+03_hp, &
               7.062198e+03_hp, 7.883422e+03_hp, 8.909992e+03_hp, 9.936521e+03_hp, &
               1.091817e+04_hp, 1.189586e+04_hp, 1.286959e+04_hp, 1.429100e+04_hp, &
               1.562600e+04_hp, 1.696090e+04_hp, 1.816190e+04_hp, 1.930970e+04_hp, &
               2.032590e+04_hp, 2.121500e+04_hp, 2.187760e+04_hp, 2.238980e+04_hp, &
               2.243630e+04_hp, 2.168650e+04_hp, 2.011920e+04_hp, 1.769300e+04_hp, &
               1.503930e+04_hp, 1.278370e+04_hp, 1.086630e+04_hp, 9.236572e+03_hp, &
               7.851231e+03_hp, 5.638791e+03_hp, 4.017541e+03_hp, 2.836781e+03_hp, &
               1.979160e+03_hp, 9.292942e+02_hp, 4.076571e+02_hp, 1.650790e+02_hp, &
               6.167791e+01_hp, 2.113490e+01_hp, 6.600001e+00_hp, 1.000000e+00_hp  &
                                                                       /)

   ! Bp [unitless] for 47 levels (48 edges)
   REAL(hp), PARAMETER          :: Bp47(48) = (/                        &
               1.000000e+00_hp, 9.849520e-01_hp, 9.634060e-01_hp, 9.418650e-01_hp, &
               9.203870e-01_hp, 8.989080e-01_hp, 8.774290e-01_hp, 8.560180e-01_hp, &
               8.346609e-01_hp, 8.133039e-01_hp, 7.919469e-01_hp, 7.706375e-01_hp, &
               7.493782e-01_hp, 7.211660e-01_hp, 6.858999e-01_hp, 6.506349e-01_hp, &
               6.158184e-01_hp, 5.810415e-01_hp, 5.463042e-01_hp, 4.945902e-01_hp, &
               4.437402e-01_hp, 3.928911e-01_hp, 3.433811e-01_hp, 2.944031e-01_hp, &
               2.467411e-01_hp, 2.003501e-01_hp, 1.562241e-01_hp, 1.136021e-01_hp, &
               6.372006e-02_hp, 2.801004e-02_hp, 6.960025e-03_hp, 8.175413e-09_hp, &
               0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
               0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
               0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
               0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp  &
                                                                       /)

   ! Ap [Pa] for 72 levels (73 edges)
   REAL(hp), PARAMETER          :: Ap72(73) = (/                        &
               0.000000e+00_hp, 4.804826e+00_hp, 6.593752e+02_hp, 1.313480e+03_hp, &
               1.961311e+03_hp, 2.609201e+03_hp, 3.257081e+03_hp, 3.898201e+03_hp, &
               4.533901e+03_hp, 5.169611e+03_hp, 5.805321e+03_hp, 6.436264e+03_hp, &
               7.062198e+03_hp, 7.883422e+03_hp, 8.909992e+03_hp, 9.936521e+03_hp, &
               1.091817e+04_hp, 1.189586e+04_hp, 1.286959e+04_hp, 1.429100e+04_hp, &
               1.562600e+04_hp, 1.696090e+04_hp, 1.816190e+04_hp, 1.930970e+04_hp, &
               2.032590e+04_hp, 2.121500e+04_hp, 2.187760e+04_hp, 2.238980e+04_hp, &
               2.243630e+04_hp, 2.168650e+04_hp, 2.011920e+04_hp, 1.769300e+04_hp, &
               1.503930e+04_hp, 1.278370e+04_hp, 1.086630e+04_hp, 9.236572e+03_hp, &
               7.851231e+03_hp, 6.660341e+03_hp, 5.638791e+03_hp, 4.764391e+03_hp, &
               4.017541e+03_hp, 3.381001e+03_hp, 2.836781e+03_hp, 2.373041e+03_hp, &
               1.979160e+03_hp, 1.645710e+03_hp, 1.364340e+03_hp, 1.127690e+03_hp, &
```

```
            9.292942e+02_hp, 7.619842e+02_hp, 6.216801e+02_hp, 5.046801e+02_hp, &
            4.076571e+02_hp, 3.276431e+02_hp, 2.620211e+02_hp, 2.084970e+02_hp, &
            1.650790e+02_hp, 1.300510e+02_hp, 1.019440e+02_hp, 7.951341e+01_hp, &
            6.167791e+01_hp, 4.758061e+01_hp, 3.650411e+01_hp, 2.785261e+01_hp, &
            2.113490e+01_hp, 1.594950e+01_hp, 1.197030e+01_hp, 8.934502e+00_hp, &
            6.600001e+00_hp, 4.758501e+00_hp, 3.270000e+00_hp, 2.000000e+00_hp, &
            1.000000e+00_hp /)

! Bp [unitless] for 72 levels (73 edges)
REAL(hp), PARAMETER          :: Bp72(73) = (/                      &
            1.000000e+00_hp, 9.849520e-01_hp, 9.634060e-01_hp, 9.418650e-01_hp, &
            9.203870e-01_hp, 8.989080e-01_hp, 8.774290e-01_hp, 8.560180e-01_hp, &
            8.346609e-01_hp, 8.133039e-01_hp, 7.919469e-01_hp, 7.706375e-01_hp, &
            7.493782e-01_hp, 7.211660e-01_hp, 6.858999e-01_hp, 6.506349e-01_hp, &
            6.158184e-01_hp, 5.810415e-01_hp, 5.463042e-01_hp, 4.945902e-01_hp, &
            4.437402e-01_hp, 3.928911e-01_hp, 3.433811e-01_hp, 2.944031e-01_hp, &
            2.467411e-01_hp, 2.003501e-01_hp, 1.562241e-01_hp, 1.136021e-01_hp, &
            6.372006e-02_hp, 2.801004e-02_hp, 6.960025e-03_hp, 8.175413e-09_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, 0.000000e+00_hp, &
            0.000000e+00_hp /)
PUBLIC TYPES:
```

REVISION HISTORY:

```
28 Sep 2015 - C. Keller   - Initialization
```

1.16.1 HCO_VertGrid_Init

Function HCO_VertGrid_Init initializes the vertical grid.

INTERFACE:

```
SUBROUTINE HCO_VertGrid_Init( am_I_Root, zGrid, RC )
```

INPUT PARAMETERS:

```
LOGICAL,        INTENT(IN   )  :: am_I_Root ! Root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(VertGrid), POINTER        :: zGrid     ! vertical grid
INTEGER,        INTENT(INOUT)  :: RC        ! Return code
```

REVISION HISTORY:

28 Sep 2015 - C. Keller - Initial version

1.16.2 HCO_VertGrid_Define

Function HCO_VertGrid_Define initializes the vertical grid.

INTERFACE:

```
SUBROUTINE HCO_VertGrid_Define( am_I_Root, HcoConfig, zGrid, nz, Ap, Bp, RC )
```

USES:

```
USE HCO_TYPES_MOD,    ONLY : ConfigObj
```

INPUT PARAMETERS:

```
LOGICAL,        INTENT(IN   )            :: am_I_Root ! Root CPU?
INTEGER,        INTENT(IN   )            :: nz        ! # of vertical levels
REAL(hp),       INTENT(IN   ), OPTIONAL  :: Ap(nz+1)  ! Ap values
REAL(hp),       INTENT(IN   ), OPTIONAL  :: Bp(nz+1)  ! Bp values
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ConfigObj),POINTER                  :: HcoConfig ! HEMCO config obj
TYPE(VertGrid), POINTER                  :: zGrid     ! vertical grid
INTEGER,        INTENT(INOUT)            :: RC        ! Return code
```

REVISION HISTORY:

28 Sep 2015 - C. Keller - Initial version

1.16.3 HCO_VertGrid_Cleanup

Function HCO_VertGrid_Cleanup cleans up the vertical grid.

INTERFACE:

```
SUBROUTINE HCO_VertGrid_Cleanup( zGrid )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(VertGrid), POINTER        :: zGrid     ! vertical grid
```

REVISION HISTORY:

28 Sep 2015 - C. Keller - Initial version

1.17 Fortran: Module Interface hco_emislist_mod.F90

Module HCO_EmisList_Mod contains routines and variables for the HEMCO emissions list EmisList. EmisList is a sorted collection of all data containers needed for emission calculation. The containers are sorted by data type, species, emission category, and emission hierarchy (in this order).

INTERFACE:

```
MODULE HCO_EMISLIST_MOD
```

USES:

```
USE HCO_ERROR_MOD
USE HCO_TYPES_MOD
USE HCO_STATE_MOD,    ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCO_GetPtr
PUBLIC  :: EmisList_Pass
 PUBLIC  :: EmisList_Update
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: EmisList_Add
PRIVATE :: Add2EmisList
```

REVISION HISTORY:

```
 04 Dec 2012 - C. Keller   - Initialization
 08 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 08 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

1.17.1 EmisList_Add

Subroutine EmisList_Add adds the passed data container Dct to EmisList. Within EmisList, Dct becomes placed with increasing data type, species ID, category and hierarchy (in this order).

INTERFACE:

```
SUBROUTINE EmisList_Add( am_I_Root, Dct, HcoState, RC )
```

USES:

```
  USE HCO_DATACONT_MOD,  ONLY : ListCont_Find
  USE HCO_LOGFILE_MOD,   ONLY : HCO_PrintDataCont
```

INPUT PARAMETERS:

```
LOGICAL,           INTENT(IN)    :: am_I_Root  ! Root CPU?
TYPE(DataCont),    POINTER       :: Dct        ! Data cont.
TYPE(HCO_State),   POINTER       :: HcoState   ! HEMCO state
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,           INTENT(INOUT) :: RC         ! Return code
```

REVISION HISTORY:

```
04 Dec 2012 - C. Keller - Initialization
02 Feb 2015 - C. Keller - Moved tIDx_Assign call to hco_readlist_mod
                          so that this module can also be used by
                          hco_clock_mod.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.17.2 Add2EmisList

Subroutine Add2EmisList adds list container Lct to EmisList. Base emission fields (Data type = 1) are sorted based on species ID, category and hierarchy (for fields of same category). Scale fields and masks are added to the end of EmisList.

INTERFACE:

```
SUBROUTINE Add2EmisList( am_I_Root, HcoState, Lct, RC )
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root
TYPE(HCO_State), POINTER       :: HcoState   ! HEMCO state
TYPE(ListCont),  POINTER       :: Lct
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
06 Dec 2012 - C. Keller - Initial version
```

1.17.3 EmisList_Pass

Subroutine EmisList_Pass passes (the ReadList) container Lct to EmisList. This routine mostly checks for additive arrays, i.e. if arrays from multiple containers have to be added together prior to emission calculation (e.g. sectoral data).

INTERFACE:

```
SUBROUTINE EmisList_Pass( am_I_Root, HcoState, Lct, RC )
```

USES:

```
USE HCO_DATACONT_MOD, ONLY : ListCont_Find
USE HCO_FILEDATA_MOD, ONLY : FileData_ArrCheck
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)    :: am_I_Root
TYPE(HCO_State),  POINTER       :: HcoState
TYPE(ListCont),   POINTER       :: Lct        ! list container
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC         ! Success
```

REVISION HISTORY:

```
28 Mar 2013 - C. Keller - Initial version
22 Dec 2014 - C. Keller - Bug fix: pass container to EmisList if
                          cID is not targetID but data cannot be
                          added to targetID because it's not the
                          home container.
23 Dec 2014 - C. Keller - Don't cleanup container anymore. This is
                          now handled in ReadList_Read.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.17.4 HCO_GetPtr_3D

Subroutine HCO_GetPtr_3D returns the 3D data pointer Ptr3D of EmisList that is associated with data container DctName. By default, the routine returns an error if the given container name is not found. This can be avoided by calling the routine with the optional argument FOUND, in which case only this argument will be set to FALSE. Similarly, the FILLED flag can be used to control the behaviour if the data container is found but empty, e.g. no data is associated with it.

This routine returns the unevaluated data field, e.g. no scale factors or masking is applied to the data. Use routine HCO_EvalFld in hco_calc_mod.F90 to get evaluated fields.

INTERFACE:

```
SUBROUTINE HCO_GetPtr_3D( am_I_Root, HcoState, DctName, Ptr3D, &
                          RC, TIDX, FOUND, FILLED )
```

USES:

```
USE HCO_DATACONT_MOD, ONLY : ListCont_Find
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )         :: am_I_Root      ! root CPU?
TYPE(HCO_State),  POINTER               :: HcoState       ! HEMCO state obj
CHARACTER(LEN=*), INTENT(IN   )         :: DctName        ! container name
INTEGER,          INTENT(IN), OPTIONAL  :: TIDX           ! time index
                                                          ! (default=1)
```

OUTPUT PARAMETERS:

```
REAL(sp),         POINTER               :: Ptr3D(:,:,:)   ! output array
LOGICAL,          INTENT(OUT), OPTIONAL :: FOUND          ! cont. found?
LOGICAL,          INTENT(OUT), OPTIONAL :: FILLED         ! array filled?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)         :: RC             ! Success/fail
```

REVISION HISTORY:

```
04 Sep 2013 - C. Keller    - Initialization
19 May 2015 - C. Keller    - Added argument FILLED
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.17.5 HCO_GetPtr_2D

Subroutine HCO_GetPtr_2D returns the 2D data pointer Ptr2D of EmisList that is associated with data container DctName. See HCO_GetPtr_3D for more details.

INTERFACE:

```
SUBROUTINE HCO_GetPtr_2D( am_I_Root, HcoState, DctName, Ptr2D, &
                          RC, TIDX, FOUND, FILLED )
```

USES:

```
USE HCO_DATACONT_MOD, ONLY : ListCont_Find
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )         :: am_I_Root   ! root CPU?
TYPE(HCO_State),  POINTER               :: HcoState    ! HEMCO state obj
CHARACTER(LEN=*), INTENT(IN   )         :: DctName     ! container name
INTEGER,          INTENT(IN), OPTIONAL  :: TIDX        ! time index
                                                       ! (default=1)
```

OUTPUT PARAMETERS:

```
REAL(sp),         POINTER               :: Ptr2D(:,:)  ! output array
LOGICAL,          INTENT(OUT), OPTIONAL :: FOUND       ! cont. found?
LOGICAL,          INTENT(OUT), OPTIONAL :: FILLED      ! array filled?
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)         :: RC          ! Success/fail
```

REVISION HISTORY:

```
    04 Sep 2013 - C. Keller    - Initialization
    19 May 2015 - C. Keller    - Added argument FILLED
    26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.18 Fortran: Module Interface hco_calc_mod.F90

Module HCO_Calc_Mod contains routines to calculate HEMCO core emissions based on the content of the HEMCO EmisList object. All emissions are in [kg/m2/s].

Emissions for the current datetime are calculated by multiplying base emissions fields with the associated scale factors. Different inventories are merged/overlayed based upon the category and hierarchy attributes assigned to the individual base fields. Within the same category, fields of higher hierarchy overwrite lower-hierarchy fields. Emissions of different categories are always added.

The assembled emission array is written into the corresponding emission rates array of the HEMCO state object: HcoStateHcoID denotes the corresponding species ID. Emis covers dimension lon, lat, lev on the HEMCO grid, i.e. unlike the emission arrays in EmisList that only cover the levels defined in the source files, Emis extends over all vertical model levels.

Negative emissions are not supported and are ignored. Surface deposition velocities are stored in HcoStatebe added therein.

In addition to emissions and surface deposition rates, HEMCO also supports concentrations (kg/m3). Data is automatically written into the concentration array HcoStateis marked as being concentration data (i.e. if DtaThe latter is automatically determined by HEMCO based upon the data units.

All emission calculation settings are passed through the HcoState options object (HcoState

- ExtNr: extension number to be considered.
- SpcMin: lower species ID (HEMCO ID) to be considered.
- SpcMax: upper species ID (HEMCO ID) to be considered. If set to -1, all species above or equal to SpcMin are considered.
- CatMin: lower emission category to be considered.
- CatMax: upper emission category to be considered. If set to -1, all categories above or equal to CatMin are considered.
- FillBuffer: if set to TRUE, the emissions will be written into buffer array HcoStateIf this option is enabled, only one species can be calculated at a time (by setting

SpcMin/SpcMax, CatMin/CatMax and/or ExtNr accordingly). This option is useful for extensions, e.g. if additional scalings need to be done on some emission fields assembled by HEMCO (e.g. PARANOX extension).

INTERFACE:

```
MODULE HCO_Calc_Mod
```

USES:

```
USE HCO_Diagn_Mod
USE HCO_Error_Mod
USE HCO_Types_Mod
USE HCO_DataCont_Mod, ONLY : Pnt2DataCont

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCO_CalcEmis
PUBLIC  :: HCO_CheckDepv
PUBLIC  :: HCO_EvalFld
PUBLIC  :: HCO_MaskFld
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: GET_CURRENT_EMISSIONS
PRIVATE :: GetMaskVal
PRIVATE :: GetDilFact
PRIVATE :: GetVertIndx
PRIVATE :: GetIdx
!PARAMETER
! Mask threshold. All mask values below this value will be evaluated
! as zero (= outside of mask), and all values including and above this
! value as inside the mask. This is only of relevance if the MaskFractions
! option is false. If MaskFractions is true, the fractional mask values are
! considered, e.g. a grid box can contribute 40% to a mask region, etc.
! The MaskFractions toggle can be set in the settings section of the HEMCO
! configuration file (Use mask fractions: true/false). It defaults to false.
REAL(sp), PARAMETER  :: MASK_THRESHOLD = 0.5_sp
=============================================================================
```

REVISION HISTORY:

```
 25 Aug 2012 - C. Keller   - Initial version.
 06 Jun 2014 - R. Yantosca - Add cosmetic changes in ProTeX headers
 08 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 29 Dec 2014 - C. Keller   - Added MASK_THRESHOLD parameter. Added option to
                             apply scale factors only over masked area.
 08 Apr 2015 - C. Keller   - Added option for fractional masks.
 11 May 2015 - C. Keller   - Added HCO_EvalFld interface.
```

1.18.1 HCO_CalcEmis

Subroutine HCO_CalcEmis calculates the 3D emission fields at current datetime for the specified species, categories, and extension number.

INTERFACE:

```
SUBROUTINE HCO_CalcEmis( am_I_Root, HcoState, UseConc, RC )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
USE HCO_ARR_MOD,      ONLY : HCO_ArrAssert
USE HCO_DATACONT_MOD, ONLY : ListCont_NextCont
USE HCO_FILEDATA_MOD, ONLY : FileData_ArrIsDefined
USE HCO_Scale_Mod,    ONLY : HCO_ScaleArr
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root  ! Root CPU?
LOGICAL,         INTENT(IN   )  :: UseConc    ! Use concentration fields?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState   ! HEMCO state object
INTEGER,         INTENT(INOUT)  :: RC         ! Return code
```

REVISION HISTORY:

```
25 Aug 2012 - C. Keller   - Initial Version
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX header
03 Aug 2014 - C. Keller   - Bug fix for adding data to diagnostics. Now
                            explicitly check for new species OR category.
21 Aug 2014 - C. Keller   - Added concentration.
14 Apr 2016 - C. Keller   - Bug fix: avoid double-counting if multiple
                            regional inventories have the same hierarchy.
19 Sep 2016 - R. Yantosca - Use .neqv. for LOGICAL comparisons
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.18.2 HCO_CheckDepv

Subroutine HCO_CheckDepv is a simple routine to check the dry deposition frequency value. This is to avoid unrealistically high deposition frequencies that may occur if grid box concentrations are very low. The deposition frequency is limited to a value that will make sure that the drydep exponent (exp(-depfreq * dt)) is still small enough to remove all species mass. The maximum limit of depfreq * dt can be defined as a HEMCO option (MaxDepExp). Its default value is 20.0.

INTERFACE:

```
SUBROUTINE HCO_CheckDepv( am_I_Root, HcoState, Depv, RC )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root  ! Root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState   ! HEMCO state object
REAL(hp),        INTENT(INOUT)  :: Depv       ! Deposition velocity
INTEGER,         INTENT(INOUT)  :: RC         ! Return code
```

REVISION HISTORY:

```
11 Mar 2015 - C. Keller   - Initial Version
```

1.18.3 Get_Current_Emissions

Subroutine Get_Current_Emissions calculates the current emissions for the specified emission container. This subroutine is only called by HCO_CalcEmis and for base emission containers, i.e. containers of type 1.

INTERFACE:

```
SUBROUTINE Get_Current_Emissions( am_I_Root, HcoState,   BaseDct,   &
                                  nI, nJ, nL, OUTARR_3D, MASK, RC, UseLL )
```

USES:

```
USE HCO_State_Mod,    ONLY : HCO_State
USE HCO_tIdx_MOD,     ONLY : tIDx_GetIndx
USE HCO_FileData_Mod, ONLY : FileData_ArrIsDefined
```

INPUT PARAMETERS:

```
LOGICAL,           INTENT(IN ) :: am_I_Root           ! Root CPU?
INTEGER,           INTENT(IN)  :: nI                  ! # of lons
INTEGER,           INTENT(IN)  :: nJ                  ! # of lats
INTEGER,           INTENT(IN)  :: nL                  ! # of levs
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState            ! HEMCO state object
TYPE(DataCont),  POINTER       :: BaseDct             ! base emission
                                                      !  container
REAL(hp),        INTENT(INOUT) :: OUTARR_3D(nI,nJ,nL) ! output array
REAL(hp),        INTENT(INOUT) :: MASK     (nI,nJ,nL) ! mask array
INTEGER,         INTENT(INOUT) :: RC
```

OUTPUT PARAMETERS:

```
INTEGER,         INTENT(  OUT), OPTIONAL :: UseLL
```

REMARKS:

```
This routine uses multiple loops over all grid boxes (base emissions
and scale factors use separate loops). In an OMP environment, this approach
seems to be faster than using only one single loop (but repeated calls to
point to containers, etc.). The alternative approach is used in routine
Get\_Current\_Emissions\_B at the end of this module and may be employed
on request.
```

REVISION HISTORY:

```
25 Aug 2012 - C. Keller   - Initial Version
09 Nov 2012 - C. Keller   - MASK update. Masks are now treated
                            separately so that multiple masks can be
                            added.
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
07 Sep 2014 - C. Keller   - Mask update. Now set mask to zero as soon as
                            on of the applied masks is zero.
03 Dec 2014 - C. Keller   - Now calculate time slice index on-the-fly.
29 Dec 2014 - C. Keller   - Added scale factor masks.
02 Mar 2015 - C. Keller   - Now check for missing values. Missing values are
                            excluded from emission calculation.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
11 May 2017 - C. Keller   - Added universal scaling
```

1.18.4 Get_Current_Emissions_b (NOT USED!!)

Subroutine Get_Current_Emissions_B calculates the current emissions for the specified emission field and passes the result to OUTARR_3D.

This subroutine is only called by HCO_CalcEmis and for fields with a valid species ID, i.e. for base emission fields.
!!! WARNING: this routine is not actively developed any more and may lag !!! behind Get_Current_Emissions

INTERFACE:

```
SUBROUTINE Get_Current_Emissions_B( am_I_Root, HcoState, BaseDct, &
                                    nI, nJ, nL, OUTARR_3D, MASK, RC )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
USE HCO_TIDX_MOD,     ONLY : tIDx_GetIndx
USE HCO_FILEDATA_MOD, ONLY : FileData_ArrIsDefined
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)    :: am_I_Root           ! Root CPU?
INTEGER,          INTENT(IN)    :: nI                  ! # of lons
INTEGER,          INTENT(IN)    :: nJ                  ! # of lats
INTEGER,          INTENT(IN)    :: nL                  ! # of levs
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState            ! HEMCO state object
TYPE(DataCont),  POINTER       :: BaseDct             ! base emission
                                                      !  container
REAL(hp),        INTENT(INOUT) :: OUTARR_3D(nI,nJ,nL) ! output array
REAL(hp),        INTENT(INOUT) :: MASK     (nI,nJ,nL) ! mask array
INTEGER,         INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
25 Aug 2012 - C. Keller   - Initial Version
09 Nov 2012 - C. Keller   - MASK update. Masks are now treated
                            separately so that multiple masks can be
                            added.
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX header
07 Sep 2014 - C. Keller   - Mask update. Now set mask to zero as soon as
                            on of the applied masks is zero.
02 Mar 2015 - C. Keller   - Now check for missing values
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.18.5 HCO_EvalFld_3D

Subroutine HCO_EvalFld_3D returns the 3D data field belonging to the emissions list data container with field name 'cName'. The returned data field is the completely evaluated field, e.g. the base field multiplied by all scale factors and with all masking being applied (as specified in the HEMCO configuration file). This distinguished this routine from HCO_GetPtr in hco_emislist_mod.F90, which returns a reference to the unevaluated data field.

INTERFACE:

```
SUBROUTINE HCO_EvalFld_3D( am_I_Root, HcoState, cName, Arr3D, RC, FOUND )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
USE HCO_DATACONT_MOD, ONLY : ListCont_Find
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root    ! Root CPU?
CHARACTER(LEN=*), INTENT(IN   )  :: cName
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState     ! HEMCO state object
REAL(hp),         INTENT(INOUT)  :: Arr3D(:,:,:) ! 3D array
INTEGER,          INTENT(INOUT)  :: RC           ! Return code
```

OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(  OUT), OPTIONAL  :: FOUND
```

REVISION HISTORY:

```
11 May 2015 - C. Keller   - Initial Version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.18.6 HCO_EvalFld_2D

Subroutine HCO_EvalFld_2D returns the 2D data field belonging to the emissions list data container with field name 'cName'. The returned data field is the completely evaluated field, e.g. the base field multiplied by all scale factors and with all masking being applied (as specified in the HEMCO configuration file). This distinguished this routine from HCO_GetPtr in hco_emislist_mod.F90, which returns a reference to the unevaluated data field.

INTERFACE:

```
SUBROUTINE HCO_EvalFld_2D( am_I_Root, HcoState, cName, Arr2D, RC, FOUND )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
USE HCO_DATACONT_MOD, ONLY : ListCont_Find
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root    ! Root CPU?
CHARACTER(LEN=*), INTENT(IN   )  :: cName
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState     ! HEMCO state object
REAL(hp),         INTENT(INOUT)  :: Arr2D(:,:)   ! 2D array
INTEGER,          INTENT(INOUT)  :: RC           ! Return code
```

OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(  OUT), OPTIONAL  :: FOUND
```

REVISION HISTORY:

```
11 May 2015 - C. Keller   - Initial Version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
01 Nov 2016 - C. Keller   - Added error trap for UseLL
```

1.18.7 GetMaskVal

Subroutine GetMaskVal is a helper routine to get the mask value at a given location.

INTERFACE:

```
SUBROUTINE GetMaskVal ( am_I_Root, Dct, I, J, MaskVal, Fractions, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root           ! Root CPU?
INTEGER,         INTENT(IN   ) :: I                   ! # of lons
INTEGER,         INTENT(IN   ) :: J                   ! # of lats
LOGICAL,         INTENT(IN   ) :: Fractions           ! Use fractions?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(DataCont),  POINTER       :: Dct                 ! Mask container
REAL(sp),        INTENT(INOUT) :: MaskVal
INTEGER,         INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
09 Apr 2015 - C. Keller   - Initial Version
```

1.18.8 HCO_MaskFld

Subroutine HCO_MaskFld is a helper routine to get the mask field with the given name. The returned mask field is fully evaluated, e.g. the data operation flag associated with this mask field is already taken into account. For instance, if the data operator of a mask field is set to 3, the returned array contains already the mirrored mask values.

INTERFACE:

```
SUBROUTINE HCO_MaskFld ( am_I_Root, HcoState, MaskName, Mask, RC, FOUND )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
USE HCO_DATACONT_MOD, ONLY : ListCont_Find
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )           :: am_I_Root    ! Root CPU?
TYPE(HCO_STATE), POINTER                 :: HcoState
CHARACTER(LEN=*),INTENT(IN   )           :: MaskName
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp),         INTENT(INOUT)           :: Mask(:,:)
INTEGER,          INTENT(INOUT)           :: RC
```

OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(  OUT), OPTIONAL :: FOUND
```

REVISION HISTORY:

```
11 Jun 2015 - C. Keller   - Initial Version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.18.9 GetVertIndx

Subroutine GetVertIndx is a helper routine to get the vertical index range of the given data field.

INTERFACE:

```
SUBROUTINE GetVertIndx ( am_I_Root, HcoState, Dct, &
                         LevDct1, LevDct2, I, J, LowLL, UppLL, RC )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,  INTENT(IN   )           :: am_I_Root   ! Root CPU?
TYPE(HCO_State), POINTER          :: HcoState    ! HEMCO state object
TYPE(DataCont),  POINTER          :: LevDct1     ! Level index 1 container
TYPE(DataCont),  POINTER          :: LevDct2     ! Level index 2 container
INTEGER,  INTENT(IN   )           :: I           ! lon index
INTEGER,  INTENT(IN   )           :: J           ! lat index
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(DataCont),  POINTER          :: Dct         ! Mask container
INTEGER,  INTENT(INOUT)           :: LowLL       ! lower level index
INTEGER,  INTENT(INOUT)           :: UppLL       ! upper level index
INTEGER,  INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
06 May 2016 - C. Keller   - Initial Version
```

1.18.10 GetEmisL

Returns the emission level read from a scale factor.

INTERFACE:

```
FUNCTION GetEmisL ( am_I_Root, HcoState, LevDct, I, J ) RESULT ( EmisL )
```

USES:

```
USE HCO_TYPES_MOD
USE HCO_STATE_MOD,    ONLY : HCO_State
USE HCO_tIdx_MOD,     ONLY : tIDx_GetIndx
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root      ! Root CPU?
TYPE(HCO_State), POINTER        :: HcoState       ! HEMCO state object
TYPE(DataCont),  POINTER        :: LevDct         ! Level index 1 container
INTEGER,         INTENT(IN   )  :: I, J           ! horizontal index
```

RETURN VALUE:

```
REAL(hp)                        :: EmisL
```

REVISION HISTORY:

```
26 Jan 2018 - C. Keller - Initial version
```

1.18.11 GetEmisLUnit

Returns the emission level unit read from a scale factor.

INTERFACE:

```
FUNCTION GetEmisLUnit ( am_I_Root, HcoState, LevDct ) RESULT( EmisLUnit )
```

USES:

```
USE HCO_TYPES_MOD
USE HCO_STATE_MOD,    ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root      ! Root CPU?
TYPE(HCO_State), POINTER        :: HcoState       ! HEMCO state object
TYPE(DataCont),  POINTER        :: LevDct         ! Level index 1 container
```

RETURN VALUE:

```
INTEGER                         :: EmisLUnit
```

REVISION HISTORY:

```
26 Jan 2018 - C. Keller - Initial version
```

1.18.12 GetIdx

Subroutine GetIdx is a helper routine to return the vertical level index for a given altitude. The altitude can be provided in level coordinates, in units of meters or as the 'PBL mixing height'.

INTERFACE:

```
SUBROUTINE GetIdx( am_I_Root, HcoState, I, J, alt, altu, lidx, RC )
```

USES:

```
USE HCO_TYPES_MOD
USE HCO_STATE_MOD,   ONLY : HCO_STATE
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root      ! Root CPU?
TYPE(HCO_State), POINTER        :: HcoState       ! HEMCO state object
INTEGER,         INTENT(IN   )  :: I, J           ! horizontal index
INTEGER,         INTENT(IN   )  :: altu           ! altitude unit
```

OUTPUT PARAMETERS:

```
INTEGER,         INTENT(  OUT)  :: lidx           ! level index
```

INPUT/OUTPUT PARAMETERS:

```
REAL(hp),        INTENT(INOUT)  :: alt            ! altitude
INTEGER,         INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
09 May 2016 - C. Keller - Initial version
```

1.18.13 GetDilFact

Subroutine GetDilFact returns the vertical dilution factor, that is the factor that is to be applied to distribute emissions into multiple vertical levels. If grid box height information are available, these are used to compute the distribution factor. Otherwise, equal weight is given to all vertical levels.

!TODO: Dilution factors are currently only weighted by grid box heights (if these information are available) but any pressure information are ignored.

INTERFACE:

```
SUBROUTINE GetDilFact ( am_I_Root, HcoState, EmisL1, EmisL1Unit, &
                        EmisL2, EmisL2Unit, I, J, L, LowLL, UppLL, &
                        DilFact, RC )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,  INTENT(IN   )           :: am_I_Root   ! Root CPU?
TYPE(HCO_State), POINTER          :: HcoState    ! HEMCO state object
INTEGER,  INTENT(IN   )           :: I           ! lon index
INTEGER,  INTENT(IN   )           :: J           ! lat index
INTEGER,  INTENT(IN   )           :: L           ! lev index
INTEGER,  INTENT(IN   )           :: LowLL       ! lower level index
INTEGER,  INTENT(IN   )           :: UppLL       ! upper level index
```

OUTPUT PARAMETERS:

```
REAL(hp), INTENT(  OUT)           :: DilFact     ! Dilution factor
```

INPUT/OUTPUT PARAMETERS:

```
REAL(hp), INTENT(INOUT)           :: EmisL1
INTEGER,  INTENT(INOUT)           :: EmisL1Unit
REAL(hp), INTENT(INOUT)           :: EmisL2
INTEGER,  INTENT(INOUT)           :: EmisL2Unit
INTEGER,  INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
06 May 2016 - C. Keller   - Initial Version
```

1.19 Fortran: Module Interface interpreter

Module interpreter is a third-party module that parses and evaluates mathematical functions, e.g. 2*sin(MM). downloaded on May 12, 2017 from http://zeus.df.ufcg.edu.br/labfit/functionparser.htm

REVISION HISTORY:

```
12 May 2017 - C. Keller   - Modified for use in HEMCO: use hp instead of realkind.
16 May 2017 - R. Yantosca - Do not use SIND, COSD, TAND functions, because
                            these are non-standard (Gfortran chokes)
16 May 2017 - R. Yantosca - Replaced TABs with spaces, cosmetic changes
```

1.20 Fortran: Module Interface hco_datacont_mod.F90

Module HCO_DATACONT_MOD contains routines and variables to handle the HEMCO data-container (DataCont) and correspoding list-container (ListCont) derived type.

DataCont holds all information of an emission field, such as emission category, emission

hierarchy, scale factors, etc. DataCont also contains a pointer to the source data (see HCO_FILEDATA_MOD) for more information on the file data object. A data-container will be created for every emission field specified in the HEMCO configuration file.

The ListCont object is a derived type used to create linked lists. It contains a pointer to one data container (Dta) and a pointer to the next element of the list (NextCont). All HEMCO lists (ConfigList, ReadList, ListCont) are built from ListCont elements.

DataCont consists of the following elements:

- cName: container name, as set in the configuration file.
- cID: container ID, defined by HEMCO.
- targetID: target ID of this container. If target ID differs from the container ID, the data will be added to the content of the container with cID = targetID (e.g. data of container 1 will be added to container 5 if it has a target ID of 5). Internal use only.
- DctType: container type. 1 for base emissions, 2 for scale factors, 3 for masks (set parameter in HCO_ERROR_MOD)
- SpcName: Species name associated with this data container, as read from the configuration file. Only relevant for base emission arrays.
- HcoID: HEMCO species ID corresponding to SpcName.
- ExtNr: Extension number. Extension number 0 is reserved for HEMCO core, other extensions can have freely defined extensions number, as specified in the configuration file. Only relevant for base emissions.
- Cat: emission category, as set in the configuration file. Only relevant for base emissions.
- Hier: emission hierarchy, as set in the configuration file. Only relevant for base emissions.
- ScalID: scale factor ID, as set in the configuration file. Only relevant for scale factors and masks.
- Oper: mathematical operator applied to scale factor. If 1, the field will be multiplied (E=BxS); if -1, division is applied (E=B/S); if 2, field will be squared (E=BxSxS). For masks, operator 3 can be used to mirror the mask data, i.e. E=Bx(1-S). Only relevant for scale factors and masks.
- Scal_cID: vector of scale factor IDs associated to a base emission field, as specified in the configuration file. Only relevant for base emissions.
- Scal_cID_set: the Scal_cID values read from the configuration file are translated to the corresponding container IDs values (the scale IDs are defined in the configuration file, container IDs are automatically set by HEMCO) to optimize container assignment operations. Scal_cID_set indicates whether or not the Scal_cID holds the container IDs or still the original scale factor IDs. For internal use only.

- Dta: a file data object, holding information about the source file, update frequency, the data arrays, etc. See HCO_FILEDATA_MOD for more information.
- DtaHome: a data container only holds a pointer to a file data object, i.e. it is possible that multiple containers share the same file data object. the DtaHome flag is used to determine whether this is the home container of this file data object. For internal use only.

INTERFACE:

```
MODULE HCO_DataCont_Mod
```

USES:

```
USE HCO_TYPES_MOD
USE HCO_Error_Mod
USE HCO_Arr_Mod

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: DataCont_Init
PUBLIC  :: DataCont_Cleanup
PUBLIC  :: cIDList_Create
PUBLIC  :: cIDList_Cleanup
PUBLIC  :: Pnt2DataCont
PUBLIC  :: ListCont_NextCont
PUBLIC  :: ListCont_Find
PUBLIC  :: ListCont_Length
PUBLIC  :: ListCont_Cleanup
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

1.20.1 DataCont_Init

Subroutine DataCont_Init initializes a new (blank) data container Dct.

INTERFACE:

```
SUBROUTINE DataCont_Init( Dct, cID )
```

INPUT PARAMETERS:

```
TYPE(DataCont),  POINTER       :: Dct
INTEGER,         INTENT(IN)    :: cID
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
```

1.20.2 DataCont_Cleanup

Subroutine DataCont_Cleanup cleans up data container Dct. If ArrOnly is set to True, this will only cleanup the data array of the container but keep all meta-data.

INTERFACE:

```
SUBROUTINE DataCont_Cleanup( Dct, ArrOnly )
```

USES:

```
USE HCO_FILEDATA_MOD, ONLY : FileData_Cleanup
```

ARGUMENTS:

```
TYPE(DataCont), POINTER               :: Dct
LOGICAL,        INTENT(IN), OPTIONAL  :: ArrOnly
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
```

1.20.3 ListCont_Cleanup

Subroutine ListCont_Cleanup cleans up list List The corresponding data container (LstContRemoveDct is set to true.

INTERFACE:

```
SUBROUTINE ListCont_Cleanup( List, RemoveDct )
```

INPUT PARAMETERS:

```
TYPE(ListCont), POINTER      :: List
LOGICAL,        INTENT(IN)   :: RemoveDct
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller: Initialization
25 Oct 2016 - R. Yantosca - Do not nullify pointers in declaration stmts
```

1.20.4 cIDList_Create

Subroutine cIDList_Create creates a vector of pointers (cIDList) pointing to all available containers of the passed List. The vector index of cIDList corresponds to the container cIDs, i.e. cIDList(3) will point to data container with cID = 3.

INTERFACE:

```
SUBROUTINE cIDList_Create( am_I_Root, HcoState, List, RC )
```

USES:

```
USE HCO_STATE_MOD, ONLY : HCO_State
```

ARGUMENTS:

```
LOGICAL,         INTENT(IN)    :: am_I_Root
TYPE(HCO_State), POINTER       :: HcoState
TYPE(ListCont),  POINTER       :: List
INTEGER,         INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
24 Aug 2012 - C. Keller - Initial Version
25 Oct 2016 - R. Yantosca - Do not nullify pointers in declaration stmts
```

1.20.5 cIDList_Cleanup

Subroutine cIDList_Cleanup cleans up cIDList.

INTERFACE:

```
SUBROUTINE cIDList_Cleanup( HcoState )
```

USES:

```
USE HCO_STATE_MOD, ONLY : HCO_State
```

ARGUMENTS:

```
TYPE(HCO_State), POINTER       :: HcoState
```

REVISION HISTORY:

```
24 Aug 2012 - C. Keller - Initial Version
```

1.20.6 Pnt2DataCont

Subroutine Pnt2DataCont returns the data container Dct with container ID cID.

INTERFACE:

```
SUBROUTINE Pnt2DataCont( am_I_Root, HcoState, cID, Dct, RC )
```

USES:

```
USE HCO_STATE_MOD, ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN)    :: am_I_Root
TYPE(HCO_State), POINTER       :: HcoState
INTEGER,         INTENT(IN)    :: cID
TYPE(DataCont), POINTER        :: Dct
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
11 Apr 2012 - C. Keller - Initial version
```

1.20.7 ListCont_NextCont

Routine ListCont_NextCont returns container Lct from data list List. This is the generic routine for cycling through the data container lists.

If Lct is empty (i.e. NULL), the first container of List is returned. If Lct already points to a list container, the pointer is advanced to the next container in that list (LctFLAG is set to HCO_SUCCESS if the return container Lct is defined, and to HCO_FAIL otherwise.

INTERFACE:

```
SUBROUTINE ListCont_NextCont( List, Lct, FLAG )
```

INPUT PARAMETERS:

```
TYPE(ListCont), POINTER       :: List
TYPE(ListCont), POINTER       :: Lct
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: FLAG
```

REVISION HISTORY:

```
11 Apr 2012 - C. Keller - Initial version
```

1.20.8 ListCont_Find_Name

Subroutine ListCont_Find_Name searches for (data) container name NME in list List and returns a pointer pointing to this container (Lct).

INTERFACE:

```
SUBROUTINE ListCont_Find_Name( List, NME, FOUND, Lct )
```

ARGUMENTS:

```
TYPE(ListCont),   POINTER              :: List  ! List to be searched
CHARACTER(LEN=*), INTENT(IN )          :: NME   ! Container name
LOGICAL,          INTENT(OUT)          :: FOUND ! Container found?
TYPE(ListCont),   POINTER, OPTIONAL    :: Lct   ! matched list container
```

REVISION HISTORY:

```
04 Dec 2012 - C. Keller: Initialization
25 Oct 2016 - R. Yantosca - Do not nullify pointers in declaration stmts
```

1.20.9 ListCont_Find_ID

Subroutine ListCont_Find_ID searches for (data) container cID or ScalID (ID) in list List and returns a pointer pointing to this (list) container (Lct).

INTERFACE:

```
SUBROUTINE ListCont_Find_ID( List, ID, IsScalID, FOUND, Lct )
```

INPUT PARAMETERS:

```
TYPE(ListCont),   POINTER           :: List     ! List to be searched
INTEGER,          INTENT(IN )       :: ID       ! cID or ScalID
INTEGER,          INTENT(IN )       :: IsScalID ! 1=ID is ScalID;
                                                !  else: ID is cID
```

OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(OUT)       :: FOUND    ! Container found?
TYPE(ListCont),   POINTER, OPTIONAL :: Lct      ! Container w/ ID
```

REVISION HISTORY:

```
04 Dec 2012 - C. Keller: Initialization
25 Oct 2016 - R. Yantosca - Do not nullify pointers in declaration stmts
```

1.20.10 ListCont_Length

Subroutine ListCont_Length returns the length of the passed list.

INTERFACE:

```
FUNCTION ListCont_Length ( List ) RESULT ( nnCont )
```

INPUT PARAMETERS:

```
TYPE(ListCont),  POINTER       :: List
INTEGER                        :: nnCont
```

REVISION HISTORY:

```
15 Feb 2016 - C. Keller: Initial version
```

1.21 Fortran: Module Interface hco_driver_mod.F90

Module HCO_Driver_Mod contains the driver routines (INIT, RUN, FINAL) for the HEMCO core module. It calls all the subroutines to initialize, execute and finalize the HEMCO core emissions calculations, i.e. all emissions not calculated in a HEMCO extension (See module hcox_driver_mod.F90 for the extensions).

Call this module at the HEMCO - model interface level to execute the HEMCO core operations.

INTERFACE:

```
MODULE HCO_Driver_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_State_Mod, ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCO_Init
PUBLIC :: HCO_Run
PUBLIC :: HCO_Final
```

REVISION HISTORY:

```
 27 May 2012 - C. Keller   - Initialization
 11 Jun 2014 - R. Yantosca - Now indended with F90 free-format
 11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

1.21.1 HCO_Run

Subroutine HCO_Run is the HEMCO core run routine. It calculates the HEMCO emissions as specified in the HEMCO configuration file. All calculation settings, such as the extension number, the lowest and highest species ID, and the lowest and highest emission category, are passed through the HEMCO options object (HcoStateThe time stamp is taken from the HEMCO clock object. Subroutine HcoClock_Set should be used to update the HEMCO clock (see module hco_clock_mod.F90). This should be done at the HEMCO - model interface level.

INTERFACE:

```
SUBROUTINE HCO_Run( am_I_Root, HcoState, Phase, RC )
```

USES:

```
USE HCO_Calc_Mod,     ONLY : HCO_CalcEmis
USE HCO_ReadList_Mod, ONLY : ReadList_Read
USE HCO_Clock_Mod,    ONLY : HcoClock_Get
USE HCO_Clock_Mod,    ONLY : HcoClock_First
USE HCO_Clock_Mod,    ONLY : HcoClock_InitTzPtr
USE HCOIO_DIAGN_MOD,  ONLY : HcoDiagn_Write
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root   ! root CPU?
INTEGER,         INTENT(IN   ) :: Phase       ! Run phase (1 or 2)
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState    ! HEMCO state object
INTEGER,         INTENT(INOUT) :: RC          ! Success or failure?
```

REVISION HISTORY:

```
27 May 2012 - C. Keller   - Initialization
16 Jul 2014 - R. Yantosca - Cosmetic changes
23 Dec 2014 - C. Keller   - ReadList_to_EmisList is now obsolete.
                            Containers are added to EmisList within
                            routine ReadList_Read.
23 Feb 2015 - R. Yantosca - Now call HcoClock_InitTzPtr on the first
                            emissions timestep to initialize the pointer
                            to the timezones data (i.e. hours from UTC)
01 Apr 2015 - C. Keller   - Added run phases
```

1.21.2 HCO_Init

Subroutine HCO_INIT initializes the HEMCO core modules. This routine assumes that the HEMCO configuration file has already been read to buffer (subroutine Config_ReadFile in HCO_CONFIG_MOD.F90) and that the HEMCO state object has already been initialized. This has to be done at the HEMCO - model interface level.

INTERFACE:

```
SUBROUTINE HCO_Init( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_Diagn_Mod,    ONLY : HcoDiagn_Init
USE HCO_tIdx_Mod,     ONLY : tIDx_Init
USE HCO_Clock_Mod,    ONLY : HcoClock_Init
USE HCO_Config_Mod,   ONLY : SetReadList
USE HCO_Scale_Mod,    ONLY : HCO_ScaleInit
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root  ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER       :: HcoState   ! HcoState object
INTEGER,          INTENT(INOUT) :: RC         ! Failure or success
```

REVISION HISTORY:

```
27 May 2012 - C. Keller   - Initialization
11 Jun 2014 - R. Yantosca - Now indended with F90 free-format
11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
16 Jul 2014 - R. Yantosca - Remove reference to gigc_errcode_mdo.F90
```

1.21.3 HCO_Final

Subroutine HCO_Final finalizes HEMCO core.

INTERFACE:

```
SUBROUTINE HCO_Final( am_I_Root, HcoState, ERROR, RC )
```

USES:

```
USE HCO_Clock_Mod,     ONLY : HcoClock_Cleanup
USE HCO_tIdx_Mod,      ONLY : tIDx_Cleanup
USE HCO_DataCont_Mod,  ONLY : cIDList_Cleanup
USE HCO_ReadList_Mod,  ONLY : ReadList_Cleanup
USE HCO_DataCont_Mod,  ONLY : ListCont_Cleanup
USE HCO_ExtList_Mod,   ONLY : ExtFinal
USE HCOIO_DIAGN_MOD,   ONLY : HcoDiagn_Write
USE HCO_Scale_Mod,     ONLY : HCO_ScaleFinal
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root  ! root CPU?
LOGICAL,          INTENT(IN   ) :: ERROR      ! Cleanup because of crash?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER       :: HcoState   ! HcoState object
INTEGER,          INTENT(INOUT) :: RC         ! Failure or success
```

REMARKS:

```
(1) ConfigFile_Cleanup also cleans up the data containers, while routine
    EmisList_Cleanup and ReadList_Cleanup only removes the pointers to
    them. Hence, we have to call these routines before ConfigFile_Cleanup!
(2) HcoState is cleaned up in the HEMCO-module interface.
```

REVISION HISTORY:

```
27 May 2012 - C. Keller   - Initialization
11 Jun 2014 - R. Yantosca - Now indended with F90 free-format
11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
16 Jul 2014 - R. Yantosca - Cosmetic changes
```

1.22 Fortran: Module Interface hco_types_mod.F90

Module HCO_Types_Mod contains HEMCO derived type definitions and global parameters. The here derived types become all bundled into the HEMCO state object, which is defined in module hco_state_mod.F90. Specific routines for the various derived types defined below can be found in the respective HEMCO modules (e.g. hco_datacont_mod.F90 for handling of HEMCO data containers and data container lists).

Prior to HEMCO v2.0, the derived type definitions were placed within the corresponding modules. Collecting them here breaks unnecessary dependencies amongst the various modules and consequently gives greater flexibility in using the modules independently.

INTERFACE:

```
MODULE HCO_TYPES_MOD
  USES:
  USE HCO_Error_Mod
  USE HCO_Arr_Mod

  IMPLICIT NONE
  PUBLIC
  !PUBLIC PARAMETERS:
  ! Maximum length of option name
  INTEGER, PARAMETER       :: OPTLEN = 1023

  ! Cycle flags. Used to determine the temporal behavior of data
  ! fields. For example, data set to 'cycle' will be recycled if
  ! the simulation date is outside of the datetime range of the
  ! source data. See data reading routine (hcoio_dataread_mod.F90)
  ! for more details.
  INTEGER, PARAMETER, PUBLIC  :: HCO_CFLAG_CYCLE    = 1
  INTEGER, PARAMETER, PUBLIC  :: HCO_CFLAG_RANGE    = 2
  INTEGER, PARAMETER, PUBLIC  :: HCO_CFLAG_EXACT    = 3
  INTEGER, PARAMETER, PUBLIC  :: HCO_CFLAG_INTER    = 4
  INTEGER, PARAMETER, PUBLIC  :: HCO_CFLAG_AVERG    = 5
  INTEGER, PARAMETER, PUBLIC  :: HCO_CFLAG_RANGEAVG = 6

  ! Data container update flags. At the moment, those only indicate
  ! if a container gets updated every time step or based upon the
  ! details specifications ('srcTime') in the HEMCO configuration file.
  INTEGER, PARAMETER, PUBLIC  :: HCO_UFLAG_FROMFILE = 1
  INTEGER, PARAMETER, PUBLIC  :: HCO_UFLAG_ALWAYS   = 2

  ! Data container types. These are used to distinguish between
  ! base emissions, scale factors and masks.
  INTEGER, PARAMETER, PUBLIC  :: HCO_DCTTYPE_BASE = 1
  INTEGER, PARAMETER, PUBLIC  :: HCO_DCTTYPE_SCAL = 2
  INTEGER, PARAMETER, PUBLIC  :: HCO_DCTTYPE_MASK = 3

  ! PBL level flag
  INTEGER, PARAMETER, PUBLIC  :: HCO_EMISL_PBL = 0
  INTEGER, PARAMETER, PUBLIC  :: HCO_EMISL_LEV = 1
  INTEGER, PARAMETER, PUBLIC  :: HCO_EMISL_M   = 2
```

PUBLIC TYPES:

```
  !=========================================================================
```

```
! HcoSpc: Derived type for HEMCO species
! Notes:
! **1 The emission molecular weight is the molecular weight of the
!     emitted compound. This value is only different to MW_g if the
!     emitted compound does not correspond to the transported species,
!     e.g. if emissions are in kg C4H10 but the corresponding species
!     is transported as mass Carbon.
! **2 MolecRatio is the ratio between # of species molecules per emitted
!       molecule, e.g. 4 if emissions are kg C4H10 but model species
!       are kg C.
!=========================================================================
TYPE :: HcoSpc
   INTEGER                 :: HcoID      ! HEMCO species ID
   INTEGER                 :: ModID      ! Model species ID
   CHARACTER(LEN= 31)      :: SpcName    ! species names
   REAL(hp)                :: MW_g       ! species molecular wt.     (g/mol)
   REAL(hp)                :: EmMW_g     ! emission molecular wt.**1 (g/mol)
   REAL(hp)                :: MolecRatio ! molecule emission ratio**2 (-)
   REAL(hp)                :: HenryK0    ! liq. over gas Henry const [M/atm]
   REAL(hp)                :: HenryCR    ! K0 temp. dependency [K]
   REAL(hp)                :: HenryPKA   ! pKa for Henry const. correction
   TYPE(Arr2D_HP), POINTER :: Depv       ! Deposition velocity [1/s]
   TYPE(Arr3D_HP), POINTER :: Emis       ! Emission flux [kg/m2/s]
   TYPE(Arr3D_HP), POINTER :: Conc       ! Concentration [v/v]
END TYPE HcoSpc

!=========================================================================
! HcoOpt: Derived type for HEMCO run options
!=========================================================================
TYPE :: HcoOpt
   INTEGER  :: ExtNr          ! ExtNr to be used
   INTEGER  :: SpcMin         ! Smallest HEMCO species ID to be considered
   INTEGER  :: SpcMax         ! Highest HEMCO species ID to be considered
   INTEGER  :: CatMin         ! Smallest category to be considered
   INTEGER  :: CatMax         ! Highest category to be considered
   LOGICAL  :: HcoWritesDiagn ! If set to .TRUE., HEMCO will schedule the
                              ! output of the default HEMCO diagnostics
                              ! (in hco_driver_mod.F90).
   LOGICAL  :: AutoFillDiagn  ! Write into AutoFill diagnostics?
   LOGICAL  :: FillBuffer     ! Write calculated emissions into buffer
                              ! instead of emission array?
   INTEGER  :: NegFlag        ! Negative value flag (from configfile):
                              ! 2 = allow negative values
                              ! 1 = set neg. values to zero and prompt warning
                              ! 0 = return w/ error if neg. value
   LOGICAL  :: PBL_DRYDEP     ! If true, dry deposition frequencies will
                              ! be calculated over the full PBL. If false,
                              ! they are calculated over the first layer only.
```

```
   REAL(hp) :: MaxDepExp      ! Maximum value of deposition freq. x time step.
   LOGICAL  :: MaskFractions  ! If TRUE, masks are treated as binary, e.g.
                              ! grid boxes are 100% inside or outside of a
                              ! mask.
   LOGICAL  :: Field2Diagn    ! When reading fields from disk, check if there
                              ! is a diagnostics with the same name and write
                              ! field to that diagnostics? Defaults to yes in
                              ! standalone mode and no in other setups.
   LOGICAL  :: VertWeight     ! if spreading 2D fields across multiple vert.
                              ! levels, weight vertical dilution factors based
                              ! upon level depths?
   LOGICAL  :: ScaleEmis      ! Scale emissions by uniform scale factors set
                              ! in HEMCO configuration file? Defaults to yes.
END TYPE HcoOpt

!=============================================================================
! VertGrid: Description of vertical grid
!=============================================================================
TYPE :: VertGrid
   INTEGER                 :: ZTYPE           ! Grid type
   REAL(hp),       POINTER :: Ap(:) => NULL() ! Hybrid sigma Ap values
   REAL(hp),       POINTER :: Bp(:) => NULL() ! Hybrid sigma Bp values
END TYPE VertGrid

!=============================================================================
! HcoGrid: Derived type for HEMCO grid. The grid edges are used for data
! interpolation. The pressure midpoints are not needed by HEMCO core but
! can be specified for the extensions through ExtList.
! NOTES:
! *  Not used in ESMF environment
! ** Only used by some extensions
!=============================================================================
TYPE :: HcoGrid
   TYPE(Arr2D_Hp), POINTER :: XMID       ! mid-points in x-direction (lon)
   TYPE(Arr2D_Hp), POINTER :: YMID       ! mid-points in y-direction (lat)
   TYPE(Arr2D_Hp), POINTER :: XEDGE      ! grid edges in x-direction (lon)*
   TYPE(Arr2D_Hp), POINTER :: YEDGE      ! grid edges in y-direction (lat)*
   TYPE(Arr3D_Hp), POINTER :: PEDGE      ! pressure edges (Pa)
   TYPE(Arr2D_Hp), POINTER :: YSIN       ! sin of y-direction grid edges*
   TYPE(Arr2D_Hp), POINTER :: AREA_M2    ! grid box areas (m2)
   TYPE(Arr2D_Hp), POINTER :: ZSFC       ! surface geopotential height (m)**
   TYPE(Arr2D_Hp), POINTER :: PSFC       ! surface pressure (Pa)
   TYPE(Arr2D_Hp), POINTER :: PBLHEIGHT  ! PBL height in m
   TYPE(Arr3D_Hp), POINTER :: BXHEIGHT_M ! grid box heights (m)**
   TYPE(VertGrid), POINTER :: ZGRID      ! vertical grid description
END TYPE HcoGrid

!=============================================================================
```

```
! HcoClock: Derived type definition for the HEMCO clock object
!=============================================================================
TYPE :: HcoClock

   ! Current time stamp (UTC)
   INTEGER            :: ThisYear        ! year
   INTEGER            :: ThisMonth       ! month
   INTEGER            :: ThisDay         ! day
   INTEGER            :: ThisHour        ! hour
   INTEGER            :: ThisMin         ! minute
   INTEGER            :: ThisSec         ! second
   INTEGER            :: ThisDOY         ! day of year
   INTEGER            :: ThisWD          ! weekday (0=Sun,...,6=Sat)
   INTEGER            :: MonthLastDay    ! Last day of month: 28,29,30,31

   ! Current local times
   INTEGER            :: ntz             ! number of time zones
   INTEGER,  POINTER  :: ThisLocYear(:)  ! local year
   INTEGER,  POINTER  :: ThisLocMonth(:) ! local month
   INTEGER,  POINTER  :: ThisLocDay(:)   ! local day
   INTEGER,  POINTER  :: ThisLocWD(:)    ! local weekday
   REAL(sp), POINTER  :: ThisLocHour(:)  ! local hour

   ! Previous time stamp (UTC)
   INTEGER            :: PrevYear
   INTEGER            :: PrevMonth
   INTEGER            :: PrevDay
   INTEGER            :: PrevHour
   INTEGER            :: PrevMin
   INTEGER            :: PrevSec
   INTEGER            :: PrevDOY
   INTEGER            :: PrevWD

   ! Current emission time stamp
   INTEGER            :: ThisEYear
   INTEGER            :: ThisEMonth
   INTEGER            :: ThisEDay
   INTEGER            :: ThisEHour
   INTEGER            :: ThisEMin
   INTEGER            :: ThisESec

   ! Previous emission time stamp
   INTEGER            :: PrevEYear
   INTEGER            :: PrevEMonth
   INTEGER            :: PrevEDay
   INTEGER            :: PrevEHour
   INTEGER            :: PrevEMin
   INTEGER            :: PrevESec
```

```
    ! Simulation year, month, day, hour, minute, second. Will only
    ! be different from current time stamp in special cases, e.g.
    ! if emission year shall be fixed.
    INTEGER            :: SimYear        ! year
    INTEGER            :: SimMonth       ! month
    INTEGER            :: SimDay         ! day
    INTEGER            :: SimHour        ! hour
    INTEGER            :: SimMin         ! minute
    INTEGER            :: SimSec         ! second

    ! total number of elapsed time steps and emission time steps
    ! LastEStep denotes the last nEmisSteps values for which
    ! emissions have been calculated.
    INTEGER            :: nSteps
    INTEGER            :: nEmisSteps
    INTEGER            :: LastEStep

    ! Pointer to gridded time zones. Will only hold data if a field 'TIMEZONES'
    ! is provided in the HEMCO configuration file.
    ! NOTE: This pointer is initialized by a call to HcoClock_InitTzPtr
    ! from the HEMCO run routine (HCO_Run, in hco_driver_mod.F90).
    ! This is necessary to avoid segmentation faults when running with
    ! OpenMP turned on. (bmy, 2/23/15)
    TYPE(Arr2D_Sp), POINTER :: TIMEZONES

    ! Fixed dates to be used for simulation dates
    INTEGER              :: FixYY    = -1
    INTEGER              :: FixMM    = -1
    INTEGER              :: Fixdd    = -1
    INTEGER              :: Fixhh    = -1

    ! Last time step?
    LOGICAL              :: isLast

 END TYPE HcoClock

 !=============================================================================
 ! HcoPhys: Derived type for HEMCO physical constants
 !=============================================================================
 TYPE :: HcoPhys
    REAL(dp) :: Avgdr   ! Avogadro number (mol-1)
    REAL(dp) :: PI      ! Pi
    REAL(dp) :: PI_180  ! Pi / 180
    REAL(dp) :: Re      ! Earth radius [m]
    REAL(dp) :: AIRMW   ! Molecular weight of air (g/mol)
    REAL(dp) :: g0      ! Gravity at surface of earth (m/s2)
    REAL(dp) :: Rd      ! Gas Constant (R) in dry air (J/K/kg)
```

```
   REAL(dp) :: Rdg0    ! Rd/g0
   REAL(dp) :: RSTARG  ! Universal gas constant [J/K/mol]
END TYPE HcoPhys

!=============================================================================
! HcoMicroPhys: Derived type for aerosol microphysics settings
!=============================================================================
TYPE :: HcoMicroPhys
   INTEGER           :: nBins              ! # of size-resolved bins
   INTEGER           :: nActiveModebins    ! # of active mode bins
   REAL(dp), POINTER :: BinBound(:)        ! Size bin boundaries
END TYPE HcoMicroPhys

! RdList contains lists to all ReadLists
TYPE :: RdList
   TYPE(ListCont), POINTER :: Once
   TYPE(ListCont), POINTER :: Always
   TYPE(ListCont), POINTER :: Year
   TYPE(ListCont), POINTER :: Month
   TYPE(ListCont), POINTER :: Day
   TYPE(ListCont), POINTER :: Hour
   INTEGER                 :: FileLun       = -1  ! LUN of file in archive
   CHARACTER(LEN=2023)     :: FileInArchive = ''  ! name of file in archive
END TYPE RdList

!-----------------------------------------------------------------------------
! DataCont: Derived type definition for HEMCO data container
!-----------------------------------------------------------------------------
TYPE :: DataCont

   ! Container information
   CHARACTER(LEN= 63)          :: cName          ! Cont. name
   INTEGER                     :: cID            ! Cont. ID
   INTEGER                     :: targetID       ! target ID
   INTEGER                     :: DctType        ! Data type
   TYPE(FileData),     POINTER :: Dta            ! data information
   INTEGER                     :: DtaHome        ! Home cont for Dta?
   CHARACTER(LEN= 31)          :: SpcName        ! Species Name
   INTEGER                     :: HcoID          ! HEMCO species ID
   INTEGER                     :: ExtNr          ! Extension #
   INTEGER                     :: Cat            ! Category
   INTEGER                     :: Hier           ! Hierarchy
   INTEGER                     :: ScalID         ! Scale factor ID
   INTEGER                     :: Oper           ! Operator
   INTEGER                     :: levScalID1     ! ID of vertical level field
   INTEGER                     :: levScalID2     ! ID of vertical level field
   INTEGER                     :: nScalID        ! # of scale factor IDs
   INTEGER,            POINTER :: Scal_cID(:)    ! assoc. scalefactor IDs
```

```
   LOGICAL                     :: Scal_cID_set   ! cIDs or scalIDs
END TYPE DataCont

!------------------------------------------------------------------------
! ListCont: Derived type definition for HEMCO list object
!------------------------------------------------------------------------
TYPE :: ListCont
   TYPE(DataCont),     POINTER :: Dct
   TYPE(ListCont),     POINTER :: NextCont
END TYPE ListCont

!------------------------------------------------------------------------
! FileData: Derived type definition for HEMCO filetype object
!------------------------------------------------------------------------
TYPE :: FileData
   CHARACTER(LEN=255)          :: ncFile    ! file path+name
   CHARACTER(LEN= 50)          :: ncPara    ! file parameter
   INTEGER                     :: ncYrs(2)  ! year range
   INTEGER                     :: ncMts(2)  ! month range
   INTEGER                     :: ncDys(2)  ! day range
   INTEGER                     :: ncHrs(2)  ! hour range
   INTEGER                     :: tShift(2) ! time stamp shift in months & seconds
   INTEGER                     :: CycleFlag ! cycle flag
   LOGICAL                     :: MustFind  ! file must be found
   INTEGER                     :: UpdtFlag  ! update flag
   LOGICAL                     :: ncRead    ! read from source?
   TYPE(Arr3D_SP),     POINTER :: V3(:)     ! vector of 3D fields
   TYPE(Arr2D_SP),     POINTER :: V2(:)     ! vector of 2D fields
   TYPE(TimeIdx),      POINTER :: tIDx      ! for time slice indexing
   CHARACTER(LEN= 31)          :: OrigUnit  ! original data units
   CHARACTER(LEN= 63)          :: ArbDimName! name of additional dimension
   CHARACTER(LEN= 63)          :: ArbDimVal ! desired value of additional dimension
   INTEGER                     :: Cover     ! data coverage
   INTEGER                     :: SpaceDim  ! space dimension: 1, 2 or 3
   INTEGER                     :: Levels    ! vertical level handling
    INTEGER                     :: Lev2D     ! level to use for 2D data
   REAL(hp)                    :: EmisL1     ! emission level 1
   REAL(hp)                    :: EmisL2     ! emission level 2
   INTEGER                     :: EmisL1Unit ! emission level 1 unit
   INTEGER                     :: EmisL2Unit ! emission level 2 unit
   INTEGER                     :: nt        ! time dimension: length of Arr
   INTEGER                     :: DeltaT    ! temp. resolution of array [h]
   LOGICAL                     :: IsLocTime ! local time?
   LOGICAL                     :: IsConc    ! concentration data?
   LOGICAL                     :: DoShare   ! shared object?
   LOGICAL                     :: IsInList  ! is in emissions list?
   LOGICAL                     :: IsTouched ! Has container been touched yet?
END TYPE FileData
```

```
!-----------------------------------------------------------------------
! TimeIdx: Derived type definition for the object that points to the
! current time slices of data within a file.  Used by hco_tidx_mod.F90.
!-----------------------------------------------------------------------
TYPE :: TimeIdx
   INTEGER                     :: TypeID
   CHARACTER(LEN=31)           :: TempRes
END TYPE TimeIdx

! The TimeIdxCollection derived type contains the pointers with the
! current valid vector indices for all defined cycling intervals.
TYPE ::  TimeIdxCollection
   TYPE(TimeIdx), POINTER :: CONSTANT
   TYPE(TimeIdx), POINTER :: HOURLY
   TYPE(TimeIdx), POINTER :: HOURLY_GRID
   TYPE(TimeIdx), POINTER :: WEEKDAY
   TYPE(TimeIdx), POINTER :: MONTHLY
END TYPE TimeIdxCollection

!-----------------------------------------------------------------------
! cIdListPnt: Derived type definition for pointing to list containers
!-----------------------------------------------------------------------
TYPE :: cIDListPnt
   TYPE(DataCont),     POINTER :: PNT ! Pointer to list container
END TYPE cIDListPnt

!-----------------------------------------------------------------------
! Variables to store (unique) scale factor IDs and species names
!-----------------------------------------------------------------------
TYPE :: ScalIDCont
   INTEGER                   :: ScalID
   TYPE(ScalIDCont), POINTER :: NEXT
END TYPE ScalIDCont

TYPE :: SpecNameCont
   CHARACTER(LEN=31)           :: SpecName
   TYPE(SpecNameCont), POINTER :: NEXT
END TYPE SpecNameCont

!-----------------------------------------------------------------------
! Derived type to store options
!-----------------------------------------------------------------------
TYPE :: Opt
   CHARACTER(LEN=OPTLEN)   :: OptName
   CHARACTER(LEN=OPTLEN)   :: OptValue
   TYPE(Opt), POINTER      :: NextOpt => NULL()
END TYPE Opt
```

```
!-----------------------------------------------------------------------
! Derived type to manage list of extensions and associated options
!-----------------------------------------------------------------------
TYPE :: Ext
   CHARACTER(LEN=255)       :: ExtName  ! Name
   CHARACTER(LEN=OPTLEN)    :: Spcs     ! Species
   INTEGER                  :: ExtNr    ! Ext. number
   TYPE(Opt), POINTER       :: Opts     ! Options linked list
   TYPE(Ext), POINTER       :: NextExt  ! next list item
END TYPE Ext

!-----------------------------------------------------------------------
! Configuration object
!-----------------------------------------------------------------------
TYPE :: ConfigObj
   CHARACTER(LEN=OPTLEN)        :: ROOT           = ''
   CHARACTER(LEN=255)           :: ConfigFileName = ''
   TYPE(ScalIDCont),   POINTER  :: ScalIDList     => NULL()
   TYPE(SpecNameCont), POINTER  :: SpecNameList   => NULL()
   TYPE(ListCont),     POINTER  :: ConfigList     => NULL()
   TYPE(Ext),          POINTER  :: ExtList        => NULL()
   TYPE(HcoErr),       POINTER  :: Err            => NULL()
   LOGICAL                      :: ConfigFileRead = .FALSE.
END TYPE ConfigObj

!-----------------------------------------------------------------------
! DiagnCont: Diagnostics container derived type declaration
!-----------------------------------------------------------------------
TYPE :: DiagnCont
   CHARACTER(LEN= 63)          :: cName          ! Cont. name
   CHARACTER(LEN=255)          :: long_name      ! ncdf long_name attribute
   INTEGER                     :: cID            ! Cont. ID
   INTEGER                     :: ExtNr          ! Extension #
   INTEGER                     :: Cat            ! Category
   INTEGER                     :: Hier           ! Hierarchy
   INTEGER                     :: HcoID          ! HEMCO species ID
   INTEGER                     :: AutoFill       ! fill automatically?
   INTEGER                     :: SpaceDim       ! Space dimension (1-3)
   REAL(sp)                    :: Scalar         ! 1D scalar
   TYPE(Arr2D_SP),     POINTER :: Arr2D          ! 2D array
   TYPE(Arr3D_SP),     POINTER :: Arr3D          ! 3D array
   REAL(sp)                    :: Total          ! Diagnostics total
   LOGICAL                     :: DtaIsPtr       ! Is data just a pointer?
   INTEGER                     :: LevIdx         ! Level index to be used
   CHARACTER(LEN= 31)          :: OutUnit        ! Output unit
   INTEGER                     :: AreaFlag       ! 2=per area, 3=per volume, 0 otherwise
   REAL(hp)                    :: AreaScal       ! Scale factor for area
```

```
   REAL(hp)                    :: MassScal       ! Scale factor for mass
   REAL(hp)                    :: ScaleFact      ! Uniform scale factor
   INTEGER                     :: TimeAvg        ! Scale flag for time unit
   INTEGER                     :: Counter        ! time steps since
                                                 ! last output
   CHARACTER(LEN= 31)          :: AvgName        ! Output averaging operation
   INTEGER                     :: AvgFlag        ! Averaging flag for
                                                 !  non-standard units
   INTEGER                     :: LastUpdateID   ! Last update time
   INTEGER                     :: nnGetCalls     ! # of Diagn_Get calls w/o update
   LOGICAL                     :: IsOutFormat    ! Data is in output format?
   INTEGER                     :: CollectionID   ! Collection diagnostics belongs to
   TYPE(DiagnCont),    POINTER :: NextCont       ! Ptr to next item in list
END TYPE DiagnCont

!-----------------------------------------------------------------------
! Diagnostcs collection derived type.
! DiagnList      : Linked list with all diagnostics container of
!                  this collection.
! nnDiag         : Number of diagnostics in this collection.
! AF_LevelDefined: Set to true if there is at least one autofill
!                  diagnostics at the given level (1-4).
! PREFIX         : Prefix to be used for diagnostics output file name.
! NX, NY, NZ     : Grid dimensions.
! TS             : Time step. This is only of relevance for emission
!                  diagnostics that are internally converted from
!                  kg/m2/s to kg/m2.
! AREA_M2        : Surface grid box areas. May be required for unit
!                  conversions.
!-----------------------------------------------------------------------
TYPE :: DiagnCollection
   TYPE(DiagnCont),       POINTER :: DiagnList          => NULL()
   INTEGER                        :: nnDiagn            =  0
   LOGICAL                        :: AF_LevelDefined(4) =  .FALSE.
   INTEGER                        :: CollectionID       = -1
   CHARACTER(LEN=255)             :: PREFIX             =  ''
   INTEGER                        :: NX                 =  0
   INTEGER                        :: NY                 =  0
   INTEGER                        :: NZ                 =  0
   INTEGER                        :: deltaYMD           =  0
   INTEGER                        :: deltaHMS           =  0
   INTEGER                        :: lastYMD            = -1
   INTEGER                        :: lastHMS            = -1
   INTEGER                        :: OutTimeStamp       = -1
   REAL(sp)                       :: TS                 =  0       ! Time step
   REAL(hp),              POINTER :: AREA_M2(:,:)       => NULL()
   TYPE(DiagnCollection), POINTER :: NextCollection     => NULL()
END TYPE DiagnCollection
```

```
TYPE :: DiagnBundle
   TYPE(DiagnCollection), POINTER :: Collections       => NULL()
   INTEGER                        :: HcoDiagnIDDefault = -999
   INTEGER                        :: HcoDiagnIDRestart = -999
   INTEGER                        :: HcoDiagnIDManual  = -999
   INTEGER                        :: nnCollections     = 0
END TYPE DiagnBundle
```

REVISION HISTORY:

```
15 Feb 2016 - C. Keller - Initial version (collected from various modules)
12 May 2017 - C. Keller - Added option ScaleEmis
```

1.23 Fortran: Module Interface hco_readlist_mod.F90

Module HCO_ReadList_Mod contains routines and variables for the HEMCO ReadList. ReadList is a collection of all data containers used by HEMCO. They are categorized according to their reading update frequency, i.e all data containers that need to be updated on an annual basis are stored in ReadList 'Year', etc. The following reading update frequencies are supported:

- Year: update every year (annual data)
- Month: update every month (monthly data)
- Day: update every day (daily data)
- Hour: update every hour (hourly data)
- Once: update only once (time-invariant data)
- Always: update every time step

INTERFACE:

```
MODULE HCO_ReadList_Mod
```

USES:

```
USE HCO_Types_Mod
USE HCO_Error_MOD
USE HCO_State_MOD,    ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: ReadList_Init
PUBLIC  :: ReadList_Read
PUBLIC  :: ReadList_Set
PUBLIC  :: ReadList_Print
PUBLIC  :: ReadList_Cleanup
PUBLIC  :: ReadList_Remove
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: DtCont_Add
PRIVATE :: ReadList_Fill
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller   - Initial version
01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
```

1.23.1 ReadList_Set

Subroutine ReadList_Set places the passed data container Dct in one of the reading lists, according to the data update frequency specified in the HEMCO configuration file. Containers are sorted with increasing container ID.

INTERFACE:

```
SUBROUTINE ReadList_Set( am_I_Root, HcoState, Dct, RC )
```

USES:

```
USE HCO_LOGFILE_MOD, ONLY : HCO_PrintDataCont
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState
TYPE(DataCont),   POINTER        :: Dct
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
```

1.23.2 ReadList_Read

Subroutine ReadList_Read makes sure that all arrays in the reading lists are up to date, i.e. it invokes the data reading calls for those lists that need to be refreshed.

INTERFACE:

```
SUBROUTINE ReadList_Read( am_I_Root, HcoState, RC, ReadAll )
```

USES:

```
USE HCO_CLOCK_MOD, ONLY : HcoClock_NewYear
USE HCO_CLOCK_MOD, ONLY : HcoClock_NewMonth
USE HCO_CLOCK_MOD, ONLY : HcoClock_NewDay
USE HCO_CLOCK_MOD, ONLY : HcoClock_NewHour
USE HCO_CLOCK_MOD, ONLY : HcoClock_First
```

INPUT PARAMETERS:

```
LOGICAL,           INTENT(IN   )  :: am_I_Root  ! root CPU?
LOGICAL, OPTIONAL, INTENT(IN   )  :: ReadAll    ! read all fields?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),   POINTER        :: HcoState   ! HEMCO state object
INTEGER,           INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
```

1.23.3 ReadList_Fill

Subroutine ReadList_Fill (re-)reads the data from all containers of the passed ReadList. In a non-ESMF environment, this routine calls the HEMCO generic (netCDF) reading and remapping routines. In an ESMF environment, the arrays are obtained through the ESMF/MAPL software framework. ReadLIst_Fill provides the interface between HEMCO and the data reading interface. See module HCOI_DATAREAD_MOD.F90 for more details on data reading.

The ReadList containers are added to EmisList immediately after data filling. This has the advantage that data arrays are immediately available through routine HCO_GetPtr. This is required for country mappings that depend on the country mask input field.

INTERFACE:

```
SUBROUTINE ReadList_Fill( am_I_Root, HcoState, ReadList, RC )
```

USES:

```
USE HCOIO_DataRead_Mod, ONLY : HCOIO_DataRead
USE HCOIO_Read_Std_Mod, ONLY : HCOIO_ReadOther
USE HCOIO_Read_Std_Mod, ONLY : HCOIO_CloseAll
USE HCO_FileData_Mod,   ONLY : FileData_ArrIsDefined
USE HCO_FileData_Mod,   ONLY : FileData_ArrIsTouched
USE HCO_EmisList_Mod,   ONLY : EmisList_Pass
USE HCO_DataCont_Mod,   ONLY : DataCont_Cleanup
USE HCO_TIDX_MOD,       ONLY : tIDx_Assign
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root  ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState   ! HEMCO state object
TYPE(ListCont),  POINTER        :: ReadList   ! Current reading list
INTEGER,         INTENT(INOUT)  :: RC         ! Success or failure?
```

REMARKS:

```
Different HCOI_DATAREAD routines may be invoked depending on the
model environment.
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Dec 2014 - C. Keller - Now pass container to EmisList immediately after
                          reading the data. Added second loop to remove
                          data arrays that are not used in EmisList.
02 Feb 2015 - C. Keller - Now call tIDx_Assign here instead of in
                          hco_emislist_mod. This way, hco_emislist_mod
                          can also be used by hco_clock_mod.
24 Mar 2015 - C. Keller - Now avoid closing/reopening the same file all
                          the time.
24 Mar 2016 - C. Keller - Remove GetFileLUN and SaveFileLUN. This is now
                          handled in hcoio_read_std_mod.F90.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.23.4 DtCont_Add

Subroutine DtCont_Add adds a new container to the specified reading list.

INTERFACE:

```
SUBROUTINE DtCont_Add( ReadList, Dct )
```

INPUT PARAMETERS:

```
TYPE(ListCont), POINTER :: ReadList
TYPE(DataCont), POINTER :: Dct
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.23.5 ReadList_Init

Subroutine ReadList_Init initializes the ReadList.

INTERFACE:

```
SUBROUTINE ReadList_Init( am_I_Root, ReadLists, RC )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(RdList),     POINTER        :: ReadLists
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
```

1.23.6 ReadList_Print

Subroutine ReadList_Print displays the content of ReadList.

INTERFACE:

```
SUBROUTINE ReadList_Print( HcoState, ReadLists, verb )
```

USES:

```
  USE HCO_LOGFILE_MOD,  ONLY : HCO_PrintList
!INPUT ARGUMENTS
  TYPE(HCO_State), POINTER       :: HcoState
  TYPE(RdList),    POINTER       :: ReadLists
  INTEGER,         INTENT(IN)    :: verb   ! verbose number
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
15 Mar 2015 - C. Keller - Added verbose number as input argument
```

1.23.7 ReadList_Remove

Subroutine ReadList_Remove removes the container given by name from the ReadList. If no container with the given name exist, nothing is done. This routine returns an error if the container already holds data.

INTERFACE:

```
SUBROUTINE ReadList_Remove( am_I_Root, HcoState, cName, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )   :: am_I_Root
TYPE(HCO_State),  POINTER         :: HcoState
CHARACTER(LEN=*), INTENT(IN   )   :: cName
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
13 Jan 2015 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.23.8 ReadList_Cleanup

Subroutine ReadList_Cleanup removes all content of ReadList. If RemoveDct is set to True, the content of the data containers will be also removed, otherwise the corresponding pointer is just nullified.

INTERFACE:

```
SUBROUTINE ReadList_Cleanup( ReadLists, RemoveDct )
```

USES:

```
USE HCO_DataCont_Mod, ONLY : ListCont_Cleanup
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )   :: RemoveDct
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(RdList),     POINTER         :: ReadLists
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
```

1.24 Fortran: Module Interface hco_arr_mod.F90

Module HCO_Arr_Mod contains routines and variables to initialize, validate, and cleanup HEMCO data arrays. HEMCO data arrays can be 2D or 3D. They can be organized as single arrays or as vector of arrays to represent an additional dimension (time).

The public data types Arr2D_Hp and Arr3D_Hp represent the 2D/3D arrays used by HEMCO. The HEMCO precision HP is defined in HCO_Error_Mod. There is also an integer 2D array (type Arr2D_I) that can be used to store integer data. Additional data types can be added as needed.

Data values are stored in array 'Val'. Val can be self-allocated or a pointer to existing data, as denoted by the Alloc flag.

At the moment, the following HEMCO structures use HEMCO arrays:

- FileData: emission data (base emissions, scale factors, masks) stored in the FileData derived type. These data are read from disk as specified in the configuration file. See HCO_FileData_Mod.F90.
- FluxArr: the HEMCO flux arrays (emissions and deposition velocities) stored in the HEMCO state object. See HCO_State_Mod.F90.
- Grid: all grid information arrays (x midpoints, y midpoints, etc.) stored in the HEMCO state object.
- ExtDat: external data required by the extensions (primarily met fields). See HCOX_State_Mod.F90.

INTERFACE:

```
MODULE HCO_Arr_Mod
```

USES:

```
USE HCO_Error_Mod

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCO_ArrInit
PUBLIC  :: HCO_ArrAssert
PUBLIC  :: HCO_ArrCleanup
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: HCO_ArrInit_3D_Hp
PRIVATE :: HCO_ArrInit_2D_Hp
PRIVATE :: HCO_ArrInit_3D_Sp
PRIVATE :: HCO_ArrInit_2D_Sp
PRIVATE :: HCO_ArrInit_2D_I
PRIVATE :: HCO_ArrVecInit_3D_Hp
```

```
PRIVATE :: HCO_ArrVecInit_2D_Hp
PRIVATE :: HCO_ArrVecInit_3D_Sp
PRIVATE :: HCO_ArrVecInit_2D_Sp
PRIVATE :: HCO_ValInit
PRIVATE :: HCO_ValInit_3D_Sp
PRIVATE :: HCO_ValInit_3D_Dp
PRIVATE :: HCO_ValInit_2D_Sp
PRIVATE :: HCO_ValInit_2D_Dp
PRIVATE :: HCO_ValInit_2D_I
PRIVATE :: HCO_ArrAssert_2D_Hp
PRIVATE :: HCO_ArrAssert_3D_Hp
PRIVATE :: HCO_ArrAssert_2D_Sp
PRIVATE :: HCO_ArrAssert_3D_Sp
PRIVATE :: HCO_ArrCleanup_3D_Hp
PRIVATE :: HCO_ArrCleanup_2D_Hp
PRIVATE :: HCO_ArrCleanup_3D_Sp
PRIVATE :: HCO_ArrCleanup_2D_Sp
PRIVATE :: HCO_ArrCleanup_2D_I
PRIVATE :: HCO_ArrVecCleanup_3D_Hp
PRIVATE :: HCO_ArrVecCleanup_2D_Hp
PRIVATE :: HCO_ArrVecCleanup_3D_Sp
PRIVATE :: HCO_ArrVecCleanup_2D_Sp
PRIVATE :: HCO_ValCleanup_3D_Sp
PRIVATE :: HCO_ValCleanup_3D_Dp
PRIVATE :: HCO_ValCleanup_2D_Sp
PRIVATE :: HCO_ValCleanup_2D_Dp
PRIVATE :: HCO_ValCleanup_2D_I
```

PUBLIC DATA MEMBERS:

```
! 2D arrays
TYPE, PUBLIC :: Arr2D_Hp
   REAL(hp), POINTER :: Val(:,:)    ! x,y
   LOGICAL           :: Alloc       ! Allocated?
END TYPE Arr2D_Hp

TYPE, PUBLIC :: Arr2D_I
   INTEGER,  POINTER :: Val(:,:)    ! x,y
   LOGICAL           :: Alloc       ! Allocated?
END TYPE Arr2D_I

TYPE, PUBLIC :: Arr2D_Sp
   REAL(sp), POINTER :: Val(:,:)    ! x,y
   LOGICAL           :: Alloc       ! Allocated?
END TYPE Arr2D_Sp

! 3D arrays
TYPE, PUBLIC :: Arr3D_Hp
   REAL(hp), POINTER :: Val(:,:,:)  ! x,y,z
```

```
   LOGICAL           :: Alloc       ! Allocated?
END TYPE Arr3D_Hp

TYPE, PUBLIC :: Arr3D_Sp
   REAL(sp), POINTER :: Val(:,:,:)  ! x,y,z
   LOGICAL           :: Alloc       ! Allocated?
END TYPE Arr3D_Sp
```

PRIVATE TYPES:

```
INTERFACE HCO_ArrInit
   MODULE PROCEDURE HCO_ArrInit_3D_Hp
   MODULE PROCEDURE HCO_ArrInit_2D_Hp
   MODULE PROCEDURE HCO_ArrInit_3D_Sp
   MODULE PROCEDURE HCO_ArrInit_2D_Sp
   MODULE PROCEDURE HCO_ArrInit_2D_I
   MODULE PROCEDURE HCO_ArrVecInit_3D_Hp
   MODULE PROCEDURE HCO_ArrVecInit_2D_Hp
   MODULE PROCEDURE HCO_ArrVecInit_3D_Sp
   MODULE PROCEDURE HCO_ArrVecInit_2D_Sp
END INTERFACE HCO_ArrInit

INTERFACE HCO_ValInit
   MODULE PROCEDURE HCO_ValInit_3D_Sp
   MODULE PROCEDURE HCO_ValInit_3D_Dp
   MODULE PROCEDURE HCO_ValInit_2D_Sp
   MODULE PROCEDURE HCO_ValInit_2D_Dp
   MODULE PROCEDURE HCO_ValInit_2D_I
END INTERFACE HCO_ValInit

INTERFACE HCO_ArrAssert
   MODULE PROCEDURE HCO_ArrAssert_2D_Hp
   MODULE PROCEDURE HCO_ArrAssert_3D_Hp
   MODULE PROCEDURE HCO_ArrAssert_2D_Sp
   MODULE PROCEDURE HCO_ArrAssert_3D_Sp
   MODULE PROCEDURE HCO_ArrAssert_2D_I
END INTERFACE HCO_ArrAssert

INTERFACE HCO_ArrCleanup
   MODULE PROCEDURE HCO_ArrCleanup_3D_Hp
   MODULE PROCEDURE HCO_ArrCleanup_2D_Hp
   MODULE PROCEDURE HCO_ArrCleanup_2D_I
   MODULE PROCEDURE HCO_ArrCleanup_3D_Sp
   MODULE PROCEDURE HCO_ArrCleanup_2D_Sp
   MODULE PROCEDURE HCO_ArrVecCleanup_3D_Hp
   MODULE PROCEDURE HCO_ArrVecCleanup_2D_Hp
   MODULE PROCEDURE HCO_ArrVecCleanup_3D_Sp
   MODULE PROCEDURE HCO_ArrVecCleanup_2D_Sp
END INTERFACE HCO_ArrCleanup
```

```
INTERFACE HCO_ValCleanup
   MODULE PROCEDURE HCO_ValCleanup_3D_Sp
   MODULE PROCEDURE HCO_ValCleanup_3D_Dp
   MODULE PROCEDURE HCO_ValCleanup_2D_Sp
   MODULE PROCEDURE HCO_ValCleanup_2D_Dp
   MODULE PROCEDURE HCO_ValCleanup_2D_I
END INTERFACE HCO_ValCleanup
```

REVISION HISTORY:

```
19 Dec 2013 - C. Keller   - Initialization
01 Jul 2014 - R. Yantosca - Corrected errors in ProTeX headers
01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
01 Oct 2014 - C. Keller   - Added Alloc flag
```

1.24.1 HCO_ArrInit_2D_Hp

Subroutine HCO_ArrInit_2D_Hp initializes the given data container 2D array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrInit_2D_Hp( Arr, nx, ny, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Hp), POINTER       :: Arr       ! Array
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.2 HCO_ArrInit_2D_Sp

Subroutine HCO_ArrInit_2D_Sp initializes the given data container 2D array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrInit_2D_Sp( Arr, nx, ny, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Sp), POINTER       :: Arr       ! Array
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.3 HCO_ArrInit_2D_I

Subroutine HCO_ArrInit_2D_I initializes the given data container integer 2D array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrInit_2D_I( Arr, nx, ny, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_I),  POINTER       :: Arr   ! Array
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.4 HCO_ArrInit_3D_Hp

Subroutine HCO_ArrInit_3D_Hp initializes the given data container 3D array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrInit_3D_Hp( Arr, nx, ny, nz, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Hp), POINTER       :: Arr   ! Array
INTEGER,        INTENT(IN)    :: nx       ! x-dim
INTEGER,        INTENT(IN)    :: ny       ! y-dim
INTEGER,        INTENT(IN)    :: nz       ! z-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC       ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.5 HCO_ArrInit_3D_Sp

Subroutine HCO_ArrInit_3D_Sp initializes the given data container 3D array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrInit_3D_Sp( Arr, nx, ny, nz, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Sp), POINTER       :: Arr   ! Array
INTEGER,        INTENT(IN)    :: nx       ! x-dim
INTEGER,        INTENT(IN)    :: ny       ! y-dim
INTEGER,        INTENT(IN)    :: nz       ! z-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT) :: RC       ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.6 HCO_ArrVecInit_2D_Hp

Subroutine HCO_ArrVecInit_2D_Hp initializes the given data container 2D array vector. nn denotes the number of 2D arrays, and nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrVecInit_2D_Hp( ArrVec, nn, nx, ny, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Hp),   POINTER       :: ArrVec(:) ! Array vector
INTEGER,          INTENT(IN)    :: nn        ! vector length
INTEGER,          INTENT(IN)    :: nx        ! x-dim
INTEGER,          INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.7 HCO_ArrVecInit_2D_Sp

Subroutine HCO_ArrVecInit_2D_Sp initializes the given data container 2D array vector. nn denotes the number of 2D arrays, and nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrVecInit_2D_Sp( ArrVec, nn, nx, ny, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Sp),   POINTER       :: ArrVec(:) ! Array vector
INTEGER,          INTENT(IN)    :: nn        ! vector length
INTEGER,          INTENT(IN)    :: nx        ! x-dim
INTEGER,          INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.8 HCO_ArrVecInit_3D_Hp

Subroutine HCO_ArrVecInit_3D_Hp initializes the given data container 3D array vector. nn denotes the number of 2D arrays, and nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrVecInit_3D_Hp( ArrVec, nn, nx, ny, nz, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Hp),   POINTER       :: ArrVec(:) ! Array vector
INTEGER,          INTENT(IN)    :: nn        ! vector length
INTEGER,          INTENT(IN)    :: nx        ! x-dim
INTEGER,          INTENT(IN)    :: ny        ! y-dim
INTEGER,          INTENT(IN)    :: nz        ! z-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.9 HCO_ArrVecInit_3D_Sp

Subroutine HCO_ArrVecInit_3D_Sp initializes the given data container 3D array vector. nn denotes the number of 2D arrays, and nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ArrVecInit_3D_Sp( ArrVec, nn, nx, ny, nz, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Sp),   POINTER       :: ArrVec(:) ! Array vector
INTEGER,          INTENT(IN)    :: nn        ! vector length
INTEGER,          INTENT(IN)    :: nx        ! x-dim
INTEGER,          INTENT(IN)    :: ny        ! y-dim
INTEGER,          INTENT(IN)    :: nz        ! z-dim
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.10 HCO_ValInit_2D_Sp

Subroutine HCO_ValInit_2D_Sp initializes the given data container 2D single precision array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ValInit_2D_Sp( Val, nx, ny, Alloc, RC )
```

INPUT PARAMETERS:

```
REAL(sp),       POINTER       :: Val(:,:)  ! Array
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,        INTENT(  OUT) :: Alloc     ! allocated?
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.11 HCO_ValInit_2D_Dp

Subroutine HCO_ValInit_2D_Dp initializes the given data container 2D double precision array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ValInit_2D_Dp( Val, nx, ny, Alloc, RC )
```

INPUT PARAMETERS:

```
REAL(dp),       POINTER       :: Val(:,:)  ! Array
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,        INTENT(  OUT) :: Alloc     ! allocated?
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.12 HCO_ValInit_2D_I

Subroutine HCO_ValInit_2D_I initializes the given data container 2D integer array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ValInit_2D_I( Val, nx, ny, Alloc, RC )
```

INPUT PARAMETERS:

```
INTEGER,        POINTER       :: Val(:,:)  ! Array
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,        INTENT(  OUT) :: Alloc     ! allocated?
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.13 HCO_ValInit_3D_Dp

Subroutine HCO_ValInit_3D_Dp initializes the given data container 3D double precision array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ValInit_3D_Dp( Val, nx, ny, nz, Alloc, RC )
```

INPUT PARAMETERS:

```
REAL(dp),       POINTER       :: Val(:,:,:)! Array
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
INTEGER,        INTENT(IN)    :: nz        ! z-dim
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,        INTENT(  OUT) :: Alloc     ! allocated?
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.14 HCO_ValInit_3D_Sp

Subroutine HCO_ValInit_3D_Sp initializes the given data container 3D single precision array. nx and ny denote the array size dimensions. If nx is set to 0, no data is allocated but Val is set to a (nullified) pointer instead.

INTERFACE:

```
SUBROUTINE HCO_ValInit_3D_Sp( Val, nx, ny, nz, Alloc, RC )
```

INPUT PARAMETERS:

```
REAL(sp),       POINTER       :: Val(:,:,:)! Array
INTEGER,        INTENT(IN)    :: nx        ! x-dim
INTEGER,        INTENT(IN)    :: ny        ! y-dim
INTEGER,        INTENT(IN)    :: nz        ! z-dim
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,        INTENT(  OUT) :: Alloc     ! allocated?
INTEGER,        INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.15 HCO_ArrAssert_3D_Hp

Routine HCO_ArrAssert_3D_Hp makes sure that the passed 3D array is allocated.

INTERFACE:

```
SUBROUTINE HCO_ArrAssert_3D_Hp( ThisArr3D, I, J, L, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Hp), POINTER         :: ThisArr3D ! 3D array
INTEGER,        INTENT(IN   )   :: I, J, L   ! Array dims
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT)   :: RC        ! Return code
```

REMARKS:

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.16 HCO_ArrAssert_3D_Sp

Routine HCO_ArrAssert_3D_Sp makes sure that the passed 3D array is allocated.

INTERFACE:

```
SUBROUTINE HCO_ArrAssert_3D_Sp( ThisArr3D, I, J, L, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Sp),  POINTER         :: ThisArr3D ! 3D array
INTEGER,         INTENT(IN   )   :: I, J, L   ! Array dims
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)   :: RC        ! Return code
```

REMARKS:

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.17 HCO_ArrAssert_2D_Hp

Routine HCO_ArrAssert_2D_Hp makes sure that the passed 2D array is allocated.

INTERFACE:

```
SUBROUTINE HCO_ArrAssert_2D_Hp( ThisArr2D, I, J, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Hp),  POINTER         :: ThisArr2D ! 2D array
INTEGER,         INTENT(IN   )   :: I, J      ! Array dims
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)   :: RC        ! Return code
```

REMARKS:

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.18 HCO_ArrAssert_2D_Sp

Routine HCO_ArrAssert_2D_Sp makes sure that the passed 2D array is allocated.

INTERFACE:

```
SUBROUTINE HCO_ArrAssert_2D_Sp( ThisArr2D, I, J, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Sp),  POINTER         :: ThisArr2D ! 2D array
INTEGER,         INTENT(IN   )   :: I, J      ! Array dims
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)   :: RC        ! Return code
```

REMARKS:

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.19 HCO_ArrAssert_2D_I

Routine HCO_ArrAssert_2D_I makes sure that the passed 2D array is allocated.

INTERFACE:

```
SUBROUTINE HCO_ArrAssert_2D_I( ThisArr2D, I, J, RC )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_I),   POINTER         :: ThisArr2D ! 2D array
INTEGER,         INTENT(IN   )   :: I, J      ! Array dims
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)   :: RC        ! Return code
```

REMARKS:

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.20 HCO_ArrCleanup_2D_Hp

Subroutine HCO_ArrCleanup_2D_Hp cleans up the given container 2D array.

INTERFACE:

```
SUBROUTINE HCO_ArrCleanup_2D_Hp( Arr, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Hp),      POINTER  :: Arr       ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate allocated array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.21 HCO_ArrCleanup_2D_Sp

Subroutine HCO_ArrCleanup_2D_Sp cleans up the given container 2D array.

INTERFACE:

```
SUBROUTINE HCO_ArrCleanup_2D_Sp( Arr, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Sp),      POINTER  :: Arr       ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate allocated array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.22 HCO_ArrCleanup_2D_I

Subroutine HCO_ArrCleanup_2D_I cleans up the given container 2D array.

INTERFACE:

```
SUBROUTINE HCO_ArrCleanup_2D_I( Arr, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_I),       POINTER  :: Arr       ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.23 HCO_ArrCleanup_3D_Hp

Subroutine HCO_ArrCleanup_3D_Hp cleans up the given container 3D array.

INTERFACE:

```
SUBROUTINE HCO_ArrCleanup_3D_Hp( Arr, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Hp),      POINTER  :: Arr       ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.24 HCO_ArrCleanup_3D_Sp

Subroutine HCO_ArrCleanup_3D_Sp cleans up the given container 3D array.

INTERFACE:

```
SUBROUTINE HCO_ArrCleanup_3D_Sp( Arr, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Sp),      POINTER  :: Arr       ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.25 HCO_ArrVecCleanup_2D_Hp

Subroutine HCO_ArrVecCleanup_2D_Hp cleans up the given container 2D array vector.

INTERFACE:

```
SUBROUTINE HCO_ArrVecCleanup_2D_Hp( ArrVec, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Hp),      POINTER  :: ArrVec(:) ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.26 HCO_ArrVecCleanup_2D_Sp

Subroutine HCO_ArrVecCleanup_2D_Sp cleans up the given container 2D array vector.

INTERFACE:

```
SUBROUTINE HCO_ArrVecCleanup_2D_Sp( ArrVec, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr2D_Sp),      POINTER  :: ArrVec(:) ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.27 HCO_ArrVecCleanup_3D_Hp

Subroutine HCO_ArrVecCleanup_3D_Hp cleans up the given container 3D array vector.

INTERFACE:

```
SUBROUTINE HCO_ArrVecCleanup_3D_Hp( ArrVec, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Hp),      POINTER  :: ArrVec(:) ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.28 HCO_ArrVecCleanup_3D_Sp

Subroutine HCO_ArrVecCleanup_3D_Sp cleans up the given container 3D array vector.

INTERFACE:

```
SUBROUTINE HCO_ArrVecCleanup_3D_Sp( ArrVec, DeepClean )
```

INPUT PARAMETERS:

```
TYPE(Arr3D_Sp),      POINTER  :: ArrVec(:) ! Array
LOGICAL, INTENT(IN), OPTIONAL :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.29 HCO_ValCleanup_2D_Dp

Subroutine HCO_ValCleanup_2D_Dp cleans up the given container 2D array. If DeepClean is set to TRUE and the array is indeed allocated (as determined by the Alloc flag), the array becomes deallocated. Otherwise, it is just nullified.

INTERFACE:

```
SUBROUTINE HCO_ValCleanup_2D_Dp( Val, Alloc, DeepClean )
```

INPUT PARAMETERS:

```
REAL(dp),            POINTER  :: Val(:,:)  ! Array
LOGICAL, INTENT(IN)           :: Alloc     ! Allocated?
LOGICAL, INTENT(IN)           :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.30 HCO_ValCleanup_2D_Sp

Subroutine HCO_ValCleanup_2D_Sp cleans up the given container 2D array. If DeepClean is set to TRUE and the array is indeed allocated (as determined by the Alloc flag), the array becomes deallocated. Otherwise, it is just nullified.

INTERFACE:

```
SUBROUTINE HCO_ValCleanup_2D_Sp( Val, Alloc, DeepClean )
```

INPUT PARAMETERS:

```
REAL(sp), POINTER    :: Val(:,:)  ! Array
LOGICAL,  INTENT(IN) :: Alloc     ! Allocated?
LOGICAL,  INTENT(IN) :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.31 HCO_ValCleanup_2D_I

Subroutine HCO_ValCleanup_2D_I cleans up the given container 2D array. If DeepClean is set to TRUE and the array is indeed allocated (as determined by the Alloc flag), the array becomes deallocated. Otherwise, it is just nullified.

INTERFACE:

```
SUBROUTINE HCO_ValCleanup_2D_I( Val, Alloc, DeepClean )
```

INPUT PARAMETERS:

```
INTEGER, POINTER    :: Val(:,:)  ! Array
LOGICAL, INTENT(IN) :: Alloc     ! Allocated?
LOGICAL, INTENT(IN) :: DeepClean ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.32 HCO_ValCleanup_3D_Dp

Subroutine HCO_ValCleanup_3D_Dp cleans up the given container 3D array. If DeepClean is set to TRUE and the array is indeed allocated (as determined by the Alloc flag), the array becomes deallocated. Otherwise, it is just nullified.

INTERFACE:

```
SUBROUTINE HCO_ValCleanup_3D_Dp( Val, Alloc, DeepClean )
```

INPUT PARAMETERS:

```
REAL(dp), POINTER    :: Val(:,:,:) ! Array
LOGICAL,  INTENT(IN) :: Alloc      ! Allocated?
LOGICAL,  INTENT(IN) :: DeepClean  ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.24.33 HCO_ValCleanup_3D_Sp

Subroutine HCO_ValCleanup_3D_Sp cleans up the given container 3D array. If DeepClean is set to TRUE and the array is indeed allocated (as determined by the Alloc flag), the array becomes deallocated. Otherwise, it is just nullified.

INTERFACE:

```
SUBROUTINE HCO_ValCleanup_3D_Sp( Val, Alloc, DeepClean )
```

INPUT PARAMETERS:

```
REAL(sp), POINTER    :: Val(:,:,:) ! Array
LOGICAL,  INTENT(IN) :: Alloc      ! Allocated?
LOGICAL,  INTENT(IN) :: DeepClean  ! Deallocate array?
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
01 Oct 2014 - C. Keller - Added Alloc flag
```

1.25 Fortran: Module Interface hco_tidx_mod.F90

Module HCO_tIdx_Mod contains routines and variables to organize and index data array time slices.

The HEMCO data containers can hold multiple 2D or 3D data arrays (aligned in a vector), providing a 4th dimension (time). During emission calculation, only the vector element ('time slice') representative for the given time will be used. Currently, the following time slice intervals are supported:

- Constant: Only one time slice (default)
- Hourly: Between 2-24 time slices. These slices will be split into even day bins, and are cycled through accordingly. the local time is used to obtain the current valid index at each longitude.
- Hourly_gridded: As hourly, but uses the same time slice index across all longitudes (based upon UTC time).
- Weekdaily: Seven time slices, representing the days of the week: Sun, Mon, ..., Sat. Uses local time.
- Monthly: 12 time slices, representing the months of the year: Jan, ..., Dec. Uses local time.

The time slice cycling frequency is automatically determined during creation of a data container - based upon the time stamp settings in the HEMCO configuration file and the time information read from the data (netCDF) file.

Spatial uniform data (i.e. nx = ny = 1) is always assigned the local time cycle intervals (hourly, weekdaily, or monthly), e.g. the local time is used at every grid box when picking the time slice at a given time. For gridded data, it's assumed that local-time effects are already taken into account and UTC time is used at all locations to select the currently valid time slice. The exception is weekdaily data, which is always assumed to be in local time.

Structure AlltIDx organizes the indexing of the vector arrays. It contains the current valid time slice index for each of the above defined time slice intervals. Each data container points to one of the elements of AlltIDx, according to the temporal dimension of its array. The values of AlltIDx become update on every HEMCO time step.

INTERFACE:

```
MODULE HCO_tIdx_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Types_Mod,  ONLY : TimeIdx
USE HCO_Types_Mod,  ONLY : TimeIdxCollection

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: tIDx_Assign
PUBLIC :: tIDx_Init
PUBLIC :: tIDx_GetIndx
PUBLIC :: tIDx_Cleanup
PUBLIC :: tIDx_IsInRange
PUBLIC :: HCO_GetPrefTimeAttr
PUBLIC :: HCO_ExtractTime
```

REMARKS:

```
The current local time implementation assumes a regular grid,
i.e. local time does not change with latitude!
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller   - Initialization
22 Aug 2013 - C. Keller   - Some time slice updates.
08 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
08 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
03 Dec 2014 - C. Keller   - Major update: now calculate the time slice
                            indeces on the fly instead of storing them in
                            precalculated vectors.
25 Feb 2015 - R. Yantosca - Comment out WEEKDAY_GRID, it is not used
                            anymore.  This avoids seg faults.
```

1.25.1 tIDx_Init

Subroutine tIDx_Init initializes the time slice index collection.

INTERFACE:

```
SUBROUTINE tIDx_Init( HcoState, RC )
```

USES:

```
USE HCO_State_Mod, ONLY : HCO_State
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState  ! Hemco state
INTEGER,         INTENT(INOUT)  :: RC        ! Return code
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller - Initialization
```

1.25.2 tIDx_Set

Subroutine tIDx_Set linkes the passed TimeIDx type to the corresponding element in the TimeIdx collection, according to the given TypeID. TypeID can be one of the following:

- 1: Constant
- 24: Hourly
- 241: Hourly_Grid
- 7: Weekday
- 71: Weekday_Grid
- 12: Monthly

INTERFACE:

```
SUBROUTINE tIDx_Set( HcoState, ctIDx, TypeID )
```

USES:

```
USE HCO_State_Mod, ONLY : HCO_State
```

INPUT PARAMETERS:

```
TYPE(HCO_State), POINTER     :: HcoState ! HEMCO state
TYPE(TimeIdx),   POINTER     :: ctIDx    ! container TimeIDx
INTEGER,         INTENT(IN)  :: TypeID   ! type ID
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller - Initialization
```

1.25.3 tIDx_Cleanup

Subroutine tIDx_Cleanup deallocates the time slice index collection.

INTERFACE:

```
SUBROUTINE tIDx_Cleanup( AlltIDx )
!Input/output arguments:

   TYPE(TimeIdxCollection), POINTER :: AlltIDx
```

REVISION HISTORY:

```
29 Dec 2012 - C. Keller - Initialization
```

1.25.4 tIDx_GetIndx

Subroutine tIDx_GetIndx calculates the current valid index values for the given file data time slice type and longitude location

INTERFACE:

```
FUNCTION tIDx_GetIndx ( am_I_Root, HcoState, Dta, I, J ) RESULT ( Indx )
```

USES:

```
USE HCO_State_Mod,    ONLY : HCO_State
USE HCO_Types_Mod,    ONLY : FileData
USE HCO_CLOCK_MOD,    ONLY : HcoClock_Get, HcoClock_GetLocal
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN) :: am_I_Root ! Longitude index of interest
TYPE(HCO_State), POINTER    :: HcoState  ! Hemco state
TYPE(FileData),  POINTER    :: Dta       ! File data object
INTEGER,         INTENT(IN) :: I         ! Longitude index of interest
INTEGER,         INTENT(IN) :: J         ! Latitude  index of interest
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER                     :: Indx      ! Index
```

REVISION HISTORY:

```
02 Dec 2014 - C. Keller - Initial version
```

1.25.5 tIDx_Assign

Subroutine tIDx_Assign assigns the time index pointer of the file data object of the passed list container to the corresponding time index element of the time index collection. The time slice cycle interval is determined from the number of time slices (length of the data array vector) and the time difference (deltaT) between them.

Note: Scale factors read directly from the HEMCO configuration file become their delta T assigned upon reading from file (see subroutine Register_Scal in hco_config_mod.F90).

INTERFACE:

```
SUBROUTINE tIDx_Assign( HcoState, Dct, RC )
```

USES:

```
USE HCO_State_Mod,    ONLY : HCO_State
USE HCO_Types_Mod,    ONLY : DataCont
```

INPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState  ! Hemco state
TYPE(DataCont),  POINTER       :: Dct       ! Data container
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC        ! Success or failure?
```

REVISION HISTORY:

```
13 Jan 2014 - C. Keller - Initial version
```

1.25.6 tIDx_IsInRange

Subroutine tIDx_IsInRange returns true if the passed datetime is within the range of the date ranges of the data container.

INTERFACE:

```
FUNCTION tIDx_IsInRange ( Lct, Yr, Mt, Dy, Hr ) RESULT ( InRange )
```

USES:

```
USE HCO_TYPES_MOD,    ONLY : ListCont
```

INPUT PARAMETERS:

```
TYPE(ListCont),  POINTER    :: Lct       ! File data object
INTEGER,         INTENT(IN) :: Yr
INTEGER,         INTENT(IN) :: Mt
INTEGER,         INTENT(IN) :: Dy
INTEGER,         INTENT(IN) :: Hr
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL                     :: InRange
```

REVISION HISTORY:

```
04 Mar 2015 - C. Keller - Initial version
```

1.25.7 HCO_GetPrefTimeAttr

Subroutine HCO_GetPrefTimeAttr returns the preferred time reading attributes for a given field, based upon the specs set in the HEMCO configuration file and the current simulation date. This routine is used to select the proper time slice to be read at the given time.

The time reading attributes are set to the value that is closest (or equal) to the current datetime but within the range provided in the configuration file. If the year, month, or day is not specified, the current simulation date values are taken. For unspecified hours, a value of -1 is returned (this allows to read all hourly slices at once).

INTERFACE:

```
SUBROUTINE HCO_GetPrefTimeAttr( am_I_Root, HcoState, Lct,    &
                                readYr,    readMt,   readDy, &
                                readHr,    readMn,   RC       )
```

USES:

```
USE HCO_STATE_MOD,     ONLY : HCO_State
USE HCO_TYPES_MOD,     ONLY : ListCont
USE HCO_TYPES_MOD,     ONLY : HCO_CFLAG_RANGE, HCO_CFLAG_EXACT
USE HCO_CLOCK_MOD,     ONLY : HcoClock_Get
USE HCO_TIMESHIFT_MOD, ONLY : TimeShift_Apply
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root ! preferred year
TYPE(HCO_State), POINTER       :: HcoState  ! List container
TYPE(ListCont),  POINTER       :: Lct       ! List container
```

OUTPUT PARAMETERS:

```
INTEGER,         INTENT(  OUT) :: readYr    ! preferred year
INTEGER,         INTENT(  OUT) :: readMt    ! preferred month
INTEGER,         INTENT(  OUT) :: readDy    ! preferred day
INTEGER,         INTENT(  OUT) :: readHr    ! preferred hour
INTEGER,         INTENT(  OUT) :: readMn    ! preferred minute
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC        ! Success or failure?
```

REVISION HISTORY:

```
13 Jan 2014 - C. Keller - Initial version
29 Feb 2016 - C. Keller - Added time shift option
03 Mar 2017 - C. Keller - Added option to deal with UTC weekdays
```

1.25.8 HCO_ExtractTime

Subroutine HCO_ExtractTime extracts the time stamp (ranges) from the passed character string. The string is expected to be in format Yr/Mt/Dy/Hr. Valid values for Yr, Mt, Dy and Hr are:

1. Range of values, separated by - sign: e.g. 2000-2010.
2. Single value: 2000
3. Wildcard character (default = *). In this case, the data interval is determined automatically by HEMCO based on the number of time slices found in the data set.
4. Time tokens: *YYYY*,MM, *DD*,HH. When reading the data, these values will be substituted by the current simulation date.

5. String 'WD'. Denotes that the data contains weekday data. It is expected that the first slice represents Sunday. Weekday data can be used in combination with annual or monthly data. In that case, there need to be seven entries for every year and/or month, respectively.

The extracted time stamp is written into the arrays ncYrs, ncMts, ncDys and ncHrs of the passed data structure Dta.

INTERFACE:

```
SUBROUTINE HCO_ExtractTime ( HcoConfig, CharStr, Dta, RC )
```

USES:

```
USE CHARPAK_MOD,        ONLY : STRSPLIT
USE HCO_TYPES_MOD,      ONLY : FileData
USE HCO_TYPES_MOD,      ONLY : ConfigObj
USE HCO_TYPES_MOD,      ONLY : HCO_UFLAG_ALWAYS
USE HCO_EXTLIST_MOD,    ONLY : HCO_GetOpt
USE HCO_TIMESHIFT_MOD,  ONLY : TimeShift_Set
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER       :: HcoConfig  ! config obj
CHARACTER(LEN=*), INTENT(IN   ) :: CharStr
TYPE(FileData),   POINTER       :: Dta
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
18 Sep 2013 - C. Keller - Initial version (update)
29 Feb 2016 - C. Keller - Added time shift option
03 Mar 2017 - C. Keller - Added option to deal with UTC weekdays
```

1.26 Fortran: Module Interface hcoio_read_std_mod.F90

Module HCOIO_read_std_mod controls data processing (file reading, unit conversion, regridding) for HEMCO in the 'standard' environment (i.e. non-ESMF).

INTERFACE:

```
MODULE HCOIO_read_std_mod
```

USES:

```
   USE HCO_Types_Mod
   USE HCO_Error_Mod
   USE HCO_CharTools_Mod
   USE HCO_State_Mod,       ONLY : Hco_State

   IMPLICIT NONE
   PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
   PUBLIC  :: HCOIO_ReadOther
   PUBLIC  :: HCOIO_CloseAll
#if !defined(ESMF_)
   PUBLIC  :: HCOIO_read_std
```

PRIVATE MEMBER FUNCTIONS:

```
   PRIVATE :: GET_TIMEIDX
   PRIVATE :: Check_AvailYMDhm
   PRIVATE :: prefYMDhm_Adjust
   PRIVATE :: Set_tIdx2
   PRIVATE :: IsClosest
   PRIVATE :: GetIndex2Interp
   PRIVATE :: GetWeights
   PRIVATE :: YMDhm2hrs
   PRIVATE :: Normalize_Area
   PRIVATE :: SrcFile_Parse
   PRIVATE :: SigmaMidToEdges
   PRIVATE :: CheckMissVal
   PRIVATE :: GetArbDimIndex
#endif
   PRIVATE :: HCOIO_ReadCountryValues
   PRIVATE :: HCOIO_ReadFromConfig
   PRIVATE :: GetDataVals
   PRIVATE :: GetSliceIdx
   PRIVATE :: FillMaskBox
   PRIVATE :: ReadMath
```

REVISION HISTORY:

```
    22 Aug 2013 - C. Keller   - Initial version
    01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
    01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
    22 Feb 2016 - C. Keller   - Split off from hcoio_dataread_mod.F90
    10 Apr 2017 - R. Yantosca - Time vectors now use YYYYMMDDhhmm format,
                                and are now all REAL(dp) instead of INTEGER(8)
    11 Apr 2017 - R. Yantosca - Added more minor fixes for robustness
```

1.26.1 HCOIO_Read_std

Reads a netCDF file and returns the regridded array in proper units. This routine uses the HEMCO generic data reading and regridding routines.

Two different regridding algorithm are used: NCREGRID for 3D data with vertical regridding, and map_a2a for all other data. map_a2a also supports index-based remapping, while this feature is currently not possible in combination with NCREGRID.

3D data is vertically regridded onto the simulation grid on the sigma interface levels. In order to calculate these levels correctly, the netCDF vertical coordinate description must adhere to the CF - conventions. See routine NC_Get_Sigma_Levels in Ncdf_Mod for more details.

A simpler vertical interpolation scheme is used if (a) the number of vertical levels of the input data corresponds to the number of levels on the simulation grid (direct mapping, no remapping), (b) the vertical level variable name (long_name) contains the word "GEOS-Chem level". In the latter case, the vertical levels of the input data is interpreted as GEOS vertical levels and mapped onto the simulation grid using routine ModelLev_Interpolate.

INTERFACE:

```
SUBROUTINE HCOIO_read_std( am_I_Root, HcoState, Lct, RC )
```

USES:

```
USE Ncdf_Mod,           ONLY : NC_Open
USE Ncdf_Mod,           ONLY : NC_Close
USE Ncdf_Mod,           ONLY : NC_Read_Var
USE Ncdf_Mod,           ONLY : NC_Read_Arr
USE Ncdf_Mod,           ONLY : NC_Get_Grid_Edges
USE Ncdf_Mod,           ONLY : NC_Get_Sigma_Levels
USE Ncdf_Mod,           ONLY : NC_ISMODELLEVEL
USE CHARPAK_MOD,        ONLY : TRANLC
USE HCO_Unit_Mod,       ONLY : HCO_Unit_Change
USE HCO_Unit_Mod,       ONLY : HCO_Unit_ScalCheck
USE HCO_Unit_Mod,       ONLY : HCO_IsUnitless
USE HCO_Unit_Mod,       ONLY : HCO_IsIndexData
USE HCO_Unit_Mod,       ONLY : HCO_UnitTolerance
USE HCO_GeoTools_Mod,   ONLY : HCO_ValidateLon
USE HCO_FileData_Mod,   ONLY : FileData_ArrCheck
USE HCO_FileData_Mod,   ONLY : FileData_Cleanup
USE HCOIO_MESSY_MOD,    ONLY : HCO_MESSY_REGRID
USE HCO_INTERP_MOD,     ONLY : REGRID_MAPA2A
USE HCO_INTERP_MOD,     ONLY : ModelLev_Check
USE HCO_CLOCK_MOD,      ONLY : HcoClock_Get
USE HCO_DIAGN_MOD,      ONLY : Diagn_Update
USE HCO_EXTLIST_MOD,    ONLY : HCO_GetOpt
USE HCO_TIDX_MOD,       ONLY : tIDx_IsInRange
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root  ! Are we on the root CPU?
TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
TYPE(ListCont),   POINTER        :: Lct        ! HEMCO list container
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller   - Initial version
27 Aug 2014 - R. Yantosca - Err msg now displays hcoio_dataread_mod.F90
01 Oct 2014 - C. Keller   - Added file name parser
03 Oct 2014 - C. Keller   - Added vertical regridding capability
12 Dec 2014 - C. Keller   - Don't do vertical regridding if data is already
                            on GEOS-Chem levels.
31 Dec 2014 - C. Keller   - Now call ModelLev_Interpolate for model remapping
                            of model levels.
15 Jan 2015 - C. Keller   - Now allow model level interpolation in
                            combination with MESSy (horizontal) regridding.
03 Feb 2015 - C. Keller   - Moved map_a2a regridding to hco_interp_mod.F90.
24 Mar 2015 - C. Keller   - Added arguments LUN and CloseFile.
27 Mar 2015 - R. Yantosca - Now use a FORMAT statement when printing the
                            filename to the Unix stdout.
08 Apr 2015 - R. Yantosca - Bug fix: set KeepSpec=.TRUE. if there is no
                            species in the container.  This prevents
                            diffs in output in sp vs mp runs.
13 Jul 2015 - C. Keller   - Write data into diagnostics right after reading
                            (if a diagnostics with the same name exists).
23 Sep 2015 - C. Keller   - Support time averaging (cycle flags A and RA).
06 Oct 2015 - C. Keller   - Support additional horizontal coordinates. Added
                            MustFind error checks (cycle flags EF and RF).
22 Nov 2015 - C. Keller   - Bug fix: now use Lun2 if reading second file.
24 Mar 2016 - C. Keller   - Simplified handling of file in buffer. Remove
                            args LUN and CloseFile.
29 Apr 2016 - R. Yantosca - Don't initialize pointers in declaration stmts
```

1.26.2 Get_TimeIdx

Returns the lower and upper time slice index (tidx1 and tidx2, respectively) to be read. These values are determined based upon the time slice information extracted from the netCDF file, the time stamp settings set in the config. file, and the current simulation date.

Return arguments wgt1 and wgt2 denote the weights to be given to the two time slices. This is only of relevance for data that shall be interpolated between two (not necessarily consecutive) time slices. In all other cases, the returned weights are negative and will be

ignored.

Also returns the time slice year and month, as these values may be used for unit conversion.

INTERFACE:

```
SUBROUTINE GET_TIMEIDX( am_I_Root, HcoState, Lct,     &
                        ncLun,     tidx1,    tidx2,   &
                        wgt1,      wgt2,     oYMDhm,   &
                        YMDhm,     YMDhm1,   RC,      &
                        Year )
```

USES:

```
USE Ncdf_Mod,      ONLY : NC_Read_Time_YYYYMMDDhhmm
USE HCO_tIdx_Mod,  ONLY : HCO_GetPrefTimeAttr
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState  ! HcoState object
TYPE(ListCont),   POINTER                 :: Lct       ! List container
INTEGER,          INTENT(IN   )           :: ncLun     ! open ncLun
INTEGER,          INTENT(IN   ), OPTIONAL  :: Year      ! year to be used
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT)           :: tidx1     ! lower time idx
INTEGER,          INTENT(  OUT)           :: tidx2     ! upper time idx
REAL(sp),         INTENT(  OUT)           :: wgt1      ! weight to tidx1
REAL(sp),         INTENT(  OUT)           :: wgt2      ! weight to tidx2
REAL(dp),         INTENT(  OUT)           :: oYMDhm     ! preferred time slice
REAL(dp),         INTENT(  OUT)           :: YMDhm      ! selected time slice
REAL(dp),         INTENT(  OUT)           :: YMDhm1     ! 1st time slice in file
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller - Initial version
27 Feb 2015 - C. Keller - Added weigths
```

1.26.3 Check_AvailYMDhm

Checks if prefYMDhm is within the range of availYMDhm and returns the location of the closest vector element that is in the past (–¿ tidx1). tidx1 is set to -1 otherwise.

INTERFACE:

```
SUBROUTINE Check_AvailYMDhm( Lct, N, availYMDhm, prefYMDhm, tidx1 )
```

INPUT PARAMETERS:

```
TYPE(ListCont),   POINTER      :: Lct
INTEGER,          INTENT(IN)   :: N
REAL(dp),         INTENT(IN)   :: availYMDhm(N)
REAL(dp),         INTENT(IN)   :: prefYMDhm
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(OUT)  :: tidx1
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller   - Initial version
11 Apr 2017 - R. Yantosca - Now epsilon-test time stamps for equality
```

1.26.4 prefYMDhm_Adjust

Adjusts prefYMDhm to the closest available time attribute. Can be adjusted for year (level=1), month (level=2), or day (level=3).

INTERFACE:

```
SUBROUTINE prefYMDhm_Adjust( N, availYMDhm, prefYMDhm, level, tidx1 )
```

INPUT PARAMETERS:

```
INTEGER   , INTENT(IN)     :: N
REAL(dp)  , INTENT(IN)     :: availYMDhm(N)
INTEGER   , INTENT(IN)     :: level
INTEGER   , INTENT(IN)     :: tidx1
```

INPUT/OUTPUT PARAMETERS:

```
REAL(dp)  , INTENT(INOUT)  :: prefYMDhm
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller   - Initial version
17 Jul 2014 - C. Keller   - Now allow to adjust year, month, or day.
10 Apr 2017 - R. Yantosca - Times are now in YYYYMMDDhhmm format
```

1.26.5 Set_tIdx2

sets the upper time slice index by selecting the range of all elements in availYMDhm with the same date (year,month,day) as availYMDh(tidx1).

INTERFACE:

```
SUBROUTINE Set_tIdx2( N, availYMDhm, tidx1, tidx2 )
```

INPUT PARAMETERS:

```
INTEGER,  INTENT(IN)  :: N               ! Number of times
REAL(dp), INTENT(IN)  :: availYMDhm(N)   ! Time stamp vector
INTEGER,  INTENT(IN)  :: tidx1           ! Lower time slice index
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,  INTENT(OUT) :: tidx2           ! Upper time slice index
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller   - Initial version
10 Apr 2017 - R. Yantosca - AvailYMDHm now uses YYYYMMDDhhmm format,
                            so divide by 1d4 instead of 1d2
```

1.26.6 IsClosest

function IsClosest returns true if the selected time index is the 'closest' one. It is defined as being closest if: (a) the currently selected index exactly matches the preferred one. (b) the time gap between the preferred time stamp and the currently selected index is at least as small as any other gap of consecutive prior time stamps.

INTERFACE:

```
FUNCTION IsClosest ( prefYMDhm, availYMDhm, nTime, ctidx1 ) RESULT ( Closest )
```

INPUT PARAMETERS:

```
REAL(dp),   INTENT(IN)  :: prefYMDhm
REAL(dp),   INTENT(IN)  :: availYMDhm(nTime)
INTEGER,    INTENT(IN)  :: nTime
INTEGER,    INTENT(IN)  :: ctidx1
```

OUTPUT PARAMETERS:

```
LOGICAL             :: Closest
```

REVISION HISTORY:

```
03 Mar 2015 - C. Keller   - Initial version
11 Apr 2017 - R. Yantosca - Now epsilon-test time stamps for equality
```

1.26.7 GetIndex2Interp

GetIndex2Interp

INTERFACE:

```
SUBROUTINE GetIndex2Interp ( am_I_Root, HcoState,   Lct,       &
                             nTime,     availYMDhm,            &
                             prefYMDhm, origYMDhm,  tidx1,     &
                             tidx2,     wgt1,       wgt2,  RC   )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)    :: am_I_Root
TYPE(HCO_State),  POINTER       :: HcoState
TYPE(ListCont),   POINTER       :: Lct
INTEGER,          INTENT(IN)    :: nTime
REAL(dp),         INTENT(IN)    :: availYMDhm(nTime)
REAL(dp),         INTENT(IN)    :: prefYMDhm
REAL(dp),         INTENT(IN)    :: origYMDhm
INTEGER,          INTENT(IN)    :: tidx1
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(OUT)   :: tidx2
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp),         INTENT(INOUT) :: wgt1
REAL(sp),         INTENT(INOUT) :: wgt2
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
02 Mar 2015 - C. Keller   - Initial version
11 Apr 2017 - R. Yantosca - Time stamp variables now use YYYYMMDDhhmm
                            fprmat and are REAL(dp) instead of INTEGER(8)
11 Apr 2017 - R. Yantosca - Now epsilon-test time stamps for equality
```

1.26.8 GetWeights

Helper function to get the interpolation weights between two datetime intervals (int1, int2) and for a given time cur.

INTERFACE:

```
SUBROUTINE GetWeights ( int1, int2, cur, wgt1, wgt2 )
```

INPUT PARAMETERS:

```
REAL(dp),         INTENT(IN   )   :: int1, int2, cur
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp),         INTENT(  OUT)   :: wgt1, wgt2
```

REVISION HISTORY:

```
04 Mar 2015 - C. Keller - Initial version
```

1.26.9 YMDhm2hrs

returns the hours of element YMDhm. For simplicity, 30 days are assigned to every month. At the moment, this routine is only called to determine the time interval between two emission time slices (DeltaT) and this approximation is good enough.

INTERFACE:

```
FUNCTION YMDhm2hrs ( YMDhm ) RESULT ( hrs )
```

INPUT PARAMETERS:

```
REAL(dp), INTENT(IN)  :: YMDhm
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER               :: hrs
```

REVISION HISTORY:

```
26 Jan 2015 - C. Keller - Initial version
```

1.26.10 Normalize_Area

Subroutine Normalize_Area normalizes the given array by the surface area calculated from the given netCDF file.

INTERFACE:

```
SUBROUTINE Normalize_Area( HcoState, Array, nlon, LatEdge, FN, RC )
```

INPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER         :: HcoState           ! HEMCO state object
INTEGER,          INTENT(IN   )   :: nlon               ! # of lon midpoints
REAL(hp),         POINTER         :: LatEdge(:)         ! lat edges
CHARACTER(LEN=*), INTENT(IN   )   :: FN                 ! filename
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp),         POINTER         :: Array(:,:,:,:) ! Data
INTEGER,          INTENT(INOUT)   :: RC             ! Return code
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller - Initial version
```

1.26.11 SrcFile_Parse

Routine SrcFile_Parse parses the source file name ('ncFile') of the provided list container Lct. In particular, it searches for tokens such as *ROOT*,YYYY, etc., within the file name and replaces those values with the intendend characters. The parsed file name is returned in string srcFile, while the original file name is retained in Lct.

It now also checks if the file exists. If the file does not exist and the file name contains date tokens, it tries to adjust the file name to the closest available date in the past. The optional flag FUTURE can be used to denote that the next available file in the future shall be selected, even if there is a file that exactly matches the preferred date time. This is useful for interpolation between fields.

INTERFACE:

```
SUBROUTINE SrcFile_Parse ( am_I_Root, HcoState, Lct, srcFile, FOUND, RC, &
                           FUTURE,    Year )
```

USES:

```
USE HCO_TIDX_MOD,         ONLY : HCO_GetPrefTimeAttr
USE HCO_TIDX_MOD,         ONLY : tIDx_IsInRange
USE HCO_CLOCK_MOD,        ONLY : HcoClock_Get
USE HCO_CLOCK_MOD,        ONLY : Get_LastDayOfMonth
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root  ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState   ! HEMCO state object
TYPE(ListCont),   POINTER                 :: Lct        ! HEMCO list container
LOGICAL,          INTENT(IN   ), OPTIONAL :: FUTURE     ! If needed, update
                                                        ! date tokens to future
INTEGER,          INTENT(IN   ), OPTIONAL :: Year       ! To use fixed year
```

OUTPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(  OUT)           :: srcFile    ! output string
LOGICAL,          INTENT(  OUT)           :: FOUND      ! Does file exist?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: RC         ! return code
```

REVISION HISTORY:

```
01 Oct 2014 - C. Keller - Initial version
23 Feb 2015 - C. Keller - Now check for negative return values in
                          HCO_GetPrefTimeAttr
06 Nov 2015 - C. Keller - Bug fix: restrict day to last day of month.
```

1.26.12 SigmaMidToEdges

Helper routine to interpolate sigma mid point values to edges. A simple linear interpolation is performed.

INTERFACE:

```
SUBROUTINE SigmaMidToEdges ( am_I_Root, HcoState, SigMid, SigEdge, RC )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root       ! Root CPU?
TYPE(HCO_State),  POINTER                 :: HcoState        ! HEMCO state obj
REAL(hp),         POINTER                 :: SigMid(:,:,:)   ! sigma levels
```

OUTPUT PARAMETERS:

```
REAL(hp),         POINTER                 :: SigEdge(:,:,:)  ! sigma edges
INTEGER,          INTENT(  OUT)           :: RC              ! return code
```

REVISION HISTORY:

```
03 Oct 2013 - C. Keller - Initial version
```

1.26.13 CheckMissVal

Checks for missing values in the passed array. Missing values of base emissions and masks are set to 0, missing values of scale factors are set to 1.

INTERFACE:

```
SUBROUTINE CheckMissVal ( Lct, Arr )
```

INPUT PARAMETERS:

```
TYPE(ListCont),   POINTER                 :: Lct
REAL(sp),         POINTER                 :: Arr(:,:,:,:)
```

REVISION HISTORY:

```
04 Mar 2015 - C. Keller - Initial version
```

1.26.14 GetArbDimIndex

Subroutine GetArbDimIndex returns the index of the arbitrary file dimension. -1 if no such dimension is defined.

INTERFACE:

```
SUBROUTINE GetArbDimIndex( am_I_Root, HcoState, Lun, Lct, ArbIdx, RC )
```

USES:

```
USE m_netcdf_io_checks
USE m_netcdf_io_get_dimlen
USE HCO_ExtList_Mod,    ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )           :: am_I_Root
TYPE(HCO_State),  POINTER                 :: HcoState
INTEGER,          INTENT(IN   )           :: Lun
TYPE(ListCont),   POINTER                 :: Lct
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT)           :: ArbIdx
INTEGER,          INTENT(  OUT)           :: RC
```

REVISION HISTORY:

```
22 Sep 2015 - C. Keller - Initial version
```

1.26.15 HCOIO_ReadOther

Subroutine HCOIO_ReadOther is a wrapper routine to read data from sources other than netCDF.

If a file name is given (ending with '.txt'), the data are assumed to hold country-specific values (e.g. diurnal scale factors). In all other cases, the data is directly read from the configuration file (scalars).

INTERFACE:

```
SUBROUTINE HCOIO_ReadOther ( am_I_Root, HcoState, Lct, RC )
```

USES:

```
!INPUT PARAMTERS:
  LOGICAL,          INTENT(IN   )    :: am_I_Root
  TYPE(HCO_State), POINTER           :: HcoState    ! HEMCO state
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ListCont),   POINTER          :: Lct
INTEGER,          INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
22 Dec 2014 - C. Keller: Initial version
```

1.26.16 HCOIO_ReadCountryValues

Subroutine HCOIO_ReadCountryValues

INTERFACE:

```
SUBROUTINE HCOIO_ReadCountryValues ( am_I_Root, HcoState, Lct, RC )
```

USES:

```
  USE inquireMod,         ONLY : findFreeLUN
  USE HCO_CHARTOOLS_MOD,  ONLY : HCO_CMT, HCO_SPC, NextCharPos
  USE HCO_EmisList_Mod,   ONLY : HCO_GetPtr
  USE HCO_FileData_Mod,   ONLY : FileData_ArrCheck
!INPUT PARAMTERS:
  LOGICAL,         INTENT(IN   )    :: am_I_Root
  TYPE(HCO_State), POINTER          :: HcoState    ! HEMCO state
```

INPUT/OUTPUT PARAMETERS:

```
  TYPE(ListCont),   POINTER         :: Lct
  INTEGER,          INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
22 Dec 2014 - C. Keller: Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.26.17 HCOIO_ReadFromConfig

Subroutine HCOIO_ReadFromConfig reads data directly from the configuration file (instead of reading it from a netCDF file). These data is always assumed to be spatially uniform, but it is possible to specify multiple time slices by separating the individual time slice values by the HEMCO separator sign ('/' by default). The time dimension of these data is either determined from the srcTime attribute or estimated from the number of time slices provided. For example, if no srcTime is specified and 24 time slices are provided, data is assumed to represent hourly data. Similarly, data is assumed to represent weekdaily or monthly data for 7 or 12 time slices, respectively.

If the srcTime attribute is defined, the time slices are determined from this attribute. Only one time dimension (year, month, day, or hour) can be defined for scalar fields!

INTERFACE:

```
SUBROUTINE HCOIO_ReadFromConfig ( am_I_Root, HcoState, Lct, RC )
```

USES:

```
  USE HCO_FILEDATA_MOD,   ONLY : FileData_ArrCheck
!INPUT PARAMTERS:
  LOGICAL,         INTENT(IN   )    :: am_I_Root
  TYPE(HCO_State), POINTER          :: HcoState    ! HEMCO state
```

INPUT/OUTPUT PARAMETERS:

```
    TYPE(ListCont),   POINTER         :: Lct
    INTEGER,          INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
   24 Jul 2014 - C. Keller: Initial version
   26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.26.18 HCOIO_CloseAll

Subroutine HCOIO_CloseAll makes sure that there is no open netCDF file left in the stream.

INTERFACE:

```
  SUBROUTINE HCOIO_CloseAll ( am_I_Root, HcoState, RC )
```

USES:

```
#if !defined(ESMF_)
    USE Ncdf_Mod,       ONLY : NC_CLOSE
#endif
  !INPUT PARAMTERS:
    LOGICAL,          INTENT(IN   )    :: am_I_Root
    TYPE(HCO_State), POINTER           :: HcoState    ! HEMCO state
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
   24 Mar 2016 - C. Keller: Initial version
```

1.26.19 GetSliceIdx

gets the time slice index to be used for data directly read from the HEMCO configuration file. prefDt denotes the preferred time attribute (year, month, or day). DtType is used to identify the time attribute type (1=year, 2=month, 3=day). The time slice index will be selected based upon those two variables. IDX is the selected time slice index. It will be set to -1 if the current simulation date is outside of the specified time range and the time cycle attribute is not enabled for this field.

INTERFACE:

```
  SUBROUTINE GetSliceIdx ( HcoState, Lct, DtType, prefDt, IDX, RC )
```

INPUT PARAMETERS:

```
    TYPE(HCO_State),  POINTER                :: HcoState
    TYPE(ListCont),   POINTER                :: Lct
    INTEGER,          INTENT(IN   )          :: DtType
    INTEGER,          INTENT(IN   )          :: prefDt
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)          :: IDX
    INTEGER,          INTENT(INOUT)          :: RC
```

REVISION HISTORY:

13 Mar 2013 - C. Keller - Initial version

1.26.20 GetDataVals

Subroutine GetDataVals extracts the data values from ValStr and writes them into vector Vals. ValStr is typically a character string read from an external ASCII file or directly from the HEMCO configuration file. Depending on the time specifications provided in the configuration file, Vals will be filled with only a subset of the values of ValStr.

INTERFACE:

```
SUBROUTINE GetDataVals ( am_I_Root, HcoState, Lct, ValStr, Vals, RC )
```

USES:

```
    USE HCO_CHARTOOLS_MOD,  ONLY : HCO_CharSplit
    USE HCO_EXTLIST_MOD,    ONLY : HCO_GetOpt
    USE HCO_UNIT_MOD,       ONLY : HCO_Unit_Change
    USE HCO_tIdx_Mod,       ONLY : HCO_GetPrefTimeAttr
!INPUT PARAMTERS:
    LOGICAL,          INTENT(IN   )   :: am_I_Root
    TYPE(HCO_State),  POINTER         :: HcoState    ! HEMCO state
    CHARACTER(LEN=*), INTENT(IN   )   :: ValStr
```

INPUT/OUTPUT PARAMETERS:

```
    TYPE(ListCont),   POINTER         :: Lct
    INTEGER,          INTENT(INOUT)   :: RC
```

OUTPUT PARAMETERS:

```
    REAL(hp),         POINTER         :: Vals(:)
```

REVISION HISTORY:

22 Dec 2014 - C. Keller: Initial version

1.26.21 FillMaskBox

Subroutine FillMaskBox fills the data array of the passed list container Lct according to the mask region provided in Vals. Vals contains the mask region of interest, denoted by the lower left and upper right grid box corners: lon1, lat1, lon2, lat2. The data array of Lct is filled such that all grid boxes are set to 1 whose mid-point is inside of the given box range.

INTERFACE:

```
SUBROUTINE FillMaskBox ( am_I_Root, HcoState, Lct, Vals, RC )
```

USES:

```
!INPUT PARAMTERS:
  LOGICAL,          INTENT(IN   )   :: am_I_Root
  TYPE(HCO_State),  POINTER         :: HcoState    ! HEMCO state
  REAL(hp)        , POINTER         :: Vals(:)
```

INPUT/OUTPUT PARAMETERS:

```
  TYPE(ListCont),   POINTER         :: Lct
  INTEGER,          INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
29 Dec 2014 - C. Keller - Initial version
19 Nov 2015 - C. Keller - Now support grid point masks
```

1.26.22 ReadMath

Subroutine ReadMath reads and evaluates a mathematical expression. Mathematical expressions can combine time-stamps with mathematical functions, e.g. to yield the sine of current simulation hour. Mathematical expressions must start with the identifier 'MATH:', followed by the actual expression. Each expression must include at least one variable (evaluated at runtime). The following variables are currently supported: YYYY (year), MM (month), DD (day), HH (hour), LH (local hour), NN (minute), SS (second), WD (weekday), LWD (local weekday), DOY (day of year). In addition, the following variables can be used: PI (3.141...), DOM (# of days of current month). For example, the following expression would yield a continuous sine curve as function of hour of day: 'MATH:sin(HH/24*PI*2)'.

For a full list of valid mathematical expressions, see module interpreter.F90.

INTERFACE:

```
SUBROUTINE ReadMath ( am_I_Root, HcoState, Lct, ValStr, Vals, N, RC )
```

USES:

```
  USE HCO_CLOCK_MOD,      ONLY : HcoClock_Get
  USE HCO_tIdx_Mod,       ONLY : HCO_GetPrefTimeAttr
  USE INTERPRETER
```

```
!INPUT PARAMTERS:
  LOGICAL,          INTENT(IN   )   :: am_I_Root
  TYPE(HCO_State),  POINTER         :: HcoState    ! HEMCO state
  TYPE(ListCont),   POINTER         :: Lct
  CHARACTER(LEN=*), INTENT(IN   )   :: ValStr
```

INPUT/OUTPUT PARAMETERS:

```
  REAL(hp),         INTENT(INOUT)   :: Vals(:)
  INTEGER,          INTENT(INOUT)   :: RC
```

OUTPUT PARAMETERS:

```
  INTEGER,          INTENT(  OUT)   :: N
```

REVISION HISTORY:

```
 11 May 2017 - C. Keller - Initial version
 07 Jul 2017 - C. Keller - Parse function before evaluation to allow
                           the usage of user-defined tokens within the
                           function.
```

1.27 Fortran: Module Interface hco_fluxarr_mod.F90

Module HCO_FluxArr_Mod contains routines to handle the HEMCO flux arrays. These are the emissions and deposition arrays listed in the HEMCO state object.

INTERFACE:

```
MODULE HCO_FluxArr_Mod
  USES:
  USE HCO_Error_Mod
  USE HCO_Arr_Mod
  USE HCO_Scale_Mod
  USE HCO_State_Mod, ONLY : HCO_State

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  PUBLIC  :: HCO_EmisAdd
  PUBLIC  :: HCO_DepvAdd
  PUBLIC  :: HCO_FluxarrReset
```

PRIVATE MEMBER FUNCTIONS:

```
  PRIVATE :: DiagnCheck
```

REMARKS:

REVISION HISTORY:

```
05 Jan 2014 - C. Keller - Initial version, adapted from hco_state_mod.F90
21 Oct 2014 - C. Keller - Added error check for negative values to HCO_EmisAdd
```

1.27.1 HCO_FluxarrReset

Routine HCO_FluxarrReset (re)sets all data arrays of the passed HEMCO state object. The (optional) argument Typ indicates whether only emissions (1), deposition (2), or concentration (3) arrays shall be reset. To reset all, set Typ to 0 (default).

INTERFACE:

```
SUBROUTINE HCO_FluxarrReset( HcoState, RC, Typ )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER                :: HcoState
INTEGER,         INTENT(INOUT)          :: RC
```

INPUT PARAMETERS:

```
INTEGER,         INTENT(IN   ), OPTIONAL :: Typ
```

REMARKS:

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
21 Aug 2014 - C. Keller - Added concentration
```

1.27.2 HCO_EmisAdd_3D_Dp

Routine HCO_EmisAdd_3D adds the 3D-array Arr3D to the emissions array of species HcoID in HEMCO object HcoState. This routine also updates all autofill diagnostics that are defined for the givne species, extension number, emission category and hierarchy.

INTERFACE:

```
SUBROUTINE HCO_EmisAdd_3D_Dp( am_I_Root, HcoState, Arr3D, HcoID, &
                              RC,        ExtNr,    Cat,   Hier,  &
                              MinDiagnLev )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER                :: HcoState
REAL(dp),        INTENT(INOUT)          :: Arr3D( HcoState%NX, &
                                                  HcoState%NY, &
                                                  HcoState%NZ )
INTEGER,         INTENT(INOUT)          :: RC
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )           :: am_I_Root
INTEGER,         INTENT(IN   )           :: HcoID
INTEGER,         INTENT(IN   ), OPTIONAL :: ExtNr
INTEGER,         INTENT(IN   ), OPTIONAL :: Cat
INTEGER,         INTENT(IN   ), OPTIONAL :: Hier
INTEGER,         INTENT(IN   ), OPTIONAL :: MinDiagnLev
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
20 Apr 2015 - C. Keller - Added DiagnCheck
12 May 2017 - C. Keller - Added option to use uniform scale factor
```

1.27.3 HCO_EmisAdd_3D_Sp

Routine HCO_EmisAdd_3D adds the 3D-array Arr3D to the emissions array of species HcoID in HEMCO object HcoState. This routine also updates all autofill diagnostics that are defined for the givne species, extension number, emission category and hierarchy.

INTERFACE:

```
SUBROUTINE HCO_EmisAdd_3D_Sp ( am_I_Root, HcoState, Arr3D, HcoID, &
                               RC,        ExtNr,    Cat,   Hier,  &
                               MinDiagnLev )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER                 :: HcoState
REAL(sp),        INTENT(INOUT)           :: Arr3D( HcoState%NX, &
                                                   HcoState%NY, &
                                                   HcoState%NZ )
INTEGER,         INTENT(INOUT)           :: RC
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )           :: am_I_Root
INTEGER,         INTENT(IN   )           :: HcoID
INTEGER,         INTENT(IN   ), OPTIONAL :: ExtNr
INTEGER,         INTENT(IN   ), OPTIONAL :: Cat
INTEGER,         INTENT(IN   ), OPTIONAL :: Hier
INTEGER,         INTENT(IN   ), OPTIONAL :: MinDiagnLev
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
20 Apr 2015 - C. Keller - Added DiagnCheck
12 May 2017 - C. Keller - Added option to use uniform scale factor
```

1.27.4 HCO_EmisAdd_2D_Dp

Routine HCO_EmisAdd_2D_Dp adds the real*8 2D-array Arr2D to the emission array of species HcoID in HEMCO object HcoState. This routine also updates all autofill diagnostics that are defined for the givne species, extension number, emission category and hierarchy.

INTERFACE:

```
SUBROUTINE HCO_EmisAdd_2D_Dp( am_I_Root, HcoState, Arr2D, HcoID, &
                              RC,        ExtNr,    Cat,   Hier,  &
                              MinDiagnLev )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER                :: HcoState
REAL(dp),        INTENT(INOUT)          :: Arr2D(HcoState%NX,HcoState%NY)
INTEGER,         INTENT(INOUT)          :: RC
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )          :: am_I_Root
INTEGER,         INTENT(IN   )          :: HcoID
INTEGER,         INTENT(IN   ), OPTIONAL :: ExtNr
INTEGER,         INTENT(IN   ), OPTIONAL :: Cat
INTEGER,         INTENT(IN   ), OPTIONAL :: Hier
INTEGER,         INTENT(IN   ), OPTIONAL :: MinDiagnLev
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
20 Apr 2015 - C. Keller - Added DiagnCheck
12 May 2017 - C. Keller - Added option to use uniform scale factor
```

1.27.5 HCO_EmisAdd_2D_Sp

Routine HCO_EmisAdd_2D_Sp adds the real*4 2D-array Arr2D to the emission array of species HcoID in HEMCO object HcoState. This routine also updates all autofill diagnostics that are defined for the givne species, extension number, emission category and hierarchy.

INTERFACE:

```
SUBROUTINE HCO_EmisAdd_2D_Sp( am_I_Root, HcoState, Arr2D, HcoID, &
                              RC,        ExtNr,    Cat,   Hier,  &
                              MinDiagnLev )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER                :: HcoState
REAL(sp),        INTENT(INOUT)          :: Arr2D(HcoState%NX,HcoState%NY)
INTEGER,         INTENT(INOUT)          :: RC
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )           :: am_I_Root
INTEGER,         INTENT(IN   )           :: HcoID
INTEGER,         INTENT(IN   ), OPTIONAL :: ExtNr
INTEGER,         INTENT(IN   ), OPTIONAL :: Cat
INTEGER,         INTENT(IN   ), OPTIONAL :: Hier
INTEGER,         INTENT(IN   ), OPTIONAL :: MinDiagnLev
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
20 Apr 2015 - C. Keller - Added DiagnCheck
12 May 2017 - C. Keller - Added option to use uniform scale factor
```

1.27.6 HCO_EmisAdd_Dp

Routine HCO_EmisAdd_Dp adds value iVal to the emission array of species HcoID in HEMCO object HcoState. The value is placed at location I, J, L of the array.

INTERFACE:

```
SUBROUTINE HCO_EmisAdd_Dp( HcoState, iVal, HcoID, I, J, L, RC )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState
REAL(dp),        INTENT(INOUT) :: iVal
INTEGER,         INTENT(INOUT) :: RC
```

INPUT PARAMETERS:

```
INTEGER,         INTENT(IN   ) :: HcoID
INTEGER,         INTENT(IN   ) :: I
INTEGER,         INTENT(IN   ) :: J
INTEGER,         INTENT(IN   ) :: L
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
12 May 2017 - C. Keller - Added option to use uniform scale factor
```

1.27.7 HCO_EmisAdd_Sp

Routine HCO_EmisAdd_Sp adds value iVal to the emission array of species HcoID in HEMCO object HcoState. The value is placed at location I, J, L of the array.

INTERFACE:

```
SUBROUTINE HCO_EmisAdd_Sp( HcoState, iVal, HcoID, I, J, L, RC )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState
REAL(sp),        INTENT(INOUT) :: iVal
INTEGER,         INTENT(INOUT) :: RC
```

INPUT PARAMETERS:

```
INTEGER,         INTENT(IN   ) :: HcoID
INTEGER,         INTENT(IN   ) :: I
INTEGER,         INTENT(IN   ) :: J
INTEGER,         INTENT(IN   ) :: L
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
12 May 2017 - C. Keller - Added option to use uniform scale factor
```

1.27.8 HCO_DepvAdd_2D_Dp

Routine HCO_DepvAdd_2D_Dp adds the real*8 2D-array Arr2D to the depostion array of species HcoID in HEMCO object HcoState.

INTERFACE:

```
SUBROUTINE HCO_DepvAdd_2D_Dp( HcoState, Arr2D, HcoID, RC )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState
INTEGER,         INTENT(INOUT) :: RC
```

INPUT PARAMETERS:

```
REAL(dp),        INTENT(IN   ) :: Arr2D(HcoState%NX,HcoState%NY)
INTEGER,         INTENT(IN   ) :: HcoID
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
```

1.27.9 HCO_DepvAdd_2D_Sp

Routine HCO_DepvAdd_2D_Sp adds the real*4 2D-array Arr2D to the depostion array of species HcoID in HEMCO object HcoState.

INTERFACE:

```
SUBROUTINE HCO_DepvAdd_2D_Sp( HcoState, Arr2D, HcoID, RC )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState
INTEGER,         INTENT(INOUT) :: RC
```

INPUT PARAMETERS:

```
REAL(sp),        INTENT(IN   ) :: Arr2D(HcoState%NX,HcoState%NY)
INTEGER,         INTENT(IN   ) :: HcoID
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
```

1.27.10 HCO_DepvAdd_Dp

Routine HCO_DepvAdd_Dp adds value iVal to the deposition array of species HcoID in HEMCO object HcoState. The value is placed at location I, J of the array.

INTERFACE:

```
SUBROUTINE HCO_DepvAdd_Dp( HcoState, iVal, HcoID, I, J, RC )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState
INTEGER,         INTENT(INOUT) :: RC
```

INPUT PARAMETERS:

```
REAL(dp),        INTENT(IN   ) :: iVal
INTEGER,         INTENT(IN   ) :: HcoID
INTEGER,         INTENT(IN   ) :: I
INTEGER,         INTENT(IN   ) :: J
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
```

1.27.11 HCO_DepvAdd_Sp

Routine HCO_DepvAdd_Sp adds value iVal to the deposition array of species HcoID in HEMCO object HcoState. The value is placed at location I, J of the array.

INTERFACE:

```
SUBROUTINE HCO_DepvAdd_Sp( HcoState, iVal, HcoID, I, J, RC )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState
INTEGER,         INTENT(INOUT) :: RC
```

INPUT PARAMETERS:

```
REAL(sp),        INTENT(IN   ) :: iVal
INTEGER,         INTENT(IN   ) :: HcoID
INTEGER,         INTENT(IN   ) :: I
INTEGER,         INTENT(IN   ) :: J
```

REVISION HISTORY:

```
01 May 2013 - C. Keller - Initial version
```

1.27.12 DiagnCheck

Subroutine DiagnCheck checks if the given emission array needs to be added to any auto-fill diagnostics. The diagnostics to be filled (if any) depend on the passed extension number, emission category and hierarchy, and the HEMCO species ID.

INTERFACE:

```
SUBROUTINE DiagnCheck( am_I_Root, HcoState, ExtNr,   Cat,     &
                       Hier,      HcoID,    Arr3D,   Arr3Dsp, &
                       Arr2D,     Arr2Dsp,  MinDiagnLev,  RC   )
```

USES:

```
USE HCO_DIAGN_MOD
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )           :: am_I_Root
INTEGER,         INTENT(IN   )           :: HcoID
INTEGER,         INTENT(IN   ), OPTIONAL :: ExtNr
INTEGER,         INTENT(IN   ), OPTIONAL :: Cat
INTEGER,         INTENT(IN   ), OPTIONAL :: Hier
INTEGER,         INTENT(IN   ), OPTIONAL :: MinDiagnLev
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER                 :: HcoState
REAL(dp),        INTENT(INOUT), OPTIONAL :: Arr3D(   HcoState%NX, &
                                                     HcoState%NY, &
                                                     HcoState%NZ )
REAL(sp),        INTENT(INOUT), OPTIONAL :: Arr3Dsp( HcoState%NX, &
                                                     HcoState%NY, &
                                                     HcoState%NZ )
REAL(dp),        INTENT(INOUT), OPTIONAL :: Arr2D(   HcoState%NX, &
                                                     HcoState%NY )
REAL(sp),        INTENT(INOUT), OPTIONAL :: Arr2Dsp( HcoState%NX, &
                                                     HcoState%NY )
INTEGER,         INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
20 Apr 2015 - C. Keller - Initial version
```

1.28 Fortran: Module Interface hco_config_mod.F90

Module HCO_Config_Mod contains routines related to the HEMCO configuration file. It reads the content of the configuration file, checks which entires therein are actually used for this simulation run, and stores these information. This occurs in two calls: Config_ReadFile and SetReadList. Config_ReadFile writes the entire content of the configuration file into buffer except for the input data associated with a disabled extension. SetReadList does many more logical checks and adds all data used by HEMCO to ReadList. Scale factors not used by any of the base emissions and base emission fields (e.g. scale factors that won't be used) are removed in this step.

All data fields are saved in individual data containers, which are organized in the ConfigList. Hence, ConfigList is a collection of all HEMCO data containers, with every container representing an entry of the configuration file. Each data container has its unique container ID for identification. All HEMCO lists (ConfigList, ReadList, EmisList) access the same containers.

The configuration file provides all source file information of the emission fields and scale factors to be used. It must be read at the beginning of a simulation run.

As of HEMCO v2.0, the ConfigList linked list sits within the HEMCO configuration object (HcoConfig). HcoConfig must be passed to all routines. This allows the parallel usage of multiple invocations of HEMCO that use different input data. HcoConfig is initialized upon reading the HEMCO configuration file (within subroutine Config_ReadFile).

INTERFACE:

```
MODULE HCO_Config_Mod
```

USES:

```
  USE HCO_ERROR_MOD
  USE HCO_DIAGN_MOD
  USE HCO_CHARTOOLS_MOD
  USE HCO_TYPES_MOD
  USE HCO_STATE_MOD,          ONLY : HCO_State

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  PUBLIC  :: SetReadList
  PUBLIC  :: Config_ReadFile
  PUBLIC  :: Config_GetnSpecies
  PUBLIC  :: Config_GetSpecNames
  !PRIVATE:
  PRIVATE :: ReadSettings
  PRIVATE :: ExtSwitch2Buffer
  PRIVATE :: ConfigList_AddCont
```

```
PRIVATE :: Config_ReadCont
PRIVATE :: RegisterPrepare
PRIVATE :: Get_targetID
PRIVATE :: Calc_Coverage
PRIVATE :: Register_Base
PRIVATE :: Register_Scal
PRIVATE :: ReadAndSplit_Line
PRIVATE :: Config_GetSpecAttr
PRIVATE :: BracketCheck
PRIVATE :: AddZeroScal
PRIVATE :: AddShadowFields
PRIVATE :: ConfigInit
PRIVATE :: ParseEmisL
```

REVISION HISTORY:

```
18 Jun 2013 - C. Keller   -  Initialization
08 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
08 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
15 Feb 2015 - C. Keller   - Added BracketCheck, AddZeroScal, AddShadowFields
15 Feb 2016 - C. Keller   - Update to v2.0: ConfigList now sits in HcoConfig
```

1.28.1 Config_Readfile

Subroutine CONFIG_READFILE reads the HEMCO configuration file, archives all HEMCO options and settings (including traceback/error setup), and creates a data container for every (used) emission field in the config. file. All containers become linked through the ConfigList linked list. Note that lists EmisList and ReadList (created lateron) will point to the same containers, but will order the containers in a manner that is most efficient for the respective purpose. Argument HcoConfig represents the HEMCO configuration object. It contains pointers to the HEMCO traceback and error information as well as a pointer to ConfigList. If undefined, HcoConfig becomes initialized as part of this routine.

INTERFACE:

```
SUBROUTINE Config_ReadFile( am_I_Root, HcoConfig, ConfigFile, Phase, RC, IsNest )
```

USES:

```
USE inquireMod,        ONLY : findFreeLUN
USE CharPak_Mod,       ONLY : STRREPL
USE HCO_EXTLIST_MOD,   ONLY : AddExt, CoreNr, ExtNrInUse
```

INPUT PARAMETERS:

```
LOGICAL,            INTENT(IN)              :: am_I_Root  ! root CPU?
TYPE(ConfigObj),    POINTER                 :: HcoConfig  ! HEMCO config obj
CHARACTER(LEN=*),   INTENT(IN)              :: ConfigFile ! Full file name
INTEGER,            INTENT(IN)              :: Phase      ! 0: all
```

```
                                                          ! 1: Settings and switches on
                                                          ! 2: fields only
    LOGICAL,            INTENT(IN   ), OPTIONAL :: IsNest     ! Nested call?
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,            INTENT(INOUT)           :: RC         ! Success?
```

REVISION HISTORY:

```
    17 Sep 2012 - C. Keller   - Initialization
    03 Jan 2014 - C. Keller   - Now use Config_ReadCont calls.
    30 Sep 2014 - R. Yantosca - Now declare LINE w/ 2047 characters.  This lets
                                us handle extra-long species lists
    13 Feb 2015 - C. Keller   - Removed section extension data: these are now
                                listed in section base emissions.
    11 Dec 2015 - C. Keller   - Read settings and extension switches even for
                                nested configuration files.
    15 Feb 2016 - C. Keller   - Now pass HcoConfig argument.
```

1.28.2 SetReadList

Subroutine SetReadList writes data to the data reading lists (ReadList). This routine assumes that the configuration file has been read beforehand (via Config_ReadFile).

INTERFACE:

```
  SUBROUTINE SetReadList( am_I_Root, HcoState, RC )
```

USES:

```
    USE HCO_DATACONT_Mod,    ONLY : cIDList_Create
    USE HCO_READLIST_Mod,    ONLY : ReadList_Init
```

INPUT PARAMETERS:

```
    LOGICAL,         INTENT(IN)    :: am_I_Root   ! root CPU?
    TYPE(HCO_State), POINTER       :: HcoState    ! HEMCO state
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,         INTENT(INOUT) :: RC          ! Error stat
```

REVISION HISTORY:

```
    18 Jun 2013 - C. Keller: Initialization
    17 Sep 2013 - C. Keller: Now get data from buffer
```

1.28.3 Config_ReadCont

Subroutine CONFIG_READCONT reads the given line into a list container. Depending on the specified data type, the line is assumed to hold base emissions, scale factors, or mask information.

INTERFACE:

```
SUBROUTINE Config_ReadCont( am_I_Root, HcoConfig, IU_HCO,   &
                            CFDIR,     DctType,   EOF,    RC )
```

USES:

```
USE HCO_EXTLIST_MOD,  ONLY : ExtNrInUse, HCO_GetOpt
USE HCO_TIDX_Mod,     ONLY : HCO_ExtractTime
USE HCO_FILEDATA_Mod, ONLY : FileData_Init
USE HCO_DATACONT_Mod, ONLY : CatMax, ZeroScalID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root ! Root CPU?
TYPE(ConfigObj),  POINTER       :: HcoConfig ! Config object
INTEGER,          INTENT(IN   ) :: IU_HCO    ! Logfile LUN
CHARACTER(LEN=*), INTENT(IN   ) :: CFDIR     ! Configuration file directory
INTEGER,          INTENT(IN   ) :: DctType   ! 1=base; 2=scale; 3=mask
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(INOUT) :: EOF       ! end of file encountered?
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT) :: RC        ! error code
```

REVISION HISTORY:

```
03 Jan 2014 - C. Keller - Initial version
29 Dec 2014 - C. Keller - Added optional 11th element for scale factors. This
                          value will be interpreted as mask field (applied to
                          this scale factor only).
27 Feb 2015 - C. Keller - Added CycleFlag 'I' (interpolation)
13 Mar 2015 - C. Keller - Added include files (nested configuration files)
                          and CFDIR argument.
23 Sep 2015 - C. Keller - Added cycle flags 'A' and 'RA' (for averaging).
06 Oct 2015 - C. Keller - Added cycle flags 'EF' and 'RF' (fields must be
                          found).
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.4 BracketCheck

Subroutine BracketCheck checks if base emission data is within a bracket and if that field shall be ignored or not. Brackets can be used to lump entires of the HEMCO configuration file into collections that can be collectively enabled or disabled. The first entry of a collection is marked adding an 'opening bracket' to the HEMCO configuration file (on the line above the entry). Opening brackets must start with three opening brackets, e.g.: '(((TEST'. Similarly, the end of a collection is marked by placing a closing bracket after the last entry of the collection: '))))TEST'. Brackets can be enabled / disabled in the EXTENSION SWITCH section of the HEMCO configuration file: # ExtNr ExtName on/off Species 0 Base : on * –¿ TEST : true

It is also possible to use 'opposite' brackets, e.g. to use a collection only if the given setting is *disabled*. This can be achieved by precede the collection word with '.not.', e.g. '(((.not.TEST' and ''))).not.TEST'. Similarly, multiple collections can be combined to be evaluated together, e.g. NAME1.or.NAME2.

INTERFACE:

```
SUBROUTINE BracketCheck( am_I_Root, HcoConfig, STAT, LINE, SKIP, RC )
```

USES:

```
USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt, GetExtNr
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)    :: am_I_Root   ! root CPU?
INTEGER,          INTENT(IN)    :: STAT        !
CHARACTER(LEN=*), INTENT(IN)    :: LINE        !
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER       :: HcoConfig ! Config object
LOGICAL,          INTENT(INOUT) :: SKIP        ! Skip
INTEGER,          INTENT(INOUT) :: RC          ! Success/failure
```

REVISION HISTORY:

```
15 Feb 2015 - C. Keller   - Initial version.
12 Mar 2015 - C. Keller   - Added 'mirror' option.
```

1.28.5 AddShadowFields

Subroutine AddShadowFields adds a shadow container for every additional category of a base emission field. These container contain the same container as the 'mother' container, but an additional scale factor of zero will be applied to them. This makes sure that no additional emissions are created by the virtue of the shadow container.

INTERFACE:

```
SUBROUTINE AddShadowFields( am_I_Root, HcoConfig, Lct, Cats, nCat, RC )
```

USES:

```
USE HCO_DATACONT_MOD,  ONLY : CatMax, ZeroScalID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)     :: am_I_Root    ! root CPU?
TYPE(ConfigObj), POINTER         :: HcoConfig    ! Config object
TYPE(ListCont),  POINTER         :: Lct          ! List container of interest
INTEGER,          INTENT(IN)     :: Cats(CatMax) ! Category numbers
INTEGER,          INTENT(IN)     :: nCat         ! number of categories
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)  :: RC           ! Success/failure
```

REVISION HISTORY:

```
15 Feb 2015 - C. Keller   - Initial version.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.6 AddZeroScal

Subroutine AddZeroScal adds a scale factor of zero to the configuration container list. This scale factor is an internal scale factor used in combination with the 'shadow' containers. Its scale factor ID is defined in Hco_DataCont_Mod and must not be used otherwise, e.g. there must not be another scale factor in the HEMCO configuration file with the same scale factor ID. Otherwise, HEMCO will create an error lateron.

INTERFACE:

```
SUBROUTINE AddZeroScal( am_I_Root, HcoConfig, RC )
```

USES:

```
USE HCO_DATACONT_MOD,  ONLY : ZeroScalID
USE HCO_DATACONT_MOD,  ONLY : ListCont_Find
USE HCO_FILEDATA_MOD,  ONLY : FileData_Init
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN)            :: am_I_Root   ! root CPU?
TYPE(ConfigObj), POINTER       :: HcoConfig   ! Config object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT)         :: RC          ! Success/failure
```

REVISION HISTORY:

```
15 Feb 2015 - C. Keller   - Initial version.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.7 ExtSwitch2Buffer

Subroutine ExtSwitch2Buffer reads the HEMCO extension switches and registers all enabled extensions.

INTERFACE:

```
SUBROUTINE ExtSwitch2Buffer( am_I_Root, HcoConfig, IU_HCO, EOF, RC )
```

USES:

```
USE CHARPAK_Mod,        ONLY : STRREPL, STRSPLIT, TRANLC
USE HCO_EXTLIST_MOD,    ONLY : AddExt, AddExtOpt, HCO_GetOpt
USE HCO_EXTLIST_MOD,    ONLY : GetExtNr
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN)    :: am_I_Root   ! root CPU?
TYPE(ConfigObj), POINTER       :: HcoConfig   ! Config object
INTEGER,         INTENT(IN)    :: IU_HCO      ! HEMCO configfile LUN
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,         INTENT(INOUT) :: EOF         ! End of file?
INTEGER,         INTENT(INOUT) :: RC          ! Success/failure
```

REVISION HISTORY:

```
17 Sep 2013 - C. Keller   - Initialization (update)
30 Sep 2014 - R. Yantosca - Declare SUBSTR and SPECS w/ 2047 characters,
                            which lets us handle extra-long species lists
21 Apr 2015 - R. Yantosca - Bug fix: now look for END_SECTION before
                            testing if the line is a comment.  This will
                            allow for tags labeled "### END SECTION".
12 Dec 2015 - C. Keller   - Added argument IgnoreIfExist to AddExtOpt to
                            make sure that nested configuration files do
                            use the settings set at highest level.
```

1.28.8 ReadSettings

Subroutine ReadSettings reads the HEMCO settings, stores them as HEMCO core extension options, and also evaluates some of the values (e.g. to initialize the HEMCO error module).

INTERFACE:

```
SUBROUTINE ReadSettings( am_I_Root, HcoConfig, IU_HCO, EOF, RC )
```

USES:

```
USE HCO_EXTLIST_MOD,    ONLY : AddExtOpt, GetExtOpt, CoreNr
USE HCO_EXTLIST_MOD,    ONLY : HCO_SetDefaultToken
USE HCO_EXTLIST_MOD,    ONLY : HCO_GetOpt
USE CHARPAK_MOD,        ONLY : STRREPL, STRSPLIT, TRANLC
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN)    :: am_I_Root   ! root CPU?
TYPE(ConfigObj), POINTER       :: HcoConfig   ! Config obj
INTEGER,         INTENT(IN)    :: IU_HCO      ! HEMCO configfile LUN
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,         INTENT(INOUT) :: EOF         ! End of file?
INTEGER,         INTENT(INOUT) :: RC          ! Success/failure
```

REVISION HISTORY:

```
17 Sep 2013 - C. Keller   - Initialization (update)
21 Apr 2015 - R. Yantosca - Bug fix: now look for END_SECTION before
                            testing if the line is a comment.  This will
                            allow for tags labeled "### END SECTION".
12 Dec 2015 - C. Keller   - Added argument IgnoreIfExist to AddExtOpt to
                            make sure that nested configuration files do
                            use the settings set at highest level.
```

1.28.9 RegisterPrepare

Subroutine RegisterPrepare extracts the spatial coverages of all mask fields as well as the HEMCO species IDs of all base emissions.

The species IDs are determined by matching the species name read from the configuration file (in ConfigList) and the species names defined in the HEMCO state object HcoState.

Mask coverages are defined based upon the passed horizontal grid extensions on this CPU (xrng and yrng).

INTERFACE:

```
SUBROUTINE RegisterPrepare( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_EXTLIST_MOD,  ONLY : ExtNrInUse
USE HCO_STATE_Mod,    ONLY : HCO_GetHcoID
USE HCO_DATACONT_MOD, ONLY : ListCont_NextCont
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root  ! Root CPU
TYPE(HCO_State),  POINTER       :: HcoState   ! HEMCO state obj.
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
18 Sep 2013 - C. Keller - Initial version (update)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.10 Register_Base

Subroutine Register_Base registers all base emission data and writes out all associated scale factor IDs.

INTERFACE:

```
SUBROUTINE Register_Base ( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_READLIST_Mod,      ONLY : ReadList_Set
USE HCO_DATACONT_Mod,      ONLY : DataCont_Cleanup
USE HCO_DATACONT_MOD,      ONLY : ListCont_NextCont
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)    :: am_I_Root   ! Are we on the root CPU?
TYPE(HCO_State),  POINTER       :: HcoState    ! HEMCO state object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC          ! Success or failure
```

REVISION HISTORY:

```
18 Jun 2013 - C. Keller: Initialization
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.11 Register_Scal

Subroutine Register_Scal registers all scale factors.

INTERFACE:

```
SUBROUTINE Register_Scal( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_ReadList_Mod,      ONLY : ReadList_Set
USE HCO_DATACONT_MOD,      ONLY : ListCont_NextCont
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)    :: am_I_Root   ! Are we on root CPU?
TYPE(HCO_State),  POINTER       :: HcoState    ! HEMCO state object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC          ! Success or failure
```

REVISION HISTORY:

```
18 Jun 2013 - C. Keller - Initialization
29 Dec 2014 - C. Keller - Now check for masks assigned to scale factors.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.12 Get_targetID

Subroutine Get_targetID returns the target ID of a container. The target ID can point to the container ID (cID) of another base field if multiple emissions shall be added together prior to emission calculation, e.g. sectoral emissions data with same species ID, category, hierarchy, extension number, scale factors, etc.

Target ID is set to -999 if there exists another inventory over the full spatial region covered by this CPU for this species but with higher hierarchy. In this case, we can ignore the current container from here onwards!

INTERFACE:

```
SUBROUTINE Get_targetID( am_I_Root, HcoState, Lct, targetID, RC )
```

USES:

```
USE HCO_DataCont_Mod, ONLY : ListCont_Find
USE HCO_DataCont_Mod, ONLY : ListCont_NextCont
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root
TYPE(HCO_State), POINTER       :: HcoState
TYPE(ListCont),  POINTER       :: Lct
```

OUTPUT PARAMETERS:

```
INTEGER,         INTENT(  OUT) :: targetID
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC
!NOTE: If data from multiple containers are added, the target ID
is always set to the lowest cID of all involved containers, i.e.
data are added to the container with the lowest cID. This makes
sure that data is not accidentally overwritten, e.g. when updating
container contents!
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller - Initialization
07 Dec 2015 - C. Keller - Make sure emissions with limited time range do
                          never erase lower hierarchy base emissions.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.13 Calc_Coverage

Function Calc_Coverage calculates the coverage of the specified lon/lat box with the area covered by the inventory. Returns 0 if no overlap, 1 if complete overlap, and -1 for partial overlap.

INTERFACE:

```
FUNCTION Calc_Coverage( msk_x1, msk_x2, msk_y1, msk_y2,  &
                        cpu_x1, cpu_x2, cpu_y1, cpu_y2 ) RESULT ( COVERAGE )
```

INPUT PARAMETERS:

```
INTEGER, INTENT(IN) :: msk_x1
INTEGER, INTENT(IN) :: msk_x2
INTEGER, INTENT(IN) :: msk_y1
INTEGER, INTENT(IN) :: msk_y2
INTEGER, INTENT(IN) :: cpu_x1
INTEGER, INTENT(IN) :: cpu_x2
INTEGER, INTENT(IN) :: cpu_y1
INTEGER, INTENT(IN) :: cpu_y2
```

RETURN VALUE:

```
INTEGER             :: COVERAGE
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Initialization
```

1.28.14 ReadAndSplit_Line

Subroutine ReadAndSplit_Line reads a line from the HEMCO config file and parses the specified columns into the passed integer and character variables. If the optional argument inLine is provided, this line will be parsed, otherwise a new line will be read from the config file. If the optional argument outLine is provided, this variable will hold the parsed line.

This routine splits the input line (or the next line of an open file with ID IU_HCO), using the HEMCO separator (default: space) as separator. The resulting elements are then passed to the specified output characters and integers. For example, to pass the 5th element of a line to variable int1, set int1cl to 5, etc. An error will be returned (STAT=100) if any of the output columns exceeds the number of line elements. The optional argument optcl can be used to denote an optional value, e.g. no error is returned if the value at position optcl cannot be read. Only one optional value can be specified.

INTERFACE:

```
SUBROUTINE ReadAndSplit_Line( AIR,     HcoConfig,              &
                                       IU_HCO,  char1, chr1cl, &
                              char2,   chr2cl,  char3, chr3cl, &
```

```
                                char4,   chr4cl,  char5, chr5cl, &
                                char6,   chr6cl,  char7, chr7cl, &
                                char8,   chr8cl,  char9, chr9cl, &
                                char10,  chr10cl,                &
                                int1,    int1cl,  int2,  int2cl, &
                                int3,    int3cl,  STAT,  inLine, &
                                outLine, optcl                    )
```

USES:

```
USE CHARPAK_Mod,        ONLY : STRREPL, STRSPLIT
```

INPUT PARAMETERS:

```
LOGICAL,             INTENT(IN   )           :: AIR
TYPE(ConfigObj),     POINTER                 :: HcoConfig
INTEGER,             INTENT(IN   )           :: IU_HCO
INTEGER,             INTENT(IN   )           :: chr1cl
INTEGER,             INTENT(IN   )           :: chr2cl
INTEGER,             INTENT(IN   )           :: chr3cl
INTEGER,             INTENT(IN   )           :: chr4cl
INTEGER,             INTENT(IN   )           :: chr5cl
INTEGER,             INTENT(IN   )           :: chr6cl
INTEGER,             INTENT(IN   )           :: chr7cl
INTEGER,             INTENT(IN   )           :: chr8cl
INTEGER,             INTENT(IN   )           :: chr9cl
INTEGER,             INTENT(IN   )           :: chr10cl
INTEGER,             INTENT(IN   )           :: int1cl
INTEGER,             INTENT(IN   )           :: int2cl
INTEGER,             INTENT(IN   )           :: int3cl
CHARACTER(LEN=255), INTENT(IN   ), OPTIONAL :: inLINE
INTEGER,             INTENT(IN   ), OPTIONAL :: optcl
```

OUTPUT PARAMETERS:

```
CHARACTER(LEN=*),   INTENT(INOUT)           :: char1
CHARACTER(LEN=*),   INTENT(INOUT)           :: char2
CHARACTER(LEN=*),   INTENT(INOUT)           :: char3
CHARACTER(LEN=*),   INTENT(INOUT)           :: char4
CHARACTER(LEN=*),   INTENT(INOUT)           :: char5
CHARACTER(LEN=*),   INTENT(INOUT)           :: char6
CHARACTER(LEN=*),   INTENT(INOUT)           :: char7
CHARACTER(LEN=*),   INTENT(INOUT)           :: char8
CHARACTER(LEN=*),   INTENT(INOUT)           :: char9
CHARACTER(LEN=*),   INTENT(INOUT)           :: char10
INTEGER,            INTENT(INOUT)           :: int1
INTEGER,            INTENT(INOUT)           :: int2
INTEGER,            INTENT(INOUT)           :: int3
CHARACTER(LEN=255), INTENT(  OUT), OPTIONAL :: outLINE
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,            INTENT(INOUT)           :: STAT
```

REVISION HISTORY:

```
28 Aug 2013 - C. Keller - Initial version
11 Dec 2013 - C. Keller - Added optional arguments inLine and outLine
29 Dec 2014 - C. Keller - Added optional argument optcl. Now use wrapper
                          routines READCHAR and READINT.
13 Mar 2015 - C. Keller - Added check for include files.
```

1.28.15 READCHAR

Subroutine READCHAR is a helper routine to read character values from the HEMCO configuration file.

INTERFACE:

```
SUBROUTINE READCHAR ( LINE, SUBSTR, N, chrcl, charout, OPT, STAT )
```

INPUT PARAMETERS:

```
CHARACTER(LEN=255), INTENT(IN   )    :: LINE
CHARACTER(LEN=255), INTENT(IN   )    :: SUBSTR(255)
INTEGER,            INTENT(IN   )    :: N
INTEGER,            INTENT(IN   )    :: chrcl
INTEGER,            INTENT(IN   )    :: OPT
```

INPUT/OUTPUT PARAMETERS:

```
CHARACTER(LEN=*),   INTENT(INOUT)    :: charout
INTEGER,            INTENT(INOUT)    :: STAT
```

REVISION HISTORY:

```
29 Dec 2014 - C. Keller   - Initial version
```

1.28.16 READINT

Subroutine READINT is a helper routine to read integer values from the HEMCO configuration file.

INTERFACE:

```
SUBROUTINE READINT ( ExtList, LINE, SUBSTR, N, intcl, intout, OPT, STAT )
```

USES:

```
USE HCO_EXTLIST_MOD, ONLY : HCO_GetOpt
```

INPUT PARAMETERS:

```
TYPE(Ext),          POINTER          :: ExtList
CHARACTER(LEN=255), INTENT(IN   )    :: LINE
CHARACTER(LEN=255), INTENT(IN   )    :: SUBSTR(255)
INTEGER,            INTENT(IN   )    :: N
INTEGER,            INTENT(IN   )    :: intcl
INTEGER,            INTENT(IN   )    :: OPT
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,            INTENT(INOUT)    :: intout
INTEGER,            INTENT(INOUT)    :: STAT
```

REVISION HISTORY:

```
29 Dec 2014 - C. Keller   - Initial version
```

1.28.17 Get_cID

Subroutine Get_cID searches the whole ConfigList for an entry with the given ScalID and returns the corresponding container ID cID.

INTERFACE:

```
SUBROUTINE Get_cID( ScalID, HcoConfig, cID, RC )
```

INPUT PARAMETERS:

```
INTEGER, INTENT(IN   )   :: scalID
TYPE(ConfigObj), POINTER :: HcoConfig
```

OUTPUT PARAMETERS:

```
INTEGER, INTENT(  OUT)   :: cID
!INPUT/OUTPUTP PARAMETERS:
INTEGER, INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
18 Sep 2013 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.18 ConfigList_AddCont

Subroutine ConfigList_AddCont adds a new (blank) container to the ConfigList list.

INTERFACE:

```
SUBROUTINE ConfigList_AddCont( Lct, List )
```

USES:

```
USE HCO_DATACONT_Mod, ONLY : DataCont_Init
USE HCO_DATACONT_Mod, ONLY : ListCont_Length
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ListCont), POINTER       :: Lct
TYPE(ListCont), POINTER       :: List
```

REVISION HISTORY:

```
17 Sep 2013 - C. Keller: Initialization (update)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.19 ScalID_Register

Subroutine ScalID_Register adds the scale factor IDs ScalIDs to the list of scale factor IDs.

INTERFACE:

```
SUBROUTINE ScalID_Register( Dct, HcoConfig, RC )
```

INPUT PARAMETERS:

```
TYPE(DataCont),  POINTER :: Dct
TYPE(ConfigObj), POINTER :: HcoConfig
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,  INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
29 Dec 2014 - C. Keller: Now add new container to end of list to allow
                         list being updated while calling Register_Scal.
```

1.28.20 ScalID2List

Subroutine ScalID2List adds the scale factor IDs ScalIDs to the list of scale factor IDs.

INTERFACE:

```
SUBROUTINE ScalID2List( ScalIDList, ID, RC )
```

INPUT PARAMETERS:

```
TYPE(ScalIDCont),   POINTER        :: ScalIDList
INTEGER,            INTENT(IN   )  :: ID
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,            INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
29 Dec 2014 - C. Keller: Now add new container to end of list to allow
                         list being updated while calling Register_Scal.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.21 ScalID_Cleanup

Subroutine ScalID_Cleanup cleans up the internal ScalID list.

INTERFACE:

```
SUBROUTINE ScalID_Cleanup( ScalIDList )
!INPUT ARGUMENTS:
TYPE(ScalIDCont),   POINTER  :: ScalIDList
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.22 SpecName_Register

Subroutine SpecName_Register adds the species name SpecName to the list of species names.

INTERFACE:

```
SUBROUTINE SpecName_Register( HcoConfig, SpecName, RC )
```

USES:

```
USE HCO_EXTLIST_MOD, ONLY : HCO_GetOpt
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),    POINTER       :: HcoConfig
CHARACTER(LEN=*),   INTENT(IN   ) :: SpecName
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.23 SpecName_Cleanup

Subroutine SpecName_Cleanup cleans up the internal SpecName list.

INTERFACE:

```
SUBROUTINE SpecName_Cleanup ( SpecNameList )
!INPUT/OUTPUT ARGUMENT:
  TYPE(SpecNameCont), POINTER       :: SpecNameList
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.24 Config_GetnSpecies

Function Config_GetnSpecies is a wrapper function to get the number of (unique) species names in SpecNameList.

INTERFACE:

```
FUNCTION Config_GetnSpecies( HcoConfig ) RESULT( nSpecies )
!INPUT ARGUMENT:
  TYPE(ConfigObj), POINTER       :: HcoConfig
```

RETURN VALUE:

```
  INTEGER :: nSpecies
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
```

1.28.25 Config_GetSpecNames

Subroutine Config_GetSpecNames is a wrapper routine to obtain the list of (unique) species names defined in SpecNameList.

INTERFACE:

```
SUBROUTINE Config_GetSpecNames( HcoConfig, SpecNames, nSpecies, RC )
!INPUT ARGUMENT:
  TYPE(ConfigObj), POINTER          :: HcoConfig
!OUTPUT PARAMTERS:
  CHARACTER(LEN=*), POINTER         :: SpecNames(:)
```

INPUT/OUTPUT PARAMETERS:

```
  INTEGER,          INTENT(INOUT)   :: nSpecies
  INTEGER,          INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
```

1.28.26 Config_getSpecAttr

Subroutine Config_GetSpecAttr returns the number of species names N and the vector of species names SpecNames. SpecNames must be of length nnSpecs, i.e. in order to obtain SpecNames, Config_getSpecAttr has to be called twice: N = 0 CALL Config_getSpecAttr (N=N, RC=RC) ALLOCATE(SpecNames(N)) CALL Config_getSpecAttr (N=N, SpecNames=SpecNames, RC=RC)

INTERFACE:

```
SUBROUTINE Config_GetSpecAttr( HcoConfig, N, SpecNames, RC )
!INPUT ARGUMENT:
  TYPE(ConfigObj),    POINTER                    :: HcoConfig
```

INPUT/OUTPUT PARAMETERS:

```
  INTEGER,            INTENT(INOUT)              :: N
  INTEGER,            INTENT(INOUT)              :: RC
```

OUTPUT PARAMETERS:

```
  CHARACTER(LEN=*),   POINTER,       OPTIONAL    :: SpecNames(:)
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

1.28.27 Check_ContNames

Function Check_Contnames compares the container names of two containers, ignoring the name 'tags', i.e. ignoring everything that follows double underscore (__). For example, two containers with names "EDGAR_NOX__PNT" and "EDGAR_NOX__MOB" are considered equal, while "EDGAR_NOX_PNT" and "EDGAR_NOX_MOB" are not.

INTERFACE:

```
FUNCTION Check_ContNames( Lct1, Lct2 ) RESULT( SameName )
```

INPUT/OUTPUT PARAMETERS:

```
  TYPE(ListCont), POINTER :: Lct1
  TYPE(ListCont), POINTER :: Lct2
```

RETURN VALUE:

```
  LOGICAL                 :: SameName
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
```

1.28.28 ExtractSrcDim

Subroutine ExtractSrcDim extracts the source dimension attribute. Specifically, it checks if the field is expected to be 2D (xy) or 3D. Default 3D data is xyz, but it is also possible to explicitly define the number of vertical levels to be read, as well as the reading direction (up or down). For example, 'xy1' will be interpreted as reading only the first level, and 'xy27' will only read the first 27 levels. To reverse the vertical axis, use e.g. 'xy-1' to read only the top level, or 'xy-27' to read the top 27 levels, with the topmost level being put into the surface level.

INTERFACE:

```
SUBROUTINE ExtractSrcDim( am_I_Root, HcoConfig, SrcDim, Dta, Lscal1, Lscal2, RC )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )   :: am_I_Root
TYPE(ConfigObj),  POINTER         :: HcoConfig
CHARACTER(LEN=*), INTENT(IN   )   :: SrcDim
TYPE(FileData),   POINTER         :: Dta
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT)   :: Lscal1
INTEGER,          INTENT(  OUT)   :: Lscal2
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
20 May 2015 - C. Keller   - Initial version
22 Jan 2016 - R. Yantosca - Bug fix, removed & in the middle of the line
                            since the PGI compiler chokes on it.
26 Jan 2018 - C. Keller   - Add L1 & L2
```

1.28.29 ConfigInit

Subroutine ConfigInit is a wrapper routine to initialize the HEMCO configuration object.

INTERFACE:

```
SUBROUTINE ConfigInit ( HcoConfig )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ConfigObj), POINTER :: HcoConfig
```

REVISION HISTORY:

```
16 Feb 2016 - C. Keller: Initialization (update)
```

1.28.30 ParseEmisL

parses the emission level.

INTERFACE:

```
SUBROUTINE ParseEmisL ( str, EmisL, EmisUnit, ScalID )
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*),  INTENT(IN ) :: str
```

INPUT/OUTPUT PARAMETERS:

```
REAL(hp),          INTENT(OUT) :: EmisL
INTEGER,           INTENT(OUT) :: EmisUnit
INTEGER,           INTENT(OUT) :: ScalID
```

REVISION HISTORY:

```
09 May 2016 - C. Keller: Intial version.
```

1.29 Fortran: Module Interface hco_error_mod.F90

Module HCO_Error_Mod contains routines and variables for error handling and logfile messages in HEMCO. It also contains definitions of some globally used parameter, such as the single/double precision as well as the HEMCO precision definitions. The HEMCO precision is used for almost all HEMCO internal data arrays and can be changed below if required.

The error settings are specified in the HEMCO configuration file and error handling is performed according to these settings. They include:

1. HEMCO logfile: all HEMCO information is written into the specified logfile. The logfile can be set to the wildcard character, in which case the standard output will be used (this may be another opened logfile).
2. Verbose: Number indicating the verbose level to be used. 0 = no verbose, 3 = very verbose. The verbose level can be set in the HEMCO configuration file. The default value is 0.
3. Warnings: Number indicating the warning level to be shown. 0 = no warnings, 3 = all warnings.

The error settings are set via subroutine HCO_ERROR_SET, called when reading section 'settings' of the HEMCO configuration file (subroutine Config_ReadFile in hco_config_mod.F90). The currently active verbose settings can be checked using subroutines HCO_IsVerb and HCO_VERBOSE_INQ. Messages can be written into the logfile using subroutine HCO_MSG. Note that the logfile actively need to be opened (HCO_LOGFILE_OPEN) before writing to it.

The verbose and warning settings are all set to false if it's not the root CPU.

As of HEMCO v2.0, all HEMCO error variables are organized in derived type object HcoErr. HcoErr is a component of the HEMCO configuration object (type ConfigObj, see hco_types_mod.F90). It must be passed explicitly to all error routines. This design allows the invocation of multiple independent HEMCO instances at the same time (which may have different HEMCO error settings). **INTERFACE:**

```
MODULE HCO_Error_Mod
```

USES:

```
  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  PUBLIC           :: HCO_ERROR
  PUBLIC           :: HCO_WARNING
  PUBLIC           :: HCO_MSG
  PUBLIC           :: HCO_ENTER
  PUBLIC           :: HCO_LEAVE
  PUBLIC           :: HCO_ERROR_SET
  PUBLIC           :: HCO_ERROR_FINAL
  PUBLIC           :: HCO_IsVerb
  PUBLIC           :: HCO_VERBOSE_INQ
  PUBLIC           :: HCO_LOGFILE_OPEN
  PUBLIC           :: HCO_LOGFILE_CLOSE
  !MODULE VARIABLES:
  ! Double and single precision definitions
  INTEGER, PARAMETER, PUBLIC  :: dp = KIND( REAL( 0.0, 8 ) ) ! Double (r8)
  INTEGER, PARAMETER, PUBLIC  :: sp = KIND( REAL( 0.0, 4 ) ) ! Single (r4)
#if defined( USE_REAL8 )
  INTEGER, PARAMETER, PUBLIC  :: hp = KIND( REAL( 0.0, 8 ) ) ! HEMCO prec r8
#else
  INTEGER, PARAMETER, PUBLIC  :: hp = KIND( REAL( 0.0, 4 ) ) ! HEMCO prec r4
#endif
  INTEGER, PARAMETER, PUBLIC  :: i4 = 4                      ! FourByteInt
  INTEGER, PARAMETER, PUBLIC  :: i8 = 8                      ! EightByteInt

  ! Error success/failure definitions
  INTEGER, PARAMETER, PUBLIC  :: HCO_SUCCESS = 0
  INTEGER, PARAMETER, PUBLIC  :: HCO_FAIL    = -999

  ! Tiny value for math operations:
  ! --> deprecated. Use TINY(1.0_hp) instead!
  REAL(hp), PARAMETER, PUBLIC :: HCO_TINY    = 1.0e-32_hp

  ! Missing value
  ! Note: define missing value as single precision because all data arrays
  ! are read/stored in single precision.
```

```
REAL(sp), PARAMETER, PUBLIC :: HCO_MISSVAL = -1.e31_sp

! HEMCO version number.
CHARACTER(LEN=12), PARAMETER, PUBLIC :: HCO_VERSION = 'v2.1.006'

INTERFACE HCO_Error
   MODULE PROCEDURE HCO_ErrorNoErr
   MODULE PROCEDURE HCO_ErrorErr
END INTERFACE HCO_Error

INTERFACE HCO_Warning
   MODULE PROCEDURE HCO_WarningNoErr
   MODULE PROCEDURE HCO_WarningErr
END INTERFACE HCO_Warning

INTERFACE HCO_MSG
   MODULE PROCEDURE HCO_MsgNoErr
   MODULE PROCEDURE HCO_MsgErr
END INTERFACE HCO_MSG
```

REVISION HISTORY:

```
 23 Sep 2013 - C. Keller   - Initialization
 12 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 12 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
 03 Mar 2015 - C. Keller   - Added HCO_CFLAG_* and HCO_DCTTYPE_*
 15 Feb 2016 - C. Keller   - Update to v2.0: error variables now
                             organized in derived type HcoErr.
 23 Nov 2016 - R. Yantosca - Now rewrite KIND definitions to prevent 4-byte
                             and 8-byte variables from being elevated
                             when using -r8 (or equivalent flags)
 29 Dec 2017 - C. Keller   - Update to v2.1.004
```

1.29.1 HCO_Error

Subroutine HCO_Error promts an error message and sets RC to HCO_FAIL. Note that this routine does not stop a run, but it will cause a stop at higher level (when RC gets evaluated).

INTERFACE:

```
SUBROUTINE HCO_ErrorErr( Err, ErrMsg, RC, THISLOC )
```

INPUT PARAMETERS:

```
 TYPE(HcoErr),     POINTER                  :: Err
 CHARACTER(LEN=*), INTENT(IN   )            :: ErrMsg
 CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL  :: THISLOC
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
    23 Sep 2013 - C. Keller - Initialization
```

1.29.2 HCO_Error

Subroutine HCO_Error promts an error message and sets RC to HCO_FAIL. Note that this routine does not stop a run, but it will cause a stop at higher level (when RC gets evaluated).

INTERFACE:

```
  SUBROUTINE HCO_ErrorNoErr( ErrMsg, RC, THISLOC )
```

INPUT PARAMETERS:

```
    CHARACTER(LEN=*), INTENT(IN   )           :: ErrMsg
    CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL  :: THISLOC
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
    23 Sep 2013 - C. Keller - Initialization
```

1.29.3 HCO_Warning

Subroutine HCO_Warning promts a warning message without forcing HEMCO to stop, i.e. return code is set to HCO_SUCCESS.

INTERFACE:

```
  SUBROUTINE HCO_WarningErr( Err, ErrMsg, RC, WARNLEV, THISLOC )
  !INPUT PARAMETERS"
    TYPE(HcoErr),     POINTER                 :: Err
    CHARACTER(LEN=*), INTENT(IN   )           :: ErrMsg
    INTEGER         , INTENT(IN   ), OPTIONAL  :: WARNLEV
    CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL  :: THISLOC
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
    23 Sep 2013 - C. Keller - Initialization
    26 Mar 2015 - C. Keller - Added warning levels
```

1.29.4 HCO_Warning

Subroutine HCO_Warning promts a warning message without forcing HEMCO to stop, i.e. return code is set to HCO_SUCCESS.

INTERFACE:

```
SUBROUTINE HCO_WarningNoErr( ErrMsg, RC, WARNLEV, THISLOC )
!INPUT PARAMETERS"
  CHARACTER(LEN=*), INTENT(IN   )            :: ErrMsg
  INTEGER         , INTENT(IN   ), OPTIONAL  :: WARNLEV
  CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL  :: THISLOC
```

INPUT/OUTPUT PARAMETERS:

```
  INTEGER,          INTENT(INOUT)            :: RC
```

REVISION HISTORY:

```
 23 Sep 2013 - C. Keller - Initialization
 26 Mar 2015 - C. Keller - Added warning levels
```

1.29.5 HCO_MSG

Subroutine HCO_MSG passes message msg to the HEMCO logfile (or to standard output if the logfile is not open). Sep1 and Sep2 denote line delimiters before and after the message, respectively. The optional argument Verb denotes the minimum verbose level associated with this message. The message will only be prompted if the verbose level on this CPU (e.g. of this Err object) is at least as high as Verb.

INTERFACE:

```
SUBROUTINE HCO_MSGErr( Err, Msg, Sep1, Sep2, Verb )
```

INPUT PARAMETERS:

```
  TYPE(HcoErr),     POINTER                  :: Err
  CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL  :: Msg
  CHARACTER(LEN=1), INTENT(IN   ), OPTIONAL  :: Sep1
  CHARACTER(LEN=1), INTENT(IN   ), OPTIONAL  :: Sep2
  INTEGER,          INTENT(IN   ), OPTIONAL  :: Verb
```

REVISION HISTORY:

```
 23 Sep 2013 - C. Keller   - Initialization
 20 May 2015 - R. Yantosca - Minor formatting fix: use '(a)' instead of *
                             to avoid line wrapping around at 80 columns.
```

1.29.6 HCO_MSG

Subroutine HCO_MSG passes message msg to the HEMCO logfile (or to standard output if the logfile is not open). Sep1 and Sep2 denote line delimiters before and after the message, respectively. The optional argument Verb denotes the minimum verbose level associated with this message. The message will only be prompted if the verbose level on this CPU (e.g. of this Err object) is at least as high as Verb.

INTERFACE:

```
SUBROUTINE HCO_MSGnoErr( Msg, Sep1, Sep2, Verb )
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN   ), OPTIONAL  :: Msg
CHARACTER(LEN=1), INTENT(IN   ), OPTIONAL  :: Sep1
CHARACTER(LEN=1), INTENT(IN   ), OPTIONAL  :: Sep2
INTEGER,          INTENT(IN   ), OPTIONAL  :: Verb
```

REVISION HISTORY:

```
23 Sep 2013 - C. Keller   - Initialization
20 May 2015 - R. Yantosca - Minor formatting fix: use '(a)' instead of *
                            to avoid line wrapping around at 80 columns.
```

1.29.7 HCO_Enter

Subroutine HCO_Enter is called upon entering a routine. It organizes the traceback handling. It is recommended to call this routine for 'big' routines but NOT for routines/functions that are frequently called, e.g. inside of loops!

Note that all subroutines calling HCO_Enter must also call HCO_Leave!

INTERFACE:

```
SUBROUTINE HCO_Enter( Err, thisLoc, RC )
```

INPUT PARAMETERS:

```
TYPE(HcoErr),     POINTER       :: Err
CHARACTER(LEN=*), INTENT(IN   ) :: thisLoc
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
23 Sep 2013 - C. Keller - Initialization
```

1.29.8 HCO_Leave

Subroutine HCO_Leave is called upon leaving a routine. It organizes the traceback handling. It is recommended to call this routine for 'big' routines but NOT for routines/functions that are frequently called, e.g. inside of loops!

Note that all subroutines calling HCO_Leave must also call HCO_Enter!

INTERFACE:

```
SUBROUTINE HCO_Leave( Err, RC )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HcoErr),     POINTER       :: Err
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
23 Sep 2013 - C. Keller - Initialization
```

1.29.9 HCO_Error_Set

Subroutine HCO_Error_Set defines the HEMCO error settings. This routine is called at the beginning of a HEMCO simulation. Its input parameter are directly taken from the HEMCO configuration file. If LogFile is set to '*' (asterik), all output is directed to the standard output.

INTERFACE:

```
SUBROUTINE HCO_ERROR_SET( am_I_Root, Err, LogFile,   &
                          Verbose,   WarningLevel, RC )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)     :: am_I_Root      ! Root CPU?
TYPE(HcoErr),     POINTER        :: Err            ! Error object
CHARACTER(LEN=*), INTENT(IN)     :: LogFile        ! logfile path+name
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: Verbose        ! verbose level
INTEGER,          INTENT(INOUT)  :: WarningLevel   ! warning level
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
23 Sep 2013 - C. Keller - Initialization
```

1.29.10 HCO_Error_Final

Subroutine HCO_Error_Final finalizes the error type.

INTERFACE:

```
SUBROUTINE HCO_Error_Final ( Err )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HcoErr),     POINTER        :: Err            ! Error object
```

REVISION HISTORY:

23 Sep 2013 - C. Keller - Initialization

1.29.11 HCO_Verbose_Inq

Function HCO_Verbose_Inq returns the HEMCO verbose number.

INTERFACE:

```
FUNCTION HCO_VERBOSE_INQ ( ERR ) RESULT ( VerbNr )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HcoErr),     POINTER        :: Err            ! Error object
```

OUTPUT PARAMETERS:

```
INTEGER :: VerbNr
```

REVISION HISTORY:

15 Mar 2015 - C. Keller - Initialization

1.29.12 HCO_IsVerb

Function HCO_IsVerb returns true if the HEMCO verbose number is equal to or larger than the passed number.

INTERFACE:

```
FUNCTION HCO_IsVerb ( Err, VerbNr ) RESULT ( IsVerb )
```

INPUT PARAMETERS:

```
TYPE(HcoErr),  POINTER    :: Err            ! Error object
INTEGER,       INTENT(IN) :: VerbNr
```

OUTPUT PARAMETERS:

```
LOGICAL                   :: IsVerb
```

REVISION HISTORY:

15 Mar 2015 - C. Keller - Initialization

1.29.13 HCO_LOGFILE_OPEN

Subroutine HCO_LOGFILE_OPEN opens the HEMCO logfile (if not yet open).

INTERFACE:

```
SUBROUTINE HCO_LogFile_Open( Err, RC )
```

USES:

```
USE inquireMod,   ONLY : findFreeLUN
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HcoErr),  POINTER    :: Err            ! Error object
INTEGER,   INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
23 Sep 2013 - C. Keller   - Initialization
14 Aug 2014 - R. Yantosca - Add FORM='FORMATTED' to the OPEN statement
                            so that the HEMCO log will be a text file
22 Jan 2016 - R. Yantosca - Line-buffer the HEMCO log file for pgfortran
```

1.29.14 HCO_LogFile_Close

Subroutine HCO_LOGFILE_CLOSE closes the HEMCO logfile. If argument ShowSummary is enabled, it will prompt a summary of the HEMCO run up to this point (number of warnings, etc.).

INTERFACE:

```
SUBROUTINE HCO_LogFile_Close( Err, ShowSummary )
```

INPUT PARAMETERS:

```
TYPE(HcoErr),  POINTER        :: Err            ! Error object
LOGICAL, INTENT(IN), OPTIONAL :: ShowSummary
```

REVISION HISTORY:

```
23 Sep 2013 - C. Keller - Initialization
```

1.30 Fortran: Module Interface hco_unit_mod.F90

Module HCO_Unit_Mod contains routines to check/convert units.

INTERFACE:

```
MODULE HCO_Unit_Mod
```

USES:

```
USE HCO_Error_Mod

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCO_Unit_Change
PUBLIC :: HCO_Unit_GetMassScal
PUBLIC :: HCO_Unit_GetAreaScal
PUBLIC :: HCO_Unit_GetTimeScal
PUBLIC :: HCO_Unit_ScalCheck
PUBLIC :: HCO_IsUnitLess
PUBLIC :: HCO_IsIndexData
PUBLIC :: HCO_UnitTolerance
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: HCO_Unit_Change_SP
PRIVATE :: HCO_Unit_Change_DP
```

REVISION HISTORY:

```
 15 May 2012 - C. Keller   - Initialization
 08 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 08 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 24 Jul 2014 - C. Keller   - Now define unitless & 'standard' units as
                             parameter
 13 Aug 2014 - C. Keller   - Interface for sp & dp arrays
 13 Mar 2015 - R. Yantosca - Add m and m2 to the "unitless" list
 16 Mar 2015 - R. Yantosca - Also allow "kg m-2 s-1" and similar units
 16 Mar 2015 - R. Yantosca - Add dobsons and dobsons/day units
 16 Jun 2015 - R. Yantosca - Add % and percent to the unitless list
 07 Jan 2016 - E. Lundgren - Update Avogadro's # to NIST 2014 value
 19 Sep 2016 - R. Yantosca - Make sure all strings are the same length in
                             the array constructor or Gfortran will choke
 10 Jun 2018 - C. Keller   - Add mol/mol to unitless quantities
```

1.30.1 HCO_Unit_Change_sp

Subroutine HCO_UNIT_CHANGE_SP is a wrapper routine to convert the values of the passed single precision array to units of (emitted) kg/m2/s.

INTERFACE:

```
SUBROUTINE HCO_Unit_Change_SP( HcoConfig,   ARRAY, UNITS, MW_IN, MW_OUT,   &
                               MOLEC_RATIO, YYYY,  MM,    AreaFlag, &
                               TimeFlag,    FACT,  RC,    KeepSpec   )
```

USES:

```
USE HCO_TYPES_MOD,    ONLY : ConfigObj
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER                :: HcoConfig
CHARACTER(LEN=*), INTENT(IN )            :: UNITS          ! Data unit
REAL(hp),         INTENT(IN )            :: MW_IN          ! MW g/mol
REAL(hp),         INTENT(IN )            :: MW_OUT         ! MW g/mol
REAL(hp),         INTENT(IN )            :: MOLEC_RATIO    ! molec. ratio
INTEGER,          INTENT(IN )            :: YYYY           ! Data year
INTEGER,          INTENT(IN )            :: MM             ! Data month
LOGICAL,          INTENT(IN ), OPTIONAL  :: KeepSpec       ! Keep input species?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)          :: AreaFlag       ! 2 if per area, 3 if per v
INTEGER,          INTENT(INOUT)          :: TimeFlag       ! 1 if per time, 0 otherwis
REAL(sp),         POINTER                :: ARRAY(:,:,:,:) ! Data
INTEGER,          INTENT(INOUT)          :: RC
```

OUTPUT PARAMETERS:

```
REAL(hp),         INTENT(  OUT), OPTIONAL :: FACT           ! Applied factor
```

REVISION HISTORY:

```
13 Aug 2014 - C. Keller - Initial Version
```

1.30.2 HCO_Unit_Change_dp

Subroutine HCO_UNIT_CHANGE_DP is a wrapper routine to convert the values of the passed double precision array to units of (emitted) kg/m2/s.

INTERFACE:

```
SUBROUTINE HCO_Unit_Change_DP( HcoConfig,   ARRAY, UNITS, MW_IN, MW_OUT,   &
                               MOLEC_RATIO, YYYY,  MM,    AreaFlag, &
                               TimeFlag,    FACT,  RC,    KeepSpec   )
```

USES:

```
USE HCO_TYPES_MOD,    ONLY : ConfigObj
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER                :: HcoConfig
CHARACTER(LEN=*), INTENT(IN )            :: UNITS          ! Data unit
REAL(hp),         INTENT(IN )            :: MW_IN          ! MW g/mol
REAL(hp),         INTENT(IN )            :: MW_OUT         ! MW g/mol
REAL(hp),         INTENT(IN )            :: MOLEC_RATIO    ! molec. ratio
INTEGER,          INTENT(IN )            :: YYYY           ! Data year
INTEGER,          INTENT(IN )            :: MM             ! Data month
LOGICAL,          INTENT(IN ), OPTIONAL  :: KeepSpec       ! Keep input species?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)           :: AreaFlag       ! 2 if per area, 3 if per v
INTEGER,          INTENT(INOUT)           :: TimeFlag       ! 1 if per time, 0 otherwis
REAL(dp),         POINTER                 :: ARRAY(:,:,:,:) ! Data
INTEGER,          INTENT(INOUT)           :: RC
```

OUTPUT PARAMETERS:

```
REAL(hp),         INTENT(  OUT), OPTIONAL :: FACT           ! Applied factor
```

REVISION HISTORY:

```
13 Aug 2014 - C. Keller - Initial Version
```

1.30.3 HCO_Unit_Factor

Subroutine HCO_UNIT_Factor calculates the conversion factor needed to conver unit UNIT to HEMCO units.

The mass is in units of kg and refers to mass of emitted species. For most compounds, this corresponds to the molecular mass of the species, but e.g. for VOCs this can be mass of carbon instead. The mass and area/volume conversion is always performed, but the time conversion is only done if a valid time string is provided. For example, if the input unit is kg/cm3 it will be converted to kg/m3, while ug/m2/year is converted to kg/m2/s. If no (valid) area/volume is given in the unit string, the return flag PerArea is set to False (True otherwise).

The input argument UNITS refers to the unit of the input data. Argument MW_IN denotes the molecular weight of the input unit (g/mol), while MW_OUT is the molecular weight of the output unit. They can differ e.g. for VOCs whose output units are in mass carbon instead of mass species. The argument MOLEC_RATIO is the coefficient used for the conversion of molecules of species to molecules of carbon. For example, MOLEC_RATIO should be set to 2 for C2H6, which will properly convert kg species (or molec species) to kg C. If the input unit is already on a per carbon basis (e.g. kgC, or molecC), no species coefficients will be applied!

Supported unit values:

MASSES:

- molec (includes molecC; molec(C); molec tracer; molecN, molec(N))
- atom or atoms (incl. atomC, etc.)
- kg, g, mg, ug, ng (incl. kgC, etc.)

TIMES:

- s, sec, hr, hour, d, day, mt, month, y, year.

- Valid formats: s, s-1, s^-1.

VOLUMES/AREAS:

- cm2, m2, km2, cm3, dm3, m3, l.
- Valid formats: cm3, cm-3, cm$\hat{3}$, cm^-3.

The following units will be ignored (no unit conversion is applied):

- unitless
- fraction
- factor
- hours
- degC
- 1

INTERFACE:

```
SUBROUTINE HCO_Unit_Factor( HcoConfig, UNITS, MW_IN,    MW_OUT,   MOLEC_RATIO, YYYY, &
                            MM,       AreaFlag, TimeFlag, Factor,      RC, KeepSpec )
```

USES:

```
USE HCO_TYPES_MOD,    ONLY : ConfigObj
USE CharPak_Mod,      ONLY : CStrip
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER              :: HcoConfig
CHARACTER(LEN=*), INTENT(IN )          :: UNITS       ! Data unit
REAL(hp),         INTENT(IN )          :: MW_IN       ! MW g/mol
REAL(hp),         INTENT(IN )          :: MW_OUT      ! MW g/mol
REAL(hp),         INTENT(IN )          :: MOLEC_RATIO ! molec. ratio
INTEGER,          INTENT(IN )          :: YYYY        ! Data year
INTEGER,          INTENT(IN )          :: MM          ! Data month
LOGICAL,          INTENT(IN ), OPTIONAL :: KeepSpec    ! Keep input species?
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT)        :: AreaFlag
INTEGER,          INTENT(  OUT)        :: TimeFlag
REAL(hp),         INTENT(  OUT)        :: Factor         ! Conversion factor
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)        :: RC
```

REVISION HISTORY:

```
23 Oct 2012 - C. Keller - Initial Version
23 May 2013 - C. Keller - Now use additive method
01 Oct 2013 - C. Keller - Now convert to kg/m2/s instead of
                          molec/cm2/s
13 Aug 2014 - C. Keller - Split off from subroutine HCO\_UNIT\_CHANGE
```

1.30.4 HCO_Unit_GetMassScal

Returns the mass scale factors for the given unit. This is the scale factor required to convert from input units to HEMCO units (i.e. kg). If KeepSpec is set to true, the molecular weight of the input data is preserved, e.g. data in kgC is kept in kgC, data in molecC is converted to kgC, etc.

INTERFACE:

```
SUBROUTINE HCO_UNIT_GetMassScal( HcoConfig, unt, MW_IN, MW_OUT, &
                                 MOLEC_RATIO, Scal, KeepSpec )
```

USES:

```
USE HCO_CharTools_Mod
USE HCO_TYPES_MOD,    ONLY : ConfigObj
```

INPUT PARAMETERS:

```
TYPE(ConfigObj),  POINTER               :: HcoConfig
CHARACTER(LEN=*), INTENT(IN)            :: unt         ! Input units
REAL(hp),         INTENT(IN)            :: MW_IN       ! MW g/mol
REAL(hp),         INTENT(IN)            :: MW_OUT      ! MW g/mol
REAL(hp),         INTENT(IN)            :: MOLEC_RATIO ! molec. ratio
LOGICAL,          INTENT(IN ), OPTIONAL :: KeepSpec    ! Keep input species?
!OUTPUT PARAMETER:
REAL(hp),         INTENT(OUT)           :: Scal        ! Scale factor
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller - Initial version
17 Sep 2014 - C. Keller - Now accept '1' as mass unit (e.g. 1/m3/s)
```

1.30.5 HCO_Unit_GetTimeScal

Returns the time scale factors for the given unit. This is the scale factor required to convert from unit 'Unit' to HEMCO units (i.e. per second).

INTERFACE:

```
SUBROUTINE HCO_Unit_GetTimeScal( unt, MM, YYYY, Scal, Flag )
```

USES:

```
USE HCO_CharTools_Mod
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN)  :: unt   ! This unit
INTEGER,          INTENT(IN)  :: MM    ! Current month
INTEGER,          INTENT(IN)  :: YYYY  ! Current year
```

OUTPUT PARAMETERS:

```
REAL(hp),         INTENT(OUT) :: Scal  ! Scale factor
INTEGER,          INTENT(OUT) :: Flag  ! 1=per time, 0 otherwise
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller - Initial version
```

1.30.6 HCO_Unit_GetAreaScal

Returns the area/volume scale factors for the given unit. This is the scale factor required to convert from unit 'Unit' to HEMCO units (i.e. per m2 or per m3).

INTERFACE:

```
SUBROUTINE HCO_Unit_GetAreaScal( unt, Scal, Flag )
```

USES:

```
USE HCO_CharTools_Mod
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN)  :: unt   ! This unit
```

OUTPUT PARAMETERS:

```
REAL(hp),         INTENT(OUT) :: Scal  ! scale factor
INTEGER,          INTENT(OUT) :: Flag  ! 2=per area, 3= per volume, 0 otherwise
```

REVISION HISTORY:

```
13 Mar 2013 - C. Keller - Initial version
```

1.30.7 HCO_Unit_ScalCheck

Check if the provided unit is unitless. Returns 0 if Unit is unitless, 1 if it's not unitless but in correct HEMCO emission units (i.e. kg/m2/s), 2 if it's in HEMCO concentration units (kg/m3), -1 otherwise.

INTERFACE:

```
FUNCTION HCO_Unit_ScalCheck( Unit ) Result ( Flag )
```

USES:

```
USE CharPak_Mod, ONLY : TRANLC
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN)  :: Unit
```

OUTPUT PARAMETERS:

```
INTEGER                         :: Flag  ! 0=ok, 1=warning, 2=error
```

REVISION HISTORY:

13 Mar 2013 - C. Keller - Initial version

1.30.8 HCO_IsUnitless

Returns TRUE if the passed string corresponds to the HEMCO unitless data unit string.

INTERFACE:

```
FUNCTION HCO_IsUnitless( Unt ) Result ( Bool )
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN)   :: Unt
```

OUTPUT PARAMETERS:

```
LOGICAL                         :: Bool
```

REVISION HISTORY:

24 Jul 2014 - C. Keller - Initial version

1.30.9 HCO_IsIndexData

Returns TRUE if the passed string corresponds to the HEMCO index data unit string.

INTERFACE:

```
FUNCTION HCO_IsIndexData( Unt ) Result ( Bool )
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN)   :: Unt
```

OUTPUT PARAMETERS:

```
LOGICAL                         :: Bool
```

REVISION HISTORY:

24 Jul 2014 - C. Keller - Initial version

1.30.10 HCO_UnitTolerance

Returns the HEMCO unit tolerance as defined in the HEMCO configuration file (under settings). Returns a default value of 0 (no tolerance) if no value is set in the configuration file.

INTERFACE:

```
FUNCTION HCO_UnitTolerance( HcoConfig ) Result ( UnitTolerance )
```

USES:

```
USE HCO_TYPES_MOD,    ONLY : ConfigObj
USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt, CoreNr
```

INPUT PARAMETERS:

```
TYPE(ConfigObj), POINTER       :: HcoConfig
```

OUTPUT PARAMETERS:

```
INTEGER                        :: UnitTolerance
```

REVISION HISTORY:

```
24 Jul 2014 - C. Keller - Initial version
```

1.31 Fortran: Module Interface hcoio_diagn_mod.F90

Module HCOIO_Diagn_Mod.F90 is the data interface module for the HEMCO diagnostics. It contains routines to write out diagnostics into a netCDF file.

In an ESMF/MAPL environment, the HEMCO diagnostics are not directly written to disk but passed to the gridded component export state, where they can be picked up by the MAPL HISTORY component.

INTERFACE:

```
MODULE HCOIO_DIAGN_MOD
```

USES:

```
USE HCO_ERROR_MOD

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HcoDiagn_Write
PUBLIC :: HCOIO_Diagn_WriteOut
```

REMARKS:

```
HEMCO diagnostics are still in testing mode. We will fully activate them
at a later time.  They will be turned on when debugging & unit testing.
```

REVISION HISTORY:

```
04 May 2014 - C. Keller   - Initial version.
11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
11 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
28 Jul 2014 - C. Keller   - Removed GC specific initialization calls and
                            moved to HEMCO core.
05 Aug 2014 - C. Keller   - Added dummy interface for ESMF.
03 Apr 2015 - C. Keller   - Added HcoDiagn_Write
```

1.31.1 HcoDiagn_Write

Subroutine HcoDiagn_Write is the wrapper routine to write out the content of the built-in HEMCO diagnostics. If input argument Restart is set to TRUE, only the restart collection will be written out. Otherwise, the default collection

INTERFACE:

```
SUBROUTINE HcoDiagn_Write( am_I_Root, HcoState, Restart, RC )
```

USES:

```
USE HCO_State_Mod,        ONLY : HCO_State
USE HCO_Clock_Mod,        ONLY : HcoClock_SetLast
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )    :: am_I_Root    ! Root CPU?
TYPE(HCO_State), POINTER          :: HcoState     ! HEMCO state object
LOGICAL,         INTENT(IN   )    :: Restart      ! write restart (enforced)?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)    :: RC           ! Return code
```

REVISION HISTORY:

```
03 Apr 2015 - C. Keller   - Initial version
01 Nov 2016 - C. Keller   - Also write out default diagnostics collection if
                            RESTART=.TRUE.
17 Oct 2017 - C. Keller   - Don't pass restart diagnostics to EXPORT state in
                            ESMF mode. They are already in the internal
                            state!
```

1.31.2 HCOIO_Diagn_WriteOut

Subroutine HCOIO_Diagn_WriteOut writes diagnostics to output. Depending on the model environment, different subroutines will be invoked.

INTERFACE:

```
SUBROUTINE HCOIO_Diagn_WriteOut( am_I_Root,   HcoState, ForceWrite,  &
                                 RC,          PREFIX,   UsePrevTime, &
                                 OnlyIfFirst, COL                    )
```

USES:

```
    USE HCO_State_Mod,        ONLY : HCO_State
#if defined(ESMF_)
    USE HCOIO_WRITE_ESMF_MOD, ONLY : HCOIO_WRITE_ESMF
#else
    USE HCOIO_WRITE_STD_MOD,  ONLY : HCOIO_WRITE_STD
#endif
```

INPUT PARAMETERS:

```
    LOGICAL,                    INTENT(IN   ) :: am_I_Root   ! root CPU?
    TYPE(HCO_State),  POINTER                 :: HcoState    ! HEMCO state object
    LOGICAL,                    INTENT(IN   ) :: ForceWrite  ! Write all diagnostics?
    CHARACTER(LEN=*), OPTIONAL, INTENT(IN   ) :: PREFIX      ! File prefix
    LOGICAL,          OPTIONAL, INTENT(IN   ) :: UsePrevTime ! Use previous time
    LOGICAL,          OPTIONAL, INTENT(IN   ) :: OnlyIfFirst ! Only write if nnDiagn is 1
    INTEGER,          OPTIONAL, INTENT(IN   ) :: COL         ! Collection Nr.
```

INPUT/OUTPUT PARAMETERS:

```
    INTEGER,          INTENT(INOUT) :: RC          ! Failure or success
```

REVISION HISTORY:

```
12 Sep 2013 - C. Keller   - Initial version
11 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
11 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
19 Feb 2015 - C. Keller   - Added optional argument OnlyIfFirst
23 Feb 2015 - R. Yantosca - Now make Arr1D REAL(sp) so that we can write
                            out lon & lat as float instead of double
06 Nov 2015 - C. Keller   - Output time stamp is now determined from
                            variable OutTimeStamp.
```

1.32 Fortran: Module Interface hco_state_mod.F90

Module HCO_State_Mod contains definitions and sub- routines for the HEMCO state derived type. The HEMCO state object (HcoState) contains all information related to the HEMCO run, such as the HEMCO clock, information on the emission grid and the data

fields to be read, details on all used species, various physical constants, etc. It also contains the final assembled 3D flux and 2D deposition arrays (to be passed to the overlaying model) and a pointer to the HEMCO configuration object (Config). The latter contains error and traceback information and holds the data fields (in the data list ConfigList).

The HEMCO state object (typically called HcoState) for a given HEMCO run must be defined on the HEMCO-model interface level (subroutine HcoState_Init).

INTERFACE:

```
MODULE HCO_State_Mod
  USES:
  USE HCO_Types_Mod
  USE HCO_Error_Mod
  USE HCO_Arr_Mod
  USE HCO_VertGrid_Mod

#if defined(ESMF_)
  USE ESMF
#endif

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  PUBLIC :: HcoState_Init
  PUBLIC :: HcoState_Final
  PUBLIC :: HCO_GetModSpcID
  PUBLIC :: HCO_GetHcoID
  PUBLIC :: HCO_GetExtHcoID

  !=========================================================================
  ! HCO_State: Main HEMCO State derived type
  !=========================================================================
  TYPE, PUBLIC :: HCO_State

     !%%%%% Species information %%%%%
     INTEGER                     :: nSpc       ! # of species
     TYPE(HcoSpc),       POINTER :: Spc(:)     ! list of species

     !%%%%% Emission grid information %%%%%
     INTEGER                     :: NX         ! # of x-pts (lons) on this CPU
     INTEGER                     :: NY         ! # of y-pts (lats) on this CPU
     INTEGER                     :: NZ         ! # of z-pts (levs) on this CPU
     TYPE(HcoGrid),      POINTER :: Grid       ! HEMCO grid information
     TYPE(HcoClock),     POINTER :: Clock      ! HEMCO clock

     ! Data array
```

```
      TYPE(Arr3D_HP),     POINTER :: Buffer3D   ! Placeholder to store temporary
                                                ! 3D array.  Emissions will be
                                                ! written into this array if
                                                ! option FillBuffer = .TRUE.

      !%%%%% Constants and timesteps %%%%%
      TYPE(HcoPhys),      POINTER :: Phys       ! Physical constants
      REAL(sp)                    :: TS_EMIS    ! Emission timestep [s]
      REAL(sp)                    :: TS_CHEM    ! Chemical timestep [s]
      REAL(sp)                    :: TS_DYN     ! Dynamic  timestep [s]

      !%%%%% Aerosol quantities %%%%%
      INTEGER                     :: nDust      ! # of dust species
      TYPE(HcoMicroPhys), POINTER :: MicroPhys  ! Microphysics settings

      !%%%%%  Run time options %%%%%
      LOGICAL                     :: isESMF     ! Are we using ESMF?
      TYPE(HcoOpt),       POINTER :: Options    ! HEMCO run options

      !%%%%% ReadLists %%%%%
      TYPE(RdList),      POINTER  :: ReadLists
      LOGICAL                     :: SetReadListCalled

      !%%%%% Emissions linked list %%%%%%
      TYPE(ListCont), POINTER     :: EmisList
      INTEGER                     :: nnEmisCont =  0 ! # of container in EmisList

      !%%%%% Data container indeces %%%%%
      ! Element i of cIDList will point to data-container with container
      ! ID i (e.g. cIDList(3) points to data-container with cID = 3).
      TYPE(cIDListPnt), POINTER   :: cIDList(:) => NULL()

      ! # of defined data containers. Will be automatically increased
      ! by one when creating a new data container (DataCont_Init)
      INTEGER                     :: nnDataCont = 0

      ! Define object based on TimeIdxCollection derived type
      TYPE(TimeIdxCollection), POINTER :: AlltIDx  => NULL()

      ! HEMCO configuration object
      TYPE(ConfigObj), POINTER    :: Config => NULL()

      ! Pointer to beginning of collections linked list
      TYPE(DiagnBundle),  POINTER :: Diagn  => NULL()

      !%%%%%  ESMF objects
#if defined(ESMF_)
      TYPE(ESMF_GridComp), POINTER :: GridComp
```

```
     TYPE(ESMF_State),    POINTER :: IMPORT
     TYPE(ESMF_State),    POINTER :: EXPORT
#endif
  END TYPE HCO_State
```

REVISION HISTORY:

```
20 Aug 2013 - C. Keller   - Initial version, adapted from
                            gigc_state_chm_mod.F90
07 Jul 2014 - R. Yantosca - Cosmetic changes
30 Sep 2014 - R. Yantosca - Add HcoMicroPhys derived type to HcoState
08 Apr 2015 - C. Keller   - Added MaskFractions to HcoState options.
13 Jul 2015 - C. Keller   - Added option 'Field2Diagn'.
15 Feb 2016 - C. Keller   - Update to v2.0
```

1.32.1 HcoState_Init

Routine HcoState_Init initializes the HEMCO state object. This initializes (nullifies) all pointers and sets all HEMCO settings and options to default values. The here defined pointers are defined/connected at the HEMCO-model interface level. The passed HEMCO configuration object (HcoConfig) must be defined, e.g. this subroutine must be called after having read (at least stage 1 of) the HEMCO configuration file (Config_ReadFile in hco_config_mod.F90).

INTERFACE:

```
SUBROUTINE HcoState_Init( am_I_Root, HcoState, HcoConfig, nSpecies, RC )
```

USES:

```
USE HCO_EXTLIST_MOD,    ONLY : GetExtOpt, CoreNr
USE HCO_UNIT_MOD,       ONLY : HCO_UnitTolerance
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)    :: am_I_Root ! root CPU?
INTEGER,          INTENT(IN)    :: nSpecies  ! # HEMCO species
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER       :: HcoState  ! HEMCO State object
TYPE(ConfigObj),  POINTER       :: HcoConfig ! HEMCO Config object
INTEGER,          INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Aug 2013 - C. Keller - Adapted from gigc_state_chm_mod.F90
07 Jan 2016 - E. Lundgren - Add physical constant RSTARG and updated
                            Avgdr and g0 to NIST 2014 values
15 Feb 2016 - C. Keller - Now pass HcoConfig object
01 Nov 2016 - C. Keller - Now nullify all pointers
12 May 2017 - C. Keller - Added option ScaleEmis
```

1.32.2 HcoState_Final

Routine HcoState_CLEANUP cleans up HcoState.

INTERFACE:

```
SUBROUTINE HcoState_Final( HcoState )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER  :: HcoState    ! HEMCO State object
```

REVISION HISTORY:

```
20 Aug 2013 - C. Keller   - Adapted from gigc_state_chm_mod.F90
24 Sep 2014 - R. Yantosca - Add an extra safety check when deallocating
                            the pointer field HcoState%Spc
```

1.32.3 HCO_GetModSpcId

Function HCO_GetModSpcId returns the model species index of a species by name. Returns -1 if given species is not found, 0 if name corresponds to the HEMCO wildcard character.

INTERFACE:

```
FUNCTION HCO_GetModSpcID( name, HcoState ) RESULT( Indx )
```

USES:

```
USE HCO_EXTLIST_MOD,     ONLY : HCO_GetOpt
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN)    :: name      ! Species name
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  INTENT(INOUT) :: HcoState  ! HEMCO State
```

RETURN VALUE:

```
INTEGER                         :: Indx      ! Index of this species
```

REVISION HISTORY:

```
20 Aug 2013 - C. Keller - Adapted from gigc_state_chm_mod.F90
```

1.32.4 HCO_GetHcoId

Function HCO_GetHcoIdHCO returns the HEMCO species index of a species by name. Returns -1 if given species is not found, 0 if name corresponds to the HEMCO wildcard character.

INTERFACE:

```
FUNCTION HCO_GetHcoID( name, HcoState ) RESULT( Indx )
```

USES:

```
USE HCO_EXTLIST_MOD,   ONLY : HCO_GetOpt
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN)   :: name         ! Species name
TYPE(HCO_State), INTENT(INOUT) :: HcoState     ! HEMCO State
```

RETURN VALUE:

```
INTEGER                      :: Indx         ! Index of this species
```

REVISION HISTORY:

```
20 Aug 2013 - C. Keller - Adapted from gigc_state_chm_mod.F90
```

1.32.5 HCO_GetExtHcoID

Subroutine HCO_GetExtHcoID returns the HEMCO species IDs and names for all species assigned to the given extension (identified by its extension number).

INTERFACE:

```
SUBROUTINE HCO_GetExtHcoID( HcoState, ExtNr, HcoIDs, &
                            SpcNames, nSpc,  RC       )
```

USES:

```
USE CHARPAK_MOD,         ONLY : STRSPLIT
USE HCO_EXTLIST_MOD,     ONLY : GetExtSpcStr
USE HCO_EXTLIST_MOD,     ONLY : HCO_GetOpt
```

INPUT PARAMETERS:

```
TYPE(HCO_State),               POINTER       :: HcoState
INTEGER,                       INTENT(IN   ) :: ExtNr       ! Extension #
```

OUTPUT PARAMETERS:

```
INTEGER,          ALLOCATABLE, INTENT(  OUT) :: HcoIDs(:)   ! Species IDs
```

INPUT/OUTPUT PARAMETERS:

```
 CHARACTER(LEN=*), ALLOCATABLE, INTENT(INOUT) :: SpcNames(:) ! Species names
 INTEGER,                       INTENT(INOUT) :: nSpc        ! # of species
 INTEGER,                       INTENT(INOUT) :: RC          ! Success/fail
```

REVISION HISTORY:

```
10 Jan 2014 - C. Keller: Initialization (update)
29 Sep 2014 - C. Keller: Now allows species lists up to 2047 instead of 255
                         characters.
```

2 HEMCO "Extensions" modules

These modules define the HEMCO Extensions. These are emissions that rely on environmental variables (e.g. temperature, pressure, sunlight, boundary layer height, etc.) supplied by GEOS-Chem.

2.1 Fortran: Module Interface hcox_gc_POPs_mod.F90

Defines the HEMCO extension for the GEOS-Chem persistent organic pollutants (POPs) specialty simulation.

INTERFACE:

```
MODULE HCOX_GC_POPs_Mod
```

USES:

```
  USE HCO_Error_Mod
  USE HCO_Diagn_Mod
  USE HCO_State_Mod,  ONLY : HCO_State   ! Derived type for HEMCO state
  USE HCOX_State_Mod, ONLY : Ext_State   ! Derived type for External state

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  PUBLIC  :: HcoX_GC_POPs_Run
  PUBLIC  :: HcoX_GC_POPs_Init
  PUBLIC  :: HcoX_Gc_POPs_Final
```

PRIVATE MEMBER FUNCTIONS:

```
  PRIVATE :: VEGEMISPOP
  PRIVATE :: LAKEEMISPOP
  PRIVATE :: SOILEMISPOP
  PRIVATE :: IS_LAND
  PRIVATE :: IS_ICE
```

REMARKS:

```
References:
============================================================================
(1 ) Zhang, Y., and S. Tao, Global atmospheric emission inventory of
     polycyclic aromatic hydrocarbons (PAHs) for 2004. Atm Env, 43, 812-819,
     2009.
(2 ) Friedman, C.L, and N.E. Selin, Long-Range Atmospheric Transport of
     Polycyclic Aromatic Hydrocarbons: A Global 3-D Model Analysis
     Including Evaluation of Arctic Sources, Environ. Sci. Technol., 46(17),
     9501-9510, 2012.
(3 ) Friedman, C.L., Y. Zhang, and N.E. Selin, Climate change and
     emissions impacts on atmospheric PAH transport to the Arctic, Environ.
     Sci. Technol., 48, 429-437, 2014.
(4 ) Friedman, C.L., J.R. Pierce, and N.E. Selin, Assessing the influence of
     secondary organic versus primary carbonaceous aerosols on long-range
     atmospheric polycyclic aromatic hydrocarbon transport, Environ. Sci.
     Technol., 48(6), 3293-3302, 2014.
```

REVISION HISTORY:

```
20 Sep 2010 - N.E. Selin    - Initial Version
04 Jan 2011 - C.L. Friedman - Expansion on initial version
19 Aug 2014 - M. Sulprizio  - Now a HEMCO extension
18 Aug 2015 - M. Sulprizio  - Add VEGEMISPOP, LAKEEMISPOP, and SOILEMISPOP
                              routines from new land_pops_mod.F written by
                              C.L. Friedman.
24 Aug 2017 - M. Sulprizio  - Remove support for GCAP
```

2.1.1 HCOX_GC_POPs_run

Subroutine HcoX_Gc_POPs_Run computes emissions of OC-phase, BC-phase, and gas-phase POPs for the GEOS-Chem POPs specialty simulation.

INTERFACE:

```
SUBROUTINE HCOX_GC_POPs_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
! HEMCO modules
USE HCO_EmisList_Mod,  ONLY : HCO_GetPtr
USE HCO_FluxArr_Mod,   ONLY : HCO_EmisAdd
USE HCO_Clock_Mod,     ONLY : HcoClock_First
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
TYPE(Ext_State),  POINTER       :: ExtState    ! Options for POPs sim
TYPE(HCO_State),  POINTER       :: HcoState    ! HEMCO state
```

INPUT/OUTPUT PARAMETERS:

```
 INTEGER,          INTENT(INOUT) :: RC          ! Success or failure?
```

REMARKS:

```
This code is based on routine EMISSPOPS in prior versions of GEOS-Chem.
```

REVISION HISTORY:

```
20 Sep 2010 - N.E. Selin  - Initial Version based on EMISSMERCURY
29 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
13 Dec 2012 - R. Yantosca - Remove reference to obsolete CMN_DEP_mod.F
25 Mar 2013 - R. Yantosca - Now accept am_I_Root, Input_Opt, State_Chm, RC
14 Apr 2014 - R. Yantosca - Prevent div-by-zero error w/ SUM_OF_ALL
19 Aug 2014 - M. Sulprizio- Now a HEMCO extension
07 Jan 2016 - E. Lundgren - Update molar gas constant to NIST 2014
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.1.2 soilemispop

Subroutine SOILEMISPOP is the subroutine for secondary POP emissions from soils.

INTERFACE:

```
SUBROUTINE SOILEMISPOP( POP_SURF, F_OC_SOIL, EPOP_SOIL, &
                        HcoState, ExtState )
```

INPUT PARAMETERS:

```
REAL(sp), DIMENSION(:,:), INTENT(IN)  :: POP_SURF   ! POP sfc conc [kg]
REAL(sp), DIMENSION(:,:), INTENT(IN)  :: F_OC_SOIL  ! Frac C in soil [g/g]
TYPE(HCO_STATE),          POINTER     :: HcoState   ! Hemco state
TYPE(Ext_State),          POINTER     :: ExtState   ! Module options
```

OUTPUT PARAMETERS:

```
REAL(hp), DIMENSION(:,:), INTENT(OUT) :: EPOP_SOIL  ! POP emissions from
                                                    ! soil [kg/m2/s]
```

REMARKS:

REVISION HISTORY:

```
21 Aug 2012 - C.L. Friedman - Initial version based on LAND_MERCURY_MOD
25 Aug 2015 - M. Sulprizio  - Moved to hcox_gc_POPs_mod.F90
02 Oct 2015 - E. Lundgren   - ExtState%POPG is now kg/kg dry air (prev kg)
07 Jan 2016 - E. Lundgren   - Update molar gas constant to NIST 2014
```

2.1.3 lakeemispop

Subroutine LAKEEMISPOP is the subroutine for secondary POP emissions from lakes.

INTERFACE:

```
SUBROUTINE LAKEEMISPOP( POP_SURF, EPOP_LAKE, HcoState, ExtState )
```

USES:

INPUT PARAMETERS:

```
REAL(sp), DIMENSION(:,:), INTENT(IN)  :: POP_SURF   ! POP sfc conc [kg]
TYPE(HCO_STATE),          POINTER     :: HcoState   ! Hemco state
TYPE(Ext_State),          POINTER     :: ExtState   ! Module options
```

OUTPUT PARAMETERS:

```
REAL(hp), DIMENSION(:,:), INTENT(OUT) :: EPOP_LAKE  ! POP emissions from
                                                    ! lakes [kg/m2/s]
```

REMARKS:

REVISION HISTORY:

```
21 Aug 2012 - C.L. Friedman - Initial version based on LAND_MERCURY_MOD
25 Aug 2015 - M. Sulprizio  - Moved to hcox_gc_POPs_mod.F90
02 Oct 2015 - E. Lundgren   - ExtState%POPG is now kg/kg dry air (prev kg)
07 Jan 2016 - E. Lundgren   - Update molar gas constant to NIST 2014
```

2.1.4 vegemispop

Subroutine VEGEMISPOP is the subroutine for secondary POP emissions from soils.

INTERFACE:

```
SUBROUTINE VEGEMISPOP( POP_SURF,  EPOP_VEG, HcoState, ExtState )
```

USES:

INPUT PARAMETERS:

```
REAL(sp), DIMENSION(:,:), INTENT(IN)  :: POP_SURF   ! POP sfc conc [kg]
TYPE(HCO_State),          POINTER     :: HcoState   ! Hemco state
TYPE(Ext_State),          POINTER     :: ExtState   ! Module options
```

OUTPUT PARAMETERS:

```
REAL(hp), DIMENSION(:,:), INTENT(OUT) :: EPOP_VEG   ! POP emissions from
                                                    ! leaves [kg/m2/s]
```

REMARKS:

REVISION HISTORY:

```
21 Aug 2012 - C.L. Friedman - Initial version based on LAND_MERCURY_MOD
25 Aug 2015 - M. Sulprizio  - Moved to hcox_gc_POPs_mod.F90
02 Oct 2015 - E. Lundgren   - ExtState%POPG is now kg/kg dry air (prev kg)
07 Jan 2016 - E. Lundgren   - Update molar gas constant to NIST 2014
```

2.1.5 is_land

Function IS_LAND returns TRUE if surface grid box (I,J) is a land box.

INTERFACE:

```
FUNCTION IS_LAND( I, J, ExtState ) RESULT ( LAND )
```

USES:

INPUT PARAMETERS:

```
INTEGER,         INTENT(IN) :: I           ! Longitude index of grid box
INTEGER,         INTENT(IN) :: J           ! Latitude  index of grid box
TYPE(Ext_State), POINTER    :: ExtState    ! Module options
```

RETURN VALUE:

```
LOGICAL                     :: LAND        ! =T if it is a land box
```

REVISION HISTORY:

```
26 Jun 2000 - R. Yantosca - Initial version
(1 ) Now use ALBEDO field to determine land or land ice boxes for GEOS-3.
     (bmy, 4/4/01)
(2 ) For 4x5 data, regridded albedo field can cause small inaccuracies
     near the poles (bmy, 4/4/01)
(3 ) Add references to CMN_SIZE and CMN, so that we can use the JYEAR
     variable to get the current year.  Also, for 1998, we need to compute
     if is a land box or not from the surface albedo, since for this
     year the LWI/SURFTYPE field is not given.  For other years than 1998,
     we use LWI(I,J) < 50 as our land box criterion.  Deleted obsolete
     code and updated comments.(mje, bmy, 1/9/02)
(4 ) Deleted GEOS-2 #ifdef statement.  GEOS-2 met fields never really
     materialized, we use GEOS-3 instead. (bmy, 9/18/02)
```

```
(5 ) Now uses function GET_YEAR from "time_mod.f".  Removed reference
     to CMN header file. (bmy, 3/11/03)
(6 ) Added code to determine land boxes for GEOS-4 (bmy, 6/18/03)
(7 ) Now modified for GEOS-5 and GCAP met fields (swu, bmy, 5/25/05)
(8 ) Now return TRUE only for land boxes (w/ no ice) (bmy, 8/10/05)
(9 ) Now use NINT to round LWI for GEOS-4/GEOS-5 (ltm, bmy, 5/9/06)
(10) Remove support for GEOS-1 and GEOS-STRAT met fields (bmy, 8/4/06)
16 Aug 2010 - R. Yantosca - Added ProTeX headers
25 Aug 2010 - R. Yantosca - Treat MERRA in the same way as GEOS-5
06 Feb 2012 - R. Yantosca - Treat GEOS-5.7.x in the same way as MERRA/GEOS-5
28 Feb 2012 - R. Yantosca - Removed support for GEOS-3
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
```

2.1.6 is_ice

Function IS_ICE returns TRUE if surface grid box (I,J) contains either land-ice or sea-ice.

INTERFACE:

```
FUNCTION IS_ICE( I, J, ExtState ) RESULT ( ICE )
```

USES:

INPUT PARAMETERS:

```
INTEGER,         INTENT(IN) :: I          ! Longitude index of grid box
INTEGER,         INTENT(IN) :: J          ! Latitude  index of grid box
TYPE(Ext_State), POINTER    :: ExtState   ! Module options
```

RETURN VALUE:

```
LOGICAL                     :: ICE        ! =T if this is an ice box
```

REVISION HISTORY:

```
09 Aug 2005 - R. Yantosca - Initial version
(1 ) Remove support for GEOS-1 and GEOS-STRAT met fields (bmy, 8/4/06)
16 Aug 2010 - R. Yantosca - Added ProTeX headers
25 Aug 2010 - R. Yantosca - Treat MERRA in the same way as GEOS-5
06 Feb 2012 - R. Yantosca - Treat GEOS-5.7.x in the same way as MERRA/GEOS-5
28 Feb 2012 - R. Yantosca - Removed support for GEOS-3
```

2.1.7 HCOX_GC_POPs_Init

Subroutine HcoX_GC_POPs_Init initializes the HEMCO GC_POPs extension.

INTERFACE:

```
SUBROUTINE HCOX_GC_POPs_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_ExtList_Mod,   ONLY : GetExtNr
USE HCO_STATE_MOD,     ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName     ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState    ! Module options
TYPE(HCO_State),  POINTER        :: HcoState    ! Hemco state
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
19 Aug 2014 - M. Sulprizio- Initial version
01 May 2015 - R. Yantosca - Bug fix: need to zero arrays after allocating
```

2.1.8 HCOX_GC_POPs_Final

Subroutine HcoX_GC_POPs_Final finalizes the HEMCO extension for the GEOS-Chem POPs specialty simulation. All module arrays will be deallocated.

INTERFACE:

```
SUBROUTINE HCOX_GC_POPs_Final()
```

REVISION HISTORY:

```
19 Aug 2014 - M. Sulprizio- Initial version
```

2.2 Fortran: Module Interface drydep_toolbox_mod.F90

Module DryEep_ToolBox_Mod contains routines used for dry deposition (and soil NOx emissions) calculations, as implemented into! the GEOS-Chem model.

INTERFACE:

```
MODULE DryDep_ToolBox_Mod
```

USES:

```
USE HCO_ERROR_MOD, ONLY : hp     ! Precision (hp)

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: BIOFIT

INTERFACE BIOFIT
  MODULE PROCEDURE BIOFIT_R4
  MODULE PROCEDURE BIOFIT_R8
END INTERFACE
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: SUNPARAM_R4
PRIVATE :: SUNPARAM_R8
```

REVISION HISTORY:

```
 14 Nov 2013 - C. Keller   - Created from BIOFIT.F and SUNPARAM.F
 09 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 09 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 11 Dec 2014 - M. Yannetti - Split BIOFIT into R4 and R8 versions
                             Split SUNPARAM into R4 and R8 versions
```

2.2.1 BIOFIT_R4

Function BioFit computes the light correction used in the dry deposition and canopy NOx modules.

INTERFACE:

```
FUNCTION BIOFIT_R4( COEFF1, XLAI1, SUNCOS1, CFRAC1, NPOLY ) RESULT ( BIO_FIT )
```

INPUT PARAMETERS:

```
  REAL*4,  INTENT(IN) :: COEFF1(NPOLY)   ! Baldocchi drydep coefficients
  REAL*4,  INTENT(IN) :: XLAI1           ! Leaf area index [cm2/cm2]
  REAL*4,  INTENT(IN) :: SUNCOS1         ! Cosine( Solar Zenith Angle )
  REAL*4,  INTENT(IN) :: CFRAC1          ! Cloud fraction [unitless]
  INTEGER, INTENT(IN) :: NPOLY           ! # of drydep coefficients
```

RETURN VALUE:

```
  REAL*4              :: BIO_FIT         ! Resultant light correction
```

REMARKS:

```
This routine is ancient code from Yuhang Wang.  It was part of the old
Harvard-GISS CTM and was ported into GEOS-Chem.  See this reference for
more information:

  Wang, Y., D.J. Jacob, and J.A. Logan, "Global simulation of tropospheric
   O3-NOx-hydrocarbon chemistry, 1. Model formulation", J. Geophys. Res.,
   103/D9, 10,713-10,726, 1998.
```

REVISION HISTORY:

```
13 Dec 2012 - R. Yantosca - Added ProTeX headers
09 Dec 2014 - R. Yantosca - Now use BIO_FIT as the return value
11 Dec 2014 - M. Yannetti - Split from BIO_FIT
```

2.2.2 BIOFIT_R8

Function BioFit computes the light correction used in the dry deposition and canopy NOx modules.

INTERFACE:

```
FUNCTION BIOFIT_R8( COEFF1, XLAI1, SUNCOS1, CFRAC1, NPOLY ) RESULT ( BIO_FIT )
```

INPUT PARAMETERS:

```
REAL*8,  INTENT(IN) :: COEFF1(NPOLY)   ! Baldocchi drydep coefficients
REAL*8,  INTENT(IN) :: XLAI1           ! Leaf area index [cm2/cm2]
REAL*8,  INTENT(IN) :: SUNCOS1         ! Cosine( Solar Zenith Angle )
REAL*8,  INTENT(IN) :: CFRAC1          ! Cloud fraction [unitless]
INTEGER, INTENT(IN) :: NPOLY           ! # of drydep coefficients
```

RETURN VALUE:

```
REAL*8              :: BIO_FIT         ! Resultant light correction
```

REMARKS:

```
This routine is ancient code from Yuhang Wang.  It was part of the old
Harvard-GISS CTM and was ported into GEOS-Chem.  See this reference for
more information:

  Wang, Y., D.J. Jacob, and J.A. Logan, "Global simulation of tropospheric
   O3-NOx-hydrocarbon chemistry, 1. Model formulation", J. Geophys. Res.,
   103/D9, 10,713-10,726, 1998.
```

REVISION HISTORY:

```
13 Dec 2012 - R. Yantosca - Added ProTeX headers
09 Dec 2014 - R. Yantosca - Now use BIO_FIT as the return value
11 Dec 2014 - M. Yannetti - Split from BIO_FIT
```

2.2.3 SunParam_r4

Subroutine SUNPARAM is called by BIOFIT to perform the light correction used in the dry deposition and canopy NOx modules.

INTERFACE:

```
SUBROUTINE SUNPARAM_R4( X )
```

DEFINED PARAMETERS:

```
INTEGER, PARAMETER    :: NN = 3  ! # of variables (LAI, SUNCOS, CLDFRC)
```

INPUT/OUTPUT PARAMETERS:

```
REAL*4, INTENT(INOUT) :: X(NN)   ! LAI, SUNCOS, or cloud fraction
```

REMARKS:

```
This routine is ancient code from Yuhang Wang.  It was part of the old
Harvard-GISS CTM and was ported into GEOS-Chem.  See this reference for
more information:

  Wang, Y., D.J. Jacob, and J.A. Logan, "Global simulation of tropospheric
   O3-NOx-hydrocarbon chemistry, 1. Model formulation", J. Geophys. Res.,
   103/D9, 10,713-10,726, 1998.
```

REVISION HISTORY:

```
13 Dec 2012 - R. Yantosca - Added ProTeX headers
11 Dec 2014 - M. Yannetti - Split into R4 and R8 versions.
```

2.2.4 SunParam_r8

Subroutine SUNPARAM is called by BIOFIT to perform the light correction used in the dry deposition and canopy NOx modules.

INTERFACE:

```
SUBROUTINE SUNPARAM_R8( X )
```

DEFINED PARAMETERS:

```
INTEGER, PARAMETER    :: NN = 3  ! # of variables (LAI, SUNCOS, CLDFRC)
```

INPUT/OUTPUT PARAMETERS:

```
REAL*8, INTENT(INOUT) :: X(NN)   ! LAI, SUNCOS, or cloud fraction
```

REMARKS:

```
This routine is ancient code from Yuhang Wang.  It was part of the old
Harvard-GISS CTM and was ported into GEOS-Chem.  See this reference for
more information:

  Wang, Y., D.J. Jacob, and J.A. Logan, "Global simulation of tropospheric
   O3-NOx-hydrocarbon chemistry, 1. Model formulation", J. Geophys. Res.,
   103/D9, 10,713-10,726, 1998.
```

REVISION HISTORY:

```
13 Dec 2012 - R. Yantosca - Added ProTeX headers
11 Dec 2014 - M. Yannetti - Split into R4 and R8 versions.
```

2.3 Fortran: Module Interface hcox_seaflux_mod.F90

Module HCOX_SeaFlux_Mod contains routines to calculate the oceanic emissions of a number of defined species. The oceanic flux is parameterized according to Liss and Slater, 1974: F = Kg * (Cair - H Cwater) where F is the net flux, Kg is the exchange velocity, Cair and Cwater are the air and aqueous concentrations, respectively, and H is the dimensionless air over water Henry constant.

This module calculates the source and sink terms separately. The source is given as flux, the sink as deposition rate: source = Kg * H * Cwater [kg m-2 s-1] sink = Kg / DEPHEIGHT [s-1]
The deposition rate is obtained by dividing the exchange velocity Kg by the deposition height DEPHEIGHT, e.g. the height over which deposition occurs. This can be either the first grid box only, or the entire planetary boundary layer. The HEMCO option 'PBL_DRYDEP' determines which option is being used.

Kg is calculated following Johnson, 2010, which is largely based on the work of Nightingale et al., 2000a/b. The salinity and seawater pH are currently set to constant global values of 35 ppt and 8.0, respectively. Since Kg is only little sensitive to these variables, this should not introduce a notable error.

This is a HEMCO extension module that uses many of the HEMCO core utilities.

Air-sea exchange is calculated for all species defined during extension initialization. For each species, the following parameter must be specified: species name, model species ID (i.e. ID of this species in the external model), parameterization type of Schmidt number in water, liquid molar volume of species, and the name of the field containing species seawater concentrations. See initialization routine for more details. To add new species to this module, the abovementioned arrays have to be extended accordingly.

References:

- Johnson, M.: A numerical scheme to calculate temperature and salinity dependent air-water transfer velocities for any gas, Ocean Science, 6, 2010.

- Liss and Slater: Flux of gases across the air-sea interface, Nature, 247, 1974.
- Nightingale et al.: In situ evaluation of air-sea gas exchange parameterizations using novel conservative and volatile tracers, Global Biogeochemical Cycles, 14, 2000a.
- Nightingale et al.: Measurements of air-sea gas transfer during an open ocean algal bloom, Geophys. Res. Lett., 27, 2000b.
- Saltzman et al.: Experimental determination of the diffusion coefficient of dimethylsulfide in water, J. Geophys. Res., 98, 1993.

INTERFACE:

```
MODULE HCOX_SeaFlux_Mod
```

USES:

```
USE HCO_Error_MOD
USE HCO_Diagn_MOD
USE HCO_State_MOD,  ONLY : HCO_State
USE HCOX_State_MOD, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCOX_SeaFlux_Init
PUBLIC  :: HCOX_SeaFlux_Run
PUBLIC  :: HCOX_SeaFlux_Final
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: Calc_SeaFlux
```

REVISION HISTORY:

```
 16 Apr 2013 - C. Keller   - Initial version
 01 Oct 2013 - C. Keller   - Now a HEMCO extension module
 11 Dec 2013 - C. Keller   - Now define container name during initialization
 01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 06 Nov 2015 - C. Keller   - Now use land type definitions instead of FRCLND
 14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
 10 Mar 2017 - M. Sulprizio- Add fix for acetone parameterization of Schmidt
                             number - use SCWPAR = 3 instead of 1
```

2.3.1 HCOX_SeaFlux_Run

Subroutine HcoX_SeaFlux_Run is the run routine to calculate oceanic emissions for the current time step.

INTERFACE:

```
SUBROUTINE HCOX_SeaFlux_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_FLUXARR_MOD,  ONLY : HCO_EmisAdd
USE HCO_FLUXARR_MOD,  ONLY : HCO_DepvAdd
USE HCO_CALC_MOD,     ONLY : HCO_EvalFld
 USE HCO_EMISLIST_MOD, ONLY : HCO_GetPtr
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root  ! root CPU?
TYPE(HCO_State), POINTER       :: HcoState   ! Output obj
TYPE(Ext_State), POINTER       :: ExtState  ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC         ! Success or failure?
```

REVISION HISTORY:

```
16 Apr 2013 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.3.2 Calc_SeaFlux

Subroutine CALC_SEAFLUX calculates oceanic emissions of the specified tracer using the parameterization described in Johnson, 2010.

The net emission flux is given by F = - Kg (Cg - Caq*H). Here, we calculate the source term (Kg * H * Caq) in units of kg/m2/s as well as the deposition velocity Kg in m/s.

INTERFACE:

```
SUBROUTINE Calc_SeaFlux( am_I_Root, HcoState, ExtState, &
                         SOURCE,    SINK,     SeaConc,   &
                         OcID,      HcoID,    RC          )
```

USES:

```
USE Ocean_ToolBox_Mod,  ONLY : CALC_KG
USE Henry_Mod,          ONLY : CALC_KH, CALC_HEFF
USE HCO_CALC_MOD,       ONLY : HCO_CheckDepv
USE HCO_GeoTools_Mod,   ONLY : HCO_LANDTYPE
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root           ! root CPU?
INTEGER,         INTENT(IN   ) :: OcID                ! ocean species ID
INTEGER,         INTENT(IN   ) :: HcoID               ! HEMCO species ID
TYPE(HCO_State), POINTER       :: HcoState            ! Output obj
TYPE(Ext_State), POINTER       :: ExtState
```

OUTPUT PARAMETERS:

```
REAL(hp),        INTENT(  OUT) :: SOURCE(HcoState%NX,HcoState%NY )
REAL(hp),        INTENT(  OUT) :: SINK  (HcoState%NX,HcoState%NY )
```

INPUT/OUTPUT PARAMETERS:

```
REAL(hp),        INTENT(INOUT) :: SeaConc(HcoState%NX,HcoState%NY )
INTEGER,         INTENT(INOUT) :: RC                 ! Error stat
```

REMARKS:

```
For now, the salinity and pH of seawater are prescribed to 35ppt and 8.0,
respectively.  The oceanic flux is not expected to be sensitive to these
parameters (which have only little variations anyway), but we may use
climatologies for these parameter at some point nevertheless!
```

REVISION HISTORY:

```
16 Apr 2013 - C. Keller   - Initial version
15 Aug 2014 - C. Keller   - Now restrict calculations to temperatures above
                            10 deg C.
03 Oct 2014 - C. Keller   - Added surface temperature limit of 45 degrees C
                            to avoid negative Schmidt numbers.
07 Oct 2014 - C. Keller   - Now use skin temperature instead of air temperature
06 Mar 2015 - C. Keller   - Now calculate deposition rate over entire PBL.
14 Oct 2015 - R. Yantosca - Pulled variables MW, VB, SCW out of the parallel
                            loop.
06 Nov 2015 - C. Keller   - Now use HCO_LANDTYPE instead of FRCLND
```

2.3.3 HCOX_SeaFlux_Init

Subroutine HCOX_SeaFlux_Init initializes all module variables, including all species - specific parameter such as the liquid molar volume (Vb), the parameterization type for the Schmidt number in water (SCWPAR) and the name of the field containing oceanic concentrations.

LiqVol is the liquid molar volume [cm3/mol]. If not stated otherwise, it is calculated using the Schroeder additive method as described in Johnson, 2010. Note that experimental values for LiqVol should be used if available!

Table 3 of Johnson, 2010: Schroeder additive method for calculating Vb. For all atoms/structural items a molecule contains, the sum of the incre- ments will give the molar volume. e.g. CH2=CH2 contains 2 car- bon atoms, 4 hydrogen atoms and 1 double bond so the Schroeder Vb is 2x7 + 4x7 + 7 = 49cm3mol-1. * applies to all kinds of cyclic features and is applied only once to ring-containing compounds irrespective of the number of rings present.

- Atom/feature Increment/cm3mole-1

- Carbon 7.0
- Hydrogen 7.0
- Oxygen 7.0
- Nitrogen 7.0
- Bromine 31.5
- Chlorine 24.5
- Fluorine 10.5
- Iodine 38.5
- Sulfur 21.0
- Ring* -7.0
- Double bond 7.0
- Triple bond 14.0

SCWPAR denotes which parameterization will be used to calculate the Schmidt number in water (in ocean_toolbox_mod). The following parameterizations are currently supported:

1. Parameterization as in Johnson, 2010 (default).
2. Parameterization for DMS according to Saltzman et al., 1993.
3. Parameterization for Acetone as in former acetone_mod.F in GC.
4. Parameterization for Acetaldehyde as in ald2_mod.F from D. Millet

The oceanic surface concentrations of all species are obtained from external fields. These field names are specified in array OcDataName. For now, we obtain these concentrations from netCDF-files through the HEMCO core module, i.e. for each species there need to be a corresponding seawater concentration data file specified in the HEMCO configuration file. Once we use a coupled (ESMF) system, these names may be used to refer to the names of the concentration fields imported from the ocean model component.

INTERFACE:

```
SUBROUTINE HCOX_SeaFlux_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_ExtList_Mod,        ONLY : GetExtNr
USE HCO_STATE_MOD,          ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root    ! root CPU?
TYPE(HCO_State),  POINTER        :: HcoState     ! Hemco State obj.
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName      ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState       ! Ext. obj.
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC          ! Return status
```

REVISION HISTORY:

```
16 Apr 2013 - C. Keller - Initial version
```

2.3.4 HCOX_SeaFlux_Final

Subroutine HCOX_SeaFlux_Final deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_SeaFlux_Final()
```

REVISION HISTORY:

```
16 Apr 2013 - C. Keller - Initial version
```

2.4 Fortran: Module Interface hcox_ch4wetland_mod.F90

Module HCOX_CH4Wetland_Mod contains routines to calculate methane emissions (including rice) from wetlands. This code is adapted from the GEOS-Chem CH4 offline simulation.

This is a HEMCO extension module that uses many of the HEMCO core utilities.

This code can be used to calculate emissions from wetlands, from rice, or both. Both sources can be enabled/disabled in the HEMCO configuration file.

This extension can calculate emissions for as many species as desired. Those can be listed in the extensions settings (see below), together with individual scale factors and masks. For example, to calculate emissions for total CH4 and two tagged CH4 species (CH4_NA and CH4_EU) with NA emissions scaled by a factor of 1.1, as well as applying the gridded factors NAFIELD and EUFIELD to CH4_NA and CH4_EU, respectively:
121 CH4_WETLANDS : on CH4/CH4_NA/CH4_EU –¿ Wetlands : true –¿ Rice : true –¿ Scaling_CH4_NA : 1.10 –¿ ScaleField_CH4_NA: NAFIELD –¿ ScaleField_CH4_EU: EUFIELD –¿ Cat_Wetlands : 1 –¿ Cat_Rice : 2
The fields NAFIELD and EUFIELD must be defined in the base emission section of the HEMCO configuration file. You can apply any scale factors/masks to that field.

Wetland and rice emissions are now emitted as separate emission categories. Default category is 1 for wetland emissions and 2 for rice emissions. These categories can be changed in the CH4_WETLANDS definitions of the HEMCO configuration file (see above). In combination with the ExtNr (121), these categories can then be used in the HEMCO diagnostics file to output wetland and rice emissions separately, e.g.: CH4_WETL 121 1 -1 2 kg/m2/s CH4_RICE 121 2 -1 2 kg/m2/s

References:

- Pickett-Heaps CA, Jacob DJ, Wecht KJ, et al. Magnitude and seasonality of wetland methane emissions from the Hudson Bay Lowlands (Canada). ACP, 11, 3773-3779, 2011.

INTERFACE:

```
MODULE HCOX_CH4WETLAND_Mod
```

USES:

```
USE HCO_Error_MOD
USE HCO_Diagn_MOD
USE HCOX_TOOLS_MOD
USE HCO_State_MOD,  ONLY : HCO_State
USE HCOX_State_MOD, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCOX_CH4WETLAND_INIT
PUBLIC  :: HCOX_CH4WETLAND_RUN
PUBLIC  :: HCOX_CH4WETLAND_FINAL
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: WETLAND_EMIS
PRIVATE :: RICE_EMIS
```

REVISION HISTORY:

```
 11 Sep 2014 - C. Keller   - Initial version
 01 Oct 2013 - C. Keller   - Now a HEMCO extension module
 11 Dec 2013 - C. Keller   - Now define container name during initialization
 01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 11 Jun 2015 - C. Keller   - Update to support multiple species with individual
                             scale factors and mask regions.
 14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
 24 Aug 2017 - M. Sulprizio- Remove support for GEOS-4, GEOS-5, MERRA
 30 Apr 2018 - C. Keller   - Add categories for wetlands and rice
```

2.4.1 HCOX_CH4WETLAND_Run

Subroutine HcoX_CH4WETLAND_Run is the run routine to calculate oceanic emissions for the current time step.

INTERFACE:

```
SUBROUTINE HCOX_CH4WETLAND_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_CALC_MOD,     ONLY : HCO_EvalFld
USE HCO_EMISLIST_MOD, ONLY : HCO_GetPtr
USE HCO_FLUXARR_MOD,  ONLY : HCO_EmisAdd
USE HCO_TYPES_MOD,    ONLY : DiagnCont
USE HCO_CLOCK_MOD,    ONLY : HcoClock_First
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root  ! root CPU?
TYPE(HCO_State), POINTER       :: HcoState   ! Output obj
TYPE(Ext_State), POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC         ! Success or failure?
```

REVISION HISTORY:

```
11 Sep 2014 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
30 Apr 2018 - C. Keller   - Rice and wetland emissions now have separate categories
```

2.4.2 WETLAND_EMIS

Subroutine WETLAND_EMIS is the driver routine for the CH4 wetland emissions. It calculates wetland emissions and writes them into the passed array CH4wtl.

INTERFACE:

```
SUBROUTINE WETLAND_EMIS ( am_I_Root, HcoState, ExtState, Inst, CH4wtl, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root  ! root CPU?
TYPE(HCO_State), POINTER       :: HcoState   ! Output obj
TYPE(Ext_State), POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MyInst),    POINTER       :: Inst
REAL(hp),        INTENT(INOUT) :: CH4wtl(HcoState%NX,HcoState%NY) ! CH4 emis
INTEGER,         INTENT(INOUT) :: RC         ! Success or failure?
```

REVISION HISTORY:

```
(1 ) Adapted by Jrme Drevet (3/06) from the BIOME-TG Wetland-Methane
     scheme provided by Jed O. Kaplan.
(2 ) CH4 Emissions from Wetland depend on:
             a - Soil Carbon content.
             b - Vegetation type
             c - Wetland area (%)
             d - Soil moisture.
     a, b, c are taken from the LPJ, a vegetation model. Data are provided
     by J.O.Kaplan. Soil moisture is read from GEOS Met input files.
(3 ) Corrected order of DO loops (bmy, 10/1/09)
08 Feb 2012 - R. Yantosca - Treat GEOS-5.7.x in the same way as MERRA
01 Mar 2012 - R. Yantosca - Now use GET_AREA_M2(I,J,L) from grid_mod.F90
07 Mar 2012 - M. Payer    - Added ProTeX headers
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
26 Sep 2013 - R. Yantosca - Renamed GEOS_57 Cpp switch to GEOS_FP
23 Jan 2014 - M. Sulprizio- Now zero wetland emissions if snow covers the
                            ground. Also updated MOIST_SCALE and EMIT_FACT.
                            (K. Wecht, C. Pickett-Heaps)
12 Feb 2014 - K. Wecht    - Updated for 0.25 x 0.3125 NA grid
09 Apr 2014 - R. Yantosca - Bug fix, extend #ifdef for MERRA met fields
11 Sep 2014 - C. Keller   - Now a HEMCO extension
12 Aug 2015 - R. Yantosca - Extend #ifdef for MERRA2 met fields
```

2.4.3 RICE_EMIS

Subroutine RICE_EMIS is the driver routine for the CH4 rice emissions. It calculates rice emissions and writes them into the passed array CH4.

INTERFACE:

```
SUBROUTINE RICE_EMIS ( am_I_Root, HcoState, ExtState, Inst, CH4rce, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root  ! root CPU?
TYPE(HCO_State), POINTER       :: HcoState   ! Output obj
TYPE(Ext_State), POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MyInst),    POINTER       :: Inst
REAL(hp),        INTENT(INOUT) :: CH4rce(HcoState%NX,HcoState%NY) ! CH4 emis
INTEGER,         INTENT(INOUT) :: RC         ! Success or failure?
```

REMARKS:

```
Rice Emissions are scaled to GEOS soil wetness.  Scaling sceme developed
   and implemented by Jerome Drevet.
Wetland emissions are modified by the presence of rice emissions.  Sceme
   developed by Jerome Drevet.
```

REVISION HISTORY:

```
(1 ) CH4 emissions from rice calculated with a routine created by Jerome
     Drevet.  Adapted as its own subroutine by Kevin Wecht (6/03/09)
(2 ) Corrected ordering of DO loops (bmy, 10/1/09)
07 Mar 2012 - M. Payer    - Added ProTeX headers
25 Mar 2013 - R. Yantosca - Now accept am_I_Root, Input_Opt, State_Chm, RC
09 Apr 2014 - R. Yantosca - Bug fix, extend #ifdef for MERRA met fields
11 Sep 2014 - C. Keller   - Now a HEMCO extension
```

2.4.4 HCOX_CH4WETLAND_INIT

Subroutine HCOX_CH4WETLAND_INIT initializes all module variables.

INTERFACE:

```
SUBROUTINE HCOX_CH4WETLAND_INIT( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_ExtList_Mod,        ONLY : GetExtNr
USE HCO_ExtList_Mod,        ONLY : GetExtOpt
USE HCO_ExtList_Mod,        ONLY : GetExtSpcVal
USE HCO_STATE_MOD,          ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root    ! root CPU?
TYPE(HCO_State),  POINTER        :: HcoState     ! Hemco State obj.
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName      ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState     ! Ext. obj.
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC           ! Return status
```

REVISION HISTORY:

```
11 Sep 2014 - C. Keller - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
30 Apr 2018 - C. Keller   - Rice and wetland emissions now have separate categories
```

2.4.5 HCOX_CH4WETLAND_Final

Subroutine HCOX_CH4WETLAND_Final deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_CH4WETLAND_Final( ExtState )
```

INPUT PARAMETERS:

```
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

REVISION HISTORY:

```
11 Sep 2014 - C. Keller - Initial version
```

2.4.6 InstGet

Subroutine InstGet returns a poiner to the desired instance.

INTERFACE:

```
SUBROUTINE InstGet ( Instance, Inst, RC, PrevInst )
```

INPUT PARAMETERS:

```
INTEGER                             :: Instance
TYPE(MyInst),     POINTER           :: Inst
INTEGER                             :: RC
TYPE(MyInst),     POINTER, OPTIONAL :: PrevInst
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.4.7 InstCreate

Subroutine InstCreate creates a new instance.

INTERFACE:

```
SUBROUTINE InstCreate ( ExtNr, Instance, Inst, RC )
```

INPUT PARAMETERS:

```
INTEGER,       INTENT(IN)       :: ExtNr
```

OUTPUT PARAMETERS:

```
INTEGER,       INTENT(  OUT)    :: Instance
TYPE(MyInst),  POINTER          :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,       INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.4.8 InstRemove

Subroutine InstRemove creates a new instance.

INTERFACE:

```
SUBROUTINE InstRemove ( Instance )
```

INPUT PARAMETERS:

```
INTEGER                          :: Instance
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.5 Fortran: Module Interface hcox_tomas_jeagle_mod.F90

Module HCOX_TOMAS_JEAGLE_Mod contains routines to calculate sea salt aerosol emissions for the TOMAS aerosol microphysics package. JKODROS - This is an update of hcox_tomas_seasalt_mod.F90 to use Jeagle emissions. Should bring TOMAS emissions in line with bulk sea salt.

This is a HEMCO extension module that uses many of the HEMCO core utilities.

References:

- Clarke, A.D., Owens, S., Zhou, J. *An ultrafine sea-salt flux from breaking waves: Implications for CCN in the remote marine atmosphere*, J. Geophys. Res., 2006.

INTERFACE:

```
MODULE HCOX_TOMAS_Jeagle_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Diagn_Mod
USE HCO_State_Mod,  ONLY : HCO_State
USE HCOX_State_Mod, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_TOMAS_Jeagle_Init
PUBLIC :: HCOX_TOMAS_Jeagle_Run
PUBLIC :: HCOX_TOMAS_Jeagle_Final
```

REVISION HISTORY:

```
01 Oct 2014 - R. Yantosca - Initial version, based on TOMAS code
20 May 2015 - J. Kodros   - Added fixes to integrate TOMAS with HEMCO
02 JUL 2015 - J. Kodros   - Updating to use scale factors from Jeagle
                            et al. (2011)
```

2.5.1 HCOX_TOMAS_Jeagle_Run

Subroutine HCOX_TOMAS_Jeagle_Run emits sea-salt into the TOMAS sectional sea-salt mass and aerosol number arrays. Sea-salt emission parameterization of Jeagle et al. (2011).

INTERFACE:

```
SUBROUTINE HCOX_TOMAS_Jeagle_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_GeoTools_Mod, ONLY : HCO_LandType
USE HCO_FluxArr_mod,  ONLY : HCO_EmisAdd
USE HCO_State_Mod,    ONLY : HCO_GetHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)    :: am_I_Root   ! root CPU?
TYPE(Ext_State),  POINTER       :: ExtState    ! Extension Options object
TYPE(HCO_State),  POINTER       :: HcoState    ! HEMCO state object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC          ! Success or failure?
```

REMARKS:

REVISION HISTORY:

```
01 Oct 2014 - R. Yantosca - Initial version, based on TOMAS SRCSALT30 code
20 May 2015 - J. Kodros   - Add seasalt number & mass to HEMCO state
20 May 2015 - R. Yantosca - Pass am_I_Root to HCO_EMISADD routine
22 May 2015 - R. Yantosca - Extend up to 40 size bins
10 Jul 2015 - R. Yantosca - Fixed minor issues in the ProTeX headers
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.5.2 HCOX_TOMAS_Jeagle_Init

Subroutine HcoX_TOMAS_Jeagle_Init initializes all extension variables.

INTERFACE:

```
SUBROUTINE HCOX_TOMAS_Jeagle_Init( am_I_Root, HcoState,     &
                                   ExtName,   ExtState, RC )
```

USES:

```
USE HCO_State_Mod,   ONLY : HCO_GetHcoID
USE HCO_STATE_MOD,   ONLY : HCO_GetExtHcoID
USE HCO_ExtList_Mod, ONLY : GetExtNr
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root   ! root CPU?
TYPE(HCO_State),  POINTER        :: HcoState    ! HEMCO state object
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName     ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState    ! Extension options object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC          ! Success or failure?
```

REVISION HISTORY:

```
15 Dec 2013 - C. Keller   - Initial version
10 Jul 2015 - R. Yantosca - Fixed minor issues in ProTeX header
24 Aug 2017 - M. Sulprizio- Remove support for GRID1x1
```

2.5.3 HCOX_TOMAS_Jeagle_Final

Subroutine HcoX_TOMAS_Jeagle_Final deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_TOMAS_Jeagle_Final
```

REVISION HISTORY:

```
15 Dec 2013 - C. Keller   - Initial version
20 May 2015 - J. Kodros   - Deallocate HcoIDs, TC1, TC2 arrays
```

2.6 Fortran: Module Interface hcox_gfed_mod.F90

Module HCOX_GFED_MOD contains routines to calculate GFED4 biomass burning emissions in HEMCO. **INTERFACE:**

```
MODULE HCOX_GFED_MOD
```

USES:

```
USE HCO_ERROR_MOD
USE HCO_DIAGN_MOD
USE HCOX_TOOLS_MOD
USE HCO_STATE_MOD,  ONLY : HCO_State
USE HCOX_State_MOD, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_GFED_Init
PUBLIC :: HCOX_GFED_Run
PUBLIC :: HCOX_GFED_Final
```

REMARKS:

```
 Monthly emissions of DM are read from disk,
 multiplied by daily and 3hourly fractions (if necessary), and then
 multiplied by the appropriate emission factors to produce biomass
 burning emissions.
All species to be used must be listed in the settings section of the HEMCO
configuration file. For every listed species, individual scale factors as
well as masks can be defined. For example, to scale FINN CO emissions by a
factor of 1.05 and restrict them to North America, as well as to scale NO
emissions by a factor of 1.5:
111     GFED              : on  NO/CO/ALK4/ACET/MEK/ALD2/PRPE/C3H8/CH2O/C2H6/SO2/NH3/BC/OC
   --> GFED4              :       true
   --> GFED_daily         :       false
   --> GFED_3hourly       :       false
   --> hydrophilic BC     :       0.2
   --> hydrophilic OC     :       0.5
   --> Mask_CO            :       NAMASK
   --> Scaling_CO         :       1.05
   --> Scaling_NO         :       1.5
Field NAMASK must be defined in section mask of the HEMCO configuration file.

For SOA_SVPOA mechanism:
* If tracers POG1 and POG2 are specified, emissions are calculated from OC,
  multiplied by a POG scale factor (Scaling_POG1, Scaling_POG2) that must be
  specified in the HEMCO configuration file.
* If tracer NAP is specified, emissions are calculated from CO, multiplied
  by a NAP scale factor (Scaling_NAP) that must be specified in the HEMCO
  configuration file.
 References:
 ============================================================================
 (1 ) Original GFED3 database from Guido van der Werf
       http://www.falw.vu/~gwerf/GFED/GFED3/emissions/
```

(2) Giglio, L., Randerson, J. T., van der Werf, G. R., Kasibhatla, P. S., Collatz, G. J., Morton, D. C., and DeFries, R. S.: Assessing variability and long-term trends in burned area by merging multiple satellite fire products, Biogeosciences, 7, 1171-1186, doi:10.5194/bg-7-1171-2010, 2010.
(3) van der Werf, G. R., Randerson, J. T., Giglio, L., Collatz, G. J., Mu, M., Kasibhatla, P. S., Morton, D. C., DeFries, R. S., Jin, Y., and van Leeuwen, T. T.: Global fire emissions and the contribution of deforestation, savanna, forest, agricultural, and peat fires (1997~@~S2009), Atmos. Chem. Phys., 10, 11707-11735, doi:10.5194/acp-10-11707-2010, 2010.

REVISION HISTORY:

```
07 Sep 2011 - P. Kasibhatla - Initial version, based on GFED2
07 Sep 2011 - R. Yantosca   - Added ProTeX headers
14 Feb 2012 - M. Payer      - Add modifications for CH4 (K. Wecht)
01 Mar 2012 - R. Yantosca   - Now reference new grid_mod.F90
06 Mar 2012 - P. Kasibhatla - Final version
01 Aug 2012 - R. Yantosca - Add reference to findFreeLUN from inqure_mod.F90
03 Aug 2012 - R. Yantosca - Move calls to findFreeLUN out of DEVEL block
14 Mar 2013 - M. Payer    - Replace NOx emissions with NO emissions as part
                            of removal of NOx-Ox partitioning
15 Dec 2013 - C. Keller   - Now a HEMCO extension. Emissions in kg/m2/s,
                            emission factors in kg/kgDM.
01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
08 Aug 2014 - R. Yantosca - Now avoid ASCII file reads for ESMF
23 Sep 2014 - C. Keller   - Increase N_SPEC to 26 (+Hg0)
12 Mar 2015 - C. Keller / P. Kasibhatla - Added GFED-4.
03 Jun 2015 - C. Keller / P. Kasibhatla - GFED-4 update: now use GFED-4
                                          specific emission factors and DM data.
14 Oct 2016 - C. Keller    - Now use HCO_EvalFld instead of HCO_GetPtr.
11 Feb 2017 - S. Farina    - Increase N_SPEC to 27 (+SOAP)
23 Mar 2017 - M. Sulprizio - Increase N_SPEC to 29 (+EOH+MTPA)
29 Mar 2018 - K. Travis    - Update GFED4 emission factors, increase to 34 species
29 Mar 2018 - K. Travis    - Remove GFED3
```

2.6.1 HCOX_GFED_Run

Subroutine HcoX_GFED_Run is the driver run routine to calculate seasalt emissions in HEMCO.

INTERFACE:

```
SUBROUTINE HCOX_GFED_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_Calc_Mod,     ONLY : HCO_EvalFld
USE HCO_EmisList_Mod, ONLY : HCO_GetPtr
USE HCO_FluxArr_MOD,  ONLY : HCO_EmisAdd
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root  ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState   ! Output obj
TYPE(Ext_State), POINTER        :: ExtState  ! Module options
INTEGER,         INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
07 Sep 2011 - P. Kasibhatla - Initial version, based on GFED2
15 Dec 2013 - C. Keller     - Now a HEMCO extension
03 Apr 2015 - C. Keller     - Humid tropical forest mask is not binary
                              any more but fraction (0.0 - 1.0).
21 Sep 2016 - R. Yantosca   - Bug fix: move WHERE statement for HUMTROP
                              into the GFED3 block to avoid segfault
10 Mar 2017 - M. Sulprizio  - Add SpcArr3D for emitting 65% of biomass
                              burning emissions into the PBL and 35% into the
                              free troposphere, following code from E.Fischer
24 Apr 2017 - M. Sulprizio  - Comment out vertical distribution of biomass
                              burning emissions for now.
12 May 2017 - M. Sulprizio  - Comment out partitioning of NO directly to PAN
                              and HNO3 for now.
```

2.6.2 HCOX_GFED_Init

Subroutine HcoX_GFED_Init initializes all extension variables.

INTERFACE:

```
SUBROUTINE HCOX_GFED_Init ( am_I_Root, HcoState, ExtName, &
                            ExtState,  RC                 )
```

USES:

```
USE HCO_STATE_MOD,          ONLY : HCO_GetHcoID
USE HCO_STATE_MOD,          ONLY : HCO_GetExtHcoID
USE HCO_ExtList_Mod,        ONLY : GetExtNr, GetExtOpt
USE HCO_ExtList_Mod,        ONLY : GetExtSpcVal
```

INPUT/OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root   ! root CPU?
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName     ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState    ! Options object
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState    ! HEMCO state object
INTEGER,          INTENT(INOUT)  :: RC          ! Return status
```

REVISION HISTORY:

```
07 Sep 2011 - P. Kasibhatla - Initial version, based on GFED2
15 Dec 2013 - C. Keller     - Now a HEMCO extension
08 Aug 2014 - R. Yantosca   - Now include hcox_gfed_include.H, which defines
                              GFED_SPEC_NAME and GFED_EMFAC arrays
11 Nov 2014 - C. Keller     - Now get hydrophilic fractions via config file
22 Apr 2015 - R. Yantosca   - Now explicitly test for "POA scale factor"
                              and "NAP scale factor" to avoid search errors
07 Jan 2016 - M. Sulprizio  - Change 'POA1' to 'POG1' to better reflect that
                              SVOC emissions are added to the gas-phase
                              species in carbon_mod.F
```

2.6.3 HCOX_GFED_Final

Subroutine HcoX_GFED_Final deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_GFED_Final
```

REVISION HISTORY:

```
07 Sep 2011 - P. Kasibhatla - Initial version, based on GFED2
15 Dec 2013 - C. Keller     - Now a HEMCO extension
```

2.7 Fortran: Module Interface hcox_tools_mod.F90

Module HCOX_Tools_Mod contains a collection of helper routines for the HEMCO extensions.

INTERFACE:

```
MODULE HCOX_TOOLS_MOD
```

USES:

```
USE HCO_ERROR_MOD

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_SCALE
!MODULE VARIABLES:
CHARACTER(LEN=31), PARAMETER, PUBLIC  :: HCOX_NOSCALE = 'none'
```

PRIVATE MEMBER FUNCTIONS:

REVISION HISTORY:

```
11 Jun 2015 - C. Keller   - Initial version
```

2.7.1 HCOX_SCALE_sp2D

Applies mask 'SCALENAME' to the passed 2D sp field.

INTERFACE:

```
SUBROUTINE HCOX_SCALE_sp2D( am_I_Root, HcoState, Arr, SCALENAME, RC )
```

USES:

```
USE HCO_CALC_MOD,   ONLY : HCO_EvalFld
USE HCO_STATE_MOD,  ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root  ! Root CPU?
TYPE(HCO_STATE),  POINTER        :: HcoState   ! HcoState obj
CHARACTER(LEN=*), INTENT(IN   )  :: SCALENAME   ! SCALE to be used
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp),         INTENT(INOUT)  :: Arr(:,:)   ! Array to be scaled
INTEGER,          INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
11 Jun 2013 - C. Keller - Initial version
```

2.7.2 HCOX_SCALE_sp3D

Applies mask 'SCALENAME' to the passed 3D sp field.

INTERFACE:

```
SUBROUTINE HCOX_SCALE_sp3D( am_I_Root, HcoState, Arr, SCALENAME, RC )
```

USES:

```
USE HCO_CALC_MOD,   ONLY : HCO_EvalFld
USE HCO_STATE_MOD,  ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root  ! Root CPU?
TYPE(HCO_STATE),  POINTER        :: HcoState   ! HcoState obj
CHARACTER(LEN=*), INTENT(IN   )  :: SCALENAME   ! SCALE to be used
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp),         INTENT(INOUT)  :: Arr(:,:,:) ! Array to be scaled
INTEGER,          INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
11 Jun 2013 - C. Keller - Initial version
```

2.7.3 HCOX_SCALE_dp2D

Applies mask 'SCALENAME' to the passed 2D dp field.

INTERFACE:

```
SUBROUTINE HCOX_SCALE_dp2D( am_I_Root, HcoState, Arr, SCALENAME, RC )
```

USES:

```
USE HCO_CALC_MOD,   ONLY : HCO_EvalFld
USE HCO_STATE_MOD,  ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root  ! Root CPU?
TYPE(HCO_STATE),  POINTER        :: HcoState   ! HcoState obj
CHARACTER(LEN=*), INTENT(IN   )  :: SCALENAME   ! SCALE to be used
```

INPUT/OUTPUT PARAMETERS:

```
REAL(dp),         INTENT(INOUT)  :: Arr(:,:)   ! Array to be scaled
INTEGER,          INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
11 Jun 2013 - C. Keller - Initial version
```

2.7.4 HCOX_SCALE_dp3D

Applies mask 'SCALENAME' to the passed 3D dp field.

INTERFACE:

```
SUBROUTINE HCOX_SCALE_dp3D( am_I_Root, HcoState, Arr, SCALENAME, RC )
```

USES:

```
USE HCO_CALC_MOD,   ONLY : HCO_EvalFld
USE HCO_STATE_MOD,  ONLY : HCO_State
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root  ! Root CPU?
TYPE(HCO_STATE),  POINTER        :: HcoState   ! HcoState obj
CHARACTER(LEN=*), INTENT(IN   )  :: SCALENAME   ! SCALE to be used
```

INPUT/OUTPUT PARAMETERS:

```
REAL(dp),         INTENT(INOUT)  :: Arr(:,:,:) ! Array to be scaled
INTEGER,          INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
11 Jun 2013 - C. Keller - Initial version
```

2.8 Fortran: Module Interface hcox_megan_mod.F90

Module HCOX_Megan_Mod contains variables and routines specifying the algorithms that control the MEGAN inventory of biogenic emissions (as implemented into the GEOS-Chem model).

This is a HEMCO extension module that uses many of the HEMCO core utilities.

MEGAN calculates gamma activity factor based upon temperature and radiation information from the past. In the original GEOS-Chem code, the initial 10-d averages were explicitly calculated during initialization of MEGAN. This is not feasible in an ESMF environment, and the following restart variables can now be provided through the HEMCO configuration file:

- T_DAVG: long-term historical temperature
- PARDR_DAVG: long-term historical direct radiation
- PARDF_DAVG: long-term historical diffuse radiation
- T_PREVDAY: short-term historical temperature

These variables are automatically searched for on the first call of the run call. If not defined, default values will be used. The values of T_DAVG, T_PREVDAY, PARDR_DAVG, and PARDF_DAVG are continuously updated at the end of the run sequence, e.g. they represent the instantaneous running average. The e-folding times to be used when calculating the short=term and long-term running averages are defined as module parameter below (parameter TAU_HOURS and TAU_DAYS).

A similar procedure is also applied to the leaf area index variables. The original GEOS-Chem MEGAN code used three LAI variables: current month LAI (LAI_CM), previous

month LAI (LAI_PM), and instantaneous LAI (LAI), which was a daily interpolation of LAI_CM and next month' LAI, (LAI_NM). The HEMCO implementation uses only the instantaneous LAI, assuming it is updated every day. The short term historical LAI is kept in memory and used to determine the LAI change over time (used to calculate the gamma leaf age). It is also updated on every time step. For the first simulation day, the previous' day LAI is taken from the restart file (field LAI_PREVDAY). If no restart variable is defined, a LAI change of zero is assumed (ckeller, 10/9/2014).

!References:

- Guenther, A., et al., *The Model of Emissions of Gases and Aerosols from Nature version 2.1 (MEGAN2.1): an extended and updated framework for modeling biogenic emissions*, Geosci. Model Dev., **5**, 1471-1792, 2012.
- Guenther, A., et al., *A global model of natural volatile organic compound emissions*, J.Geophys. Res., **100**, 8873-8892, 1995.
- Wang, Y., D. J. Jacob, and J. A. Logan, *Global simulation of tropospheric O3-Nox-hydrocarbon chemistry: 1. Model formulation*, J. Geophys. Res., **103**, D9, 10713-10726, 1998.
- Guenther, A., B. Baugh, G. Brasseur, J. Greenberg, P. Harley, L. Klinger, D. Serca, and L. Vierling, *Isoprene emission estimates and uncertanties for the Central African EXPRESSO study domain*, J. Geophys. Res., **104**, 30,625-30,639, 1999.
- Guenther, A. C., T. Pierce, B. Lamb, P. Harley, and R. Fall, *Natural emissions of non-methane volatile organic compounds, carbon monoxide, and oxides of nitrogen from North America*, Atmos. Environ., **34**, 2205-2230, 2000.
- Guenther, A., and C. Wiedinmyer, *User's guide to Model of Emissions of Gases and Aerosols from Nature*. http://cdp.ucar.edu. (Nov. 3, 2004)
- Guenther, A., *AEF for methyl butenol*, personal commucation. (Nov, 2004)
- Sakulyanontvittaya, T., T. Duhl, C. Wiedinmyer, D. Helmig, S. Matsunaga, M. Potosnak, J. Milford, and A. Guenther, *Monoterpene and sesquiterpene emission estimates for the United States*, Environ. Sci. Technol, **42**, 1623-1629, 2008.

INTERFACE:

```
MODULE HCOX_MEGAN_MOD
```

USES:

```
USE HCO_ERROR_MOD
USE HCO_DIAGN_MOD
USE HCOX_State_MOD,    ONLY : Ext_State
USE HCO_STATE_MOD,     ONLY : HCO_STATE

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCOX_Megan_Init
PUBLIC  :: HCOX_Megan_Run
PUBLIC  :: HCOX_Megan_Final
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: GET_MEGAN_EMISSIONS  ! dbm, new MEGAN driver routine
                                ! for all compounds (6/21/2012)
 PRIVATE :: UPDATE_T_DAY
 PRIVATE :: UPDATE_T_15_AVG
PRIVATE :: GET_MEGAN_PARAMS
PRIVATE :: GET_MEGAN_AEF
PRIVATE :: GET_GAMMA_PAR_PCEEA
PRIVATE :: GET_GAMMA_T_LI
PRIVATE :: GET_GAMMA_T_LD
PRIVATE :: GET_GAMMA_LAI
PRIVATE :: GET_GAMMA_AGE
PRIVATE :: GET_GAMMA_SM
PRIVATE :: CALC_NORM_FAC
PRIVATE :: SOLAR_ANGLE
PRIVATE :: FILL_RESTART_VARS
PRIVATE :: CALC_AEF
PRIVATE :: GET_GAMMA_CO2  ! (Tai, Jan 2013)
```

REVISION HISTORY:

```
(1 ) Original code (biogen_em_mod.f) by Dorian Abbot (6/2003).  Updated to
     latest algorithm and modified for the standard code by May Fu
     (11/2004).
(2 ) All emission are currently calculated using TS from DAO met field.
     TS is the surface air temperature, which should be carefully
     distinguished from TSKIN. (tmf, 11/20/2004)
(3 ) In GEOS4, the TS used here are the T2M in the A3 files, read in
     'a3_read_mod.f'.
(4 ) Bug fix: change #if block to also cover GCAP met fields (bmy, 12/6/05)
(5 ) Remove support for GEOS-1 and GEOS-STRAT met fields (bmy, 8/4/06)
(6 ) Bug fix: Skip Feb 29th if GCAP in INIT_MEGAN (phs, 9/18/07)
(7 ) Added routine GET_AEF_05x0666 to read hi-res AEF data for the GEOS-5
     0.5 x 0.666 nested grid simulations (yxw, dan, bmy, 11/6/08)
17 Dec 2009 - R. Yantosca - Added ProTeX headers
09 Mar 2010 - R. Yantosca - Minor bug fix in GET_EMMONOT_MEGAN
17 Mar 2010 - H. Pye      - AEF_SPARE must be a scalar local variable
                            in GET_EMMONOT_MEGAN for parallelization.
20 Aug 2010 - R. Yantosca - Move CMN_SIZE to top of module
20 Aug 2010 - R. Yantosca - Now set DAY_DIM = 24 for MERRA, since the
                            surface temperature is now an hourly field.
01 Sep 2010 - R. Yantosca - Bug fix in INIT_MEGAN: now only read in
```

```
                              NUM_DAYS (instead of 15) days of sfc temp data
    22 Nov 2011 - R. Yantosca - Do not use erroneous AEF's for nested grids
    06 Dec 2011 - E. Fischer  - Added Acetone emissions
    28 Feb 2012 - R. Yantosca - Removed support for GEOS-3
    01 Mar 2012 - R. Yantosca - Now reference new grid_mod.F90
    01 Mar 2012 - R. Yantosca - Use updated GET_LOCALTIME from time_mod.F
    11 Apr 2012 - R. Yantosca - Replace lai_mod.F with modis_lai_mod.F90
    13 Aug 2013 - M. Sulprizio- Modifications for updated SOA sim (H. Pye):
                                 Add sesquiterpenes to MEGAN group;
                                 Add plant functional types (PFT_xx);
                                 Rename GET_EMMONOG_MEGAN to GET_EMTERP_MEGAN;
                                 Add routines READ_PFT and GET_AEF_GEN
    20 Aug 2013 - R. Yantosca - Removed "define.h", this is now obsolete
    26 Sep 2013 - R. Yantosca - Renamed GEOS_57 Cpp switch to GEOS_FP
    05 Oct 2013 - C. Keller   - Now a HEMCO extension
    04 Aug 2014 - C. Keller   - Added 'manual' diagnostics for Acetone.
    09 Oct 2014 - C. Keller   - Now use only GC_LAI (keep prev. LAI in memory)
    22 Dec 2014 - C. Keller   - Now use flexible precision (hp) everywhere.
                                Option to read temperature/irradiation from
                                restart.
    26 Jan 2015 - M. Sulprizio- Update from D. Millet (19 Jan 2013): Streamlined
                                computations into a single driver routine
                                and updated emissions according to MEGAN 2.1 as
                                described in:
                                Guenther et al., The Model of Emissions of Gases
                                and Aerosols from Nature version 2.1 (MEGAN2.1):
                                an extended and updated framework for modeling
                                biogenic emissions, GMD, 5, 1471-1492, 2012.
    12 Feb 2015 - M. Sulprizio- Remove GET_AEF_GEN routine. We now calculate AEFs
                                for FARN, BCAR, and OSQT in CALC_AEF using
                                parameters from Guenther et al., 2012.
    18 Feb 2015 - M. Sulprizio- Remove LPECCA logical flag since we use this
                                scheme exclusively now.
                                Restore emissions of individual MEGAN species to
                                diagnostics for consistency with pre-HEMCO code.
    10 Jun 2015 - M. Sulprizio- Bug fix for SOA simulation: Now convert AEFs for
                                sesquiterpenes to kg/m2/s.
    15 Sep 2015 - M. Sulprizio- Add CO2 inhibition effect on isoprene emissions
                                from Amos Tai (Jan 2013)
    05 Nov 2015 - C. Keller   - Reorganize restart variables to running averages.
    08 Dec 2015 - C. Keller   - Now treat previous' day LAI as running avg, too.
    14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
    05 Oct 2015 - M. Sulprizio- Activate MEGAN ethanol emissions for PAN updates
                                from E. Fischer
    17 Jul 2017 - C. Keller   - Now normalize LAI by PFTs.
```

2.8.1 HCOX_Megan_Run

Subroutine HCOX_MEGAN_Run is the driver routine for the MEGAN model within the new emissions structure. Note that all diagnostics are commented since those are still written as part of the old emission structure.

INTERFACE:

```
SUBROUTINE HCOX_Megan_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_FLUXARR_MOD,     ONLY : HCO_EmisAdd
USE HCO_CLOCK_MOD,       ONLY : HcoClock_First
USE HCO_CLOCK_MOD,       ONLY : HcoClock_Rewind
USE HCO_CLOCK_MOD,       ONLY : HcoClock_NewHour
USE HCO_CLOCK_MOD,       ONLY : HcoClock_NewDay
USE HCO_Restart_Mod,     ONLY : HCO_RestartWrite
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN)    :: am_I_Root
TYPE(Ext_State), POINTER       :: ExtState
TYPE(HCO_State), POINTER       :: HcoState
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
05 Aug 2013 - C. Keller   - Initial version
08 Oct 2014 - C. Keller   - Now use HEMCO clock to get days between months
09 Oct 2014 - C. Keller   - Now use only GC_LAI. This makes days between months
                            obsolete
16 Oct 2014 - C. Keller   - Initialize flux arrays to avoid float invalid
 9 Mar 2015 - R. Yantosca - Bug fix: add EMIS_ALD2 to $OMP PRIVATE clause
08 May 2015 - C. Keller   - Now read/write restart variables from here to
                            accomodate replay runs in GEOS-5.
30 Sep 2015 - C. Keller   - Now add OCPI flux to BIOGENIC_OCPI diagnostics.
12 Dec 2015 - C. Keller   - Apply e-folding times to past conditions.
07 Jan 2016 - E. Lundgren - Update Avogadro's # to NIST 2014
01 Jun 2016 - R. Yantosca - Bug fix: Now use ExtNrMono in the call to
                            Hco_EmisAdd for the MONX (in order to make MONX
                            show up as nonzero in HEMCO diagnostics).
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
02 Mar 2018 - M. Sulprizio- Remove species MONX and use MTPA/MTPO/LIMO;
                            Consolidate MEGAN_SOA option into MEGAN_Mono
                            since there is essentially no difference between
                            the two now that SOA (MTPA/MTPO/LIMO) is standard
```

2.8.2 Get_Megan_Emissions

Subroutine Get_Megan_Emissions computes biogenic emissions in units of [kgC/m2/s] or [kg/m2/s] using the MEGAN inventory. (dbm, 12/2012)

INTERFACE:

```
SUBROUTINE GET_MEGAN_EMISSIONS( am_I_Root, HcoState, ExtState,
&                               Inst, I, J, CMPD, MEGAN_EMIS, RC )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)  :: am_I_Root
TYPE(HCO_STATE),  POINTER     :: HcoState
TYPE(Ext_State),  POINTER     :: ExtState
TYPE(MyInst),     POINTER     :: Inst
INTEGER,          INTENT(IN)  :: I, J      ! GEOS-Chem lon & lat indices
CHARACTER(LEN=*), INTENT(IN)  :: CMPD      ! Compound name (dbm,6/21/2012)
```

OUTPUT PARAMETERS:

```
REAL(hp),         INTENT(OUT) :: MEGAN_EMIS ! VOC emission in kgC/m2/s
                                            ! or kg/m2/s, depending on
                                            ! units the compound is
                                            ! carried in
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC
```

REMARKS:

```
References (see above for full citations):
============================================================================
(1 ) Guenther et al, 1995, 1999, 2000, 2004, 2006
(2 ) Wang,    et al, 1998
(3 ) Guenther et al, 2007, MEGAN v2.1 User mannual
(4 ) Guenther et al, 2012 GMD MEGANv2.1 description and associated code at
                           http://acd.ucar.edu/~guenther/MEGAN/
```

REVISION HISTORY:

```
(1 ) Original code by Dorian Abbot (9/2003).  Updated to the latest
     algorithm and modified for the standard code by May Fu (11/20/04)
(2 ) All MEGAN biogenic emission are currently calculated using TS from DAO
     met field. TS is the surface air temperature, which should be
     carefully distinguished from TSKIN. (tmf, 11/20/04)
(3 ) Restructing of function & implementation of activity factors (mpb,2009)
17 Dec 2009 - R. Yantosca - Added ProTeX headers
11 Apr 2012 - R. Yantosca - Now use data from modis_lai_mod.F90
11 Apr 2012 - R. Yantosca - Cosmetic changes
```

```
26 Jan 2015 - M. Sulprizio- Update from D. Millet (21 Jun 2012): New driver
                            routine for all MEGAN compounds
15 Sep 2015 - M. Sulprizio- Add CO2 inhibition effect on isoprene emissions
                            from Amos Tai (Jan 2013)
17 Jul 2017 - C. Keller   - Now normalize LAI by PFT's.
```

2.8.3 Get_Megan_Params

Subroutine Get_Megan_Params returns the emission parameters for each MEGAN compound needed to compute emissions. Called from GET_MEGAN_EMISSIONS.

INTERFACE:

```
 SUBROUTINE GET_MEGAN_PARAMS( am_I_Root, HcoState,
&                             CPD,   BTA,   LIDF,  C_T1,  C_EO,
&                             A_NEW, A_GRO, A_MAT, A_OLD, BI_DIR,
&                             RC )
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN) :: am_I_Root ! Root CPU?
TYPE(HCO_State),  POINTER    :: HcoState
CHARACTER(LEN=*), INTENT(IN) :: CPD       ! Compound name
```

INPUT/OUTPUT PARAMETERS:

```
REAL(hp), INTENT(INOUT) :: BTA    ! Beta coefficient for temperature activity
                                  ! factor for light-independent fraction
REAL(hp), INTENT(INOUT) :: LIDF   ! Light-dependent fraction of emissions
REAL(hp), INTENT(INOUT) :: C_T1   ! CT1 parameter for temperature activity
                                  ! factor for light-dependent fraction
REAL(hp), INTENT(INOUT) :: C_EO   ! Ceo parameter for temperature activity
                                  ! factor for light-dependent fraction
REAL(hp), INTENT(INOUT) :: A_NEW  ! Relative emission factor (new leaves)
REAL(hp), INTENT(INOUT) :: A_GRO  ! Relative emission factor (growing leaves)
REAL(hp), INTENT(INOUT) :: A_MAT  ! Relative emission factor (mature leaves)
REAL(hp), INTENT(INOUT) :: A_OLD  ! Relative emission factor (old leaves)
LOGICAL,  INTENT(INOUT) :: BI_DIR ! Logical flag to indicate bidirectional exchange
INTEGER,  INTENT(INOUT) :: RC
```

REMARKS:

```
References (see above for full citations):
============================================================================
(1 ) Guenther et al, (GMD 2012) and associated MEGANv2.1 source code
```

REVISION HISTORY:

```
(1 ) Created by dbm 07/2012
```

2.8.4 Get_Megan_AEF

Function Get_Megan_AEF returns the appropriate AEF value for a given compound and grid square.

INTERFACE:

```
SUBROUTINE GET_MEGAN_AEF(am_I_Root,HcoState,Inst,I,J,CPD,EMFAC,RC)
```

INPUT PARAMETERS:

```
LOGICAL,           INTENT(IN)  :: am_I_Root   ! Root CPU?
TYPE(HCO_State),   POINTER     :: HcoState
TYPE(MyInst),      POINTER     :: Inst
INTEGER,           INTENT(IN)  :: I, J        ! Lon & lat indices
CHARACTER(LEN=*),  INTENT(IN)  :: CPD         ! Compound name
```

OUTPUT PARAMETERS:

```
REAL(hp),          INTENT(OUT) :: EMFAC       ! MEGAN base emission factor
                                              ! (kgC/m2/s or kg/m2/s)
                                              ! for grid cell (I,J)
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,           INTENT(INOUT) :: RC        ! Return code
```

REMARKS:

```
References (see above for full citations):
============================================================================
(1 ) Guenther et al, 2012, MEGANv2.1 source code
```

REVISION HISTORY:

```
(1 ) Created 11/2012 by dbm
```

2.8.5 Get_Gamma_PAR_PCEEA

Computes the PCEEA gamma activity factor with sensitivity to LIGHT.

INTERFACE:

```
 FUNCTION GET_GAMMA_PAR_PCEEA( am_I_Root, HcoState, ExtState,
&                              Inst, I, J, Q_DIR_2,
&                              Q_DIFF_2, PARDR_AVG_SIM,
&                              PARDF_AVG_SIM )
&              RESULT( GAMMA_P_PCEEA )
```

USES:

```
      USE HCO_CLOCK_MOD, ONLY : HcoClock_Get, HcoClock_GetLocal
```

INPUT PARAMETERS:

```
      LOGICAL,         INTENT(IN) :: am_I_Root
      TYPE(HCO_State), POINTER    :: HcoState
      TYPE(Ext_State), POINTER    :: ExtState
      TYPE(MyInst),    POINTER    :: Inst
      INTEGER,         INTENT(IN) :: I,  J            ! Lon & lat indices
      REAL(sp),        INTENT(IN) :: PARDR_AVG_SIM    ! Avg direct PAR [W/m2]
      REAL(sp),        INTENT(IN) :: PARDF_AVG_SIM    ! Avg diffuse PAR [W/m2]
      REAL(hp),        INTENT(IN) :: Q_DIR_2          ! Direct PAR [umol/m2/s]
      REAL(hp),        INTENT(IN) :: Q_DIFF_2         ! Diffuse PAR [umol/m2/s]
```

RETURN VALUE:

```
      REAL(hp)                    :: GAMMA_P_PCEEA    ! GAMMA factor for light
```

REMARKS:

```
    References (see above for full citations):
    ============================================================================
    (1 ) Guenther et al, 2006
    (2 ) Guenther et al, 2007, MEGAN v2.1 user guide
```

REVISION HISTORY:

```
    (1 ) Here PAR*_AVG_SIM is the average light conditions over the simulation
         period. I've set this = 10 days to be consistent with temperature & as
         outlined in Guenther et al, 2006. (mpb,2009)
    (2 ) Code was taken & adapted directly from the MEGAN v2.1 source code.
         (mpb,2009)
    17 Dec 2009 - R. Yantosca - Added ProTeX headers
    01 Mar 2012 - R. Yantosca - Now use GET_YMID(I,J,L) from grid_mod.F90
    01 Mar 2012 - R. Yantosca - Now use GET_LOCALTIME(I,J,L) from time_mod.F90
```

2.8.6 Solar_Angle

Function SOLAR_ANGLE computes the local solar angle for a given day of year, latitude and longitude (or local time). Called from routine Get_Gamma_P_Pecca.

INTERFACE:

```
      FUNCTION SOLAR_ANGLE( HcoState, Inst, DOY, SHOUR, LAT )
     & RESULT(SINbeta)
```

INPUT PARAMETERS:

```
! Arguments
TYPE(HCO_State),   POINTER    :: HcoState
TYPE(MyInst),      POINTER    :: Inst
INTEGER,           INTENT(IN) :: DOY       ! Day of year
REAL(hp),          INTENT(IN) :: SHOUR     ! Local time
REAL(hp),          INTENT(IN) :: LAT       ! Latitude
```

RETURN VALUE:

```
REAL(hp)                      :: SINbeta   ! Sin of the local solar angle
```

REMARKS:

```
References (see above for full citations):
(1 ) Guenther et al, 2006
(2 ) Guenther et al, MEGAN v2.1 user mannual 2007-09
```

REVISION HISTORY:

```
(1 ) This code was taken directly from the MEGAN v2.1 source code.(mpb,2009)
17 Dec 2009 - R. Yantosca - Added ProTeX headers
```

2.8.7 Get_Gamma_T_LI

Function Get_Gamma_T_LI computes the temperature activity factor (GAMMA_T_LI) for the light-independent fraction of emissions

INTERFACE:

```
FUNCTION GET_GAMMA_T_LI( T, BETA ) RESULT( GAMMA_T_LI )
```

INPUT PARAMETERS:

```
! Current leaf temperature, the surface air temperature field (TS)
! is assumed equivalent to the leaf temperature over forests.
REAL(hp),  INTENT(IN) :: T

! Temperature factor per species
REAL(hp),  INTENT(IN) :: BETA
```

RETURN VALUE:

```
! Activity factor for the light-independent fraction of emissions
REAL(hp)              :: GAMMA_T_LI
```

REMARKS:

```
GAMMA_T =  exp[Beta*(T - T_Standard)]

           where Beta   = temperature dependent parameter
                 Ts     = standard temperature (normally 303K, 30C)
```

```
References (see above for full citations):
=============================================================================
(1 ) Guenther et al, 2006
(2 ) Guenther et al, MEGAN user mannual 2007-08
(3 ) Guenther et al., GMD 2012 and MEGANv2.1 source code.
```

REVISION HISTORY:

```
(1 ) Original code by Michael Barkley (2009).
     Note: If T = Ts  (i.e. standard conditions) then GAMMA_T = 1
17 Dec 2009 - R. Yantosca - Added ProTeX headers
(2 ) Modified to GAMMA_T_LI (dbm, 6/21/2012)
```

2.8.8 Get_Gamma_T_LD

Function Get_Gamma_T_LD computes the temperature sensitivity for the light-dependent fraction of emissions.

INTERFACE:

```
      FUNCTION GET_GAMMA_T_LD( T, PT_15, PT_1, CT1, CEO )
     &     RESULT( GAMMA_T_LD )
```

INPUT PARAMETERS:

```
      ! Current leaf temperature [K], the surface air temperature field (TS)
      ! is assumed equivalent to the leaf temperature over forests.
      REAL(hp), INTENT(IN) :: T

       ! Average leaf temperature over the past 15 days
      REAL(sp), INTENT(IN) :: PT_15

      ! Average leaf temperature over the past arbitray day(s).
      ! This is not used at present
      REAL(sp), INTENT(IN) :: PT_1

      ! Compound-specific parameters for light-dependent temperature activity
      ! factor (dbm, 6/21/2012)
      REAL(hp), INTENT(IN) :: CT1, CEO
```

RETURN VALUE:

```
      ! Temperature activity factor for the light-dependent fraction of
      ! emissions
      REAL(hp)             :: GAMMA_T_LD
```

REMARKS:

```
References (see above for full citations):
(1 ) Guenther et al, 1995
(2 ) Guenther et al, 2006
(3 ) Guenther et al, MEGAN v2.1 user mannual 2007-08
(4 ) Guenther et al., GMD 2012 and MEGANv2.1 source code.
```

REVISION HISTORY:

```
(1 ) Includes the latest MEGAN v2.1 temperature algorithm (mpb, 2009).
     Note, this temp-dependence is the same for the PCEEA & hybrid models.
17 Dec 2009 - R. Yantosca - Added ProTeX headers
(2 ) Modified to gamma_t_ld and to permit compound specific parameters
     CT1 and Ceo (dbm, 6/21/2012)
07 Jan 2016 - Update ideal gas constant to NIST 2014 value
```

2.8.9 Get_Gamma_Lai

Function Get_Gamma_Lai computes the gamma exchange activity factor which is sensitive to leaf area (= GAMMA_LAI).

INTERFACE:

```
      FUNCTION GET_GAMMA_LAI( CMLAI, BIDIREXCH )
     &   RESULT( GAMMA_LAI )
```

INPUT PARAMETERS:

```
      REAL(hp),     INTENT(IN) :: CMLAI       ! Current month's LAI [cm2/cm2]
      LOGICAL,      INTENT(IN) :: BIDIREXCH   ! Logical flag indicating whether
                                              ! the compound undergoes bidirectional
                                              ! exchange
```

RETURN VALUE:

```
      REAL(hp)             :: GAMMA_LAI
```

REMARKS:

```
References (see above for full citations):
=============================================================================
(1 ) Guenther et al, 2006
(2 ) Guenther et al, MEGAN user mannual 2007-08
(3 ) Guenther et al., GMD 2012 and MEGANv2.1 source code.
```

REVISION HISTORY:

```
(1 ) Original code by Dorian Abbot (9/2003).  Modified for the standard
      code by May Fu (11/2004)
(2 ) Update to publically released (as of 11/2004) MEGAN algorithm and
      modified for the standard code by May Fu (11/2004).
(3 ) Algorithm is based on the latest MEGAN v2.1 User's Guide (mpb,2009)
(4 ) Updated to treat bidirectional exchange compounds appropriately (dbm, 6/2012)
17 Dec 2009 - R. Yantosca - Added ProTeX headers
```

2.8.10 Get_Gamma_Age

Function Get_Gamma_Age computes the gamma exchange activity factor which is sensitive to leaf age (= Gamma_Age).

INTERFACE:

```
 FUNCTION GET_GAMMA_AGE( CMLAI, PMLAI, DBTWN, TT,
&                        AN, AG, AM, AO )
&         RESULT( GAMMA_AGE )
```

INPUT PARAMETERS:

```
 REAL(hp), INTENT(IN) :: CMLAI     ! Current month's LAI [cm2/cm2]
 REAL(hp), INTENT(IN) :: PMLAI     ! Previous months LAI [cm2/cm2]
 REAL(hp), INTENT(IN) :: DBTWN     ! Number of days between
 REAL(sp), INTENT(IN) :: TT        ! Daily average temperature [K]
 REAL(hp), INTENT(IN) :: AN        ! Relative emiss factor (new leaves)
 REAL(hp), INTENT(IN) :: AG        ! Relative emiss factor (growing leaves)
 REAL(hp), INTENT(IN) :: AM        ! Relative emiss factor (mature leaves)
 REAL(hp), INTENT(IN) :: AO        ! Relative emiss factor (old leaves)
```

RETURN VALUE:

```
 REAL(hp)             :: GAMMA_AGE ! Activity factor
```

REMARKS:

```
References (see above for full citations):
=============================================================================
(1 ) Guenther et al, 2006
(2 ) Guenther et al, MEGAN user mannual 2007-08
(3 ) Guenther et al., GMD 2012 and MEGANv2.1 source code
```

REVISION HISTORY:

```
(1 ) Original code by Dorian Abbot (9/2003). Modified for the standard
     code by May Fu (11/2004)
(2 ) Update to publically released (as of 11/2004) MEGAN algorithm and
     modified for the standard code by May Fu (11/2004).
(3 ) Algorithm is based on the latest User's Guide (tmf, 11/19/04)
(4 ) Renamed & now includes specific relative emission activity factors for
     each BVOC based on MEGAN v2.1 algorithm (mpb,2008)
(5 ) Now calculate TI (number of days after budbreak required to induce
     iso. em.) and TM (number of days after budbreak required to reach
     peak iso. em. rates) using the daily average temperature, instead
     of using fixed values (mpb,2008)
     NOTE: Can create 20% increases in tropics (Guenther et al 2006)
(6 ) Implemented change for the calculation of FGRO if ( CMLAI > PMLAI ),
```

```
      i.e. if LAI has increased with time, and used new values for
      all foilage fractions if ( CMLAI = PMLAI ). Also removed TG variable
      as not now needed. (mpb,2000)
(7 ) Updated to pass leaf age activity factors as arguments (dbm, 6/2012)
17 Dec 2009 - R. Yantosca - Added ProTeX headers
13 Aug 2013 - M. Sulprizio- Updated for sesquiterpenes (H. Pye)
```

2.8.11 get_gamma_sm

Function GET_GAMMA_SM computes activity factor for soil moisture

INTERFACE:

```
      FUNCTION GET_GAMMA_SM( ExtState, I, J, CMPD )
     &            RESULT( GAMMA_SM )
```

INPUT PARAMETERS:

```
      TYPE(Ext_State),  POINTER     :: ExtState
      INTEGER,          INTENT(IN)  :: I, J     ! GEOS-Chem lon & lat indices
      CHARACTER(LEN=*), INTENT(IN)  :: CMPD     ! Compound name (dbm, 6/21/2012)
```

RETURN VALUE:

```
      REAL(hp)                      :: GAMMA_SM ! Activity factor
```

REMARKS:

```
   References (see above for full citations):
   ============================================================================
   (1 ) Guenther et al, ACP 2006
   (2 ) Guenther et al., GMD 2012 and MEGANv2.1 source code
```

REVISION HISTORY:

```
   (1 ) Created by dbm (6/2012). We are not currently using a soil moisture
        effect for isoprene. For all compounds other than acetaldehyde and
        ethanol, gamma_sm =1 presently.
   16 Apr 2015 - C. Keller   - Now restrict GWETROOT to values between 0.0 and
                               1.0. This only seems to be a problem within the
                               GEOS-5 ESM, where GWETROOT values over the ocean
                               become 1e+15 (= missing value).
   12 Aug 2015 - R. Yantosca - Extend #ifdef for MERRA2 meteorology

  ---------------------------------------------------------------------------
  BOC
```

LOCAL VARIABLES:

```
       REAL(hp)  :: GWETROOT

       !=================================================================
       ! GET_GAMMA_SM begins here!
       !=================================================================

       ! By default gamma_sm is 1.0
       GAMMA_SM = 1.0_hp

       ! Error trap: GWETROOT must be between 0.0 and 1.0 (ckeller, 4/16/15)
       GWETROOT = MIN(MAX(ExtState%GWETROOT%Arr%Val(I,J),0.0_hp),1.0_hp)

       IF ( TRIM( CMPD ) == 'ALD2' .OR. TRIM ( CMPD ) == 'EOH' ) THEN

          ! GWETROOT = degree of saturation or wetness in the root-zone
          ! (top meter of soil). This is defined as the ratio of the volumetric
          ! soil moisture to the porosity. We use a soil moisture activity factor
          ! for ALD2 to account for stimulation of emission by flooding.
          ! (Millet et al., ACP 2010)
          ! Constant value of 1.0 for GWETROOT = 0-0.9, increasing linearly to
          ! 3.0 at GWETROOT =1.
          GAMMA_SM = MAX( 20.0_hp * GWETROOT - 17.0_hp, 1.0_hp)

       ENDIF

       ! return to calling program
       END FUNCTION GET_GAMMA_SM

  EOC
  ------------------------------------------------------------------------------
                   Harvard-NASA Emissions Component (HEMCO)                   !
  ------------------------------------------------------------------------------
%///////////////////////////////////////////////////////////////

\mbox{}\hrulefill\


  \subsubsection [get\_gamma\_co2] {get\_gamma\_co2}


  Function GET\_GAMMA\_CO2 computes the CO2 activity factor
    associated with CO2 inhibition of isoprene emission. Called from
    GET\_MEGAN\_EMISSIONS only.
  \\
  \\{\bf INTERFACE:}
\begin{verbatim}       FUNCTION GET_GAMMA_CO2( CO2a ) RESULT( GAMMA_CO2 )
```

INPUT PARAMETERS:

```
REAL(hp), INTENT(IN) :: CO2a       ! Atmospheric CO2 conc [ppmv]
```

RETURN VALUE:

```
REAL(hp)             :: GAMMA_CO2  ! CO2 activity factor [unitless]
```

LOCAL VARIABLES:

```
REAL(hp)             :: CO2i       ! Intercellular CO2 conc [ppmv]
REAL(hp)             :: ISMAXi     ! Asymptote for intercellular CO2
REAL(hp)             :: HEXPi      ! Exponent for intercellular CO2
REAL(hp)             :: CSTARi     ! Scaling coef for intercellular CO2
REAL(hp)             :: ISMAXa     ! Asymptote for atmospheric CO2
REAL(hp)             :: HEXPa      ! Exponent for atmospheric CO2
REAL(hp)             :: CSTARa     ! Scaling coef for atmospheric CO2
LOGICAL              :: LPOSSELL   ! Use Possell & Hewitt (2011)?
LOGICAL              :: LWILKINSON ! Use Wilkinson et al. (2009)?
```

REMARKS:

```
References:
============================================================================
(1 ) Heald, C. L., Wilkinson, M. J., Monson, R. K., Alo, C. A.,
     Wang, G. L., and Guenther, A.: Response of isoprene emission
     to ambient co(2) changes and implications for global budgets,
     Global Change Biology, 15, 1127-1140, 2009.
(2 ) Wilkinson, M. J., Monson, R. K., Trahan, N., Lee, S., Brown, E.,
     Jackson, R. B., Polley, H. W., Fay, P. A., and Fall, R.: Leaf
     isoprene emission rate as a function of atmospheric CO2
     concentration, Global Change Biology, 15, 1189-1200, 2009.
(3 ) Possell, M., and Hewitt, C. N.: Isoprene emissions from plants
     are mediated by atmospheric co2 concentrations, Global Change
     Biology, 17, 1595-1610, 2011.
```

REVISION HISTORY:

```
(1 ) Implemented in the standard code by A. Tai (Jun 2012).
15 Sep 2015 - M. Sulprizio- Implemented into hcox_megan_mod.F
```

2.8.12 CALC_NORM_FAC

Function CALC_NORM_FAC calculates the normalization factor needed to compute emissions. Called from GET_MEGAN_EMISSIONS.

INTERFACE:

```
SUBROUTINE CALC_NORM_FAC( am_I_Root, Inst, RC )
```

INPUT PARAMETERS:

```
      LOGICAL,          INTENT(IN)     :: am_I_Root
      TYPE(MyInst),     POINTER        :: Inst
   !INPUT/OUTPUT PARAMETERS
      INTEGER,          INTENT(INOUT)  :: RC
```

REMARKS:

```
References (see above for full citations):
=============================================================================
(1 ) Guenther et al, (GMD 2012) and associated MEGANv2.1 source code
```

REVISION HISTORY:

```
(1 ) Created by dbm 11/2012. We calculate only 1 normalization factor for all
     compounds based on the isoprene gamma values. Formally there should be a
     different normalization factor for each compound, but we are following
     Alex Guenther's approach here and the MEGAN source code.
     "Hi Dylan, sorry for being so slow to get back to you.
      Since the change is only a few percent or less, I didn't
      bother to assign a different normalization factor to each
      compound.  Since the MEGAN canopy environment model also
      has 8 different canopy types (tropical broadleaf tree,
      conifer tree, etc.) then to be correct we should have a
      different CCE for each canopy type for each compound class
      (which would be 160 slightly different values of CCE)."
07 Jan 2016 - E. Lundgren - Update ideal gas constant to NIST 2014 value
```

2.8.13 Fill_Restart_Vars

Subroutine FILL_RESTART_VARS fills the megan restart variables.

INTERFACE:

```
      SUBROUTINE FILL_RESTART_VARS( am_I_Root, HcoState,
     &                              ExtState, Inst, RC )
```

USES:

```
      USE HCO_Restart_Mod,    ONLY : HCO_RestartGet
```

INPUT PARAMETERS:

```
      LOGICAL,          INTENT(IN   )  :: am_I_Root
      TYPE(HCO_State), POINTER         :: HcoState
      TYPE(Ext_State), POINTER         :: ExtState
      TYPE(MyInst),     POINTER        :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
      INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
19 Dec 2014 - C. Keller    - Initial version
09 Mar 2015 - C. Keller    - Now use HCO_RestartGet to fill restarts
```

2.8.14 CALC_AEF

Subroutine CALC_AEF reads Emission Factors for all biogenic VOC species from disk.

INTERFACE:

```
SUBROUTINE CALC_AEF( am_I_Root, HcoState, ExtState, Inst, RC )
```

USES:

```
! References to F90 modules
USE HCO_EMISLIST_MOD,    ONLY : HCO_GetPtr
USE HCO_CALC_MOD,        ONLY : HCO_EvalFld
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN)    :: am_I_Root
TYPE(Ext_State), POINTER       :: ExtState
TYPE(HCO_State), POINTER       :: HcoState
TYPE(MyInst),    POINTER       :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC
```

REMARKS:

```
Reference: (5 ) Guenther et al, 2004
```

REVISION HISTORY:

```
(1 ) Original code by Dorian Abbot (9/2003).  Modified for the standard
     code by May Fu (11/2004)
(2 ) AEF detailed in the latest MEGAN User's Guide (tmf, 11/19/04)
(3 ) Bug fix (tmf, 11/30/04)
(4 ) Now reads 1x1 files and regrids to current resolution (bmy, 10/24/05)
(5 ) Uses new v2.1 emission factors maps for isoprene, MBO and 7 monoterpene
   species, download in 2009. (mpb,2009)
(6 ) Now use 2.1 emission factors for isoprene, MBO, and 7 monoterpenes. EFs for
     other compounds are computed by reading in the PFT fractions and multiplying
     the fractions by corresponding EF values. (dbm, 11/2012)
(7 ) Also, now read in the EF values already gridded to model resolution.
     (dbm, 11/2012)
17 Dec 2009 - R. Yantosca - Added ProTeX headers
10 Jun 2015 - M. Sulprizio- Bug fix: Now convert sesquiterpenes (FARN, BCAR,
                            OSQT) from ug compound/m2/h to kg C/m2/s
17 Jul 2017 - C. Keller   - PFT values (ARRAY_16) are now stored in Inst object
```

2.8.15 HCOX_Megan_Init

Subroutine HCOX_Megan_Init allocates and initializes all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_Megan_Init( am_I_Root, HcoState, ExtName,
&                           ExtState,    RC               )
```

USES:

```
USE HCO_STATE_MOD,    ONLY : Hco_GetHcoID
USE HCO_STATE_MOD,    ONLY : HCO_GetExtHcoID
USE HCO_ExtList_Mod,  ONLY : GetExtNr, GetExtOpt
USE HCO_Restart_Mod,  ONLY : HCO_RestartDefine
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
TYPE(HCO_State),  POINTER        :: HcoState
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName
TYPE(Ext_State),  POINTER        :: ExtState
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT)           :: RC
```

REVISION HISTORY:

```
(1 ) Change the logic in the #if block for G4AHEAD. (bmy, 12/6/05)
(2 ) Bug fix: skip Feb 29th if GCAP (phs, 9/18/07)
(3 ) Now call GET_AEF_05x0666 for GEOS-5 nested grids (yxw,dan,bmy, 11/6/08)
17 Dec 2009 - R. Yantosca - Added ProTeX headers
26 Aug 2010 - R. Yantosca - Now reference merra_a1_mod.f
01 Sep 2010 - R. Yantosca - Now read in NUM_DAYS of sfc temp data (this had
                            been hardwired to 15 days previously)
07 Feb 2011 - R. Yantosca - Fix typos: make sure to zero out the proper
                            PARDF_* and PARDR_* arrays after allocation
22 Nov 2011 - R. Yantosca - Do not use erroneous AEF's for nested grids
08 Feb 2012 - R. Yantosca - Now read surface temperature for GEOS-5.7.x
28 Feb 2012 - R. Yantosca - Removed support for GEOS-3
11 Apr 2012 - R. Yantosca - Now remove the call to INIT_LAI; we shall
                            initialize the LAI arrays from main.F
03 Aug 2012 - R. Yantosca - Move calls to findFreeLUN out of DEVEL block
11 Apr 2013 - R. Yantosca - Now pass directory info with Input_Opt
18 Feb 2015 - M. Sulprizio- Now allocate AEF arrays for species not read
                            from file
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
17 Jul 2017 - C. Keller   - Added optional setting NORMLAI
```

2.8.16 HCOX_Megan_Final

Subroutine HCOX_Megan_Final deallocates all allocated arrays at the end of a GEOS-Chem model run.

INTERFACE:

```
      SUBROUTINE HCOX_MEGAN_FINAL ( am_I_Root, HcoState, ExtState, RC )
   !USES
```

INPUT PARAMETERS:

```
      LOGICAL,         INTENT(IN   )  :: am_I_Root     ! Root CPU?
      TYPE(HCO_State), POINTER        :: HcoState      ! HEMCO State obj
      TYPE(Ext_State), POINTER        :: ExtState      ! Extension State obj
```

INPUT/OUTPUT PARAMETERS:

```
      INTEGER,         INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
    17 Dec 2009 - R. Yantosca  - Added ProTeX headers
    16 Aug 2013 - C. Keller    - Now a HEMCO extension
    18 Feb 2015 - M. Sulprizio - Added AEF arrays
    09 Mar 2015 - C. Keller    - Write variables to internal state in ESMF mode
```

2.8.17 InstGet

Subroutine InstGet returns a pointer to the desired instance.

INTERFACE:

```
      SUBROUTINE InstGet ( Instance, Inst, RC, PrevInst )
```

INPUT PARAMETERS:

```
      INTEGER                             :: Instance
      TYPE(MyInst),     POINTER           :: Inst
      INTEGER                             :: RC
      TYPE(MyInst),     POINTER, OPTIONAL :: PrevInst
```

REVISION HISTORY:

```
    18 Feb 2016 - C. Keller   - Initial version
```

2.8.18 InstCreate

Subroutine InstCreate adds a new instance to the list of instances, assigns a unique instance number to this new instance, and archives this instance number to output argument Instance.

INTERFACE:

```
SUBROUTINE InstCreate ( ExtNr, Instance, Inst, RC )
```

INPUT PARAMETERS:

```
INTEGER,       INTENT(IN)       :: ExtNr
```

OUTPUT PARAMETERS:

```
INTEGER,       INTENT(  OUT)    :: Instance
TYPE(MyInst),  POINTER          :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,       INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.8.19 InstRemove

Subroutine InstRemove removes an instance from the list of instances.

INTERFACE:

```
SUBROUTINE InstRemove ( Instance )
```

INPUT PARAMETERS:

```
INTEGER                         :: Instance
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.9 Fortran: Module Interface hcox_finn_mod.F90

Module HCOX_FINN_MOD contains routines and variables to calculate FINN biomass burning emissions in HEMCO. **INTERFACE:**

```
MODULE HcoX_FINN_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Diagn_Mod
USE HCOX_TOOLS_MOD
USE HCO_State_Mod,  ONLY : HCO_State
USE HCOX_State_Mod, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_FINN_Init
PUBLIC :: HCOX_FINN_Run
PUBLIC :: HCOX_FINN_Final
```

REMARKS:

```
 Emissions of biomass burning species are read at monthly or daily
 resolution. Note: no emission factors are used here - emissions of
 individual species are given in input files. Emissions on the FINN 0.5x0.5
 degree grid are regridded to the current model grid.
                                                                             .
 FINN biomass burning emissions are computed for the following gas-phase
 and aerosol-phase species:
                                                                             .
    (1 ) NOx  [ kg/m2/s]     (13) BC   [kgC/m2/s]
    (2 ) CO   [ kg/m2/s]     (14) OC   [kgC/m2/s]
    (3 ) ALK4 [kgC/m2/s]     (15) MGLY [ kg/m2/s]
    (4 ) ACET [kgC/m2/s]     (16) BENZ [kgC/m2/s]
    (5 ) MEK  [kgC/m2/s]     (17) TOLU [kgC/m2/s]
    (6 ) ALD2 [kgC/m2/s]     (18) C2H4 [kgC/m2/s]
    (7 ) PRPE [kgC/m2/s]     (19) C2H2 [kgC/m2/s]
    (8 ) C3H8 [kgC/m2/s]     (20) GLYC [ kg/m2/s]
    (9 ) CH2O [ kg/m2/s]     (21) HAC  [ kg/m2/s]
    (10) C2H6 [kgC/m2/s]     (22) CO2  [ kg/m2/s]
    (11) SO2  [ kg/m2/s]     (23) CH4  [ kg/m2/s]
    (12) NH3  [ kg/m2/s]     (24)
 All species to be used must be listed in the settings section of the HEMCO
 configuration file. For every listed species, individual scale factors as
 well as masks can be defined. For example, to scale FINN CO emissions by a
 factor of 1.05 and restrict them to North America, as well as to scale NO
 emissions by a factor of 1.5:
114     FINN              : on    NO/CO/ALK4/ACET/MEK/ALD2/PRPE/C3H8/CH2O/C2H6/SO2/NH3/BC
    --> FINN_daily        :       false
    --> hydrophilic BC    :       0.2
    --> hydrophilic OC    :       0.5
    --> Mask_CO           :       NAMASK
    --> Scaling_CO        :       1.05
    --> Scaling_NO        :       1.5
```

```
Field NAMASK must be defined in section mask of the HEMCO configuration file.

 References:
 ============================================================================
 (1 ) Original FINN database from Christine Wiedinmyer
       http://bai.acd.ucar.edu/Data/fire/
 (2 ) Wiedinmyer, C., Akagi, S.K., Yokelson, R.J., Emmons, L.K.,
      Al-Saadi, J.A., Orlando, J.J., and Soja, A.J.: The Fire
      INventory from NCAR (FINN): a high resolution global model to
      estimate the emissions from open burning, Geoscientific Model
      Development, 4, 625-641, doi:10.5194/gmd-4-625-2011, 2011.
```

REVISION HISTORY:

```
 02 Jan 2013 - J. Mao & J.A. Fisher - Initial version, based on GFED3
 01 Oct 2013 - J.A. Fisher - Update to only use one input file
 05 May 2014 - J.A. Fisher - Replace NOx emissions with NO emissions as part
                             of removal of NOx-Ox partitioning
 18 Jun 2014 - C. Keller   - Now a HEMCO extension.
 03 Jul 2014 - C. Keller   - Added 13 new FINN species
 11 Aug 2014 - R. Yantosca - Now get emission factors and NMOC ratios from
                             hard-coded statements in hcox_finn_include.H
 11 Aug 2014 - R. Yantosca - Now use F90 free-form indentation
 11 Aug 2014 - R. Yantosca - Cosmetic changes to ProTeX subroutine headers
 11 Jun 2015 - C. Keller   - Update to include individual scale factors and
                             masks.
 14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
```

2.9.1 HCOX_FINN_Run

Subroutine HCOX_FINN_Run computes the FINN biomass burning emissions for the current date.

INTERFACE:

```
SUBROUTINE HCOX_FINN_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
 USE HCO_EmisList_mod,  ONLY : HCO_GetPtr
 USE HCO_Calc_Mod,      ONLY : HCO_EvalFld
 USE HCO_FluxArr_mod,   ONLY : HCO_EmisAdd
 USE HCO_State_mod,     ONLY : HCO_GetHcoID
 USE HCO_Clock_mod,     ONLY : HcoClock_Get
 USE HCO_Clock_mod,     ONLY : HcoClock_First
 USE HCO_Clock_mod,     ONLY : HcoClock_NewMonth, HcoClock_NewDay
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root  ! root CPU?
TYPE(Ext_State), POINTER        :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState   ! Output obj
INTEGER,         INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
02 Jan 2012 - J. Mao & J. Fisher - Initial version, based on GFED3
18 Jun 2014 - C. Keller          - Now a HEMCO extension.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
10 Mar 2017 - M. Sulprizio- Add SpcArr3D for emitting 65% of biomass
                            burning emissions into the PBL and 35% into the
                            free troposphere, following code from E.Fischer
24 Apr 2017 - M. Sulprizio- Comment out vertical distribution of biomass
                            burning emissions for now.
```

2.9.2 HCOX_FINN_Init

Subroutine HCOX_FINN_INIT initializes all module arrays and variables.

INTERFACE:

```
SUBROUTINE HCOX_FINN_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_State_Mod,   ONLY : HCO_GetHcoID
USE HCO_State_Mod,   ONLY : HCO_GetExtHcoID
USE HCO_ExtList_Mod, ONLY : GetExtNr, GetExtOpt
USE HCO_ExtList_Mod, ONLY : GetExtSpcVal
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root   ! root CPU?
TYPE(HCO_State),  POINTER        :: HcoState    ! HEMCO state object
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName     ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState    ! Extensions object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC          ! Return status
```

REVISION HISTORY:

```
02 Jan 2013 - J. Mao & J. Fisher - Initial version, based on GFED3
05 May 2014 - J.A. Fisher - Replace NOx emissions with NO emissions as part
                            of removal of NOx-Ox partitioning
18 Jun 2014 - C. Keller   - Now a HEMCO extension.
11 Aug 2014 - R. Yantosca - Now get FINN emission factors and species names
                            from include file hcox_finn_include.H.
11 Nov 2014 - C. Keller   - Now get hydrophilic fractions through config file
```

2.9.3 HCOX_FINN_Final

Subroutine HCOX_FINN_FINAL deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_FINN_FINAL()
```

REVISION HISTORY:

```
02 Jan 2013 - J. Mao & J. Fisher - Initial version, based on GFED3
18 Jun 2014 - C. Keller          - Now a HEMCO extension.
```

2.10 Fortran: Module Interface hcox_dust_dead_mod.F

Module hcox_dust_dead_mod.F contains routines and variables from Charlie Zender's DEAD dust mobilization model. Most routines are from Charlie Zender, but have been modified and/or cleaned up for inclusion into GEOS-Chem.

This is a HEMCO extension module that uses many of the HEMCO core utilities.

NOTE: The current (dust) code was validated at 2 x 2.5 resolution. We have found that running at 4x5 we get much lower (50emissions than at 2x2.5. Recommend we either find a way to scale the U* computed in the dust module, or run a 1x1 and store the the dust emissions, with which to drive lower resolution runs. – Duncan Fairlie, 1/25/07

(We'll) implement the [dust] code in the standard [GEOS-Chem] model and put a warning about expected low bias when the simulation is run at 4x5. Whoever is interested in running dust at 4x5 in the future can deal with making the fix. – Daniel Jacob, 1/25/07

!REFERENCES:

- Zender, C. S., Bian, H., and Newman, D.: Mineral Dust Entrainment and Deposition (DEAD) model: Description and 1990s dust climatology, Journal of Geophysical Research: Atmospheres, 108, 2003.

INTERFACE:

```
MODULE HCOX_DUSTDEAD_MOD
```

USES:

```
USE HCO_ERROR_MOD
USE HCO_DIAGN_MOD
USE HCOX_State_MOD,    ONLY : Ext_State
USE HCO_STATE_MOD,     ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_DustDead_Run
PUBLIC :: HCOX_DustDead_Init
PUBLIC :: HCOX_DustDead_Final
PUBLIC :: HCOX_DustDead_GetFluxTun
```

REVISION HISTORY:

```
(1 ) Added parallel DO loop in GET_ORO (bmy, 4/14/04)
(2 ) Now references "directory_mod.f" (bmy, 7/20/04)
(3 ) Fixed typo in ORO_IS_LND for PGI compiler (bmy, 3/1/05)
(4 ) Modified for GEOS-5 and GCAP met fields (swu, bmy, 8/16/05)
(5 ) Now make sure all USE statements are USE, ONLY (bmy, 10/3/05)
(6 ) Now uses GOCART source function (tdf, bmy, 1/25/07)
(7 ) Modifications for 0.5 x 0.667 grid (yxw, dan, bmy, 11/6/08)
(8 ) Updates for nested grids (amv, bmy, 12/18/09)
01 Mar 2012 - R. Yantosca - Now reference new grid_mod.F90
25 Nov 2013 - C. Keller   - Now a HEMCO extension
06 Oct 2014 - C. Keller   - Allow mass flux tuning factor be set in
                            configuration file.
08 Jul 2015 - M. Sulprizio- Now include dust alkalinity source (tdf 04/10/08)
07 Jan 2016 - E. Lundgren - Change dry air gas constant and molec wt to
                            match GC values and update acc due to gravity
                            and universal gas constant to NIST 2014 values
14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
24 Aug 2017 - M. Sulprizio- Remove support for GEOS-4, GEOS-5, and MERRA
17 Oct 2017 - C. Keller   - Now estimate default scale factor from
                            model resolution.
03 Apr 2018 - S. Philip   - Add anthropogenic PM2.5 dust emission (AFCID)
```

2.10.1 HCOX_DustDead_Run

Subroutine HcoX_DustDead_Run is the driver routine for the HEMCO DEAD dust extension.

INTERFACE:

```
SUBROUTINE HCOX_DustDead_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_CALC_MOD,      ONLY : HCO_EvalFld, HCO_CalcEmis
USE HCO_FLUXARR_MOD,   ONLY : HCO_EmisAdd
USE HCO_CLOCK_MOD,     ONLY : HcoClock_Get
USE HCO_CLOCK_MOD,     ONLY : HcoClock_First
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root
TYPE(Ext_State), POINTER        :: ExtState    ! Module options
TYPE(HCO_State), POINTER        :: HcoState    ! Hemco state
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
08 Apr 2004 - T. D. Fairlie - Initial version
(1 ) Added OpenMP parallelization, added comments (bmy, 4/8/04)
(2 ) Bug fix: DSRC needs to be held PRIVATE (bmy, 4/14/04)
(3 ) Now references DATA_DIR from "directory_mod.f" (bmy, 7/20/04)
(4 ) Now make sure all USE statements are USE, ONLY (bmy, 10/3/05)
(5 ) Bug fix: It should be SNOW/1d3 not SNOW*1d3 (tdf, bmy, 11/18/05)
(6 ) Updated output statement (bmy, 1/23/07)
(7 ) Use SNOMAS (m H2O) for GEOS-5 (bmy, 1/24/07)
25 Aug 2010 - R. Yantosca - Treat MERRA in the same way as for GEOS-5
25 Aug 2010 - R. Yantosca - Added ProTeX headers
03 Sep 2010 - R. Yantosca - Bug fix, SNOMAS was mislabled in GEOS-5
                            and has units of mm H2O instead of m H2O
                            so we need to convert to m H2O.
08 Feb 2012 - R. Yantosca - Treat GEOS-5.7.x in the same way as MERRA
01 Mar 2012 - R. Yantosca - Now use GET_YMID_R(I,J,L) from grid_mod.F90
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
25 Nov 2013 - C. Keller   - Now a HEMCO extension
06 Oct 2014 - C. Keller   - Now calculate pressure center from edges.
26 Jun 2015 - E. Lundgren - Add L. Zhang new dust size distribution scheme
08 Jul 2015 - M. Sulprizio- Now include dust alkalinity source (tdf 04/10/08)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
03 Apr 2018 - S. Philip   - Add anthropogenic PM2.5 dust emission (AFCID)
```

2.10.2 HCOX_DustDead_Init

Subroutine HcoX_DustDead_Init initializes the HEMCO DUST_DEAD extension.

INTERFACE:

```
SUBROUTINE HCOX_DustDead_Init ( am_I_Root, HcoState, ExtName,
&                               ExtState,  RC               )
```

USES:

```
USE HCO_ExtList_Mod,    ONLY : GetExtNr, GetExtOpt
USE HCO_STATE_MOD,      ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
TYPE(HCO_State),  POINTER        :: HcoState   ! Hemco state
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName    ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState     ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
25 Nov 2013 - C. Keller   - Now a HEMCO extension
14 Aug 2014 - R. Yantosca - Now always print out extension info
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.10.3 HCOX_DustDead_Final

Subroutine HcoX_DustDead_Final finalizes the HEMCO DUST_DEAD extension.

INTERFACE:

```
SUBROUTINE HCOX_DustDead_Final ( ExtState )
```

INPUT PARAMETERS:

```
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

REVISION HISTORY:

```
25 Nov 2013 - C. Keller - Now a HEMCO extension
!NOTES:
```

2.10.4 InstGet

Subroutine InstGet returns a pointer to the desired instance.

INTERFACE:

```
SUBROUTINE InstGet ( Instance, Inst, RC, PrevInst )
```

INPUT PARAMETERS:

```
INTEGER                            :: Instance
TYPE(MyInst),     POINTER          :: Inst
INTEGER                            :: RC
TYPE(MyInst),     POINTER, OPTIONAL :: PrevInst
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.10.5 InstCreate

Subroutine InstCreate creates a new instance.

INTERFACE:

```
SUBROUTINE InstCreate ( ExtNr, Instance, Inst, RC )
```

INPUT PARAMETERS:

```
INTEGER,       INTENT(IN)       :: ExtNr
```

OUTPUT PARAMETERS:

```
INTEGER,       INTENT(  OUT)    :: Instance
TYPE(MyInst),  POINTER          :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,       INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.10.6 InstRemove

Subroutine InstRemove creates a new instance.

INTERFACE:

```
SUBROUTINE InstRemove ( Instance )
```

INPUT PARAMETERS:

```
INTEGER                         :: Instance
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.11 Fortran: Module Interface hcox_finn_mod.F90

Module HCOX_FINN_MOD contains routines and variables to calculate FINN biomass burning emissions in HEMCO. **INTERFACE:**

```
MODULE HcoX_FINN_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Diagn_Mod
USE HCOX_TOOLS_MOD
USE HCO_State_Mod,  ONLY : HCO_State
USE HCOX_State_Mod, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_FINN_Init
PUBLIC :: HCOX_FINN_Run
PUBLIC :: HCOX_FINN_Final
```

REMARKS:

```
 Emissions of biomass burning species are read at monthly or daily
 resolution. Note: no emission factors are used here - emissions of
 individual species are given in input files. Emissions on the FINN 0.5x0.5
 degree grid are regridded to the current model grid.
                                                                             .
 FINN biomass burning emissions are computed for the following gas-phase
 and aerosol-phase species:
                                                                             .
    (1 ) NOx  [ kg/m2/s]     (13) BC   [kgC/m2/s]
    (2 ) CO   [ kg/m2/s]     (14) OC   [kgC/m2/s]
    (3 ) ALK4 [kgC/m2/s]     (15) MGLY [ kg/m2/s]
    (4 ) ACET [kgC/m2/s]     (16) BENZ [kgC/m2/s]
    (5 ) MEK  [kgC/m2/s]     (17) TOLU [kgC/m2/s]
    (6 ) ALD2 [kgC/m2/s]     (18) C2H4 [kgC/m2/s]
    (7 ) PRPE [kgC/m2/s]     (19) C2H2 [kgC/m2/s]
    (8 ) C3H8 [kgC/m2/s]     (20) GLYC [ kg/m2/s]
    (9 ) CH2O [ kg/m2/s]     (21) HAC  [ kg/m2/s]
    (10) C2H6 [kgC/m2/s]     (22) CO2  [ kg/m2/s]
    (11) SO2  [ kg/m2/s]     (23) CH4  [ kg/m2/s]
    (12) NH3  [ kg/m2/s]     (24)
 All species to be used must be listed in the settings section of the HEMCO
 configuration file. For every listed species, individual scale factors as
 well as masks can be defined. For example, to scale FINN CO emissions by a
 factor of 1.05 and restrict them to North America, as well as to scale NO
 emissions by a factor of 1.5:
114     FINN              : on    NO/CO/ALK4/ACET/MEK/ALD2/PRPE/C3H8/CH2O/C2H6/SO2/NH3/BC
    --> FINN_daily        :       false
    --> hydrophilic BC    :       0.2
    --> hydrophilic OC    :       0.5
    --> Mask_CO           :       NAMASK
    --> Scaling_CO        :       1.05
    --> Scaling_NO        :       1.5
```

```
Field NAMASK must be defined in section mask of the HEMCO configuration file.

 References:
 ============================================================================
 (1 ) Original FINN database from Christine Wiedinmyer
       http://bai.acd.ucar.edu/Data/fire/
 (2 ) Wiedinmyer, C., Akagi, S.K., Yokelson, R.J., Emmons, L.K.,
      Al-Saadi, J.A., Orlando, J.J., and Soja, A.J.: The Fire
      INventory from NCAR (FINN): a high resolution global model to
      estimate the emissions from open burning, Geoscientific Model
      Development, 4, 625-641, doi:10.5194/gmd-4-625-2011, 2011.
```

REVISION HISTORY:

```
 02 Jan 2013 - J. Mao & J.A. Fisher - Initial version, based on GFED3
 01 Oct 2013 - J.A. Fisher - Update to only use one input file
 05 May 2014 - J.A. Fisher - Replace NOx emissions with NO emissions as part
                             of removal of NOx-Ox partitioning
 18 Jun 2014 - C. Keller   - Now a HEMCO extension.
 03 Jul 2014 - C. Keller   - Added 13 new FINN species
 11 Aug 2014 - R. Yantosca - Now get emission factors and NMOC ratios from
                             hard-coded statements in hcox_finn_include.H
 11 Aug 2014 - R. Yantosca - Now use F90 free-form indentation
 11 Aug 2014 - R. Yantosca - Cosmetic changes to ProTeX subroutine headers
 11 Jun 2015 - C. Keller   - Update to include individual scale factors and
                             masks.
 14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
```

2.11.1 HCOX_FINN_Run

Subroutine HCOX_FINN_Run computes the FINN biomass burning emissions for the current date.

INTERFACE:

```
SUBROUTINE HCOX_FINN_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
 USE HCO_EmisList_mod,  ONLY : HCO_GetPtr
 USE HCO_Calc_Mod,      ONLY : HCO_EvalFld
 USE HCO_FluxArr_mod,   ONLY : HCO_EmisAdd
 USE HCO_State_mod,     ONLY : HCO_GetHcoID
 USE HCO_Clock_mod,     ONLY : HcoClock_Get
 USE HCO_Clock_mod,     ONLY : HcoClock_First
 USE HCO_Clock_mod,     ONLY : HcoClock_NewMonth, HcoClock_NewDay
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root  ! root CPU?
TYPE(Ext_State), POINTER        :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState   ! Output obj
INTEGER,         INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
02 Jan 2012 - J. Mao & J. Fisher - Initial version, based on GFED3
18 Jun 2014 - C. Keller          - Now a HEMCO extension.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
10 Mar 2017 - M. Sulprizio- Add SpcArr3D for emitting 65% of biomass
                            burning emissions into the PBL and 35% into the
                            free troposphere, following code from E.Fischer
24 Apr 2017 - M. Sulprizio- Comment out vertical distribution of biomass
                            burning emissions for now.
```

2.11.2 HCOX_FINN_Init

Subroutine HCOX_FINN_INIT initializes all module arrays and variables.

INTERFACE:

```
SUBROUTINE HCOX_FINN_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_State_Mod,   ONLY : HCO_GetHcoID
USE HCO_State_Mod,   ONLY : HCO_GetExtHcoID
USE HCO_ExtList_Mod, ONLY : GetExtNr, GetExtOpt
USE HCO_ExtList_Mod, ONLY : GetExtSpcVal
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root   ! root CPU?
TYPE(HCO_State),  POINTER        :: HcoState    ! HEMCO state object
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName     ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState    ! Extensions object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC          ! Return status
```

REVISION HISTORY:

```
02 Jan 2013 - J. Mao & J. Fisher - Initial version, based on GFED3
05 May 2014 - J.A. Fisher - Replace NOx emissions with NO emissions as part
                            of removal of NOx-Ox partitioning
18 Jun 2014 - C. Keller   - Now a HEMCO extension.
11 Aug 2014 - R. Yantosca - Now get FINN emission factors and species names
                            from include file hcox_finn_include.H.
11 Nov 2014 - C. Keller   - Now get hydrophilic fractions through config file
```

2.11.3 HCOX_FINN_Final

Subroutine HCOX_FINN_FINAL deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_FINN_FINAL()
```

REVISION HISTORY:

```
02 Jan 2013 - J. Mao & J. Fisher - Initial version, based on GFED3
18 Jun 2014 - C. Keller          - Now a HEMCO extension.
```

2.12 Fortran: Module Interface hcox_custom_mod.F90

Customizable HEMCO emission extension.

INTERFACE:

```
MODULE HCOX_Custom_Mod
```

USES:

```
USE HCO_Error_MOD
USE HCO_Diagn_MOD
USE HCOX_State_MOD, ONLY : Ext_State
USE HCO_State_MOD,  ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_Custom_Run
PUBLIC :: HCOX_Custom_Init
PUBLIC :: HCOX_Custom_Final
```

REVISION HISTORY:

```
13 Dec 2013 - C. Keller   - Initial version
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
06 Jun 2014 - R. Yantosca - Now indented with F90 free-format
```

2.12.1 HCOX_Custom_Run

Subroutine HCOX_Custom_Run is the driver routine for the customizable HEMCO extension.

INTERFACE:

```
SUBROUTINE HCOX_Custom_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_FluxArr_Mod,  ONLY : HCO_EmisAdd
USE HCO_GeoTools_Mod, ONLY : HCO_LANDTYPE
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
TYPE(Ext_State), POINTER       :: ExtState    ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState    ! Hemco state
INTEGER,         INTENT(INOUT) :: RC          ! Success or failure
```

REMARKS:

REVISION HISTORY:

```
13 Dec 2013 - C. Keller   - Initial version
05 Jun 2014 - R. Yantosca - Now store the results of HCO_LANDTYPE
                            in a PRIVATE variable for the !OMP loop
05 Jun 2014 - R. Yantosca - Cosmetic changes
06 Jun 2014 - R. Yantosca - Now indented with F90 free-format
```

2.12.2 HCOX_Custom_Init

Subroutine HCOX_Custom_Init initializes the HEMCO CUSTOM extension.

INTERFACE:

```
SUBROUTINE HCOX_Custom_Init( am_I_Root, HcoState, ExtName, &
                             ExtState,  RC                 )
```

USES:

```
USE HCO_ExtList_Mod,    ONLY : GetExtNr
USE HCO_STATE_MOD,      ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root
CHARACTER(LEN=*), INTENT(IN   ) :: ExtName    ! Extension name
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER       :: HcoState   ! Hemco state
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
13 Dec 2013 - C. Keller   - Now a HEMCO extension
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
06 Jun 2014 - R. Yantosca - Now indented using F90 free-format
```

2.12.3 HCOX_Custom_Final

Subroutine HCOX_Custom_Final finalizes the HEMCO CUSTOM extension.

INTERFACE:

```
SUBROUTINE HCOX_Custom_Final
```

REVISION HISTORY:

```
13 Dec 2013 - C. Keller   - Now a HEMCO extension
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
06 Jun 2014 - R. Yantosca - Now indended with F90 free-format
```

2.13 Fortran: Module Interface hcox_gc_RnPbBe_mod.F90

Defines the HEMCO extension for the GEOS-Chem Rn-Pb-Be specialty simulation.

This extension parameterizes emissions of Rn and/or Pb based upon the literature given below. The emission fields become automatically added to the HEMCO emission array of the given species. It is possible to select only one of the two species (Rn or Pb) in the HEMCO configuration file. This may be useful if a gridded data inventory shall be applied to one of the species (through the standard HEMCO interface).

INTERFACE:

```
MODULE HCOX_GC_RnPbBe_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Diagn_Mod
USE HCO_State_Mod,  ONLY : HCO_State   ! Derived type for HEMCO state
USE HCOX_State_Mod, ONLY : Ext_State   ! Derived type for External state

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HcoX_GC_RnPbBe_Run
PUBLIC  :: HcoX_GC_RnPbBe_Init
PUBLIC  :: HcoX_Gc_RnPbBe_Final
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: Init_7Be_Emissions
```

REMARKS:

```
References:
============================================================================
(1 ) Liu,H., D.Jacob, I.Bey, and R.M.Yantosca, Constraints from 210Pb
      and 7Be on wet deposition and transport in a global three-dimensional
      chemical tracer model driven by assimilated meteorological fields,
      JGR, 106, D11, 12,109-12,128, 2001.
(2 ) Jacob et al.,Evaluation and intercomparison of global atmospheric
      transport models using Rn-222 and other short-lived tracers,
      JGR, 1997 (102):5953-5970
(3 ) Dorothy Koch, JGR 101, D13, 18651, 1996.
(4 ) Lal, D., and B. Peters, Cosmic ray produced radioactivity on the
      Earth. Handbuch der Physik, 46/2, 551-612, edited by K. Sitte,
      Springer-Verlag, New York, 1967.
```

REVISION HISTORY:

```
07 Jul 2014 - R. Yantosca - Initial version
15 Aug 2014 - C. Keller   - Targets now in hp precision. Cosmetic changes
21 Aug 2014 - R. Yantosca - Add Pb as a species
21 Aug 2014 - R. Yantosca - Add HEMCO species indices as module variables
04 Sep 2014 - R. Yantosca - Remove IDTPb; Pb210 only has a chemical source
04 Sep 2014 - R. Yantosca - Modified for GCAP simulation
05 Nov 2014 - C. Keller   - Now allow Rn or Pb to be not specified.
07 Jan 2016 - E. Lundgren - Update Avogadro's # to NIST 2014 value
24 Aug 2017 - M. Sulprizio- Remove support for GCAP
```

2.13.1 HCOX_Gc_RnPbBe_run

Subroutine HcoX_Gc_RnPbBe_Run computes emissions of 222Rn and 7Be for the GEOS-Chem Rn-Pb-Be specialty simulation.

INTERFACE:

```
SUBROUTINE HCOX_Gc_RnPbBe_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
! HEMCO modules
USE HCO_FluxArr_Mod, ONLY : HCO_EmisAdd
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
TYPE(Ext_State),  POINTER       :: ExtState    ! Options for Rn-Pb-Be sim
TYPE(HCO_State),  POINTER       :: HcoState    ! HEMCO state
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT) :: RC          ! Success or failure?
```

REMARKS:

```
This code is based on routine EMISSRnPbBe in prior versions of GEOS-Chem.
```

REVISION HISTORY:

```
07 Jul 2014 - R. Yantosca - Initial version
03 Sep 2014 - R. Yantosca - Bug fix: Prevent div-by-zero errors
06 Oct 2014 - C. Keller   - Now calculate pressure centers from edges.
29 Oct 2014 - R. Yantosca - Use latitude centers of the grid box to
                            facilitate running in ESMF/MPI environment
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.13.2 HCOX_Gc_RnPbBe_Init

Subroutine HcoX_Gc_RnPbBe_Init initializes the HEMCO GC_Rn-Pb-Be extension.

INTERFACE:

```
SUBROUTINE HCOX_Gc_RnPbBe_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_ExtList_Mod, ONLY : GetExtNr
USE HCO_State_Mod,   ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName     ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState    ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState    ! Hemco state
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
07 Jul 2014 - R. Yantosca - Initial version
21 Aug 2014 - R. Yantosca - Now define HEMCO indices as well
04 Sep 2014 - R. Yantosca - Activate ExtState%TROPP for GCAP simulation
```

2.13.3 HCOX_Gc_RnPbBe_Final

Subroutine HcoX_Gc_RnPbBe_Final finalizes the HEMCO extension for the GEOS-Chem Rn-Pb-Be specialty simulation. All module arrays will be deallocated.

INTERFACE:

```
SUBROUTINE HCOX_Gc_RnPbBe_Final()
```

REVISION HISTORY:

```
13 Dec 2013 - C. Keller   - Now a HEMCO extension
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
06 Jun 2014 - R. Yantosca - Now indended with F90 free-format
```

2.13.4 Init_7Be_Emissions

Subroutine Init_7Be_Emissions initializes the 7Be emissions from Lal & Peters on 33 pressure levels. This data used to be read from a file, but we have now hardwired it to facilitate I/O in the ESMF environment.

INTERFACE:

```
SUBROUTINE Init_7Be_Emissions()
```

REMARKS:

```
(1) Reference: Lal, D., and B. Peters, Cosmic ray produced radioactivity
     on the Earth. Handbuch der Physik, 46/2, 551-612, edited by K. Sitte,
      Springer-Verlag, New York, 1967.

(2) In prior versions of GEOS-Chem, this routine was named READ_7BE, and
    it read the ASCII file "7Be.Lal".   Because this data set is not placed
    on a lat/lon grid, ESMF cannot regrid it.  To work around this, we now
    hardwire this data in module arrays rather than read it from disk.

(3) Units of 7Be emissions are [stars/g air/s].
    Here, "stars" = # of nuclear disintegrations of cosmic rays

(4) Original data from Lal & Peters (1967), w/ these modifications:
    (a) Replace data at (0hPa, 70S) following Koch 1996:
        (i ) old value = 3000
        (ii) new value = 1900
    (b) Copy data from 70S to 80S and 90S at all levels
```

REVISION HISTORY:

```
07 Aug 2002 - H. Liu - Initial version
(1 ) This code was split off from routine EMISSRnPbBe below. (bmy, 8/7/02)
```

```
(2 ) Now reference DATA_DIR from "directory_mod.f" (bmy, 7/19/04)
08 Dec 2009 - R. Yantosca - Added ProTeX headers
01 Aug 2012 - R. Yantosca - Add reference to findFreeLUN from inqure_mod.F90
02 Jul 2014 - R. Yantosca - Now hardwire the data instead of reading it
                            from an ASCII file.  This facilitates ESMF I/O.
07 Jul 2014 - R. Yantosca - Now renamed to INIT_7Be_Emissions and added
                            as a HEMCO extension
07 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 8 Aug 2014 - R. Yantosca - Now split off into hcox_gc_RnPbBe_include.H
05 Nov 2014 - C. Keller   - Converted from double-precision to flexible
                            (HEMCO) precision hp.
26 Feb 2015 - R. Yantosca - Now inline the code that used to be in the
                            include file hcox_gc_RnPbBe_include.H.  This
                            will result in faster compilation.
08 Jan 2016 - R. Yantosca - Change 54_hp to 54.0_hp to avoid error
```

2.13.5 SLQ

Subroutine SLQ is an interpolation subroutine from a Chinese reference book (says Hongyu Liu).

INTERFACE:

```
SUBROUTINE SLQ( X, Y, Z, N, M, U, V, W )
```

INPUT PARAMETERS:

```
INTEGER :: N        ! First dimension of Z
INTEGER :: M        ! Second dimension of Z
REAL(hp)  :: X(N)     ! X-axis coordinate on original grid
REAL(hp)  :: Y(M)     ! Y-axis coordinate on original grid
REAL(hp)  :: Z(N,M)   ! Array of data on original grid
REAL(hp)  :: U        ! X-axis coordinate for desired interpolated value
REAL(hp)  :: V        ! Y-axis coordinate for desired interpolated value
```

OUTPUT PARAMETERS:

```
REAL(hp)  :: W        ! Interpolated value of Z array, at coords (U,V)
```

REMARKS:

```
This routine was taken from the old RnPbBe_mod.F.
```

REVISION HISTORY:

```
17 Mar 1998 - H. Liu      - Initial version
(1 ) Added to "RnPbBe_mod.f" (bmy, 7/16/01)
(2 ) Removed duplicate definition of IQ.  Added comments. (bmy, 11/15/01)
08 Dec 2009 - R. Yantosca - Added ProTeX headers
 7 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
04 Dec 2014 - M. Yannetti - Added PRECISION_MOD
```

2.14 Fortran: Module Interface hemcox_dustginoux_mod.F90

Paul GINOUX dust source function. This subroutine updates the surface mixing ratio of dust aerosols for NDSTBIN size bins. The uplifting of dust depends in space on the source function, and in time and space on the soil moisture and surface wind speed (10 meters). Dust is uplifted if the wind speed is greater than a threshold velocity which is calculated with the formula of Marticorena et al. (JGR, v.102, pp 23277-23287, 1997). To run this subroutine you need the source function which can be obtained by contacting Paul Ginoux at ginoux@rondo.gsfc.nasa.gov/ If you are not using GEOS DAS met fields, you will most likely need to adapt the adjusting parameter.

This is a HEMCO extension module that uses many of the HEMCO core utilities.

References:

1. Ginoux, P., M. Chin, I. Tegen, J. Prospero, B. Hoben, O. Dubovik, and S.-J. Lin, "Sources and distributions of dust aerosols simulated with the GOCART model", J. Geophys. Res., 2001

2. Chin, M., P. Ginoux, S. Kinne, B. Holben, B. Duncan, R. Martin, J. Logan, A. Higurashi, and T. Nakajima, "Tropospheric aerosol optical thickness from the GOCART model and comparisons with satellite and sunphotometers measurements", J. Atmos Sci., 2001.

AUTHOR:

```
Paul Ginoux (ginoux@rondo.gsfc.nasa.gov)
```

INTERFACE:

```
MODULE HCOX_DustGinoux_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Diagn_Mod
USE HCO_State_Mod,  ONLY : HCO_State
USE HCOX_State_Mod, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HcoX_DustGinoux_Run
PUBLIC :: HcoX_DustGinoux_Init
PUBLIC :: HcoX_DustGinoux_Final
PUBLIC :: HcoX_DustGinoux_GetChDust
```

REVISION HISTORY:

```
08 Apr 2004 - T. D. Fairlie - Initial version
(1 ) Added OpenMP parallelization (bmy, 4/8/04)
(2 ) Now references DATA_DIR from "directory_mod.f" (bmy, 7/20/04)
25 Aug 2010 - R. Yantosca - Added ProTeX headers
01 Mar 2012 - R. Yantosca - Now use GET_AREA_M2(I,J,L) from grid_mod.F90
01 Aug 2012 - R. Yantosca - Add reference to findFreeLUN from inqure_mod.F90
03 Aug 2012 - R. Yantosca - Move calls to findFreeLUN out of DEVEL block
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
26 Feb 2013 - R. Yantosca - Now accept Input_Opt via the arg list
11 Dec 2013 - C. Keller   - Now a HEMCO extension.
29 Sep 2014 - R. Yantosca - Now make NBINS a variable and not a parameter
29 Sep 2014 - R. Yantosca - Now use F90 free-format indentation
08 Jul 2015 - M. Sulprizio- Now include dust alkalinity source (tdf 04/10/08)
14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
```

2.14.1 HCOX_DustGinoux_Run

Subroutine HcoX_DustGinoux_Run is the driver routine for the Paul Ginoux dust source function HEMCO extension.

INTERFACE:

```
SUBROUTINE HcoX_DustGinoux_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_Calc_Mod,     ONLY : HCO_EvalFld
USE HCO_EmisList_Mod, ONLY : HCO_GetPtr
USE HCO_FluxArr_Mod,  ONLY : HCO_EmisAdd
USE HCO_Clock_Mod,    ONLY : HcoClock_First
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root   ! Are we on the root CPU?
TYPE(Ext_State), POINTER        :: ExtState    ! Options for this ext
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState    ! HEMCO state object
INTEGER,         INTENT(INOUT)  :: RC          ! Success or failure?
```

REMARKS:

```
SRCE_FUNK Source function                              (-)
          for 1: Sand, 2: Silt, 3: Clay

DUSTDEN   Dust density                                 (kg/m3)
DUSTREFF  Effective radius                             (um)
AD        Air mass for each grid box                   (kg)
```

```
  NTDT      Time step                                    (s)
  W10m      Velocity at the anemometer level (10meters)  (m/s)
  GWET      Surface wetness                              (-)
                                                                 .
Dust properties used in GOCART
                                                                 .
Size classes: 01-1, 1-1.8, 1.8-3, 3-6 (um)
Radius: 0.7, 1.5, 2.5, 4  (um)
Density: 2500, 2650, 2650, 2650 (kg/m3)
```

REVISION HISTORY:

```
08 Apr 2004 - T. D. Fairlie - Initial version
(1 ) Added OpenMP parallelization (bmy, 4/8/04)
(2 ) Now references DATA_DIR from "directory_mod.f" (bmy, 7/20/04)
25 Aug 2010 - R. Yantosca - Added ProTeX headers
01 Mar 2012 - R. Yantosca - Now use GET_AREA_M2(I,J,L) from grid_mod.F90
01 Aug 2012 - R. Yantosca - Add reference to findFreeLUN from inqure_mod.F90
03 Aug 2012 - R. Yantosca - Move calls to findFreeLUN out of DEVEL block
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
26 Feb 2013 - R. Yantosca - Now accept Input_Opt via the arg list
11 Dec 2013 - C. Keller   - Now a HEMCO extension
29 Sep 2014 - R. Yantosca - Bug fix: SRCE_CLAY should have been picked when
                            M=3 but was picked when M=2.  Now corrected.
26 Jun 2015 - E. Lundgren - Add L. Zhang new dust size distribution scheme
08 Jul 2015 - M. Sulprizio- Now include dust alkalinity source (tdf 04/10/08)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
07 Jul 2017 - R. Yantosca - Bug fix: Skip DustAlk IF block unless that
                            extension has been turned on in the config file
```

2.14.2 HCOX_DustGinoux_Init

Subroutine HcoX_DustGinoux_Init initializes the HEMCO DUSTGINOUX extension.

INTERFACE:

```
SUBROUTINE HcoX_DustGinoux_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_ExtList_Mod, ONLY : GetExtNr, GetExtOpt
USE HCO_State_Mod,   ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root  ! Are we on the root CPU?
TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO State object
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName    ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState   ! Extension options
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC         ! Success or failure?
```

REVISION HISTORY:

```
11 Dec 2013 - C. Keller   - Now a HEMCO extension
26 Sep 2014 - R. Yantosca - Updated for TOMAS
29 Sep 2014 - R. Yantosca - Now initialize NBINS from HcoState%N_DUST_BINS
```

2.14.3 HCOX_DustGinoux_Final

Subroutine HcoX_DustGinoux_Final finalizes the HEMCO DUSTGINOUX extension.

INTERFACE:

```
SUBROUTINE HcoX_DustGinoux_Final()
```

REVISION HISTORY:

```
11 Dec 2013 - C. Keller - Now a HEMCO extension
```

2.14.4 HCOX_DustGinoux_GetChDust

Function HCOX_DustGinoux_GetChDust returns the CH_DUST parameter for the current simulation type.

INTERFACE:

```
FUNCTION HCOX_DustGinoux_GetChDust() RESULT( CH_DUST )
```

RETURN VALUE:

```
REAL*8 :: CH_DUST
```

REMARKS:

```
The logic in the #ifdefs may need to be cleaned up later on.  We have
just replicated the existing code in pre-HEMCO versions of dust_mod.F.
```

REVISION HISTORY:

```
11 Dec 2013 - C. Keller   - Initial version
25 Sep 2014 - R. Yantosca - Updated for TOMAS
24 Aug 2017 - M. Sulprizio- Remove support for GRID1x1
```

2.15 Fortran: Module Interface hcox_paranox_mod.F90

Module HCOX_PARANOX_MOD contains routines to compute ship emissions and associated concentrations of NO, NO2, HNO3 and O3 from NO ship emission data. This follows the implementation of the PARANOX ship plume model in GEOS-Chem.

This module calculates production rates of NO, NO2, HNO3, and O3, as well as loss rates of O3 and HNO3. All fluxes are in kg species/m2/s. The O3 and HNO3 loss fluxes are not converted to a deposition velocity, but rather saved out as mass fluxes (kg/m2/s) into diagnostics 'PARANOX_O3_DEPOSITION_FLUX' and 'PARANOX_HNO3_DEPOSITION_FLUX', respectively. In order to use them, they must be imported explicitly via routine Diagn_Get (from module hco_diagn_mod.F90). This approach avoids problems with uncrealistically high loss rates for loss ambient air concentrations of O3 or HNO3.

The PARANOx look-up-table can be provided in netCDF or ASCII (txt) format. The latter is particularly useful for running PARANOx in an ESMF environment, where 7-dimensional netCDF files are currently not supported. The input data format can be specified in the HEMCO configuration file (in the PARANOx extensions section). The txt-files can be generated from the previously read netCDF data using subroutine WRITE_LUT_TXTFILE.

References:

- Vinken, G. C. M., Boersma, K. F., Jacob, D. J., and Meijer, E. W.: Accounting for non-linear chemistry of ship plumes in the GEOS-Chem global chemistry transport model, Atmos. Chem. Phys., 11, 11707-11722, doi:10.5194/acp-11-11707-2011, 2011.

The initial look up tables (LUT) distributed with GEOS-Chem v9-01-03 used 7 input variables: Temperature, J(NO2), J(O1D), solar elevation angles at emission time and 5 hours later, and ambient concentrations of NOx and O3. This version was documented by Vinken et al. (2011). Subsequently, we added wind speed as an input variable. We also use J(OH) rather than J(O1D) to index the LUT (C. Holmes, 6 May 2013)
The LUTs contain 3 quantities: FracNOx : The fraction of NOx emitted from ships that remains as NOx after 5 hours of plume aging. mol/mol OPE : Ozone production efficiency, mol(O3)/mol(HNO3) The net production of O3 per mole of ship NOx oxidized over 5 hours of plume aging. Can be negative! Defined as OPE = [P(O3) - L(O3)] / P(HNO3), where each P and L term is an integral over 5 hours. Net O3 production in the plume is E(NOx) * (1-FracNOx) * OPE, where E(NOx) is the emission rate of NOx from the ship (e.g. units: mol/s). MOE : Methane oxidation efficiency, mol(CH4)/mol(NOx) The net oxidation of CH4 per mole of NOx emitted from ships over 5 hours of plume aging. Defined as MOE = L(CH4) / E(NOx).

The solar elevation angles 5 hours ago are calculated using HEMCO subroutine HCO_GetSUNCOS. This is the same routine that is used to calculate the solar zenith angles for the current time.

INTERFACE:

```
MODULE HCOX_ParaNOx_MOD
```

USES:

```
  USE HCO_Error_MOD
```

```
USE HCO_Diagn_MOD
USE HCO_State_MOD,  ONLY : HCO_State
USE HCOX_State_MOD, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCOX_ParaNOx_Run
PUBLIC  :: HCOX_ParaNOx_Init
PUBLIC  :: HCOX_ParaNOx_Final
```

PRIVATE MEMBER FUNCTIONS:

REMARKS:

Adapted from the code in GeosCore/paranox_mod.F prior to GEOS-Chem v10-01.

REVISION HISTORY:

```
06 Aug 2013 - C. Keller   - Initial version
03 Jun 2013 - C. Holmes   - Rewritten to include wind speed in the look-up
                            table and to take input from netCDF
15 Oct 2013 - C. Keller   - Now a HEMCO extension
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
06 Jun 2014 - R. Yantosca - Now indended with F90 free-format
25 Jun 2014 - R. Yantosca - Now pass the look-up-table filenames
15 Jul 2014 - C. Holmes   - Make module variables allocatable, since they
                            are used only in full chemistry simulations.
22 Jul 2014 - R. Yantosca - Added shadow copy of FAST-JX function FJXFUNC
28 Jul 2014 - C. Keller   - Now pass J-Values through ExtState. This makes
                            the FJXFUNC shadow copy obsolete
13 Aug 2014 - C. Keller   - Added manual diagnostics
16 Oct 2014 - C. Keller   - Now store SUNCOSmid values internally over the
                            past 5 hours and use these values for SUNCOSmid5.
                            This is required for standalone mode.
05 Feb 2015 - C. Keller   - Modified to bring in the updates from Chris
                            Holmes (input data in netCDF format, include
                            wind speed, calculated dry deposition freq.
                            using whole troposheric column mass).
23 Feb 2015 - C. Keller   - Historic j-values can now be provided through
                            HEMCO configuration file.
10 Apr 2015 - C. Keller   - Now exchange deposition fluxes via diagnostics.
                            Keep units of kg/m2/s for loss rates.
20 Apr 2016 - M. Sulprizio- Get J(OH) directly from FAST-JX and remove all
                            references to J(O1D). In FlexChem, adjustment of
                            photolysis rates are now done in routine
                            PHOTRATE_ADJ (found in GeosCore/fast_jx_mod.F).
14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
```

2.15.1 HCOX_ParaNOx_Run

Subroutine HCOX_ParaNOx_Run is the driver routine to calculate ship NOx emissions for the current time step. Emissions in [kg/m2/s] are added to the emissions array of the passed

INTERFACE:

```
SUBROUTINE HCOX_ParaNOx_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_Calc_Mod, ONLY : HCO_CalcEmis
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
TYPE(Ext_State), POINTER       :: ExtState    ! External data fields
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState    ! HEMCO State object
INTEGER,         INTENT(INOUT) :: RC          ! Success or failure?
```

REVISION HISTORY:

```
06 Aug 2013 - C. Keller   - Initial Version
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
06 Jun 2014 - R. Yantosca - Now indended with F90 free-format
28 Jul 2014 - C. Keller   - Now call Hco_CalcEmis instead of Hco_Run.
```

2.15.2 Evolve_Plume

Subroutine EVOLVE_PLUME performs plume dilution and chemistry of ship NO emissions for every grid box and writes the resulting NO, HNO3 and O3 emission (production) rates into State_Chm

INTERFACE:

```
SUBROUTINE Evolve_Plume( am_I_Root, ExtState, ShipNoEmis, HcoState, RC )
```

USES:

```
USE HCO_Types_Mod,    ONLY : DiagnCont
USE HCO_FluxArr_mod,  ONLY : HCO_EmisAdd
USE HCO_FluxArr_mod,  ONLY : HCO_DepvAdd
USE HCO_Clock_Mod,    ONLY : HcoClock_First
USE HCO_Restart_Mod,  ONLY : HCO_RestartGet
USE HCO_Restart_Mod,  ONLY : HCO_RestartWrite
USE HCO_Calc_Mod,     ONLY : HCO_CheckDepv
USE HCO_GeoTools_Mod, ONLY : HCO_GetSUNCOS
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root          ! Root CPU?
TYPE(Ext_State), POINTER        :: ExtState           ! External data
```

INPUT/OUTPUT PARAMETERS:

```
REAL(hp),        INTENT(INOUT)  :: ShipNoEmis(:,:,:)  ! Emissions
TYPE(HCO_State), POINTER        :: HcoState           ! HEMCO State obj
INTEGER,         INTENT(INOUT)  :: RC                 ! Success or failure
```

REVISION HISTORY:

```
06 Aug 2013 - C. Keller   - Initial Version
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
06 Jun 2014 - R. Yantosca - Now indended with F90 free-format
24 Jun 2014 - R. Yantosca - Now pass LUT_FILENAME to READ_PARANOX_LUT
22 Jul 2014 - R. Yantosca - Comment out debug print statements
28 Jul 2014 - C. Keller   - Now get J-values through ExtState
12 Aug 2014 - R. Yantosca - READ_PARANOX_LUT is now called from Init phase
10 Nov 2014 - C. Keller   - Added div-zero error trap for O3 deposition.
25 Nov 2014 - C. Keller   - Now convert NO fluxes to HNO3 and O3 using
                            corresponding molecular weight ratios. Safe
                            division check for O3 deposition calculation.
08 May 2015 - C. Keller   - Now read/write restart variables from here to
                            accomodate replay runs in GEOS-5.
25 May 2015 - C. Keller   - Now calculate SC5 via HCO_GetSUNCOS
29 Mar 2016 - C. Keller   - Bug fix: archive O3 deposition as positive flux.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
12 May 2017 - C. Keller   - Force option ScaleEmis to off.
```

2.15.3 HCOX_ParaNOx_Init

Subroutine HcoX_ParaNOx_Init initializes the HEMCO PARANOX extension.

INTERFACE:

```
SUBROUTINE HCOX_ParaNOx_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_Chartools_Mod, ONLY : HCO_CharParse
USE HCO_State_MOD,     ONLY : HCO_GetHcoID
USE HCO_State_MOD,     ONLY : HCO_GetExtHcoID
USE HCO_ExtList_Mod,   ONLY : GetExtNr
USE HCO_ExtList_Mod,   ONLY : GetExtOpt
USE HCO_Restart_Mod,   ONLY : HCO_RestartDefine
 USE ParaNOx_Util_Mod,  ONLY : Read_ParaNOx_LUT
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName       ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState      ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState      ! HEMCO state object
INTEGER,          INTENT(INOUT)  :: RC            ! Success or failure?
```

REVISION HISTORY:

```
06 Aug 2013 - C. Keller   - Initial Version
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTex Headers
06 Jun 2014 - R. Yantosca - Now indented using F90 free-format
13 Aug 2014 - R. Yantosca - Now read the PARANOX look-up tables here
14 Aug 2014 - R. Yantosca - Minor fix, read the PARANOX look-up tables
                            after displaying text about PARANOX extension
16 Oct 2014 - C. Keller   - Added error check after READ_PARANOX_LUT
17 Oct 2014 - C. Keller   - Now parse input files via HCO_CharParse
17 Apr 2015 - C. Keller   - Now assign PARANOX_SUNCOS1 to SC5(:,:,1), etc.
25 May 2015 - C. Keller   - Now calculate SC5 via HCO_GetSUNCOS
```

2.15.4 HCOX_ParaNOx_Final

Subroutine HcoX_ParaNox_Final finalizes the HEMCO PARANOX extension.

INTERFACE:

```
SUBROUTINE HCOX_ParaNOx_Final( am_I_Root, HcoState, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root     ! Root CPU?
TYPE(HCO_State), POINTER         :: HcoState      ! HEMCO State obj
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
06 Aug 2013 - C. Keller - Initial Version
06 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
06 Jun 2014 - R. Yantosca - Now indended with F90 free-format
```

2.15.5 read_paranox_lut_nc

Subroutine READ_PARANOX_LUT_NC reads look-up tables in netCDF format for use in the PARANOX ship plume model (G.C.M. Vinken)

INTERFACE:

```
SUBROUTINE READ_PARANOX_LUT_NC ( am_I_Root, HcoState, RC )
```

USES:

```
!INPUT ARGUMENTS:
 LOGICAL, INTENT(IN   )       :: am_I_Root
 TYPE(HCO_State), POINTER     :: HcoState    ! HEMCO State object
!INPUT/OUTPUT ARGUMENTS:
 INTEGER, INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
06 Feb 2012 - M. Payer    - Initial version modified from code provided by
                            G.C.M. Vinken
01 Aug 2012 - R. Yantosca - Add reference to findFreeLUN from inqure_mod.F90
03 Aug 2012 - R. Yantosca - Move calls to findFreeLUN out of DEVEL block
03 Jun 2013 - C. Holmes   - Rewritten to include wind speed in the look-up
                            table and to take input from netCDF
```

2.15.6 read_lut_ncfile

Subroutine READ_LUT_NCFILE reads look up tables for use in the PARANOX ship plume model (C. Holmes)

INTERFACE:

```
SUBROUTINE READ_LUT_NCFILE( am_I_Root, HcoState, FILENAME, FNOX, DNOx, OPE, MOE, &
                            T, JNO2, O3, SEA0, SEA5, JRATIO, NOX )
```

USES:

```
 ! Modules for netCDF read
 USE m_netcdf_io_open
 USE m_netcdf_io_get_dimlen
 USE m_netcdf_io_read
 USE m_netcdf_io_readattr
 USE m_netcdf_io_close

#  include "netcdf.inc"
```

INPUT PARAMETERS:

```
 LOGICAL,          INTENT(IN)  :: am_I_Root
 TYPE(HCO_State), POINTER      :: HcoState    ! HEMCO State object
 CHARACTER(LEN=*),INTENT(IN)   :: FILENAME
```

OUTPUT PARAMETERS:

```
REAL*4, INTENT(OUT), DIMENSION(:,:,:,:,:,:) :: FNOX,OPE,MOE,DNOx
REAL*4, INTENT(OUT), OPTIONAL :: T(:),      JNO2(:), O3(:)
REAL*4, INTENT(OUT), OPTIONAL :: SEA0(:),   SEA5(:)
REAL*4, INTENT(OUT), OPTIONAL :: JRATIO(:), NOX(:)
```

REVISION HISTORY:

```
06 Feb 2012 - M. Payer    - Initial version modified from code provided by
                            G.C.M. Vinken
01 Aug 2012 - R. Yantosca - Add reference to findFreeLUN from inqure_mod.F90
03 Aug 2012 - R. Yantosca - Move calls to findFreeLUN out of DEVEL block
03 Jun 2013 - C. Holmes   - Rewritten to include wind speed in the look-up
                            table and to take input from netCDF
```

2.15.7 read_paranox_lut_txt

Subroutine READ_PARANOX_LUT_TXT reads look-up tables in txt format for use in the PARANOX ship plume model (G.C.M. Vinken)

INTERFACE:

```
SUBROUTINE READ_PARANOX_LUT_TXT ( am_I_Root, HcoState, RC )
```

USES:

```
!INPUT ARGUMENTS:
 LOGICAL,         INTENT(IN   ) :: am_I_Root
 TYPE(HCO_State), POINTER       :: HcoState    ! HEMCO State object
!INPUT/OUTPUT ARGUMENTS:
 INTEGER, INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
05 Feb 2015 - C. Keller   - Initial version modified from code provided by
                            G.C.M. Vinken and C. Holmes
```

2.15.8 read_lut_txtfile

Subroutine READ_LUT_TXTFILE reads look up tables for use in the PARANOX ship plume model (C. Holmes)

INTERFACE:

```
SUBROUTINE READ_LUT_TXTFILE( am_I_Root, HcoState, FILENAME, FNOX, DNOx, OPE, MOE, RC, &
                             T, JNO2, O3, SEA0, SEA5, JRATIO, NOX )
```

USES:

```
USE inquireMod, ONLY : findFreeLUN
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN)  :: am_I_Root
TYPE(HCO_State), POINTER     :: HcoState    ! HEMCO State object
CHARACTER(LEN=*),INTENT(IN)  :: FILENAME
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT)       :: RC
```

OUTPUT PARAMETERS:

```
REAL*4, INTENT(OUT), TARGET, DIMENSION(:,:,:,:,:,:,:) :: FNOX,OPE,MOE,DNOx
REAL*4, INTENT(OUT), OPTIONAL :: T(:),      JNO2(:), O3(:)
REAL*4, INTENT(OUT), OPTIONAL :: SEA0(:),   SEA5(:)
REAL*4, INTENT(OUT), OPTIONAL :: JRATIO(:), NOX(:)
```

REVISION HISTORY:

```
05 Feb 2015 - C. Keller   - Initial version modified from code provided by
                            G.C.M. Vinken and C. Holmes
```

2.15.9 write_lut_txtfile

write_lut_txtfile

INTERFACE:

```
SUBROUTINE WRITE_LUT_TXTFILE( am_I_Root, HcoState, FILENAME, FNOX, DNOx, OPE, MOE, RC, &
                              T, JNO2, O3, SEA0, SEA5, JRATIO, NOX )
```

USES:

```
USE inquireMod, ONLY : findFreeLUN
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN)     :: am_I_Root
TYPE(HCO_State), INTENT(INOUT)  :: HcoState          ! HEMCO state obj
CHARACTER(LEN=*),INTENT(IN)     :: FILENAME
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT)       :: RC
```

OUTPUT PARAMETERS:

```
REAL*4, INTENT(IN), TARGET, DIMENSION(:,:,:,:,:,:,:) :: FNOX,OPE,MOE,DNOx
REAL*4, INTENT(IN), OPTIONAL :: T(:),      JNO2(:), O3(:)
REAL*4, INTENT(IN), OPTIONAL :: SEA0(:),   SEA5(:)
REAL*4, INTENT(IN), OPTIONAL :: JRATIO(:), NOX(:)
```

REVISION HISTORY:

```
05 Feb 2015 - C. Keller   - Initial version modified from code provided by
                            G.C.M. Vinken and C. Holmes
```

2.15.10 INTERPOL_LINWEIGHTS

Subroutine INTERPOL_LINWEIGHTS finds the array elements and weights for piecewise 1-D linear interpolation. The input array of NODES must be in monotonic ascending order. (C. Holmes 3/27/2014)
If Y is an array containing values of a function evaluated at the points given in NODES, then its interpolated value at the point VALUESIN will be Y(VALUEIN) = Y(INDICES(1)) * WEIGHTS(1) + Y(INDICES(2)) * WEIGHTS(2)
This subroutine finds indices of consecutive nodes that bracket VALUEIN and weights such that VALUEIN = NODES(INDICES(1)) * WEIGHTS(1) + NODES(INDICES(1)+1) * (1-WEIGHTS(1))
For convenience, the returned values of INDICES and WEIGHTS are 2-element arrays, where INDICES(2) = INDICES(1)+1 and WEIGHTS(2) = 1 - WEIGHTS(1)

INTERFACE:

```
SUBROUTINE INTERPOL_LINWEIGHTS( NODES, VALUEIN, INDICES, WEIGHTS )
```

USES:

INPUT PARAMETERS:

```
REAL*4,INTENT(IN)   :: NODES(:), VALUEIN
```

OUTPUT PARAMETERS:

```
! These arrays are always 2 elements each, but declaring
! as deferred shape avoids array temporaries
INTEGER,INTENT(OUT) :: INDICES(:)
REAL*4, INTENT(OUT) :: WEIGHTS(:)
```

REVISION HISTORY:

```
03 Jun 2013 - C. Holmes      - Initial version
```

2.15.11 paranox_lut

Subroutine PARANOX_LUT returns fractional remainder of ship NOx (FNOx), fraction of NOx that dry deposits as NOy species (DNOx), ozone production efficiency (OPE), and methane oxidation efficiency (MOE) after 5-hrs of plume aging. Values are taken taken from a lookup table using piecewise linear interpolation. The look-up table is derived from the PARANOx gaussian plume model (Vinken et al. 2011; Holmes et al. 2014) (G.C.M. Vinken, KNMI, June 2010; C. Holmes June 2013)
The lookup table uses 8 input variables: TEMP : model temperature, K JNO2 : J(NO2) value, 1/s O3 : concentration O3 in ambient air, ppb SEA0 : solar elevation angle at emission time 5 hours ago, degree SEA5 : solar elevation angle at this time, degree JRatio : ratio J(OH)/J(NO2), unitless NOx : concentration NOx in ambient air, ppt WS : wind speed, m/s

In GEOS-Chem v9-01-03 through v9-02, the effects of wind speed on FNOx and OPE were not included (wind speed set at 6 m/s). The JRatio also used J(O1D) rather than J(OH); this has only a small effect on interpolated values. To reproduce the behavior of these earlier versions, modify code below marked with ******* and call READ_PARANOX_LUT_v913 in emissions_mod.F

INTERFACE:

```
SUBROUTINE PARANOX_LUT(  am_I_Root, ExtState,  HcoState, &
                         I, J, RC,  FNOX, DNOx, OPE, MOE_OUT )
```

USES:

```
USE HCO_STATE_MOD,        ONLY : HCO_State
USE HCOX_STATE_MOD,       ONLY : Ext_State
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN)         :: am_I_Root
TYPE(Ext_State), POINTER    :: ExtState
TYPE(HCO_State), POINTER    :: HcoState
INTEGER, INTENT(IN)         :: I, J        ! Grid indices
```

OUTPUT PARAMETERS:

```
REAL*8, INTENT(OUT)           :: FNOX ! fraction of NOx remaining, mol/mol
REAL*8, INTENT(OUT)           :: DNOX ! fraction of NOx deposited, mol/mol
REAL*8, INTENT(OUT)           :: OPE  ! net OPE, mol(net P(O3))/mol(P(HNO3))
REAL*8, INTENT(OUT), OPTIONAL :: MOE_OUT ! net MOE, mol(L(CH4))/mol(E(NOx))
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT)        :: RC   ! Return code
```

REVISION HISTORY:

```
   Jun 2010 - G.C.M. Vinken - Initial version
03 Jun 2013 - C. Holmes     - Heavily modified and simplified from previous
                              LUT interpolation code by G.C.M. Vinken and
                              M. Payer. LUT now includes wind speed.
04 Feb 2015 - C. Keller     - Updated for use in HEMCO.
24 Sep 2015 - E. Lundgren   - ExtState vars O3, NO2, and NO now in
                              kg/kg dry air (previously kg)
07 Jan 2016 - E. Lundgren   - Update H2O molec wt to match GC value
20 Apr 2016 - M. Sulprizio  - Remove calculation of J(OH). We now get J(OH),
                              the effective rate for O3+hv(+H2O)->OH+OH,
                              directly from FAST-JX. In FlexChem, adjustment
                              of the photolysis rates are done in routine
                              PHOTRATE_ADJ (found in GeosCore/fast_jx_mod.F).
20 Sep 2016 - R. Yantosca   - Replace non-standard ASIND function with ASIN,
                              and convert to degrees (divide by PI/180)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.16 Fortran: Module Interface hcox_seasalt_mod.F90

Module HCOX_SeaSalt_Mod contains routines to calculate sea salt aerosol emissions, following the implementation in GEOS-Chem. Emission number densities of the fine and coarse mode sea salt aerosols are written into diagnostic containers 'SEASALT_DENS_FINE' and 'SEASALT_DENS_COARSE', respectively.

This is a HEMCO extension module that uses many of the HEMCO core utilities.

INTERFACE:

```
MODULE HCOX_SeaSalt_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Diagn_Mod
USE HCO_State_Mod,  ONLY : HCO_State
USE HCOX_State_Mod, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_SeaSalt_Init
PUBLIC :: HCOX_SeaSalt_Run
PUBLIC :: HCOX_SeaSalt_Final
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: EMIT_SSABr2
```

REVISION HISTORY:

```
 15 Dec 2013 - C. Keller   - Now a HEMCO extension module
 09 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 09 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 09 Jul 2015 - E. Lundgren - Add marine organoc aerosols (B.Gantt, M.Johnson)
```

2.16.1 HCOX_SeaSalt_Run

Subroutine HcoX_SeaSalt_Run is the driver run routine to calculate SeaSalt emissions in HEMCO.

INTERFACE:

```
SUBROUTINE HCOX_SeaSalt_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_FluxArr_Mod,      ONLY : HCO_EmisAdd
USE HCO_GeoTools_Mod,     ONLY : HCO_LANDTYPE
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root  ! root CPU?
TYPE(HCO_State), POINTER       :: HcoState   ! Output obj
TYPE(Ext_State), POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC         ! Success or failure?
```

REMARKS:

References:

```
==========================================================================
(1 ) Chin, M., P. Ginoux, S. Kinne, B. Holben, B. Duncan, R. Martin,
     J. Logan, A. Higurashi, and T. Nakajima, "Tropospheric aerosol
     optical thickness from the GOCART model and comparisons with
     satellite and sunphotometers measurements", J. Atmos Sci., 2001.
(2 ) Gong, S., L. Barrie, and J.-P. Blanchet, "Modeling sea-salt
     aerosols in the atmosphere. 1. Model development", J. Geophys. Res.,
     v. 102, 3805-3818, 1997.
(3 ) Gong, S. L., "A parameterization of sea-salt aerosol source function
     for sub- and super-micron particles", Global Biogeochem.  Cy., 17(4),
     1097, doi:10.1029/2003GB002079, 2003.
(4 ) Jaegle, L., P.K. Quinn, T.S. Bates, B. Alexander, J.-T. Lin, "Global
     distribution of sea salt aerosols: New constraints from in situ and
     remote sensing observations", Atmos. Chem. Phys., 11, 3137-3157,
     doi:10.5194/acp-11-3137-2011.
```

REVISION HISTORY:

```
(1 ) Now references SALA_RREDGE_um and SALC_RREDGE_um from "tracer_mod.f"
     (bmy, 7/20/04)
(2 ) Now references GET_FRAC_OF_PBL and GET_PBL_TOP_L from "pbl_mix_mod.f".
     Removed reference to header file CMN.  Removed reference to
     "pressure_mod.f".  (bmy, 2/22/05)
(3 ) Now also compute alkalinity and number density of SeaSalt emissions.
     (bec, bmy, 4/13/05)
(4 ) Now references XNUMOL & XNUMOLAIR from "tracer_mod.f" (bmy, 10/25/05)
(5 ) The source function is for wet aerosol radius (RH=80%, with a radius
     twice the size of dry aerosols) so BETHA should be set to 2
     instead of 1.  Also now use LOG10 instead of LOG in the expressions
     for the SeaSalt base source, since we need the logarithm to the base
     10. (jaegle, bec, bmy, 11/23/09)
(6 ) Update to use the Gong (2003) source function (jaegle 5/11/11)
(7 ) Apply an empirical sea surface temperature dependence to Gong (2003)
     (jaegle 5/11/11)
22 Dec 2011 - M. Payer    - Added ProTeX headers
```

```
01 Mar 2012 - R. Yantosca - Now use GET_AREA_M2(I,J,L) from grid_mod.F90
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
15 Dec 2013 - C. Keller   - Now a HEMCO extension
09 Jul 2015 - E. Lundgren - Add marine organic aerosols (B.Gantt, M.Johnson)
19 Oct 2015 - C. Keller   - Now pass I and J index to EMIT_SSABr2 to support
                            curvilinear grids.
22 Oct 2015 - E. Lundgren - Bug fix: include CHLR in OMP PRIVATE statement
```

2.16.2 HCOX_SeaSalt_Init

Subroutine HcoX_SeaSalt_Init initializes all extension variables.

INTERFACE:

```
SUBROUTINE HCOX_SeaSalt_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_State_Mod,          ONLY : HCO_GetHcoID
USE HCO_STATE_MOD,          ONLY : HCO_GetExtHcoID
USE HCO_ExtList_Mod,        ONLY : GetExtNr
USE HCO_ExtList_Mod,        ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root   ! root CPU?
TYPE(HCO_State),  POINTER        :: HcoState    ! HEMCO state object
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName     ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState    ! Options object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC          ! Return status
```

REVISION HISTORY:

```
15 Dec 2013 - C. Keller   - Initial version
07 Oct 2014 - C. Keller   - Allow wind scale factor be set in config file
09 Jul 2015 - E. Lundgren - Add marine organic aerosols (B.Gantt, M.Johnson)
21 Sep 2016 - R. Yantosca - Bug fix: don't initialize SS_DEN before
                            it is allocated.  This causes a segfault.
29 Dec 2017 - C. Keller   - Bug fix: define index location of BrSALA and
                            BrSALC based upon # of species ID entries.
```

2.16.3 HCOX_SeaSalt_Final

Subroutine HcoX_SeaSalt_Final deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_SeaSalt_Final
```

REVISION HISTORY:

```
15 Dec 2013 - C. Keller - Initial version
```

2.16.4 Emit_SsaBr2

Subroutine Emit_SsaBr2 calculates aerosol emissions of Br2.

INTERFACE:

```
SUBROUTINE Emit_SsaBr2( am_I_Root,  ExtState, HcoState, ilon, &
                        ilat, rmid, p_kgsalt, br2_emiss_kg, RC )
!USE:
  USE HCO_Clock_Mod, ONLY : HcoClock_Get
```

INPUT PARAMETERS:

```
  LOGICAL,          INTENT(IN   )  :: am_I_Root ! root CPU?
  TYPE(Ext_State), POINTER         :: ExtState  ! Module options
  TYPE(HCO_State), POINTER         :: HcoState  ! Output obj
  INTEGER,          INTENT(IN)     :: ilon      ! Grid longitude index
  INTEGER,          INTENT(IN)     :: ilat      ! Grid latitude index
  REAL*8,           INTENT(IN)     :: rmid      ! Dry radius of aerosol
  REAL*8,           INTENT(IN)     :: p_kgsalt  ! SeaSalt aerosol production [kgNaCl]
```

OUTPUT PARAMETERS:

```
  INTEGER,          INTENT(INOUT)  :: RC           ! Success or failure?
  REAL*8,           INTENT(OUT)    :: br2_emiss_kg ! Br2 emissions [kg NaCl]
```

REMARKS:

```
References:
============================================================================
(1)  Parrella, J. P., Jacob, D. J., Liang, Q., Zhang, Y., Mickley, L. J.,
      Miller, B., Evans, M. J., Yang, X., Pyle, J. A., Theys, N., and Van
      Roozendael, M.: Tropospheric bromine chemistry: implications for
      present and pre-industrial ozone and mercury, Atmos. Chem. Phys., 12,
      6723-6740, doi:10.5194/acp-12-6723-2012, 2012.
(2 ) Yang, X., Cox, R. A., Warwick, N. J., Pyle, J. A., Carver, G. D.,
      O'Connor, F. M., and Savage, N. H.: Tropospheric bromine chemistry and
      its impacts on ozone: A model study, J. Geophys. Res., 110, D23311,
      doi:10.1029/2005JD006244, 2005.
(2 ) Yang, X., Pyle, J. A., and Cox, R. A.: Sea salt aerosol production and
      bromine release: Role of snow on sea ice, Geophys. Res. Lett., 35,
      L16815, doi:10.1029/2008GL034536, 2008.
```

REVISION HISTORY:

```
02 Mar 2010 - J. Parrella - Initial version
22 May 2012 - M. Payer    - Added ProTeX headers
08 Aug 2012 - M. Payer    - Modified for size-dependent depletion factors
                            from Yang et al. (2008)
07 Aug 2013 - C. Keller   - Moved to SeaSalt_mod.F
15 Dec 2013 - C. Keller   - Now a HEMCO extension
19 Oct 2015 - C. Keller   - Now use lon and lat index to work on curvilinear
                            grids.
```

2.17 Fortran: Module Interface hcox_state_mod.F90

Module HCOX_State_Mod contains routines and variables to organize the extensions state type ExtState. ExtState contains the logical switches for each extension (denoting whether or not it is enabled) as well as pointers to all met fields used by the extensions. ExtState is passed to all extension modules, and the met fields defined in here are thus available to all extensions. Additional met fields (and extension switches) can be added as required.
This module contains the routines to initialize and finalize the ExtState object, but doesn't link the met field pointers to the corresponding fields. This is done in the HEMCO-model interface routines (e.g. hcoi_standalone_mod.F90, hcoi_gc_main_mod.F90). Newly added met fields will only work if the corresponding pointer assignments are added to these interface routines!

INTERFACE:

```
MODULE HCOX_STATE_MOD
```

USES:

```
USE HCO_ERROR_MOD
USE HCO_ARR_MOD

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: ExtStateInit
PUBLIC :: ExtStateFinal
PUBLIC :: ExtDat_Set
!DERIVED TYPES:
!=========================================================================
! ExtDat_*: Derived types containing pointers to the met field arrays
! (Arr) and a logical flag whether or not the field is used by any of
! the extensions (DoUse).  Arrays can be 3D reals or 2D reals or integer
! All real values are of default precision! (df), as specified in
! HCO\_ERROR\_MOD.  You can add more types if necessary.
!=========================================================================
```

```
   ! 2D real, default precision
   TYPE, PUBLIC :: ExtDat_2R
      TYPE(Arr2D_HP), POINTER :: Arr
      LOGICAL                 :: DoUse
      LOGICAL                 :: FromList
   END TYPE ExtDat_2R

   ! 2D real, single precision
   TYPE, PUBLIC :: ExtDat_2S
      TYPE(Arr2D_SP), POINTER :: Arr
      LOGICAL                 :: DoUse
      LOGICAL                 :: FromList
   END TYPE ExtDat_2S

   ! 2D integer
   TYPE, PUBLIC :: ExtDat_2I
      TYPE(Arr2D_I),  POINTER :: Arr
      LOGICAL                 :: DoUse
      LOGICAL                 :: FromList
   END TYPE ExtDat_2I

   ! 3D real, default precision
   TYPE, PUBLIC :: ExtDat_3R
      TYPE(Arr3D_HP), POINTER :: Arr
      LOGICAL                 :: DoUse
      LOGICAL                 :: FromList
   END TYPE ExtDat_3R

   ! 3D real, single precision
   TYPE, PUBLIC :: ExtDat_3S
      TYPE(Arr3D_SP), POINTER :: Arr
      LOGICAL                 :: DoUse
      LOGICAL                 :: FromList
   END TYPE ExtDat_3S
   !=========================================================================
   ! Ext_State: Derived type declaration for the State object containing
   ! pointers to all met fields and related quantities used by the HEMCO
   ! extensions. An 'Ext_State' type called ExtState is defined at the
   ! beginning of a HEMCO run and populated according to the specifications
   ! set in the configuration file.  You can add more fields if necessary.
   !=========================================================================
   TYPE, PUBLIC :: Ext_State

      !-----------------------------------------------------------------
      ! Extension switches (enabled?)
      ! NOTE: When adding a new extension, don't forget to initialize this
      ! switch in subroutine ExtStateInit below!
      !-----------------------------------------------------------------
```

```
LOGICAL                 :: Custom         ! Customizable ext.
INTEGER                 :: DustDead       ! DEAD dust model
LOGICAL                 :: DustGinoux     ! Ginoux dust emissions
LOGICAL                 :: DustAlk        ! Dust alkalinity
INTEGER                 :: LightNOx       ! Lightning NOx
LOGICAL                 :: ParaNOx        ! PARANOX ship emissions
INTEGER                 :: SoilNOx        ! Soil NOx emissions
INTEGER                 :: Megan          ! MEGAN biogenic emissions
LOGICAL                 :: SeaFlux        ! air-sea exchange
LOGICAL                 :: SeaSalt        ! Seasalt emissions
LOGICAL                 :: MarinePOA      ! Marine organic aerosols
LOGICAL                 :: GFED           ! GFED biomass burning
LOGICAL                 :: FINN           ! FINN biomass burning
LOGICAL                 :: GC_RnPbBe      ! GEOS-Chem Rn-Pb-Be simulation
LOGICAL                 :: GC_POPs        ! GEOS-Chem POPs simulation
INTEGER                 :: Wetland_CH4    ! Methane emiss from wetlands
LOGICAL                 :: TOMAS_Jeagle   ! TOMAS Jeagle sea salt
INTEGER                 :: TOMAS_DustDead ! TOMAS sectional Dead Dust
INTEGER                 :: AeroCom        ! AeroCom volcano
LOGICAL                 :: Inorg_Iodine   ! Oceanic inorganic iodine emissions

!-----------------------------------------------------------------------
! Data directory
!-----------------------------------------------------------------------
CHARACTER(LEN=255)      :: DATA_DIR    ! Directory for data

!-----------------------------------------------------------------------
! Met fields
!-----------------------------------------------------------------------
TYPE(ExtDat_2R),  POINTER :: U10M        ! E/W 10m wind speed [m/s]
TYPE(ExtDat_2R),  POINTER :: V10M        ! N/S 10m wind speed [m/s]
TYPE(ExtDat_2R),  POINTER :: ALBD        ! Surface albedo [-]
TYPE(ExtDat_2R),  POINTER :: WLI         ! 0=water, 1=land, 2=ice
TYPE(ExtDat_2R),  POINTER :: T2M         ! 2m Sfce temperature [K]
TYPE(ExtDat_2R),  POINTER :: TSKIN       ! Surface skin temperature [K]
TYPE(ExtDat_2R),  POINTER :: GWETROOT    ! Root soil wetness [1]
TYPE(ExtDat_2R),  POINTER :: GWETTOP     ! Top soil moisture [-]
TYPE(ExtDat_2R),  POINTER :: SNOWHGT     ! Snow height [mm H2O = kg H2O/m2]
TYPE(ExtDat_2R),  POINTER :: SNODP       ! Snow depth [m ]
TYPE(ExtDat_2R),  POINTER :: SNICE       ! Fraction of snow/ice [1]
TYPE(ExtDat_2R),  POINTER :: USTAR       ! Friction velocity [m/s]
TYPE(ExtDat_2R),  POINTER :: Z0          ! Sfc roughness height [m]
TYPE(ExtDat_2R),  POINTER :: TROPP       ! Tropopause pressure [Pa]
TYPE(ExtDat_2R),  POINTER :: SUNCOS      ! COS (SZA)
TYPE(ExtDat_2R),  POINTER :: SZAFACT     ! current SZA/total daily SZA
TYPE(ExtDat_2R),  POINTER :: PARDR       ! direct photsyn radiation [W/m2]
TYPE(ExtDat_2R),  POINTER :: PARDF       ! diffuse photsyn radiation [W/m2]
TYPE(ExtDat_2R),  POINTER :: PSC2_WET    ! Interpolated sfc pressure [hPa]
```

```
TYPE(ExtDat_2R),  POINTER :: RADSWG      ! surface radiation [W/m2]
TYPE(ExtDat_2R),  POINTER :: FRCLND      ! Olson land fraction [-]
TYPE(ExtDat_2R),  POINTER :: FRLAND      ! land fraction [-]
TYPE(ExtDat_2R),  POINTER :: FROCEAN     ! ocean fraction [-]
TYPE(ExtDat_2R),  POINTER :: FRLAKE      ! lake fraction [-]
TYPE(ExtDat_2R),  POINTER :: FRLANDIC    ! land ice fraction [-]
TYPE(ExtDat_2R),  POINTER :: CLDFRC      ! cloud fraction [-]
TYPE(ExtDat_2R),  POINTER :: JNO2        ! J-Value for NO2 [1/s]
TYPE(ExtDat_2R),  POINTER :: JOH         ! J-Value for O3->OH  [1/s]
TYPE(ExtDat_2R),  POINTER :: LAI         ! daily leaf area index [cm2/cm2]
TYPE(ExtDat_2R),  POINTER :: CHLR        ! daily chlorophyll-a [mg/m3]
INTEGER,          POINTER :: PBL_MAX     ! Max height of PBL [level]
TYPE(ExtDat_3R),  POINTER :: CNV_MFC     ! Convective cloud mass flux [kg/m2/s]
TYPE(ExtDat_3R),  POINTER :: FRAC_OF_PBL ! Fraction of grid box in PBL
TYPE(ExtDat_3R),  POINTER :: SPHU        ! Spec. humidity [kg H2O/kg total air]
TYPE(ExtDat_3R),  POINTER :: TK          ! Air temperature [K]
TYPE(ExtDat_3R),  POINTER :: AIR         ! Dry air mass [kg]
TYPE(ExtDat_3R),  POINTER :: AIRVOL      ! Air volume [m3]
TYPE(ExtDat_3R),  POINTER :: AIRDEN      ! Dry air density [kg/m3]
TYPE(ExtDat_3R),  POINTER :: O3          ! O3 mass [kg/kg dry air]
TYPE(ExtDat_3R),  POINTER :: NO          ! NO mass [kg/kg dry air]
TYPE(ExtDat_3R),  POINTER :: NO2         ! NO2 mass [kg/kg dry air]
TYPE(ExtDat_3R),  POINTER :: HNO3        ! HNO3 mass [kg/kg dry air]
TYPE(ExtDat_3R),  POINTER :: POPG        ! POPG mass [kg/kg dry air]

!-----------------------------------------------------------------------
! Deposition parameter
! DRY_TOTN and WET_TOTN are the total (dry/wet) deposited N since the
! last emission timestep. Even though these numbers are per second,
! they may represent accumulated deposition velocities if chemistry
! and/or dynamic timestep are not equal to the emission timestep.
! These values are used by the soil NOx module. Note that it is assumed
! that DRY_TOTN and WET_TOTN are summed over chemistry and transport
! timesteps, respectively!
!-----------------------------------------------------------------------
TYPE(ExtDat_2R),  POINTER :: DRY_TOTN    ! Dry deposited N   [molec/cm2/s]
TYPE(ExtDat_2R),  POINTER :: WET_TOTN    ! Wet deposited N   [kg N/s]
REAL(hp),         POINTER :: DRYCOEFF(:) ! Baldocci drydep coeff.

!-----------------------------------------------------------------------
! Constants for POPs emissions module
!-----------------------------------------------------------------------
REAL(dp)                  :: POP_DEL_H   ! Delta H [J/mol]
REAL(dp)                  :: POP_DEL_Hw  ! Delta Hw [J/mol]
REAL(dp)                  :: POP_HSTAR   ! Henry's law constant [atm/M/L]
REAL(dp)                  :: POP_KOA     ! POP octanol-water partition coef
REAL(dp)                  :: POP_KBC     ! POP BC-air partition coeff.
REAL(dp)                  :: POP_XMW     ! POP molecular weight [kg/mol]
```

```
    !-------------------------------------------------------------------
    ! Fields used in ESMF environment only. These arrays won't be used
    ! in a classic environment. They become filled in HCO_SetExtState_ESMF
    ! in hcoi_esmf_mod.F90 (called from within hcoi_gc_main_mod.F90).
    !-------------------------------------------------------------------
    TYPE(ExtDat_3S),  POINTER :: BYNCY       ! Buoyancy
    TYPE(ExtDat_2S),  POINTER :: LFR         ! Lightning flash rate
    TYPE(ExtDat_2R),  POINTER :: CNV_FRC     ! convective fraction (filled
                                             ! from State_Met)
END TYPE Ext_State
```

PRIVATE MEMBER FUNCTIONS:

REVISION HISTORY:

```
02 Oct 2013 - C. Keller   - Initial version
23 Jun 2014 - R. Yantosca - Now add DATA_DIR to Ext_State declaration
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
27 Jun 2014 - C. Keller   - Added FINN biomass burning extension
07 Jul 2014 - R. Yantosca - Modified for GEOS-Chem Rn-Pb-Be simulation
28 Jul 2014 - C. Keller   - Added J-Values for NO2 and O3 to state obj.
20 Aug 2014 - M. Sulprizio- Modified for GEOS-Chem POPs emissions module
01 Oct 2014 - R. Yantosca - Modified for TOMAS sea salt emissions module
11 Dec 2014 - M. Yannetti - Updated DRYCOEFF to REAL(hp)
10 Mar 2015 - C. Keller   - Fields can now be in HEMCO precision or single
                            precision. Single precision is useful for
                            fields used in ESMF setting.
03 Apr 2015 - C. Keller   - Added ExtDat_Set.
21 Feb 2016 - C. Keller   - Update to HEMCO v2.0
03 Mar 2016 - C. Keller   - Added CNV_FRC
20 Apr 2016 - M. Sulprizio- Change JO1D pointer to JOH to reflect that it now
                            points to the effective O3 + hv -> 2OH rates
01 Nov 2016 - M. Sulprizio- Rename TOMAS sea salt to TOMAS Jeagle (J. Kodros)
17 Oct 2017 - C. Keller   - Add lightning flash rate
```

2.17.1 ExtStateInit

Initializes all fields of the ExtState object.

INTERFACE:

```
SUBROUTINE ExtStateInit( ExtState, RC )
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(Ext_State), POINTER        :: ExtState   ! ExtState object
INTEGER,         INTENT(INOUT)  :: RC         ! Success or failure?
```

REMARKS:

You can add more initialization statements as is necessary.

REVISION HISTORY:

15 Dec 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers

---

2.17.2 ExtStateFinal

Finalizes the ExtState object. This removes all defined pointer links (i.e. nullifies ExtDat%Arr), but does not deallocate the target array!

INTERFACE:

```
SUBROUTINE ExtStateFinal( ExtState )
```

INPUT PARAMETERS:

```
TYPE(Ext_State), POINTER  :: ExtState
```

REVISION HISTORY:

03 Oct 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
09 Jul 2015 - E. Lundgren - Add chlorophyll-a (CHLR)

2.17.3 ExtDat_Init_2R

Subroutine ExtDat_Init_2R initializes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Init_2R ( ExtDat, RC )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_2R), POINTER       :: ExtDat
INTEGER,         INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers

---

2.17.4 ExtDat_Init_2S

Subroutine ExtDat_Init_2S initializes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Init_2S ( ExtDat, RC )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_2S), POINTER       :: ExtDat
INTEGER,         INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.5 ExtDat_Init_2I

Subroutine ExtDat_Init_2I initializes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Init_2I ( ExtDat, RC )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_2I), POINTER       :: ExtDat
INTEGER,         INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.6 ExtDat_Init_3R

Subroutine ExtDat_Init_3R initializes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Init_3R ( ExtDat, RC )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_3R), POINTER       :: ExtDat
INTEGER,         INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.7 ExtDat_Init_3S

Subroutine ExtDat_Init_3S initializes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Init_3S ( ExtDat, RC )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_3S), POINTER       :: ExtDat
INTEGER,         INTENT(INOUT) :: RC        ! Return code
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.8 ExtDat_Cleanup_2R

Subroutine ExtDat_Cleanup_2R removes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Cleanup_2R ( ExtDat )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_2R), POINTER       :: ExtDat
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.9 ExtDat_Cleanup_2S

Subroutine ExtDat_Cleanup_2S removes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Cleanup_2S ( ExtDat )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_2S), POINTER       :: ExtDat
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.10 ExtDat_Cleanup_2I

Subroutine ExtDat_Cleanup_2I removes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Cleanup_2I ( ExtDat )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_2I), POINTER       :: ExtDat
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.11 ExtDat_Cleanup_3R

Subroutine ExtDat_Cleanup_3R removes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Cleanup_3R( ExtDat )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_3R), POINTER       :: ExtDat
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.12 ExtDat_Cleanup_3S

Subroutine ExtDat_Cleanup_3S removes the given ExtDat type.

INTERFACE:

```
SUBROUTINE ExtDat_Cleanup_3S( ExtDat )
```

INPUT PARAMETERS:

```
TYPE(ExtDat_3S), POINTER       :: ExtDat
```

REVISION HISTORY:

```
20 Apr 2013 - C. Keller - Initial version
23 Jun 2014 - R. Yantosca - Now use F90 freeform indentation
23 Jun 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
```

2.17.13 ExtDat_Set_2R

Subroutine ExtDat_Set_2R sets/updates the data array of an ExtDat object.

INTERFACE:

```
SUBROUTINE ExtDat_Set_2R ( am_I_Root, HcoState, ExtDat,  &
                           FldName,   RC,       First,   &
                           Trgt,      Filled,   NotFillOk )
```

USES:

```
USE HCO_ARR_MOD,        ONLY : HCO_ArrAssert
USE HCO_STATE_MOD,      ONLY : HCO_State
USE HCO_CALC_MOD,       ONLY : HCO_EvalFld
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )                 :: am_I_Root
TYPE(HCO_State),  POINTER                       :: HcoState
TYPE(ExtDat_2R),  POINTER                       :: ExtDat
CHARACTER(LEN=*), INTENT(IN   )                 :: FldName
INTEGER,          INTENT(INOUT)                 :: RC
LOGICAL,          INTENT(IN   ), OPTIONAL       :: First
REAL(hp),         POINTER      , OPTIONAL       :: Trgt(:,:)
LOGICAL,          INTENT(  OUT), OPTIONAL       :: Filled
LOGICAL,          INTENT(IN   ), OPTIONAL       :: NotFillOk
```

REVISION HISTORY:

```
03 Apr 2015 - C. Keller - Initial version
11 May 2015 - C. Keller - Now use HCO_EvalFld instead of HCO_GetPtr. This
                          allows the application of scale factors to
                          ExtState fields read through the HEMCO interface.
```

2.17.14 ExtDat_Set_2S

Subroutine ExtDat_Set_2S sets/updates the data array of an ExtDat object.

INTERFACE:

```
SUBROUTINE ExtDat_Set_2S ( am_I_Root, HcoState, ExtDat,  &
                           FldName,   RC,       First,   &
                           Trgt,      Filled,   NotFillOk )
```

USES:

```
USE HCO_ARR_MOD,        ONLY : HCO_ArrAssert
USE HCO_STATE_MOD,      ONLY : HCO_State
USE HCO_CALC_MOD,       ONLY : HCO_EvalFld
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )                  :: am_I_Root
TYPE(HCO_State),  POINTER                        :: HcoState
TYPE(ExtDat_2S),  POINTER                        :: ExtDat
CHARACTER(LEN=*), INTENT(IN   )                  :: FldName
INTEGER,          INTENT(INOUT)                  :: RC
LOGICAL,          INTENT(IN   ), OPTIONAL        :: First
REAL(sp),         POINTER      , OPTIONAL        :: Trgt(:,:)
LOGICAL,          INTENT(  OUT), OPTIONAL        :: Filled
LOGICAL,          INTENT(IN   ), OPTIONAL        :: NotFillOk
```

REVISION HISTORY:

```
03 Apr 2015 - C. Keller - Initial version
11 May 2015 - C. Keller - Now use HCO_EvalFld instead of HCO_GetPtr. This
                          allows the application of scale factors to
                          ExtState fields read through the HEMCO interface.
```

2.17.15 ExtDat_Set_2I

Subroutine ExtDat_Set_2I sets/updates the data array of an ExtDat object.

INTERFACE:

```
SUBROUTINE ExtDat_Set_2I ( am_I_Root, HcoState, ExtDat, &
                           FldName,   RC,       First,  &
                           Trgt,      Filled,   NotFillOk )
```

USES:

```
USE HCO_ARR_MOD,        ONLY : HCO_ArrAssert
USE HCO_STATE_MOD,      ONLY : HCO_State
USE HCO_CALC_MOD,       ONLY : HCO_EvalFld
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )                  :: am_I_Root
TYPE(HCO_State),  POINTER                        :: HcoState
TYPE(ExtDat_2I),  POINTER                        :: ExtDat
CHARACTER(LEN=*), INTENT(IN   )                  :: FldName
INTEGER,          INTENT(INOUT)                  :: RC
LOGICAL,          INTENT(IN   ), OPTIONAL        :: First
INTEGER,          POINTER,       OPTIONAL        :: Trgt(:,:)
LOGICAL,          INTENT(  OUT), OPTIONAL        :: Filled
LOGICAL,          INTENT(IN   ), OPTIONAL        :: NotFillOk
```

REVISION HISTORY:

```
03 Apr 2015 - C. Keller - Initial version
11 May 2015 - C. Keller - Now use HCO_EvalFld instead of HCO_GetPtr. This
                          allows the application of scale factors to
                          ExtState fields read through the HEMCO interface.
```

2.17.16 ExtDat_Set_3R

Subroutine ExtDat_Set_3R sets/updates the data array of an ExtDat object.

INTERFACE:

```
SUBROUTINE ExtDat_Set_3R ( am_I_Root, HcoState, ExtDat, FldName,   &
                           RC,        First,    Trgt,   OnLevEdge, &
                           Filled,    NotFillOk                    )
```

USES:

```
USE HCO_ARR_MOD,        ONLY : HCO_ArrAssert
USE HCO_STATE_MOD,      ONLY : HCO_State
USE HCO_CALC_MOD,       ONLY : HCO_EvalFld
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )                  :: am_I_Root
TYPE(HCO_State),  POINTER                        :: HcoState
TYPE(ExtDat_3R),  POINTER                        :: ExtDat
CHARACTER(LEN=*), INTENT(IN   )                  :: FldName
INTEGER,          INTENT(INOUT)                  :: RC
LOGICAL,          INTENT(IN   ), OPTIONAL        :: First
REAL(hp),         POINTER      , OPTIONAL        :: Trgt(:,:,:)
LOGICAL,          INTENT(IN   ), OPTIONAL        :: OnLevEdge
LOGICAL,          INTENT(  OUT), OPTIONAL        :: Filled
LOGICAL,          INTENT(IN   ), OPTIONAL        :: NotFillOk
```

REVISION HISTORY:

```
03 Apr 2015 - C. Keller - Initial version
11 May 2015 - C. Keller - Now use HCO_EvalFld instead of HCO_GetPtr. This
                          allows the application of scale factors to
                          ExtState fields read through the HEMCO interface.
```

2.17.17 ExtDat_Set_3S

Subroutine ExtDat_Set_3S sets/updates the data array of an ExtDat object.

INTERFACE:

```
SUBROUTINE ExtDat_Set_3S ( am_I_Root, HcoState, ExtDat, FldName,   &
                           RC,        First,    Trgt,   OnLevEdge, &
                           Filled,    NotFillOk                    )
```

USES:

```
USE HCO_ARR_MOD,        ONLY : HCO_ArrAssert
USE HCO_STATE_MOD,      ONLY : HCO_State
USE HCO_CALC_MOD,       ONLY : HCO_EvalFld
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )                  :: am_I_Root
TYPE(HCO_State),  POINTER                        :: HcoState
TYPE(ExtDat_3S),  POINTER                        :: ExtDat
CHARACTER(LEN=*), INTENT(IN   )                  :: FldName
INTEGER,          INTENT(INOUT)                  :: RC
LOGICAL,          INTENT(IN   ), OPTIONAL        :: First
REAL(sp),         POINTER      , OPTIONAL        :: Trgt(:,:,:)
LOGICAL,          INTENT(IN   ), OPTIONAL        :: OnLevEdge
LOGICAL,          INTENT(  OUT), OPTIONAL        :: Filled
LOGICAL,          INTENT(IN   ), OPTIONAL        :: NotFillOk
```

REVISION HISTORY:

```
03 Apr 2015 - C. Keller - Initial version
11 May 2015 - C. Keller - Now use HCO_EvalFld instead of HCO_GetPtr. This
                          allows the application of scale factors to
                          ExtState fields read through the HEMCO interface.
```

2.18 Fortran: Module Interface hcox_Iodine_mod.F90

Module HCOX_Iodine_Mod contains routines to calculate oceanic iodine emissions (HOI and I2), following carpenter et al. (2014). The emission is parameterised herein using online feilds for O3, 10 metre wind speed, and ocean surface iodide concentration (parameterised from STT following Chance et al (2014)).

This is a HEMCO extension module that uses many of the HEMCO core utilities.

INTERFACE:

```
MODULE HCOX_Iodine_Mod
```

USES:

```
  USE HCO_Error_Mod
  USE HCO_Diagn_Mod
  USE HCO_State_Mod,  ONLY : HCO_State
  USE HCOX_State_Mod, ONLY : Ext_State

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  PUBLIC :: HCOX_Iodine_Init
  PUBLIC :: HCOX_Iodine_Run
  PUBLIC :: HCOX_Iodine_Final
```

PRIVATE MEMBER FUNCTIONS:

```
N/A
```

REVISION HISTORY:

```
15 Mar 2013 - T. Sherwen - Initial implementation (v9-3-01)
15 Jul 2015 - T. Sherwen - Now a HEMCO extension module
```

2.18.1 HCOX_Iodine_Run

Subroutine HcoX_Iodine_Run is the driver run routine to calculate ocean inorganic iodine emissions in HEMCO.

INTERFACE:

```
SUBROUTINE HCOX_Iodine_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_FluxArr_Mod,      ONLY : HCO_EmisAdd
USE HCO_GeoTools_Mod,     ONLY : HCO_LANDTYPE
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root  ! root CPU?
TYPE(HCO_State), POINTER       :: HcoState   ! Output obj
TYPE(Ext_State), POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT) :: RC         ! Success or failure?
```

REMARKS:

```
References:
============================================================================
(1) Carpenter et al. 2013, https://doi.org/10.1038/ngeo1687
(2) Chance et al. 2014, https://doi.org/10.1039/c4em00139g
(3) Macdonal et al. 2014, https://doi.org/10.5194/acp-14-5841-2014
(4) Sherwen et al. 2016a, https://doi.org/10.5194/acp-16-1161-2016
(5) Sherwen et al. 2016b, https://doi.org/10.5194/acp-16-12239-2016
```

REVISION HISTORY:

```
15 Mar 2013 - T. Sherwen - Initial implementation (v9-3-01)
15 Jul 2015 - T. Sherwen - Now a HEMCO extension module
```

2.18.2 HCOX_Iodine_Init

Subroutine HcoX_Iodine_Init initializes all extension variables.

INTERFACE:

```
SUBROUTINE HCOX_Iodine_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_State_Mod,          ONLY : HCO_GetHcoID
USE HCO_STATE_MOD,          ONLY : HCO_GetExtHcoID
USE HCO_ExtList_Mod,        ONLY : GetExtNr
USE HCO_ExtList_Mod,        ONLY : GetExtOpt
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root   ! root CPU?
TYPE(HCO_State),  POINTER        :: HcoState    ! HEMCO state object
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName     ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState    ! Options object
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC          ! Return status
```

REVISION HISTORY:

```
15 Mar 2013 - T. Sherwen - Initial implementation (v9-3-01)
15 Jul 2015 - T. Sherwen - Now a HEMCO extension module adapted from hcox_seasalt_mod
11 Oct 2017 - R. Yantosca - Fixed typo in comment character (# instead of !)
27 Nov 2017 - C. Keller   - Now output messages to HEMCO logfile
```

2.18.3 HCOX_Iodine_Final

Subroutine HcoX_Iodine_Final deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_Iodine_Final
```

REVISION HISTORY:

```
15 Mar 2013 - T. Sherwen - Initial implementation (v9-3-01)
15 Jul 2015 - T. Sherwen - Now a HEMCO extension module adapted from hcox_seasalt_final
```

2.19 Fortran: Module Interface hcox_driver_mod.F90

Module hcox_driver_mod.F90 contains the driver routines (INIT, RUN, FINAL) for the HEMCO extensions. It determines the extensions to be used (based on the settings specified in the configuration file) and invokes the respective extension module calls.

Call this module at the HEMCO - model interface level to execute the HEMCO extensions.

INTERFACE:

```
MODULE HCOX_Driver_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_State_Mod,  ONLY : HCO_State
USE HCOX_State_Mod, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_Init
PUBLIC :: HCOX_Run
PUBLIC :: HCOX_Final

PRIVATE :: HCOX_DiagnDefine
PRIVATE :: HCOX_DiagnFill
```

REMARKS:

```
(1) The extension option objects (e.g. meteorological variables) are
    defined in the HEMCO - model interface module and passed to this
    module.
(2) To add/remove HEMCO extensions from a model application, just
    add/remove the corresponding initialize, run, and finalize calls
    in the respective driver routines!
```

REVISION HISTORY:

```
 15 Dec 2013 - C. Keller   - Initial version
 01 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 01 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
```

2.19.1 HCOX_Init

Subroutine HCOX_Init is the driver routine to initialize all enabled HEMCO extensions.
INTERFACE:

```
   SUBROUTINE HCOX_Init( amIRoot, HcoState, ExtState, RC )
```

USES:

```
    USE HCOX_State_MOD,         ONLY : ExtStateInit
    USE HCOX_Custom_Mod,        ONLY : HCOX_Custom_Init
    USE HCOX_SeaFlux_Mod,       ONLY : HCOX_SeaFlux_Init
    USE HCOX_ParaNOx_Mod,       ONLY : HCOX_ParaNOx_Init
    USE HCOX_LightNox_Mod,      ONLY : HCOX_LightNox_Init
    USE HCOX_SoilNox_Mod,       ONLY : HCOX_SoilNox_Init
    USE HCOX_DustDead_Mod,      ONLY : HCOX_DustDead_Init
    USE HCOX_DustGinoux_Mod,    ONLY : HCOX_DustGinoux_Init
    USE HCOX_SeaSalt_Mod,       ONLY : HCOX_SeaSalt_Init
    USE HCOX_GFED_Mod,          ONLY : HCOX_GFED_Init
    USE HCOX_MEGAN_Mod,         ONLY : HCOX_MEGAN_Init
    USE HCOX_Finn_Mod,          ONLY : HCOX_FINN_Init
    USE HCOX_GC_RnPbBe_Mod,     ONLY : HCOX_GC_RnPbBe_Init
    USE HCOX_GC_POPs_Mod,       ONLY : HCOX_GC_POPs_Init
    USE HCOX_CH4WetLand_MOD,    ONLY : HCOX_CH4WETLAND_Init
    USE HCOX_AeroCom_Mod,       ONLY : HCOX_AeroCom_Init
    USE HCOX_Iodine_Mod,        ONLY : HCOX_Iodine_Init
#if defined( TOMAS )
    USE HCOX_TOMAS_Jeagle_Mod,   ONLY : HCOX_TOMAS_Jeagle_Init
    USE HCOX_TOMAS_DustDead_Mod, ONLY : HCOX_TOMAS_DustDead_Init
#endif
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )  :: amIRoot        ! Are we on root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
    TYPE(HCO_State),  POINTER        :: HcoState       ! HEMCO state object
    TYPE(Ext_State),  POINTER        :: ExtState       ! HEMCO extension object
    INTEGER,          INTENT(INOUT)  :: RC             ! Failure or success
```

REMARKS:

```
    By default we will call routine ExtStateInit, which initializes the
    ExtState object.  In some instances, we may want to call ExtStateInit
    from a higher-level routine.  For example, this allows us to pass
    via ExtState additional scalar quantities from the driving model to
    HEMCO.  To do this, you will need to (1) call ExtStateInit separately,
    and (2) set the optional argument NoExtStateInit=.FALSE. flag in the
    call to HCOX_INIT.
```

REVISION HISTORY:

```
    12 Sep 2013 - C. Keller   - Initial version
    07 Jul 2014 - R. Yantosca - Now init GEOS-Chem Rn-Pb-Be emissions module
    20 Aug 2014 - M. Sulprizio- Now init GEOS-Chem POPs emissions module
    01 Oct 2014 - R. Yantosca - Now init TOMAS sea salt emissions module
    01 Nov 2016 - M. Sulprizio- Rename TOMAS sea salt to TOMAS Jeagle (J. Kodros)
```

2.19.2 HCOX_Run

Subroutine HCOX_Run is the driver routine to run the HEMCO extensions. All enabled emission extensions are executed, and the emissions calculated therein become added to the respective flux arrays in HcoState.

INTERFACE:

```
SUBROUTINE HCOX_Run( amIRoot, HcoState, ExtState, RC )
```

USES:

```
    USE HCO_Clock_Mod,          ONLY : HcoClock_Get
    USE HCOX_Custom_Mod,        ONLY : HCOX_Custom_Run
    USE HCOX_SeaFlux_Mod,       ONLY : HCOX_SeaFlux_Run
    USE HCOX_ParaNox_Mod,       ONLY : HCOX_ParaNox_Run
    USE HCOX_LightNox_Mod,      ONLY : HCOX_LightNox_Run
    USE HCOX_SoilNox_Mod,       ONLY : HCOX_SoilNox_Run
    USE HCOX_DustDead_Mod,      ONLY : HCOX_DustDead_Run
    USE HCOX_DustGinoux_Mod,    ONLY : HCOX_DustGinoux_Run
    USE HCOX_SeaSalt_Mod,       ONLY : HCOX_SeaSalt_Run
    USE HCOX_Megan_Mod,         ONLY : HCOX_Megan_Run
    USE HCOX_GFED_Mod,          ONLY : HCOX_GFED_Run
    USE HcoX_FINN_Mod,          ONLY : HcoX_FINN_Run
    USE HCOX_GC_RnPbBe_Mod,     ONLY : HCOX_GC_RnPbBe_Run
    USE HCOX_GC_POPs_Mod,       ONLY : HCOX_GC_POPs_Run
    USE HCOX_CH4WetLand_mod,    ONLY : HCOX_CH4Wetland_Run
    USE HCOX_AeroCom_Mod,       ONLY : HCOX_AeroCom_Run
    USE HCOX_Iodine_Mod,        ONLY : HCOX_Iodine_Run
#if defined( TOMAS )
    USE HCOX_TOMAS_Jeagle_Mod,   ONLY : HCOX_TOMAS_Jeagle_Run
    USE HCOX_TOMAS_DustDead_Mod, ONLY : HCOX_TOMAS_DustDead_Run
#endif
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )  :: amIRoot    ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
    TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
    TYPE(Ext_State),  POINTER        :: ExtState   ! Extension options object
    INTEGER,          INTENT(INOUT)  :: RC         ! Failure or success
  !NOTE:
  The ExtState object contains all extension option objects. In particular,
  it contains the pointers to all met fields used by the extensions. These
  pointers have to be set in the HEMCO-model interface module beforehand!
```

REVISION HISTORY:

```
    15 Dec 2013 - C. Keller   - Initial version
    07 Jul 2014 - R. Yantosca - Now Run GEOS-Chem Rn-Pb-Be emissions module
    20 Aug 2014 - M. Sulprizio- Now run GEOS-Chem POPs emissions module
    01 Oct 2014 - R. Yantosca - Now run TOMAS sea salt emissions module
    01 Nov 2016 - M. Sulprizio- Rename TOMAS sea salt to TOMAS Jeagle (J. Kodros)
```

2.19.3 HCOX_Final

Subroutine HCOX_Final finalizes all HEMCO extensions.

INTERFACE:

```
   SUBROUTINE HCOX_Final( am_I_Root, HcoState, ExtState, RC )
```

USES:

```
    USE HCOX_State_Mod,         ONLY : ExtStateFinal
    USE HCOX_Custom_Mod,        ONLY : HCOX_Custom_Final
    USE HCOX_SeaFlux_Mod,       ONLY : HCOX_SeaFlux_Final
    USE HCOX_ParaNOx_Mod,       ONLY : HCOX_PARANOX_Final
    USE HCOX_LightNox_Mod,      ONLY : HCOX_LightNox_Final
    USE HCOX_SoilNox_Mod,       ONLY : HCOX_SoilNox_Final
    USE HCOX_DustDead_Mod,      ONLY : HCOX_DustDead_Final
    USE HCOX_DustGinoux_Mod,    ONLY : HCOX_DustGinoux_Final
    USE HCOX_SeaSalt_Mod,       ONLY : HCOX_SeaSalt_Final
    USE HCOX_MEGAN_Mod,         ONLY : HCOX_MEGAN_Final
    USE HCOX_GFED_Mod,          ONLY : HCOX_GFED_Final
    USE HcoX_FINN_Mod,          ONLY : HcoX_FINN_Final
    USE HCOX_GC_RnPbBe_Mod,     ONLY : HCOX_GC_RnPbBe_Final
    USE HCOX_GC_POPs_Mod,       ONLY : HCOX_GC_POPs_Final
    USE HCOX_CH4WetLand_Mod,    ONLY : HCOX_CH4Wetland_Final
    USE HCOX_AeroCom_Mod,       ONLY : HCOX_AeroCom_Final
    USE HCOX_Iodine_Mod,        ONLY : HCOX_Iodine_Final
#if defined( TOMAS )
    USE HCOX_TOMAS_Jeagle_Mod,   ONLY : HCOX_TOMAS_Jeagle_Final
    USE HCOX_TOMAS_DustDead_Mod, ONLY : HCOX_TOMAS_DustDead_Final
#endif
```

INPUT PARAMETERS:

```
    LOGICAL,          INTENT(IN   )  :: am_I_Root  ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
    TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
    TYPE(Ext_State),  POINTER        :: ExtState   ! Extension options object
    INTEGER,          INTENT(INOUT)  :: RC         ! Failure or success
```

REVISION HISTORY:

```
12 Sep 2013 - C. Keller   - Initial version
07 Jul 2014 - R. Yantosca - Now finalize GEOS-Chem Rn-Pb-Be emissions pkg
20 Aug 2014 - M. Sulprizio- Now finalize GEOS-Chen POPs emissions module
01 Oct 2014 - R. Yantosca - Now finalize TOMAS sea salt emissions module
09 Mar 2015 - C. Keller   - Now pass HcoState since it is needed by some
                            finalization calls.
01 Nov 2016 - M. Sulprizio- Rename TOMAS sea salt to TOMAS Jeagle (J. Kodros)
```

2.19.4 HCOX_DiagnDefine

Subroutine HCOX_DiagnDefine creates custom-defined diagnostics. This is very preliminary and for testing only.

INTERFACE:

```
SUBROUTINE HCOX_DiagnDefine( am_I_Root, HcoState, ExtState, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root    ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
TYPE(Ext_State),  POINTER        :: ExtState   ! Extension options object
INTEGER,          INTENT(INOUT)  :: RC         ! Failure or success
```

REVISION HISTORY:

```
19 Feb 2015 - C. Keller   - Initial version
```

2.19.5 DgnDefine

Helper routine to define a target diagnostics.

INTERFACE:

```
SUBROUTINE DgnDefine ( am_I_Root, HcoState, DgnName, Trgt2D, RC )
```

USES:

```
USE HCO_DIAGN_MOD, ONLY : Diagn_Create
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root    ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
CHARACTER(LEN=*), INTENT(IN   )  :: DgnName      ! diagnostics name
REAL(sp),         INTENT(IN   )  :: Trgt2D(HcoState%NX,HcoState%NY)
INTEGER,          INTENT(INOUT)  :: RC         ! Failure or success
```

REVISION HISTORY:

```
19 Feb 2015 - C. Keller   - Initial version
```

2.19.6 HCOX_DiagnFill

Subroutine HCOX_DiagnFill fills custom-defined diagnostics.

INTERFACE:

```
SUBROUTINE HCOX_DiagnFill( am_I_Root, HcoState, ExtState, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root    ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER        :: HcoState   ! HEMCO state object
TYPE(Ext_State),  POINTER        :: ExtState   ! Extension options object
INTEGER,          INTENT(INOUT)  :: RC         ! Failure or success
```

REVISION HISTORY:

```
19 Feb 2015 - C. Keller   - Initial version
```

2.20 Fortran: Module Interface hcox_dust_dead_mod.F

Module hcox_dust_dead_mod.F contains routines and variables from Charlie Zender's DEAD dust mobilization model. Most routines are from Charlie Zender, but have been modified and/or cleaned up for inclusion into GEOS-Chem.

This is a HEMCO extension module that uses many of the HEMCO core utilities.

NOTE: The current (dust) code was validated at 2 x 2.5 resolution. We have found that running at 4x5 we get much lower (50emissions than at 2x2.5. Recommend we either find a way to scale the U* computed in the dust module, or run a 1x1 and store the the dust emissions, with which to drive lower resolution runs. – Duncan Fairlie, 1/25/07

(We'll) implement the [dust] code in the standard [GEOS-Chem] model and put a warning about expected low bias when the simulation is run at 4x5. Whoever is interested in running dust at 4x5 in the future can deal with making the fix. – Daniel Jacob, 1/25/07

!REFERENCES:

- Zender, C. S., Bian, H., and Newman, D.: Mineral Dust Entrainment and Deposition (DEAD) model: Description and 1990s dust climatology, Journal of Geophysical Research: Atmospheres, 108, 2003.

INTERFACE:

```
MODULE HCOX_TOMAS_DustDead_Mod
```

USES:

```
USE HCO_ERROR_MOD
USE HCO_DIAGN_MOD
USE HCOX_State_MOD,    ONLY : Ext_State
USE HCO_STATE_MOD,     ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_TOMAS_DustDead_Run
PUBLIC :: HCOX_TOMAS_DustDead_Init
PUBLIC :: HCOX_TOMAS_DustDead_Final
PUBLIC :: HCOX_TOMAS_DustDead_GetFluxTun
```

REVISION HISTORY:

```
(1 ) Added parallel DO loop in GET_ORO (bmy, 4/14/04)
(2 ) Now references "directory_mod.f" (bmy, 7/20/04)
(3 ) Fixed typo in ORO_IS_LND for PGI compiler (bmy, 3/1/05)
(4 ) Modified for GEOS-5 and GCAP met fields (swu, bmy, 8/16/05)
(5 ) Now make sure all USE statements are USE, ONLY (bmy, 10/3/05)
(6 ) Now uses GOCART source function (tdf, bmy, 1/25/07)
(7 ) Modifications for 0.5 x 0.667 grid (yxw, dan, bmy, 11/6/08)
(8 ) Updates for nested grids (amv, bmy, 12/18/09)
01 Mar 2012 - R. Yantosca - Now reference new grid_mod.F90
25 Nov 2013 - C. Keller   - Now a HEMCO extension
06 Oct 2014 - C. Keller   - Allow mass flux tuning factor be set in
                            configuration file.
07 Jan 2016 - E. Lundgren - Change dry air gas constant and molec wt to
                            match GC values and update acc due to gravity
                            and universal gas constant to NIST 2014 values
14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
24 Aug 2017 - M. Sulprizio- Remove support for GEOS-4, GEOS-5, and MERRA
```

2.20.1 HCOX_TOMAS_DustDead_Run

Subroutine HcoX_DustDead_Run is the driver routine for the HEMCO DEAD dust extension.

INTERFACE:

```
SUBROUTINE HCOX_TOMAS_DustDead_Run( am_I_Root, ExtState, HcoState
& , RC )
```

USES:

```
USE HCO_CALC_MOD,      ONLY : HCO_EvalFld
USE HCO_FLUXARR_MOD,   ONLY : HCO_EmisAdd
USE HCO_CLOCK_MOD,     ONLY : HcoClock_Get
USE HCO_CLOCK_MOD,     ONLY : HcoClock_First
USE HCO_State_Mod,     ONLY : HCO_GetHcoID
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root
TYPE(Ext_State), POINTER        :: ExtState    ! Module options
TYPE(HCO_State), POINTER        :: HcoState    ! Hemco state
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
08 Apr 2004 - T. D. Fairlie - Initial version
(1 ) Added OpenMP parallelization, added comments (bmy, 4/8/04)
(2 ) Bug fix: DSRC needs to be held PRIVATE (bmy, 4/14/04)
(3 ) Now references DATA_DIR from "directory_mod.f" (bmy, 7/20/04)
(4 ) Now make sure all USE statements are USE, ONLY (bmy, 10/3/05)
(5 ) Bug fix: It should be SNOW/1d3 not SNOW*1d3 (tdf, bmy, 11/18/05)
(6 ) Updated output statement (bmy, 1/23/07)
(7 ) Use SNOMAS (m H2O) for GEOS-5 (bmy, 1/24/07)
25 Aug 2010 - R. Yantosca - Treat MERRA in the same way as for GEOS-5
25 Aug 2010 - R. Yantosca - Added ProTeX headers
03 Sep 2010 - R. Yantosca - Bug fix, SNOMAS was mislabled in GEOS-5
                            and has units of mm H2O instead of m H2O
                            so we need to convert to m H2O.
08 Feb 2012 - R. Yantosca - Treat GEOS-5.7.x in the same way as MERRA
01 Mar 2012 - R. Yantosca - Now use GET_YMID_R(I,J,L) from grid_mod.F90
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
25 Nov 2013 - C. Keller   - Now a HEMCO extension
06 Oct 2014 - C. Keller   - Now calculate pressure center from edges.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.20.2 HCOX_TOMAS_DustDead_Init

Subroutine HcoX_DustDead_Init initializes the HEMCO DUST_DEAD extension.

INTERFACE:

```
SUBROUTINE HCOX_TOMAS_DustDead_Init ( am_I_Root, HcoState
&                            ,ExtName, ExtState,    RC     )
```

USES:

```
USE HCO_ExtList_Mod,    ONLY : GetExtNr, GetExtOpt
USE HCO_STATE_MOD,      ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
TYPE(HCO_State),  POINTER        :: HcoState   ! Hemco state
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName    ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState     ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
25 Nov 2013 - C. Keller   - Now a HEMCO extension
14 Aug 2014 - R. Yantosca - Now always print out extension info
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.20.3 HCOX_TOMAS_DustDead_Final

Subroutine HcoX_DustDead_Final finalizes the HEMCO DUST_DEAD extension.

INTERFACE:

```
SUBROUTINE HCOX_TOMAS_DustDead_Final( ExtState )
```

INPUT PARAMETERS:

```
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

REVISION HISTORY:

```
 25 Nov 2013 - C. Keller - Now a HEMCO extension
!NOTES:
```

2.20.4 InstGet

Subroutine InstGet returns a pointer to the desired instance.

INTERFACE:

```
SUBROUTINE InstGet ( Instance, Inst, RC, PrevInst )
```

INPUT PARAMETERS:

```
INTEGER                              :: Instance
TYPE(MyInst),     POINTER            :: Inst
INTEGER                              :: RC
TYPE(MyInst),     POINTER, OPTIONAL :: PrevInst
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.20.5 InstCreate

Subroutine InstCreate creates a new instance.

INTERFACE:

```
SUBROUTINE InstCreate ( ExtNr, Instance, Inst, RC )
```

INPUT PARAMETERS:

```
INTEGER,       INTENT(IN)       :: ExtNr
```

OUTPUT PARAMETERS:

```
INTEGER,       INTENT(  OUT)    :: Instance
TYPE(MyInst),  POINTER          :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,       INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.20.6 InstRemove

Subroutine InstRemove creates a new instance.

INTERFACE:

```
SUBROUTINE InstRemove ( Instance )
```

INPUT PARAMETERS:

```
INTEGER                         :: Instance
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.21 Fortran: Module Interface hcox_template_mod.F90

Module hcox_template_mod.F90 is a HEMCO extension template. It serves as a starting point for a new emission extension within HEMCO. Specifically, it provides the framework to use multiple 'instances' of the extension at the same time.

INTERFACE:

```
MODULE HCOX_<yourname>_mod
```

USES:

```
USE HCO_Error_MOD
USE HCO_Diagn_MOD
USE HCOX_TOOLS_MOD
USE HCOX_State_MOD, ONLY : Ext_State
USE HCO_State_MOD,  ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_<yourmame>_Run
PUBLIC :: HCOX_<yourname>_Init
PUBLIC :: HCOX_<yourname>_Final
```

PRIVATE MEMBER FUNCTIONS:

REVISION HISTORY:

```
19 Feb 2016 - C. Keller   - Initial version
```

2.21.1 HCOX_¡yourname¿_Run

Write a description here

INTERFACE:

```
SUBROUTINE HCOX_<yourname>_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_FluxArr_Mod,  ONLY : HCO_EmisAdd
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
TYPE(Ext_State), POINTER       :: ExtState    ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState    ! Hemco state
INTEGER,         INTENT(INOUT) :: RC          ! Success or failure
```

REMARKS:

REVISION HISTORY:

```
19 Feb 2016 - C. Keller   - Initial version
```

2.21.2 HCOX_¡yourname¿_Init

Write a description here

INTERFACE:

```
SUBROUTINE HCOX_<yourname>_Init( am_I_Root, HcoState, ExtName, &
                                 ExtState,  RC                 )
```

USES:

```
USE HCO_ExtList_Mod,    ONLY : GetExtNr
USE HCO_ExtList_Mod,    ONLY : GetExtOpt
USE HCO_ExtList_Mod,    ONLY : GetExtSpcVal
USE HCO_STATE_MOD,      ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root
CHARACTER(LEN=*), INTENT(IN   ) :: ExtName    ! Extension name
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER       :: HcoState   ! Hemco state
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller   - Initial version
```

2.21.3 HCOX_¡yourname¿_Final

Write a description here

INTERFACE:

```
SUBROUTINE HCOX_AeroCom_Final ( ExtState )
```

INPUT PARAMETERS:

```
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller   - Initial version
```

2.21.4 InstGet

Subroutine InstGet returns a pointer to the desired instance.

INTERFACE:

```
SUBROUTINE InstGet ( Instance, Inst, RC, PrevInst )
```

INPUT PARAMETERS:

```
INTEGER                            :: Instance
TYPE(MyInst),     POINTER          :: Inst
INTEGER                            :: RC
TYPE(MyInst),     POINTER, OPTIONAL :: PrevInst
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.21.5 InstCreate

Subroutine InstCreate adds a new instance to the list of instances, assigns a unique instance number to this new instance, and archives this instance number to output argument Instance.

INTERFACE:

```
SUBROUTINE InstCreate ( ExtNr, Instance, Inst, RC )
```

INPUT PARAMETERS:

```
INTEGER,       INTENT(IN)       :: ExtNr
```

OUTPUT PARAMETERS:

```
INTEGER,       INTENT(  OUT)    :: Instance
TYPE(MyInst),  POINTER          :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,       INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.21.6 InstRemove

Subroutine InstRemove removes an instance from the list of instances.

INTERFACE:

```
SUBROUTINE InstRemove ( Instance )
```

INPUT PARAMETERS:

```
INTEGER                         :: Instance
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.21.7 hcox_finn_include.H

Include file with FINN emission factor data that was originally contained in files FINN_EFratios_CO2.csv and FINN_VOC_speciation.csv. We transform these data into hardwired F90 assignment statements so that we can avoid reading ASCII files in the ESMF environment. **REMARKS:**

```
ABOUT THIS FILE:
----------------
This file was created by script HEMCO/Extensions/Preprocess/finn.pl.
This script can be executed with the following command:
  cd HEMCO/Extensions/Preprocess
  make finn
This will regenerate this include file from the original data and
automatically place it in the HEMCO/Extensions directory.
White space has been removed in order to reduce the file size as much
as possible.  If you have to recreate this file, then it is easier to
generate via the Perl script than to try to hand edit the code below.
```

REVISION HISTORY:

```
11 Aug 2014 - R. Yantosca - Initial version
10 Jul 2015 - R. Yantosca - Fixed minor issues in ProTeX header
24 Nov 2017 - B. Henderson - Changed to v1.5/v1.6/v2 consistent
                             based on Wiedinmeyer csv
```

2.22 Fortran: Module Interface ocean_toolbox_mod.F90

Module Ocean_ToolBox_Mod contains functions and routines to calculate the ocean exchange velocity for any gas, according to Johnson, 2010.
References:

- M.T. Johnson: "A numerical scheme to calculate temperature and salinity dependent air-water transfer velocities for any gas", Ocean Sci. 6, 913-932, 2010.
- Liss and Slater: Flux of gases across the air-sea interface, Nature, 247, 1974.
- Laliberte, M: "Model for calculating the viscosity of aqueous solutions", Journal of Chemical & Engineering Data, 52, 2007.
- Millero and Poisson: "International one-atmosphere equation of state of seawater", Deep Sea Res. Pt A, 28, 1981.
- Wilke and Chang: "Correlation of diffusion coefficients in dilute solutions", AIChE Journal, 1, 1955.
- Hayduk and Minhas, "Correlations for prediction of molecular diffusivities in liquids, Can. J. Chem. Eng., 60, 1982,
- E. Fuller et al.: "New method for prediction of binary gas-phase diffusion coefficients", Industrial & Engineering Chemistry, 58, 1966.
- Saltzman et al.: Experimental determination of the diffusion coefficient of dimethylsulfide in water, J. Geophys. Res., 98, 1993.

INTERFACE:

```
MODULE Ocean_ToolBox_Mod
```

USES:

```
IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: Calc_Kg
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: Calc_Ka
PRIVATE :: Calc_Kl
PRIVATE :: N_SW
PRIVATE :: P_SW
PRIVATE :: V_SW
PRIVATE :: D_WC
PRIVATE :: D_HM
PRIVATE :: Schmidt_W
PRIVATE :: Schmidt_Saltzmann
PRIVATE :: Schmidt_Acet
```

```
PRIVATE :: Schmidt_Ald2
PRIVATE :: N_Air
PRIVATE :: P_Air
PRIVATE :: V_Air
PRIVATE :: D_Air
PRIVATE :: Schmidt_G
! PARAMETER
INTEGER, PARAMETER :: OC_SUCCESS = 0
INTEGER, PARAMETER :: OC_FAIL    = -999
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
03 Oct 2-14 - C. Keller: Added error trap for negative Schmidt numbers
10 Mar 2017 - M. Sulprizio- Add function Schmidt_Ald2
```

2.22.1 Calc_Kg

Subroutine Calc_Kg is the wrapper routine to calculate the exchange velocity Kg used for calculating the ocean-air flux (cf. Liss & Slater 1974) as:

F = Kg (Cg - H * Cl)

where Cg and Cl are the bulk gas and liquid concentrations and H is the Henry constant (H= Cgs/Cls).

1/Kg = 1/ka + H/Kl = Ra + Rl.

Note that Kg is returned in m/s and not cm h-1, as is usually reported for exchange velocities!

INTERFACE:

```
SUBROUTINE Calc_Kg( T, P, V, SALT, H, VB, MW, SCW, KG, RC, RA_OVER_RL, VERBOSE )
```

INPUT PARAMETERS:

```
REAL*8,  INTENT(IN   )  :: T    ! Surface temperature     [C]
REAL*8,  INTENT(IN   )  :: P    ! Surface pressure        [Pa]
REAL*8,  INTENT(IN   )  :: V    ! Surface wind speed      [m/s]
REAL*8,  INTENT(IN   )  :: SALT ! Salinity                [PSU]
REAL*8,  INTENT(IN   )  :: H    ! Henry constant          [-]
REAL*8,  INTENT(IN   )  :: VB   ! Liquid mol. volume      [cm3/mol]
REAL*8,  INTENT(IN   )  :: MW   ! Molecular weight        [g/mol]
INTEGER, INTENT(IN   )  :: SCW  ! Parameterization type
                                ! for Schmidt number in water
LOGICAL, INTENT(IN   ), OPTIONAL  :: VERBOSE   ! turn on verbose output
```

OUTPUT PARAMETERS:

```
REAL*8,  INTENT(  OUT)  :: KG   ! Exchange velocity       [ms-1]
INTEGER, INTENT(  OUT)  :: RC   ! Error code
REAL*8,  INTENT(  OUT), OPTIONAL  :: RA_OVER_RL ! Ra/Rl   [-]
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
21 May 2013 - C. Keller: SCW added to argument list
15 Aug 2014 - C. Keller: Now limit temperature to -40 degC to avoid overflow
                         error. Also added error trap for temperatures
                         between -10.7 and -10.9 degrees that cause a div-zero
                         error in subroutine N_SW.
```

2.22.2 Calc_Ka

Calc_Ka returns the air exchange velocity KA.

INTERFACE:

```
FUNCTION Calc_Ka(T,P,V,MW,VB,VERB) RESULT(KA)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN)  :: T,P,V, MW, VB!T in C, P in Pa
LOGICAL, INTENT(IN) :: VERB
```

RETURN VALUE:

```
REAL*8              :: KA
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.3 Calc_Kl

Calc_Kl calculates the water exchange velocity Kl following Nightingale et al., Geophysical Research Letters, 2000.

INTERFACE:

```
FUNCTION Calc_Kl(T,V,S,VB,SCW,VERB) RESULT(K)
```

INPUT PARAMETERS:

```
REAL*8,  INTENT(IN) :: T,S,V,VB
INTEGER, INTENT(IN) :: SCW
LOGICAL, INTENT(IN) :: VERB
```

RETURN VALUE:

```
REAL*8              :: K
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.4 N_SW

N_SW returns the dynamic seawater viscosity following Laliberte, 2007.

INTERFACE:

```
FUNCTION N_SW(T,S) RESULT(N)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T,S !temperature (C) and salinity
```

RETURN VALUE:

```
REAL*8             :: N   ! Dynamic viscosity
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller   - Adapted from F. Paulot
 1 Jul 2014 - R. Yantosca - Bug fix: Don't take LOG(NI) if NI is zero
```

2.22.5 P_SW

P_SW returns the seawater density following Millero and Poisson, 1981.

INTERFACE:

```
FUNCTION P_SW(T,S) RESULT(P)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T,S
```

RETURN VALUE:

```
REAL*8             :: P
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.6 V_SW

V_SW returns the ???

INTERFACE:

```
FUNCTION V_SW(T,S) RESULT(V)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T,S
```

RETURN VALUE:

```
REAL*8             :: V
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.7 D_WC

D_WC returns the (water) diffusion coefficient following Wilke and Chang, 1955.

INTERFACE:

```
FUNCTION D_WC(T,S,VB) RESULT(D)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T,S,VB
```

RETURN VALUE:

```
REAL*8             :: D
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.8 D_HM

D_HM returns the (water) diffusivity following Hayduk and Minhas, 1982.

INTERFACE:

```
FUNCTION D_HM(T,S,VB) RESULT(D)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T,S,VB
```

RETURN VALUE:

```
REAL*8             :: D
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.9 Schmidt_W

Schmidt_W returns the Schmidt number of the gas in the water following Johnson, 2010.

INTERFACE:

```
FUNCTION Schmidt_W(T,S,VB) RESULT(SC)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T,S,VB
```

RETURN VALUE:

```
REAL*8             :: SC
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.10 Schmidt_Saltzmann

Schmidt_Saltzmann returns the Schmidt number of the gas in the water calculated according to Saltzmann et al., 1993.

INTERFACE:

```
FUNCTION Schmidt_Saltzmann(T) RESULT(SC)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T   ! Temperature in C
```

RETURN VALUE:

```
REAL*8             :: SC
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.11 Schmidt_Acet

Schmidt_Acet returns the Schmidt number of acetone.

INTERFACE:

```
FUNCTION Schmidt_Acet(T) RESULT(SC)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T   ! Temperature in C
```

RETURN VALUE:

```
REAL*8             :: SC
!NOTES:
 Coefficients for fitting the Schmidt number for acetone [unitless]
   A0 =  3287.687d0
   A1 = -136.2176d0
   A2 =  2.20642d0
   A3 = -0.01410642d0
```

REVISION HISTORY:

```
11 Aug 2013 - C. Keller: Initial version
```

2.22.12 Schmidt_Ald2

Schmidt_Ald2 returns the Schmidt number of acetaldehyde.

INTERFACE:

```
FUNCTION Schmidt_Ald2(T) RESULT(SC)
```

INPUT PARAMETERS:

```
 REAL*8, INTENT(IN) :: T   ! Temperature in C
```

RETURN VALUE:

```
 REAL*8             :: SC
!NOTES:
 Coefficients for fitting the Schmidt number for acetaldehyde [unitless]
 Derived using polynomial fit (code provided by qli, same as used
 for acetone, methanol)
 and partial molal volume of acetaldehyde at its normal boiling
 temperature (51.8 cm3/g/mole) calculated using Le Bas method
 see "The Properties of Gases and Liquids", Reid, Prausnitz, Sherwood.
   A0 = 2581.709d0
   A1 = -106.9671d0
   A2 = 1.73263d0
   A3 = -0.0110773d0
```

REVISION HISTORY:

```
 10 Mar 2017 - M. Sulprizio- Initial version
```

2.22.13 N_Air

N_Air returns the dynamic air viscosity.

INTERFACE:

```
FUNCTION N_Air(T) RESULT(N)
```

INPUT PARAMETERS:

```
 REAL*8, INTENT(IN) :: T
```

RETURN VALUE:

```
 REAL*8             :: N
```

REVISION HISTORY:

```
 11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.14 P_Air

P_Air returns the kinematic air viscosity.

INTERFACE:

```
FUNCTION P_Air(T) RESULT(P)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T
```

RETURN VALUE:

```
REAL*8             :: P
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.15 V_Air

V_Air returns the kinematic air viscosity (m2/s).

INTERFACE:

```
FUNCTION V_Air(T) RESULT(V)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T
```

RETURN VALUE:

```
REAL*8             :: V
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.16 D_Air

D_Air returns the gas phase diffusion coefficient according to Fuller et al., 1966.

INTERFACE:

```
FUNCTION D_Air(T,P,MW,VB) RESULT(D)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T  !T in C
REAL*8, INTENT(IN) :: P  !P in Pa
REAL*8, INTENT(IN) :: MW !MW in g/mol
REAL*8, INTENT(IN) :: VB !Liq. molar volume (cm3/mol)
```

RETURN VALUE:

```
REAL*8             :: D
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.22.17 Schmidt_G

Schmidt_G returns the schmidt number of the gas in the air.

INTERFACE:

```
FUNCTION Schmidt_G(T,P,MW,VB) RESULT(SC)
```

INPUT PARAMETERS:

```
REAL*8, INTENT(IN) :: T, P, MW, VB
```

RETURN VALUE:

```
REAL*8             :: SC
```

REVISION HISTORY:

```
11 Apr 2013 - C. Keller: Adapted from F. Paulot
```

2.23 Fortran: Module Interface hcox_lightnox_mod.F90

Module HCOX_LightNOx_Mod contains routines to compute NO lightning emissions, according to the GEOS-Chem lightning algorithms.

This is a HEMCO extension module that uses many of the HEMCO core utilities. In particular, the LIS-OTD local redistribution factors are now read through the HEMCO framework, and the corresponding netCDF input file is specified in the HEMCO configuration file. The table of cumulative distribution functions used to vertically distribute lightning NOx emissions is specified in the extension switch section of the configuration file.

References:

- Murray, L. T., Jacob, D. J., Logan, J. A., Hudman, R. C., and Koshak, W. J.: *Optimized regional and interannual variability of lightning in a global chemical transport model con- strained by LIS/OTD satellite data*, J. Geophys. Res., Atmospheres, 117, 2012.
- Ott, L. E., K. E. Pickering, G. L. Stenchikov, D. J. Allen, A. J. DeCaria, B. Ridley, R.-F. Lin, S. Lang, and W.-K. Tao, *Production of lightning NOx and its vertical distribution calculated from three-dimensional cloud-scale chemical transport model simulations*, J. Geophys. Res., 115, D04301, 2010.

INTERFACE:

```
MODULE HCOX_LightNOx_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Diagn_Mod
USE HCOX_TOOLS_MOD
USE HCO_State_Mod,  ONLY : HCO_State
USE HCOX_State_MOD, ONLY : Ext_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCOX_LightNOX_Run
PUBLIC  :: HCOX_LightNOX_Final
PUBLIC  :: HCOX_LightNOX_Init
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: LIGHTNOX
PRIVATE :: LIGHTDIST
PRIVATE :: FLASHES_CTH
PRIVATE :: GET_IC_CG_RATIO
PRIVATE :: GET_OTD_LIS_SCALE
```

PUBLIC DATA MEMBERS:

REMARKS:

```
%%% NOTE: MFLUX and PRECON methods are now deprecated (ltm, bmy, 7/9/09)
                                                                             .
References:
=============================================================================
(1 ) Price & Rind (1992), JGR, vol. 97, 9919-9933.
(2 ) Price & Rind (1994), M. Weather Rev, vol. 122, 1930-1939.
(3 ) Allen & Pickering (2002), JGR, 107, D23, 4711, doi:10.1029/2002JD002066
(4 ) Hudman et al (2007), JGR, 112, D12S05, doi:10.1029/2006JD007912
(5 ) Sauvage et al, 2007, ACP,
      http://www.atmos-chem-phys.net/7/815/2007/acp-7-815-2007.pdf
(6 ) Ott et al., (2010), JGR
(7 ) Allen et al., (2010), JGR
(8 ) Murray et al., (2011), in prep.
```

REVISION HISTORY:

```
14 Apr 2004 - L. Murray, R. Hudman - Initial version
(1 ) Based on "lightnox_nox_mod.f", but updated for near-land formulation
     and for CTH, MFLUX, PRECON parameterizations (ltm, bmy, 5/10/06)
(2 ) Now move computation of IC/CG flash ratio out of routines FLASHES_CTH,
     FLASHES_MFLUX, FLASHES_PRECON, and into routine GET_IC_CG_RATIO.
     Added a fix in LIGHTDIST for pathological grid boxes.  Set E_IC_CG=1
     according to Allen & Pickering [2002].  Rename OTDSCALE array to
     OTD_REG_REDIST, and also add OTD_LOC_REDIST array.  Now scale
     lightnox to 6 Tg N/yr for both 2x25 and 4x5.  Rename routine
     GET_OTD_LIS_REDIST to GET_REGIONAL_REDIST.  Add similar routine
     GET_LOCAL_REDIST.  Removed GET_OTD_LOCp AL_REDIST.  Bug fix: divide
     A_M2 by 1d6 to get A_KM2. (rch, ltm, bmy, 2/22/07)
(3 ) Rewritten for separate treatment of LNOx emissions at tropics &
     midlatitudes, based on Hudman et al 2007.  Removed obsolete
     variable E_IC_CG. (rch, ltm, bmy, 3/27/07)
(4 ) Changes implemented in this version (ltm, bmy, 10/3/07)
     * Revert to not classifying near-land as land
     * Eliminate NOx emisisons per path length entirely
     * Scale tropics to 260 mol/fl constraint from Randall Martin's
        4.4 Tg and OTD-LIS avg ann flash rate
     * Remove top-down scaling (remove the three functions)
     * Allow option of mid-level scaling to match global avg ann flash
        rate between G-C and OTD-LIS 11-year climatology (new function)
     * Local Redist now a la Murray et al, 2007 in preparation (monthly)
     * Replace GEMISNOX (from CMN_NOX) with module variable EMIS_LI_NOx
(5 ) Added MFLUX, PRECON redistribution options (ltm, bmy, 11/29/07)
(6 ) Updated OTD/LIS scaling for GEOS-5 to get more realistic totals
     (ltm, bmy, 2/20/08)
(7 ) Now add the proper scale factors for the GEOS-5 0.5 x 0.666 grid
     and the GEOS-3 1x1 nested N. America grid in routine
     GET_OTD_LIS_SCALE. (yxw, dan, ltm, bmy, 11/14/08)
(8 ) Added quick fix for GEOS-5 reprocessed met fields (ltm, bmy, 2/18/09)
(9 ) Added quick fix for GEOS-5 years 2004, 2005, 2008 (ltm, bmy, 4/29/09)
(10) Updated OTD/LIS scaling for GEOS-5 reprocessed data (ltm, bmy, 7/10/09)
(11) Updated for GEOS-4 1 x 1.25 grid (lok, ltm, bmy, 1/13/10)
(12) Reprocessed for CLDTOPS calculation error; Updated Ott vertical
     profiles; Removal of depreciated options, e.g., MFLUX and PRECON;
     GEOS5 5.1.0 vs. 5.2.0 special treatment; MERRA; Other changes.
     Please see PDF on wiki page for full description of lightnox
     changes to v9-01-01. (ltm, 1/25/11)
13 Aug 2010 - R. Yantosca - Add modifications for MERRA
10 Nov 2010 - L. Murray   - Updated OTD/LIS local scaling for MERRA 4x5
10 Nov 2010 - R. Yantosca - Added ProTeX headers
02 Feb 2012 - R. Yantosca - Added modifications for GEOS-5.7.x met fields
01 Mar 2012 - R. Yantosca - Now reference new grid_mod.F90
03 Aug 2012 - R. Yantosca - Move calls to findFreeLUN out of DEVEL block
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
22 Jul 2014 - R. Yantosca - Now hardwire the Lesley Ott et al CDF's in
```

```
                            lightning_cdf_mod.F90.  This avoids having to
                            read an ASCII input in the ESMF environment.
    13 Jan 2015 - L. Murray   - Add most recent lightning updates to HEMCO version
    26 Feb 2015 - R. Yantosca - Restore reading the lightning CDF's from an
                            ASCII file into the PROFILE array.  This helps
                            to reduce compilation time.
    31 Jul 2015 - C. Keller   - Added option to define scalar/gridded scale
                            factors via HEMCO configuration file.
    14 Oct 2016 - C. Keller   - Now use HCO_EvalFld instead of HCO_GetPtr.
    02 Dec 2016 - M. Sulprizio- Update WEST_NS_DIV from 23d0 to 35d0 (K. Travis)
    16 Feb 2017 - L. Murray   - Updated BETA factors for all GEOS-FP/MERRA-2
                            products fields available by v11-01 release
                            (through Dec. 2016), and latest version of
                            LIS/OTD satellite climatology.
    24 Aug 2017 - M. Sulprizio- Remove support for GCAP, GEOS-4, GEOS-5 and MERRA
    17 Oct 2017 - C. Keller   - Add option to use GEOS-5 lightning flash rate
                            (LFR). Autoselection of flash rate scale factor.
```

2.23.1 HCOX_LightNOx_Run

Subroutine HCOX_LIGHTNOX_RUN is the driver routine to calculate lightnox NOx emissions and return them to the HEMCO driver routine.

INTERFACE:

```
SUBROUTINE HCOX_LightNOx_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_FluxArr_Mod,  ONLY : HCO_EmisAdd
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
TYPE(Ext_State), POINTER         :: ExtState    ! Module options
TYPE(HCO_State), POINTER         :: HcoState   ! Hemco options
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
09 Oct 1997 - R. Yantosca - Initial version
(1 ) Remove IOFF, JOFF from the argument list.  Also remove references
     to header files "CMN_O3" and "comtrid.h" (bmy, 3/16/00)
(2 ) Now use allocatable array for ND32 diagnostic (bmy, 3/16/00)
(3 ) Now reference BXHEIGHT from "dao_mod.f".  Updated comments, cosmetic
     changes.  Replace LCONVM with the parameter LLCONVM. (bmy, 9/18/02)
```

```
(4 ) Removed obsolete reference to "CMN".  Now bundled into
     "lightnox_mod.f" (bmy, 4/14/04)
(5 ) Renamed from EMLIGHTNOX_NL to EMLIGHTNOX.  Now replace GEMISNOX
     (from CMN_NOX) with module variable EMIS_LI_NOx. (ltm, bmy, 10/3/07)
10 Nov 2010 - R. Yantosca - Added ProTeX headers
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
25 Mar 2013 - R. Yantosca - Now accept State_Chm
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
07 Oct 2013 - C. Keller   - Now allow OTD-LIS scale factor to be set
                            externally. Check for transition to Sep 2008.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.23.2 LightNOx

Subroutine LIGHTNOX uses Price & Rind's formulation for computing NOx emission from lightnox (with various updates).

INTERFACE:

```
SUBROUTINE LightNOx( am_I_Root, HcoState, ExtState, Inst, RC )
```

USES:

```
USE HCO_Calc_Mod,     ONLY : HCO_EvalFld
USE HCO_EmisList_Mod, ONLY : HCO_GetPtr
USE HCO_GeoTools_Mod, ONLY : HCO_LANDTYPE
USE HCO_Clock_Mod,    ONLY : HcoClock_Get
USE HCO_Clock_Mod,    ONLY : HcoClock_First
USE HCO_ExtList_Mod,  ONLY : GetExtOpt
USE HCO_Types_Mod,    ONLY : DiagnCont
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root
TYPE(HCO_State), POINTER        :: HcoState  ! Output obj
TYPE(Ext_State), POINTER        :: ExtState    ! Module options
TYPE(MyInst   ), POINTER        :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
10 May 2006 - L. Murray - Initial version
(1 ) Now recompute the cold cloud thickness according to updated formula
     from Lee Murray.  Rearranged argument lists to routines FLASHES_CTH,
     FLASHES_MFLUX, FLASHES_PRECON.  Now call READ_REGIONAL_REDIST and
     READ_LOCAL_REDIST. Updated comments accordingly.  Now apply
     FLASH_SCALE to scale the total lightnox NOx to 6 Tg N/yr.  Now apply
     OTD/LIS regional or local redistribution (cf. B. Sauvage) to the ND56
     diagnostic. lightnox redistribution to the ND56 diag.  Renamed
     REGSCALE variable to REDIST.  Bug fix: divide A_M2 by 1d6 to get
     A_KM2. (rch, ltm, bmy, 2/14/07)
(2 ) Rewritten for separate treatment of LNOx emissions at tropics &
     midlatitudes (rch, ltm, bmy, 3/27/07)
(3 ) Remove path-length algorithm.  Renamed from LIGHTNOX_NL to LIGHTNOX.
     Other improvements. (ltm, bmy, 9/24/07)
(4 ) Remove depreciated options; Update to new Ott et al vertical profiles;
     Reprocessed for bug in CLDTOPS calculation. See PDF on wiki for
     full description of changes for v9-01-01. (ltm, bmy, 1/25,11)
10 Nov 2010 - R. Yantosca - Added ProTeX headers
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
06 Oct 2014 - C. Keller   - Now calculate pressure centers from edges.
16 Jan 2015 - R. Yantosca - Bug fix: TmpScale should be REAL(dp)
11 Mar 2015 - C. Keller   - Now determine LTOP from buoyancy for grid boxes
                            where convection is explicitly resolved. For now,
                            this will only work in an ESMF environment.
31 Jul 2015 - C. Keller   - Take into account scalar/gridded scale factors
                            defined in HEMCO configuration file.
03 Mar 2016 - C. Keller   - Use buoyancy in combination with convective
                            fraction CNV_FRC (ESMF only).
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.23.3 LightDist

Subroutine LightDist reads in the CDF used to partition the column lightnox NOx into the GEOS-Chem vertical layers.

INTERFACE:

```
SUBROUTINE LightDist( I, J, LTOP, H0, XLAT, TOTAL, VERTPROF, &
                      ExtState, HcoState, SFCTYPE, cMt, MTYPE, Inst )
```

INPUT PARAMETERS:

```
INTEGER,         INTENT(IN)  :: I          ! Longitude index
INTEGER,         INTENT(IN)  :: J          ! Latitude index
INTEGER,         INTENT(IN)  :: LTOP       ! Level of conv cloud top
REAL*8,          INTENT(IN)  :: H0         ! Conv cloud top height [m]
```

```
REAL*8,          INTENT(IN)  :: XLAT       ! Latitude value [degrees]
REAL*8,          INTENT(IN)  :: TOTAL      ! Column Total # of LNOx molec
TYPE(Ext_State), POINTER     :: ExtState   ! Module options
TYPE(HCO_State), POINTER     :: HcoState   ! Hemco state object
INTEGER,         INTENT(IN)  :: SFCTYPE    ! Surface type
INTEGER,         INTENT(IN)  :: cMt        ! Current month
TYPE(MyInst),    POINTER     :: Inst       ! Hemco state object
```

OUTPUT PARAMETERS:

```
REAL*8,          INTENT(OUT) :: VERTPROF(HcoState%NZ) ! Vertical profile
INTEGER,         INTENT(OUT) :: MTYPE                 ! lightning type
```

REMARKS:

References:

(1) Pickering et al., JGR 103, 31,203 - 31,316, 1998.
(2) Ott et al., JGR, 2010
(3) Allen et al., JGR, 2010

REVISION HISTORY:

18 Sep 2002 - M. Evans - Initial version (based on Yuhang Wang's code)
(1) Use functions IS_LAND and IS_WATER to determine if the given grid box is over land or water. These functions work for all DAO met field data sets. (bmy, 4/2/02)
(2) Renamed M2 to LTOP and THEIGHT to H0 for consistency w/ variable names w/in "lightnox.f". Now read the "light_dist.dat.geos3" file for GEOS-3 directly from the DATA_DIR/lightnox_NOx_200203/ subdirectory. Now read the "light_dist.dat" file for GEOS-1, GEOS-STRAT directly from the DATA_DIR/lightnox_NOx_200203/ subdirectory. Added descriptive comment header. Now trap I/O errors across all platforms with subroutine "ioerror.f". Updated comments, cosmetic changes. Redimension FRAC(NNLIGHT) to FRAC(LLPAR). (bmy, 4/2/02)
(3) Deleted obsolete code from April 2002. Now reference IU_FILE and IOERROR from "file_mod.f". Now use IU_FILE instead of IUNIT as the file unit number. (bmy, 6/27/02)
(4) Now reference BXHEIGHT from "dao_mod.f" (bmy, 9/18/02)
(5) Bug fix: add GEOS_4 to the #if block (bmy, 3/4/04)
(6) Now bundled into "lightnox_mod.f". CDF's are now read w/in routine INIT_LIGHTNOX to allow parallelization (bmy, 4/14/04)
(7) Now references DATA_DIR from "directory_mod.f" (bmy, 7/20/04)
(8) Now uses near-land formulation (ltm, bmy, 5/10/06)
(9) Added extra safety check for pathological boxes (bmy, 12/11/06)
(10) Remove the near-land formulation, except for PRECON (ltm, bmy, 9/24/07)
(11) Now use the Ott et al. [2010] profiles, and apply consistently with GMI model [Allen et al., 2010] (ltm, bmy, 1/25/11).
10 Nov 2010 - R. Yantosca - Added ProTeX headers

```
01 Mar 2012 - R. Yantosca - Now use GET_AREA_CM2(I,J,L) from grid_mod.F90
15 Jun 2012 - Nielsen - INQUIRE finds free logical unit number for IU_FILE
09 Nov 2012 - M. Payer    - Replaced all met field arrays with State_Met
                            derived type object
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
```

2.23.4 Flashes_CTH

Subroutine Flashes_CTH determines the rate of lightnox flashes per minute based on the height of convective cloud tops, and the intra-cloud to cloud-ground strike ratio.

INTERFACE:

```
SUBROUTINE Flashes_CTH( I, J, HEIGHT, FLASHRATE, SFCTYPE )
```

INPUT PARAMETERS:

```
 INTEGER, INTENT(IN)  :: I           ! Longitude index
 INTEGER, INTENT(IN)  :: J           ! Latitude index
 REAL*8,  INTENT(IN)  :: HEIGHT      ! Height of conv cloud top [m]
 INTEGER, INTENT(IN)  :: SFCTYPE     ! Surface type
```

OUTPUT PARAMETERS:

```
 REAL*8,  INTENT(OUT) :: FLASHRATE   ! LightNOX flash rate [flashes/min]
```

REVISION HISTORY:

```
10 May 2006 - L. Murray - Initial version
(1  ) Subroutine renamed from FLASHES (ltm, bmy, 5/10/06)
(2  ) Remove CCTHICK, IC_CG_RATIO as arguments.  Remove computation of
      IC_CG_RATIO and move that to GET_IC_CG_RATIO. (ltm, bmy, 12/11/06)
(3  ) Remove the near-land formulation (i.e. use function IS_LAND
      instead of IS_NEAR).(ltm, bmy, 9/24/07)
10 Nov 2010 - R. Yantosca - Added ProTeX headers
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
```

2.23.5 Get_IC_CG_Ratio

Function Get_IC_CG_Ratio calculates the Intra-Cloud (IC) and Cloud-to-Ground (CG) lightnox flash ratio based on the method of Price and Rind 1993, which is calculated from the cold-cloud depth (CCTHICK).

INTERFACE:

```
FUNCTION Get_IC_CG_ratio( CCTHICK ) RESULT( IC_CG_RATIO )
```

INPUT PARAMETERS:

```
REAL*8,  INTENT(IN) :: CCTHICK       ! Cold cloud thickness [m]
```

RETURN VALUE:

```
REAL*8              :: IC_CG_RATIO   ! Intra-cloud/cloud-ground ratio
```

REVISION HISTORY:

```
11 Dec 2006 - R. Yantosca - Initial version
(1 ) Split off from FLASHES_CTH, FLASHES_MFLUX, FLASHES_PRECON into this
     separate function (ltm, bmy, 12/11/06)
(2 ) Bug fix for XLF compiler (morin, bmy, 7/8/09)
10 Nov 2010 - R. Yantosca - Added ProTeX headers
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
```

2.23.6 Get_OTD_LIS_Scale

Function GET_OTD_LIS_SCALE returns a met-field dependent scale factor which is to be applied to the lightnox flash rate to bring the annual average flash rate to match that of the OTD-LIS climatology (45.9 flashes/sec). Computed by running the model over the 11-year OTD-LIS campaign window and comparing the average flash rates, or as many years as are available.

INTERFACE:

```
SUBROUTINE Get_OTD_LIS_Scale( am_I_Root, HcoState, BETA, RC )
```

USES:

```
USE HCO_Clock_Mod, ONLY : HcoClock_Get
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN   )   :: am_I_Root  ! Root CPU?
TYPE(HCO_State), POINTER :: HcoState   ! HEMCO state obj
REAL*8,  INTENT(  OUT)   :: BETA       ! Scale factor
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT)   :: RC         ! Suc
```

REVISION HISTORY:

```
24 Sep 2007 - L. Murray - Initial version
(1 ) Added MFLUX, PRECON scaling for GEOS-4.  Also write messages for met
     field types/grids where scaling is not defined. (ltm, bmy, 11/29/07)
(2 ) Now use different divisor for local redist (ltm, bmy, 2/20/08)
(3 ) Now compute the proper scale factor for GEOS-5 0.5 x 0.666 grids
     and the GEOS-3 1x1 nested NA grid (yxw, dan, ltm, bmy, 11/14/08)
(4 ) Added "quick fix" for reprocessed GEOS-5 met fields to be used when
     the IN_CLOUD_OD switch is turned on. (ltm, bmy, 2/18/09)
(5 ) Added "quick fix" for 2004, 2005, 2008 OTD/LIS (ltm, bmy, 4/29/09)
```

```
(6 ) Updated scale factors for GEOS-5 based on 4+ years of data.  Remove
     temporary fixes. (bmy, 7/10/09)
(7 ) Modification for GEOS-4 1 x 1.25 grid (lok, ltm, bmy, 1/13/10)
(8 ) Reprocessed for error in CLDTOPS field; Updated for GEOS
     5.1.0 vs. 5.2.0; MERRA added; (ltm, bmy, 1/25/11)
10 Nov 2010 - R. Yantosca - Added ProTeX headers
02 Feb 2012 - R. Yantosca - Compute BETA for MERRA 2 x 2.5
02 Feb 2012 - R. Yantosca - Compute BETA for GEOS-5.7.x
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
04 Nov 2014 - Y. X. Wang  - Define BETA, ANN_AVG_FLASHRATE for the
                            GEOS-FP 025x03125 NESTED_CH grid
14 Jan 2015 - L. Murray   - Updated GEOS-FP files through Oct 2014
01 Apr 2015 - R. Yantosca - Cosmetic changes
01 Apr 2015 - R. Yantosca - Bug fix: GRID025x0325 should be GRID025x03125
01 Mar 2016 - L. Murray   - Add preliminary values for MERRA-2 4x5, NA, CH
19 Jul 2016 - L. Murray   - Add preliminary values for MERRA-2 2x2.5
24 Sep 2017 - L. Murray   - Removed legacy resolutions. Updated LIS/OTD
                            HRMC climatology. Final global MERRA-2 values.
                            Updated GEOS-FP and regional MERRA-2 values.
```

2.23.7 HCOX_LightNOx_Init

Subroutine HCOX_LIGHTNOX_INIT allocates all module arrays. It also reads the lightnox CDF data from disk before the first lightnox timestep.

INTERFACE:

```
SUBROUTINE HCOX_LightNOx_Init( am_I_Root, HcoState, ExtName, ExtState, RC )
```

USES:

```
USE HCO_Chartools_Mod, ONLY : HCO_CharParse
USE HCO_ExtList_Mod,   ONLY : GetExtNr
USE HCO_ExtList_Mod,   ONLY : GetExtOpt
USE HCO_ExtList_Mod,   ONLY : GetExtSpcVal
USE HCO_State_Mod,     ONLY : HCO_GetHcoID
USE HCO_State_Mod,     ONLY : HCO_GetExtHcoID
USE HCO_ReadList_Mod,  ONLY : ReadList_Remove
USE inquireMod,        ONLY : findfreeLUN
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
TYPE(HCO_State),  POINTER        :: HcoState   ! Hemco options
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName    ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState     ! Module options
```

OUTPUT PARAMETERS:

```
INTEGER,          INTENT(  OUT)  :: RC
```

REVISION HISTORY:

```
14 Apr 2004 - R. Yantosca - Initial version
(1 ) Now reference DATA_DIR from "directory_mod.f"
(2 ) Now call GET_MET_FIELD_SCALE to initialize the scale factor for
     each met field type and grid resolution (bmy, 8/25/05)
(3 ) Now make sure all USE statements are USE, ONLY (bmy, 10/3/05)
(4 ) Now get the box area at 30N for MFLUX, PRECON (lth, bmy, 5/10/06)
(5 ) Rename OTDSCALE to OTD_REG_REDIST.  Also add similar array
     OTD_LOC_REDIST.  Now call GET_FLASH_SCALE_CTH, GET_FLASH_SCALE_MFLUX,
     GET_FLASH_SCALE_PRECON depending on the type of lightnox param used.
     Updated comments.  (ltm, bmy, 1/31/07)
(6 ) Removed near-land stuff.  Renamed from HCOX_LightNOX_Init_NL to
     HCOX_LightNOX_Init.  Now allocate EMIS_LI_NOx. (ltm, bmy, 10/3/07)
(7 ) Also update location of PDF file to lightnox_NOx_200709 directory.
     (bmy, 1/24/08)
(8 ) Read in new Ott profiles from lightnox_NOx_201101. Remove
     depreciated options. (ltm, bmy, 1/25/11)
10 Nov 2010 - R. Yantosca - Added ProTeX headers
01 Mar 2012 - R. Yantosca - Removed reference to GET_YEDGE
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
26 Feb 2015 - R. Yantosca - Now re-introduce reading the CDF table from an
                            ASCII file (reduces compilation time)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.23.8 hcox_lightnox_final

Subroutine HCOX_LIGHTNOX_FINAL deallocates all module arrays.

INTERFACE:

```
SUBROUTINE HCOX_LightNOx_Final( ExtState )
```

INPUT PARAMETERS:

```
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

REVISION HISTORY:

```
14 Apr 2004 - R. Yantosca - Initial version
(1 ) Now deallocates OTDSCALE (ltm, bmy, 5/10/06)
(2 ) Rename OTDSCALE to OTD_REG_REDIST.  Now deallocate OTD_LOC_REDIST.
     (bmy, 1/31/07)
(3 ) Renamed from HCOX_LightNOX_Final_NL to HCOX_LightNOX_Final.
     Now deallocate EMIS_LI_NOx. (ltm, bmy, 10/3/07)
```

```
(4 ) Remove depreciated options. (ltm, bmy, 1/25/11)
10 Nov 2010 - R. Yantosca - Added ProTeX headers
22 Oct 2013 - C. Keller   - Now a HEMCO extension.
22 Jul 2014 - R. Yantosca - PROFILE is now set in lightning_cdf_mod.F90
26 Feb 2015 - R. Yantosca - Now re-introduce PROFILE, as we read the CDF
                            table from an ASCII file (reduces compile time)
```

2.23.9 InstGet

Subroutine InstGet returns a pointer to the desired instance.

INTERFACE:

```
SUBROUTINE InstGet ( Instance, Inst, RC, PrevInst )
```

INPUT PARAMETERS:

```
INTEGER                              :: Instance
TYPE(MyInst),     POINTER            :: Inst
INTEGER                              :: RC
TYPE(MyInst),     POINTER, OPTIONAL  :: PrevInst
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.23.10 InstCreate

Subroutine InstCreate adds a new instance to the list of instances, assigns a unique instance number to this new instance, and archives this instance number to output argument Instance.

INTERFACE:

```
SUBROUTINE InstCreate ( ExtNr, Instance, Inst, RC )
```

INPUT PARAMETERS:

```
INTEGER,       INTENT(IN)       :: ExtNr
```

OUTPUT PARAMETERS:

```
INTEGER,       INTENT(  OUT)    :: Instance
TYPE(MyInst),  POINTER          :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,       INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.23.11 InstRemove

Subroutine InstRemove removes an instance from the list of instances.

INTERFACE:

```
SUBROUTINE InstRemove ( Instance )
```

INPUT PARAMETERS:

```
 INTEGER                         :: Instance
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.24 Fortran: Module Interface hcox_soilnox_mod.F90

Module HCOX_SoilNOx_Mod contains routines to compute soil NOx emissions. We follow the implementation in GEOS-Chem by Hudman et al 2012.

INTERFACE:

```
MODULE HCOX_SoilNOx_Mod
```

USES:

```
USE HCO_ERROR_Mod
USE HCO_CHARTOOLS_MOD
USE HCO_DIAGN_Mod
USE HCOX_TOOLS_MOD
USE HCOX_State_Mod,     ONLY : Ext_State
USE HCO_STATE_Mod,      ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_SoilNOx_Run
PUBLIC :: HCOX_SoilNOx_Init
PUBLIC :: HCOX_SoilNOx_Final
```

REMARKS:

```
This is a HEMCO extension module that uses many of the HEMCO core
utilities.

 Original codes from:
   HARVARD ATMOSPHERIC CHEMISTRY MODELING GROUP
   MODULE FOR SOIL NOX EMISSIONS
```

```
  by Yuhang Wang, Gerry Gardner, and Prof. Daniel Jacob
Updated model code:
  by  Rynda Hudman, Neil Moore, Randall Martin, and Bram Maasakkers
                                                                             .
The soil NOx code has been updated from the original implementation
of Yienger & Levy [1995] from  Wang et al., [1998] as summarized below.
                                                                             .
Old:
ENOx   = f( T, biome, w/d)  x Pulse(precip) x canopy uptake + FERT
                                                                             .
New:
ENOx   = f( T, biome, WFPS, Fert)  x Pulse(dryspell) x canopy uptake
                                                                             .
1 - Update moisture treatment: soil moisture as a continuous variable
using WFPS rather than discrete wet/dry states and purely exponential
T impact (impact = -1. Tg N/yr)
                                                                             .
2 - Update to Fertilizer:  new fertilizer maps including chemical and
manure fertilizer from Potter et al., [2010] distributed using MODIS EVI
seasonality, online-N deposition as a fertilizer source, and N-fertilizer
source subject to T, WFPS, and pulsing like other N (impact = +1.3 Tg N/yr)
                                                                             .
3- Update Pulsing Scheme: Yan et al., [2005] (shorter, stronger pulses)
(impact = +1. Tg N/yr). Also added restart file containing dry spell
information to properly account for dry spell length in continuing runs.
                                                                             .
References:
==============================================================================
(1 ) Wang, Y., D.J. Jacob, and J.A. Logan,  Global simulation of
      tropospheric O3-NOx-hydrocarbon chemistry, 1. Model formulation,
      J. Geophys. Res., 103/D9, 10, 713-10,726, 1998.
(2 ) Yienger, J.J, and H. Levy, Empirical model of global soil-biogenic
      NOx emissions, J. Geophys. Res., 100, D6, 11,447-11464, June 20, 1995.
(3 ) Yan, X., T. Ohara, and H. Akimoto, Statistical modeling of global
      soil NOx emissions, Global Biogeochem. Cycles, 19, GB3019,
      doi:10.1029/2004GB002276, 2005.
(4 ) Potter, P., Ramankutty, N., Bennett, E., and Donner, S.:
      Characterizing the Spatial Patterns of Global Fertilizer Application
      and Manure Production, Earth Interactions, 14, 1-22,
      10.1175/2009EI288.1, 2010.
(5 ) Moore, N.E., Improving global bottom-up biogenic soil NOx inventories,
      Master's Thesis, Dalhousie University, 2007.
(6 ) Hudman, R.C., N.E. Moore, A.K. Mebust, R.V. Martin, A.R. Russell,
      L.C. Valin, and R.C Cohen, Steps toward a mechanistic model of global
      soil nitric oxide emissions: implementation and space
      based-constraints, Atmos. Chem. Phys., 12, 7779-7795,
      doi:10.5194/acp-12-7779-2012, 2012.
```

REVISION HISTORY:

```
17 Aug 2009 - R. Yantosca     - Columnized and cleaned up
17 Aug 2009 - R. Yantosca     - Added ProTeX headers
31 Jan 2011 - R. Hudman       - Added new code12259.perceus-ucb0
31 Jan 2011 - R. Hudman       - Updated headers
29 Aug 2012 - J.D. Maasakkers - Implemented Jacob and Bakwin CRF
29 Aug 2012 - J.D. Maasakkers - Adapted code to work with new (online
                                regridded) landfraction, climate and
                                fertilizer data
29 Aug 2012 - J.D. Maasakkers - Removed all unused Wang et al. code
                                (comments)
04 Nov 2013 - C. Keller       - Moved all soil NOx routines into one
                                module. Now a HEMCO extension.
28 Jul 2014 - C. Keller       - Now allow DRYCOEFF to be read through
                                configuration file (as setting)
11 Dec 2014 - M. Yannetti     - Changed REAL*8 to REAL(hp)
14 Oct 2016 - C. Keller       - Now use HCO_EvalFld instead of HCO_GetPtr.
```

2.24.1 HCOX_SoilNOx_Run

Subroutine HcoX_SoilNox_Run is the driver routine to calculate ship NOx emissions for the current time step. Emissions in [kg/m2/s] are added to the emissions array of the passed

INTERFACE:

```
SUBROUTINE HCOX_SoilNOx_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_Types_Mod,      ONLY : DiagnCont
USE HCO_CLOCK_MOD,      ONLY : HcoClock_First
USE HCO_CLOCK_MOD,      ONLY : HcoClock_Rewind
USE HCO_FLuxArr_Mod,    ONLY : HCO_EmisAdd
USE HCO_EmisList_Mod,   ONLY : HCO_GetPtr
USE HCO_Calc_Mod,       ONLY : HCO_EvalFld
USE HCO_ExtList_Mod,    ONLY : GetExtOpt
USE HCO_ExtList_Mod,    ONLY : HCO_GetOpt
USE HCO_Restart_Mod,    ONLY : HCO_RestartGet
USE HCO_Restart_Mod,    ONLY : HCO_RestartWrite
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root
TYPE(Ext_State), POINTER        :: ExtState    ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER        :: HcoState   ! Output obj
INTEGER,         INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
05 Nov 2013 - C. Keller - Initial Version
08 May 2015 - C. Keller - Now read/write restart variables from here to
                          accomodate replay runs in GEOS-5.
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
29 Mar 2017 - M. Sulprizio- Read DEP_RESERVOIR_DEFAULT field from file for
                            use when when DEP_RESERVOIR is not found in the
                            HEMCO restart file
```

2.24.2 HCOX_SoilNOx_Init

Subroutine HcoX_SoilNox_Init initializes the HEMCO SOILNOX extension.

INTERFACE:

```
SUBROUTINE HCOX_SoilNOx_Init( am_I_Root, HcoState, ExtName, &
                              ExtState,    RC                 )
```

USES:

```
USE HCO_ExtList_Mod,        ONLY : GetExtNr, GetExtOpt
USE HCO_ExtList_Mod,        ONLY : GetExtSpcVal
USE HCO_STATE_MOD,          ONLY : HCO_GetExtHcoID
USE HCO_Restart_Mod,        ONLY : HCO_RestartDefine
```

ARGUMENTS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root
TYPE(HCO_State),  POINTER        :: HcoState   ! Output obj
CHARACTER(LEN=*), INTENT(IN   )  :: ExtName    ! Extension name
TYPE(Ext_State),  POINTER        :: ExtState     ! Module options
INTEGER,          INTENT(INOUT)  :: RC
```

REMARKS:

REVISION HISTORY:

```
05 Nov 2013 - C. Keller   - Initial Version
12 May 2015 - R. Yantosca - Cosmetic changes
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.24.3 HCOX_SoilNOx_Final

Subroutine HcoX_SoilNOx_Final finalizes the HEMCO SOILNOX extension.

INTERFACE:

```
SUBROUTINE HCOX_SoilNOx_Final( am_I_Root, HcoState, ExtState, RC )
!USES
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root     ! Root CPU?
TYPE(HCO_State), POINTER        :: HcoState      ! HEMCO State obj
TYPE(Ext_State),  POINTER       :: ExtState      ! Extension state
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,         INTENT(INOUT)  :: RC
```

REVISION HISTORY:

```
 05 Nov 2013 - C. Keller - Initial Version
!NOTES:
```

2.24.4 HCOX_SoilNOx_GetFertScale

Function HCOX_SoilNOx_GETFERTSCALE returns the scale factor applied to fertilizer NOx emissions.

INTERFACE:

```
FUNCTION HCOX_SoilNOx_GetFertScale() RESULT ( FERT_SCALE )
```

ARGUMENTS:

```
REAL(hp) :: FERT_SCALE
```

REMARKS:

REVISION HISTORY:

```
 11 Dec 2013 - C. Keller   - Initial version
 12 May 2015 - R. Yantosca - Bug fix: PGI expects routine name to end w/ ()
!EOP
------------------------------------------------------------------------------
BOC
```

LOCAL VARIABLES:

```
! Scale factor so that fertilizer emission = 1.8 Tg N/yr
! (Stehfest and Bouwman, 2006)
! before canopy reduction
FERT_SCALE = 0.0068
! Value calculated by running the 2x2.5 model
! For now, use this value for all resolutions since regular soil NOx
! emissions change with resolution as well (J.D. Maasakkers)
```

```
   END FUNCTION HCOX_SoilNOx_GetFertScale
  EOC
  ------------------------------------------------------------------------------
                   Harvard-NASA Emissions Component (HEMCO)                   !
  ------------------------------------------------------------------------------
%//////////////////////////////////////////////////////////////

\mbox{}\hrulefill\


  \subsubsection [Soil\_NOx\_Emission] {Soil\_NOx\_Emission}


  Subroutine Soil\_NOx\_Emission computes the emission of soil and
    fertilizer NOx for the GEOS-Chem model.
  \\
  \\{\bf INTERFACE:}
\begin{verbatim}   SUBROUTINE Soil_NOx_Emission( ExtState,  Inst, &
                                TS_EMIS,   I, J, &
                                SOILFRT,   &
                                GWET_PREV, DRYPERIOD, &
                                PFACTOR,   SOILNOx,   &
                                DEPN,      FERTDIAG,  &
                                UNITCONV,  R_CANOPY )
```

INPUT PARAMETERS:

```
    TYPE(Ext_State), POINTER :: ExtState      ! Module options
    TYPE(MyInst),    POINTER :: Inst          ! Instance object
    REAL*4,   INTENT(IN)  :: TS_EMIS          ! Emission timestep [s]
    INTEGER,  INTENT(IN)  :: I                ! grid box lon index
    INTEGER,  INTENT(IN)  :: J                ! grid box lat index
    REAL(hp), INTENT(IN)  :: DEPN             ! Dry Dep Fert term [kg/m2]
    REAL(hp), INTENT(IN)  :: SOILFRT          ! Fertilizer emissions [kg/m2]
    REAL(hp), INTENT(IN)  :: UNITCONV         ! ng N to kg NO

    !Input parameters for the canopy reduction factor
    REAL(hp), INTENT(IN)  :: R_CANOPY(:)      ! Resist. of canopy to NOx [1/s]
```

OUTPUT PARAMETERS:

```
    REAL(hp), INTENT(OUT) :: SOILNOx          ! Soil NOx emissions [kg/m2/s]
    REAL(sp), INTENT(OUT) :: GWET_PREV        ! Soil Moisture Prev timestep
    REAL(sp), INTENT(OUT) :: DRYPERIOD        ! Dry period length in hours
    REAL(sp), INTENT(OUT) :: PFACTOR          ! Pulsing Factor
    REAL(hp), INTENT(OUT) :: FERTDIAG         ! Fert emissions [kg/m2/s]
```

REMARKS:

```
R_CANOPY is computed in routine GET_CANOPY_NOX of "canopy_nox_mod.f".
This was originally in the GEOS-Chem dry deposition code, but was split
off in order to avoid an ugly code dependency between the dry deposition
and soil NOx codes.

As of v9-02, this module uses the MODIS/Koppen biome types instead
of the Olson land type / biome type, making it different from the original
dry deposition code (J.D. Maasakkers)
```

REVISION HISTORY:

```
17 Aug 2009 - R. Yantosca - Columnized and cleaned up
17 Aug 2009 - R. Yantosca - Added ProTeX headers
31 Jan 2011 - R. Hudman   - New Model added
23 Oct 2012 - M. Payer    - Now reference Headers/gigc_errcode_mod.F90
12 May 2015 - R. Yantosca - Cosmetic changes
```

2.24.5 Get_Canopy_NOx

Subroutine Get_Canopy_NOx computes the bulk surface resistance of the canopy to NOx. This computation was originally done within legacy routine DEPVEL (in "drydep_mod.f"). Moving this computation to Get_Canopy_NOx now allows for a totally clean separation between dry deposition routines and emissions routines in GEOS-Chem.

INTERFACE:

```
SUBROUTINE Get_Canopy_NOx( am_I_Root, HcoState, ExtState, Inst, RC )
```

USES:

```
USE Drydep_Toolbox_Mod, ONLY : BIOFIT
```

ARGUMENTS:

```
LOGICAL,         INTENT(IN   )  :: am_I_Root
TYPE(HCO_State), POINTER        :: HcoState
TYPE(Ext_State), POINTER        :: ExtState
TYPE(MyInst),    POINTER        :: Inst
INTEGER,         INTENT(INOUT)  :: RC
```

REMARKS:

```
For backwards compatibility, the bulk surface resistance is stored
in common block array CANOPYNOX in "commsoil.h".  Leave it like this
for the time being...we'll clean it up when we fix all of the soil
NOx routines.
```

REVISION HISTORY:

```
22 Jun 2009 - R. Yantosca     - Split off from "drydep_mod.f"
14 Jun 2012 - J.D. Maasakkers - Rewritten as a function of the
                                   MODIS/Koppen biometype
09 Nov 2012 - M. Payer        - Replaced all met field arrays with State_Met
                                  derived type object
13 Dec 2012 - R. Yantosca     - Removed ref to obsolete CMN_DEP_mod.F
28 Jul 2014 - C. Keller       - Added error trap for DRYCOEFF
11 Dec 2014 - M. Yannetti     - Added BIO_RESULT
12 May 2015 - R. Yantosca     - Cosmetic changes
```

2.24.6 DiffG

Function DiffG calculates the molecular diffusivity [m2/s] in air for a gas X of molecular weight XM [kg] at temperature TK [K] and pressure PRESS [Pa].

INTERFACE:

```
FUNCTION DiffG( TK, PRESS, XM ) RESULT( DIFF_G )
```

INPUT PARAMETERS:

```
REAL(hp), INTENT(IN) :: TK      ! Temperature [K]
REAL(hp), INTENT(IN) :: PRESS   ! Pressure [hPa]
REAL(hp), INTENT(IN) :: XM      ! Molecular weight of gas [kg]
```

RETURN VALUE:

```
REAL(hp)             :: DIFF_G  ! Molecular diffusivity [m2/s]
```

REMARKS:

```
We specify the molecular weight of air (XMAIR) and the hard-sphere molecular
radii of air (RADAIR) and of the diffusing gas (RADX).  The molecular
radius of air is given in a Table on p. 479 of Levine [1988].  The Table
also gives radii for some other molecules.  Rather than requesting the user
to supply a molecular radius we specify here a generic value of 2.E-10 m for
all molecules, which is good enough in terms of calculating the diffusivity
as long as molecule is not too big.
```

REVISION HISTORY:

```
22 Jun 2009 - R. Yantosca - Copied from "drydep_mod.f"
07 Jan 2016 - E. Lundgren - Update Avogadro's # to NIST 2014 value
```

2.24.7 Get_Dep_N

Subroutine GET_DEP_N sums dry and wet deposition since prev. timestep and calculates contribution to fertilizer N source. Output is in kg NO/m2.

INTERFACE:

```
SUBROUTINE Get_Dep_N( I, J, ExtState, HcoState, Inst, DEP_FERT )
```

INPUT PARAMETERS:

```
INTEGER,  INTENT(IN)         :: I
INTEGER,  INTENT(IN)         :: J
TYPE(Ext_State), POINTER     :: ExtState
TYPE(HCO_State), POINTER     :: HcoState
TYPE(MyInst),    POINTER     :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
! Dep emitted as Fert [kgNO/m2]
REAL(hp) ,  INTENT(INOUT) :: DEP_FERT
```

REVISION HISTORY:

```
23 Oct 2012 - M. Payer    - Added ProTeX headers
```

2.24.8 Source_DryN

Subroutine SOURCE_DRYN gets dry deposited Nitrogen since last emission time step, converts to kg NO/m2/s.

INTERFACE:

```
FUNCTION Source_Dryn( I, J, ExtState, HcoState, Inst ) RESULT( DRYN )
```

INPUT PARAMETERS:

```
INTEGER,         INTENT(IN) :: I
INTEGER,         INTENT(IN) :: J
TYPE(Ext_State), POINTER    :: ExtState   ! Module options
TYPE(HCO_State), POINTER    :: HcoState   ! Output obj
TYPE(MyInst),    POINTER    :: Inst
```

RETURN VALUE:

```
REAL(hp)                     :: DRYN       ! Dry dep. N since prev timestep
```

REVISION HISTORY:

```
23 Oct 2012 - M. Payer    - Added ProTeX headers
```

2.24.9 Source_WetN

Subroutine Source_WetN gets wet deposited Nitrogen since last emission time step, converts to kg NO/m2/s.

INTERFACE:

```
FUNCTION Source_WetN( I, J, ExtState, HcoState ) RESULT(WETN )
```

INPUT PARAMETERS:

```
INTEGER,         INTENT(IN) :: I
INTEGER,         INTENT(IN) :: J
TYPE(Ext_State), POINTER    :: ExtState   ! Module options
TYPE(HCO_State), POINTER    :: HcoState   ! Output obj
```

RETURN VALUE:

```
REAL(hp)                     :: WETN       ! Dry dep. N since prev timestep
```

REVISION HISTORY:

```
23 Oct 2012 - M. Payer    - Added ProTeX headers
```

2.24.10 SoilTemp

Function SoilTemp computes the temperature-dependent term of the soil NOx emissions in ng N/m2/s and converts to molec/cm2/s

INTERFACE:

```
FUNCTION SoilTemp( NN, TC, GWET ) RESULT( SOIL_TEMP )
```

INPUT PARAMETERS:

```
INTEGER,  INTENT(IN) :: NN           ! Soil biome type
REAL(hp), INTENT(IN) :: TC           ! Surface air temperature [C]
REAL(hp), INTENT(IN) :: GWET         ! Top soil moisture
```

RETURN VALUE:

```
REAL(hp)             :: SOIL_TEMP    ! Temperature-dependent term of
                                     ! soil NOx emissions [unitless]
```

REMARKS:

```
Based on Ormeci et al., [1999] and Otter et al., [1999]
there exists and entirely exponential relationship between
temperature and soil NOx emissions at constant soil moisture
Therefore we use the following relationship based
on Yienger and Levy et al., [1995] for temperatures 0-30C:


    f(T) =  exp( 0.103+/-0.04 * T )
      in ng N/m2/s


where T is the temperature in degrees Celsius....Below
0 C, we assume emissions are zero because they are insignificant
```

```
   for the purposes of this global source. ...

References:
=============================================================================
(1 ) Ormeci, B., S. L. Sanin, and J. J. Pierce, Laboratory study of
      NO flux from agricultural soil: Effects of soil moisture, pH,
      and temperature, J. Geophys. Res., 104 ,16211629, 1999.
(2 ) Otter, L. B., W. X. Yang, M. C. Scholes, and F. X. Meixner,
      Nitric oxide emissions from a southern African savanna, J.
      Geophys. Res., 105 , 20,69720,706, 1999.
(3 ) Yienger, J.J, and H. Levy, Empirical model of global soil-biogenic
      NOx emissions, J. Geophys. Res., 100, D6, 11,447-11464, June 20, 1995.
```

REVISION HISTORY:

```
17 Aug 2009 - R. Yantosca - Initial Version
17 Aug 2009 - R. Yantosca - Added ProTeX headers
31 Jan 2011 - R. Hudman   - Added new soil T dependance
31 Jan 2011 - R. Hudman   - Updated headers
12 May 2015 - R. Yantosca - Cosmetic changes
```

2.24.11 SoilWet

Function SoilWet returns the soil moisture scaling of soil NOx emissions (values from 0-1).

INTERFACE:

```
FUNCTION SoilWet( GWET, ARID, NONARID ) RESULT( WETSCALE )
```

INPUT PARAMETERS:

```
  ! Top soil wetness [unitless]
  REAL(hp), INTENT(IN) :: GWET

  ! Fraction of arid & non-arid soil in the gridbox
  REAL(hp), INTENT(IN) :: ARID
  REAL(hp), INTENT(IN) :: NONARID
!RETURN_VALUE:

  ! A scaling term between 0-1 based on soil moisture
  REAL(hp)             :: WETSCALE
```

REMARKS:

```
Soil moisture and temperature and now decoupled, the temperature
term is scaled with a value from 0-1 based on water filled pore space
WFPS in top-soil.

From N.E. Moore thesis:
```

```
The response of SNOx is not monotonic to WFPS. SNOx are low for the
extreme values of WFPS (0 and 1). For low values, emissions are
substrate-limited. For high values, emissions are trapped and cannot
diffuse to the surface [Yan et al., 2005]. SNOx dependence on soil
moisture is best described as a Poisson function [Parsons et al., 1996;
Otter et al., 1999; Pierce and Aneja, 2000; Kirkman et al., 2001;
van Dijk and Meixner, 2001; van Dijk et al., 2002]:

   scaling = a*x*exp(-b*x^2)

where the values of a and b are chosen such that the maximum value
(unity) occurs for WFPS=0.3, which laboratory and field measurements have
found to be the optimal value for emissions in most soils. The typical
range of values are 0.2 (arid) up to 0.45 (floodplain)
[Yang and Meixner, 1997; Ormeci et al., 1999].

Rice paddies no longer have to be scaled as in the Yienger & Levy model.

References:
===========================================================================
(1 ) Galbally, I. E., and R. Roy, Loss of fixed nitrogen from soils
      by nitric oxide exhalation, Nature, 275 , 734735, 1978.
(2 ) Kirkman, G. A., W. X. Yang, and F. X. Meixner, Biogenic nitric
      oxide emissions upscaling: An approach for Zimbabwe, Global
      Biogeochemical Cycles, 15 ,1005 1020, 2001.
(3 ) Ormeci, B., S. L. Sanin, and J. J. Pierce, Laboratory study of NO
      flux from agricultural soil: Effects of soil moisture, pH, and
      temperature, J. Geophys. Res., 104 , 16211629, 1999.
(4 ) Otter, L. B., W. X. Yang, M. C. Scholes, and F. X. Meixner,
      Nitric oxide emissions from a southern African savanna, J.
      Geophys. Res., 105 , 20,69720,706, 1999.
(5 ) Parsons, D. A., M. C. Scholes, R. J. Scholes, and J. S. Levine,
      Biogenic NO emissions from savanna soils as a function of fire
      regime, soil type, soil nitrogen, and water status, J. Geophys.
      Res., 101 , 23,68323,688, 1996.
(6 ) Pierce, J. J., and V. P. Aneja, Nitric oxide emissions from
      engineered soil systems, Journal of Environmental Engineering,
      pp. 225232, 2000.
(7 ) van Dijk, S. M., and J. H. Duyzer, Nitric oxide emissions from
      forest soils, J. Geophys. Res., 104 , 15,95515,961, 1999.
(8 ) van Dijk, S. M., and F. X. Meixner, Production and consumption of
      NO in forest and pasture soils from the Amazon basin, Water, Air,
      and Soil Pollution: Focus 1 , pp. 119130, 2001.
(9 ) Yang, W. X., and F. X. Meixner, Gaseous Nitrogen Emissions from
      Grasslands, CAB Int., Wallingford, UK, 1997, 67-71.
```

REVISION HISTORY:

```
17 Aug 2009 - R. Yantosca - Columnized and cleaned up
```

```
17 Aug 2009 - R. Yantosca - Added ProTeX headers
31 Jan 2011 - R. Hudman   - Rewrote scaling scheme
31 Jan 2011 - R.Hudman    - Updated ProTeX headers
```

2.24.12 SoilCrf

Computes the canopy reduction factor for the soil NOx emissions according to Jacob % Bakwin [1991] (and as used in Wang et al [1998]).

INTERFACE:

```
FUNCTION SoilCrf( K, LAI, CPYNOX, WINDSQR, SUNCOS ) RESULT( SOIL_CRF )
```

INPUT PARAMETERS:

```
 INTEGER, INTENT(IN) :: K          ! Soil biome type
 REAL(hp),  INTENT(IN) :: LAI        ! Leaf area index [cm2/cm2]
 REAL(hp),  INTENT(IN) :: CPYNOX     ! Bulk sfc resistance to NOx [1/s]
 REAL(hp),  INTENT(IN) :: WINDSQR    ! Square of sfc wind speed [m2/s2]
 REAL(hp),  INTENT(IN) :: SUNCOS     ! Cosine of solar zenith angle
!RETURN_VALUE:

 REAL(hp)              :: SOIL_CRF   ! Canopy reduction factor (see below)
```

REMARKS:

```
Also note, CANOPYNOX (the bulk surface resistance to NOx) is computed
in routine GET_CANOPY_NOx (in "canopy_nox_mod.f") and is passed here
as an argument.
```

REVISION HISTORY:

```
17 Aug 2009 - R. Yantosca - Initial Version
```

2.24.13 FertAdd

Function FertAdd computes fertilizer emissions

INTERFACE:

```
FUNCTION FertAdd( SOILFRT, DEPN ) RESULT( FERT_ADD )
```

INPUT PARAMETERS:

```
 REAL(hp), INTENT(IN) :: DEPN       ! N emissions from deposition
 REAL(hp), INTENT(IN) :: SOILFRT    ! N emissions from fertilizers
                                    !  read in from disk and passed
                                    !  here as an argument [ng N/m2/s]
!RETURN_VALUE:

 REAL(hp)             :: FERT_ADD   ! Total Fert emissions
```

REMARKS:

```
We use a new spatially explicit data set of chemical and manure fert
(native resolution 0.5\B0x0.5\B0) from Potter et al., [2010]
distributed using MODIS EVI seasonality as described in
N.E. Moore thesis, and Hudman et al., in prep.

In previous model, fertilizer emissions were emitted instantaneously as
2.5% of applied fertilizer, independent of soil moisture/soil temperature,
so that they were constant over the growing season.

Similar to the YL  parameterization, we now treat fertilizer emissions
as part of the Aw. If we treat the wet biome coefficient as a measure of
available N multiplied by a mean emission rate, we can treat fertilizer
N in the same manner.

AW = SOILAW(BinewsoilAWS_08112011_emissonlyome) + N available in soil
     x mean emission rate

Instead of choosing an emission rate for each box equivalent to 2.5%
of applied N yearly as done in the YL scheme, we chose the mean emission
rate so that the total global above canopy SNOx due to fertilizer matches
observed estimates of fertilizer emissions of 1.8 Tg N yr-1 from Stehfest
and Bouman [2006].  This treatment allows for interannual and daily
variability in the strength  of response to temperature and precipitation.
Note: this scaling must be set for each resolution.

References:
===========================================================================
(1 ) Potter, P., Ramankutty, N., Bennett, E.,  and Donner, S.:
      Characterizing the Spatial Patterns of Global Fertilizer
      Application and Manure Production, Earth Interactions,
      in press, 2010.
(2 ) Stehfest, E. and L. Bouwman, N2O and NO emission from
      agricultural fields and soils under natural vegetation:
      summarizing available measurement data and modeling
      of global annual emissions, Nutrient Cycling in Agroecosystems
      (2006), 74:207-228 DOI 10.1007/s10705-006-9000-7.
```

REVISION HISTORY:

```
17 Aug 2009 - R. Yantosca - Columnized and cleaned up
17 Aug 2009 - R. Yantosca - Added ProTeX headers
31 Jan 2011 - R. Hudman   - Rewrote pulsing scheme
31 Jan 2011 - R. Hudman   - Updated ProTex headers
```

2.24.14 Pulsing

Function Pulsing calculates the increase (or "pulse") of soil NOx emission that happens after preciptiation falls on dry soil. . According to Yan et al., [2005] , this pulsing process is thought to be due to a release of inorganic nitrogen trapped on top of the dry soil and a subsequent reactivation of water-stressed bacteria, which then metabolize the excess nitrogen. This can happen in seasonally dry grasslands and savannahs or over freshly fertilized fields.

INTERFACE:

```
FUNCTION Pulsing( GWET,      TS_EMIS,             &
                  GWET_PREV, PFACTOR, DRYPERIOD ) &
                  RESULT( THE_PULSING )
```

INPUT PARAMETERS:

```
REAL(hp), INTENT(IN)    :: GWET        ! Soil Moisture
REAL*4,   INTENT(IN)    :: TS_EMIS     ! Emissions timestep [s]
```

INPUT/OUTPUT PARAMETERS:

```
REAL(sp), INTENT(INOUT) :: GWET_PREV   ! Soil Moisture Prev timestep
REAL(sp), INTENT(INOUT) :: PFACTOR     ! Pulsing Factor
REAL(sp), INTENT(INOUT) :: DRYPERIOD   ! Dry period length in hours
```

RETURN VALUE:

```
REAL(hp)                :: THE_PULSING ! Factor to multiply baseline
                                       ! emissions by to account for
                                       ! soil pulsing of all types
```

REMARKS:

```
Soil NOx emissions consist of baseline emissions plus discrete "pulsing"
episodes.  We follow thw Yan et al., [2005] algorithm, where the pulse
(relative to the flux prewetting) is determined by the antecedent dry
period, with a simple logarithmic relationship,

   PFACTOR = 13.01 ln ( DRYPERIOD ) -  53.6

where PFACTOR is the magnitude of peak flux relative to prewetting flux,
and DRYPERIOD  is the length of the antecedent dry period in hours.

The pulse decays with

   PFACTOR = PFACTOR * EXP( -0.068e+0_hp * DTSRCE )

References:
=============================================================================
```

```
(1 ) Yan, X., T. Ohara, and H. Akimoto (2005), Statistical modeling of
     global soil NOx emissions, Global Biogeochem. Cycles, 19, GB3019,
     doi:10.1029/2004GB002276.Section 2.3.3
```

REVISION HISTORY:

```
17 Aug 2009 - R. Yantosca - Columnized and cleaned up
17 Aug 2009 - R. Yantosca - Added ProTeX headers
31 Jan 2011 - R. Hudman   - Rewrote pulsing scheme
31 Jan 2011 - R. Hudman   - Updated ProTex header
29 May 2013 - R. Yantosca - Bug fix: prevent log(0) from happening
21 Oct 2014 - C. Keller   - Limit PFACTOR to 1.
12 May 2015 - R. Yantosca - Cosmetic changes
```

2.24.15 InstGet

Subroutine InstGet returns a pointer to the desired instance.

INTERFACE:

```
SUBROUTINE InstGet ( Instance, Inst, RC, PrevInst )
```

INPUT PARAMETERS:

```
INTEGER                            :: Instance
TYPE(MyInst),     POINTER          :: Inst
INTEGER                            :: RC
TYPE(MyInst),     POINTER, OPTIONAL :: PrevInst
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.24.16 InstCreate

Subroutine InstCreate adds a new instance to the list of instances, assigns a unique instance number to this new instance, and archives this instance number to output argument Instance.

INTERFACE:

```
SUBROUTINE InstCreate ( ExtNr, Instance, Inst, RC )
```

INPUT PARAMETERS:

```
INTEGER,       INTENT(IN)       :: ExtNr
```

OUTPUT PARAMETERS:

```
INTEGER,       INTENT(  OUT)    :: Instance
TYPE(MyInst),  POINTER          :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,        INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.24.17 InstRemove

Subroutine InstRemove removes an instance from the list of instances.

INTERFACE:

```
SUBROUTINE InstRemove ( ExtState )
```

INPUT PARAMETERS:

```
TYPE(Ext_State),  POINTER       :: ExtState      ! Extension state
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.25 Fortran: Module Interface hcox_aerocom_mod.F90

Module HCOX_AeroCom_Mod.F90 is a HEMCO extension to use AeroCom volcano emissions from ascii tables. This module reads the daily AeroCom tables and emits the emissions according to the information in this file.

INTERFACE:

```
MODULE HCOX_AeroCom_Mod
```

USES:

```
USE HCO_Error_MOD
USE HCO_Diagn_MOD
USE HCOX_TOOLS_MOD
USE HCOX_State_MOD, ONLY : Ext_State
USE HCO_State_MOD,  ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HCOX_AeroCom_Run
PUBLIC :: HCOX_AeroCom_Init
PUBLIC :: HCOX_AeroCom_Final
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: ReadVolcTable
PRIVATE :: EmitVolc
```

REMARKS:

```
Each AeroCom table is expected to list the volcano location, sulfur
emissions (in kg S/s), and the volcano elevation as well as the
volcano plume column height. These entries need be separated by space
characters. For example:
                                                                      .
###  LAT (-90,90), LON (-180,180), SULFUR [kg S/s], ELEVATION [m], CLOUD_COLUMN_HEIGHT [m
### If elevation=cloud_column_height, emit in layer of elevation
### else, emit in top 1/3 of cloud_column_height
volcano::
50.170 6.850 3.587963e-03 600. 600.
::
                                                                      .
The sulfur read from table is emitted as the species defined in the
AeroCom settings section. More than one species can be provided. Mass
sulfur is automatically converted to mass of emitted species (using the
emitted molecular weight and molecular ratio of the corresponding HEMCO
species). Additional scale factors can be defined in the settings section
by using the (optional) setting 'Scaling_<SpecName>'.
For example, to emit SO2 and BrO from volcanoes, with an additional scale
factor of 1e-4 kg BrO / kgS for BrO, use the following setting:
115     AeroCom_Volcano   : on    SO2/BrO
   --> Scaling_BrO       :       1.0e-4
   --> Volcano_Source    :       OMI
   --> AeroCom_Table     :       $ROOT/VOLCANO/v2018-03/$YYYY/so2_volcanic_emissions_Carn
                                                                      .
This extension was originally added for usage within GEOS-5 and AeroCom
volcanic emissions, but has been modified to work with OMI-based volcanic
emissions from Ge et al. (2016).
                                                                      .
When using this extension, you should turn off any other volcano emission
inventories!
                                                                      .
 References:
 ============================================================================
 (1 ) Ge, C., J. Wang, S. Carn, K. Yang, P. Ginoux, and N. Krotkov,
      Satellite-based global volcanic SO2 emissions and sulfate direct
      radiative forcing during 2005-2012, J. Geophys. Res. Atmos., 121(7),
      3446-3464, doi:10.1002/2015JD023134, 2016.
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller   - Initial version
28 Mar 2018 - M. Sulprizio- Update to allow for OMI-based volcanic emissions;
                            Added Volcano_Source option to specify AeroCom
                            or OMI emissions because the latter are only
                            currently available for 2005-2012
```

2.25.1 HCOX_AeroCom_Run

Subroutine HCOX_AeroCom_Run is the driver routine for the customizable HEMCO extension.

INTERFACE:

```
SUBROUTINE HCOX_AeroCom_Run( am_I_Root, ExtState, HcoState, RC )
```

USES:

```
USE HCO_FluxArr_Mod,  ONLY : HCO_EmisAdd
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
TYPE(Ext_State), POINTER       :: ExtState    ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State), POINTER       :: HcoState    ! Hemco state
INTEGER,         INTENT(INOUT) :: RC          ! Success or failure
```

REMARKS:

REVISION HISTORY:

```
04 Jun 2015 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.25.2 HCOX_AeroCom_Init

Subroutine HCOX_AeroCom_Init initializes the HEMCO CUSTOM extension.

INTERFACE:

```
SUBROUTINE HCOX_AeroCom_Init( am_I_Root, HcoState, ExtName, &
                              ExtState,  RC                 )
```

USES:

```
USE HCO_ExtList_Mod,    ONLY : GetExtNr
USE HCO_ExtList_Mod,    ONLY : GetExtOpt
USE HCO_ExtList_Mod,    ONLY : GetExtSpcVal
USE HCO_STATE_MOD,      ONLY : HCO_GetExtHcoID
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root
CHARACTER(LEN=*), INTENT(IN   ) :: ExtName    ! Extension name
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER       :: HcoState   ! Hemco state
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller   - Initial version
```

2.25.3 HCOX_AeroCom_Final

Subroutine HCOX_AeroCom_Final finalizes the HEMCO AeroCom extension.

INTERFACE:

```
SUBROUTINE HCOX_AeroCom_Final ( ExtState )
```

INPUT PARAMETERS:

```
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller   - Initial version
```

2.25.4 ReadVolcTable

Subroutine ReadVolcTable reads the AeroCom volcano table of the current day.

INTERFACE:

```
SUBROUTINE ReadVolcTable ( am_I_Root, HcoState, ExtState, Inst, RC )
```

USES:

```
USE HCO_CharTools_Mod
USE inquireMod,         ONLY : findfreeLun
USE HCO_CLOCK_MOD,      ONLY : HcoClock_NewDay
USE HCO_CLOCK_MOD,      ONLY : HcoClock_Get
USE HCO_GeoTools_MOD,   ONLY : HCO_GetHorzIJIndex
USE HCO_EXTLIST_MOD,    ONLY : HCO_GetOpt
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER       :: HcoState   ! Hemco state
TYPE(MyInst),     POINTER       :: Inst
INTEGER,          INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller   - Initial version
28 Mar 2018 - M. Sulprizio- Add check for OMI-based emissions and use closest
                            available year if simulation year is outside
                            2005-2012
```

2.25.5 EmitVolc

Subroutine EmitVolc reads the AeroCom volcano table of the current day.

INTERFACE:

```
SUBROUTINE EmitVolc ( am_I_Root, HcoState, ExtState, Inst, SO2d, SO2e, RC )
```

USES:

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root
TYPE(Ext_State),  POINTER       :: ExtState   ! Module options
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_State),  POINTER       :: HcoState   ! Hemco state
TYPE(MyInst),     POINTER       :: Inst
INTEGER,          INTENT(INOUT) :: RC
```

OUTPUT PARAMETERS:

```
REAL(sp),         INTENT(  OUT) :: SO2e(HcoState%NX,HcoState%NY,HcoState%NZ)
REAL(sp),         INTENT(  OUT) :: SO2d(HcoState%NX,HcoState%NY,HcoState%NZ)
```

REVISION HISTORY:

```
04 Jun 2015 - C. Keller   - Initial version
```

2.25.6 InstGet

Subroutine InstGet returns a poiner to the desired instance.

INTERFACE:

```
SUBROUTINE InstGet ( Instance, Inst, RC, PrevInst )
```

INPUT PARAMETERS:

```
INTEGER                            :: Instance
TYPE(MyInst),     POINTER          :: Inst
INTEGER                            :: RC
TYPE(MyInst),     POINTER, OPTIONAL :: PrevInst
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
```

2.25.7 InstCreate

Subroutine InstCreate creates a new instance.

INTERFACE:

```
SUBROUTINE InstCreate ( ExtNr, Instance, Inst, RC )
```

INPUT PARAMETERS:

```
INTEGER,       INTENT(IN)       :: ExtNr
```

OUTPUT PARAMETERS:

```
INTEGER,       INTENT(  OUT)    :: Instance
TYPE(MyInst),  POINTER          :: Inst
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,       INTENT(INOUT)    :: RC
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

2.25.8 InstRemove

Subroutine InstRemove creates a new instance.

INTERFACE:

```
SUBROUTINE InstRemove ( Instance )
```

INPUT PARAMETERS:

```
INTEGER                         :: Instance
```

REVISION HISTORY:

```
18 Feb 2016 - C. Keller   - Initial version
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

3 HEMCO "Interfaces" modules

These modules contain routines that are used to connect HEMCO to other models, or to use HEMCO in standalone mode.

3.1 Fortran: Module Interface hcoi_esmf_mod

Module HCOI_ESMF_MOD is the HEMCO-ESMF interface module.

INTERFACE:

```
MODULE HCOI_ESMF_MOD
```

USES:

```
  USE HCO_ERROR_MOD
  USE HCO_Types_Mod

#if defined (ESMF_)
#include "MAPL_Generic.h"
  USE ESMF
  USE MAPL_Mod

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  ! ESMF environment only:
  PUBLIC :: HCO_SetServices
  PUBLIC :: HCO_SetExtState_ESMF
  PUBLIC :: HCO_Imp2Ext
```

PRIVATE MEMBER FUNCTIONS:

```
  PRIVATE :: Diagn2Exp
  PRIVATE :: HCO_Imp2Ext2R
  PRIVATE :: HCO_Imp2Ext2S
  PRIVATE :: HCO_Imp2Ext2I
  PRIVATE :: HCO_Imp2Ext3R
  PRIVATE :: HCO_Imp2Ext3S
```

REVISION HISTORY:

```
10 Oct 2014 - C. Keller   - Initial version
```

3.1.1 HCO_SetServices

Subroutine HCO_SetServices registers all required HEMCO data so that it can be imported through the ESMF import state. This routine determines all required HEMCO input fields from the HEMCO configuration file. Note that each file needs an equivalent ESMF-style entry in the registry file (typically ExtData.rc). Otherwise, ESMF won't read these files and HEMCO will fail when attempting to get pointers to these data arrays.

The field names provided in ExtData.rc must match the names in the HEMCO configuration file! Also, all time settings (average and update interval) and data units need to be properly specified in ExtData.rc. For now, ExtData.rc and HEMCO configuration file need to be synchronized manually. The pyHEMCO interface will automate this process!

This routine also prepares an emissions export field for every species found in the HEMCO configuration file. These export fields will only be filled if specified so in the MAPL History registry. The corresponding HEMCO diagnostics must be created separately via Diagn_Create (e.g. in hcoi_gc_diagn_mod.F90).

INTERFACE:

```
SUBROUTINE HCO_SetServices( am_I_Root,  GC, HcoConfig, &
                            ConfigFile, RC )
```

USES:

```
USE HCO_TYPES_MOD,    ONLY : ListCont
USE HCO_DATACONT_MOD, ONLY : ListCont_NextCont
USE HCO_CONFIG_MOD,   ONLY : Config_ReadFile
USE HCO_EXTLIST_MOD,  ONLY : GetExtOpt
USE HCO_CONFIG_MOD,   ONLY : Config_GetnSpecies
USE HCO_CONFIG_MOD,   ONLY : Config_GetSpecNames
USE HCO_DIAGN_MOD,    ONLY : DiagnFileOpen
USE HCO_DIAGN_MOD,    ONLY : DiagnFileGetNext
USE HCO_DIAGN_MOD,    ONLY : DiagnFileClose
```

ARGUMENTS:

```
LOGICAL,              INTENT(IN   )           :: am_I_Root
TYPE(ESMF_GridComp), INTENT(INOUT)            :: GC
TYPE(ConfigObj),      POINTER                 :: HcoConfig
CHARACTER(LEN=*),     INTENT(IN   )           :: ConfigFile
INTEGER,              INTENT(  OUT)           :: RC
```

REVISION HISTORY:

```
29 Aug 2013 - C. Keller - Initial version.
10 Sep 2015 - C. Keller - Added RESTART=MAPL_RestartSkip.
21 Feb 2016 - C. Keller - Update to v2.0, added default diagnostics (optional)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
16 Mar 2018 - E. Lundgren - Expand log write to specify reading HEMCO
                            diagnostic config file and adding HEMCO exports
```

3.1.2 Diagn2Exp

Subroutine Diagn2Exp is a helper routine to add a potential HEMCO diagnostics to the Export state.

INTERFACE:

```
SUBROUTINE Diagn2Exp( GC, SNAME, LNAME, UNITS, NDIM, RC )
```

USES:

ARGUMENTS:

```
TYPE(ESMF_GridComp), INTENT(INOUT)   :: GC
CHARACTER(LEN=*),    INTENT(IN   )   :: SNAME
CHARACTER(LEN=*),    INTENT(IN   )   :: LNAME
CHARACTER(LEN=*),    INTENT(IN   )   :: UNITS
INTEGER,             INTENT(IN   )   :: NDIM
INTEGER,             INTENT(  OUT)   :: RC
```

REVISION HISTORY:

```
05 Jan 2015 - C. Keller - Initial version
```

3.1.3 HCO_SetExtState_ESMF

Subroutine HCO_SetExtState_ESMF tries to populate some fields of the ExtState object from the ESMF import state.

INTERFACE:

```
SUBROUTINE HCO_SetExtState_ESMF( am_I_Root, HcoState, ExtState, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : Hco_State
USE HCOX_STATE_MOD,  ONLY : Ext_State
```

ARGUMENTS:

```
LOGICAL,              INTENT(IN   )   :: am_I_Root
TYPE(HCO_State),      POINTER         :: HcoState
TYPE(Ext_State),      POINTER         :: ExtState
INTEGER,              INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
06 Mar 2015 - C. Keller - Initial version
```

3.1.4 HCO_Imp2Ext2S

Subroutine HCO_Imp2Ext copies fields from the import state to the HEMCO ExtState object.

INTERFACE:

```
SUBROUTINE HCO_Imp2Ext2S( am_I_Root, HcoState, ExtDat, FldName, RC )
```

USES:

```
USE HCO_ARR_MOD,     ONLY : HCO_ArrAssert
USE HCO_STATE_MOD,   ONLY : Hco_State
USE HCOX_STATE_MOD,  ONLY : ExtDat_2S
```

ARGUMENTS:

```
LOGICAL,              INTENT(IN   )   :: am_I_Root
CHARACTER(LEN=*),     INTENT(IN   )   :: FldName
TYPE(HCO_State),      POINTER         :: HcoState
TYPE(ExtDat_2S),      POINTER         :: ExtDat
INTEGER,              INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
08 Feb 2016 - C. Keller - Initial version
11 Apr 2017 - C. Keller - It's now ok if field not found.
```

3.1.5 HCO_Imp2Ext3S

Subroutine HCO_Imp2Ext copies fields from the import state to the HEMCO ExtState object.

INTERFACE:

```
SUBROUTINE HCO_Imp2Ext3S( am_I_Root, HcoState, ExtDat, FldName, RC )
```

USES:

```
USE HCO_ARR_MOD,     ONLY : HCO_ArrAssert
USE HCO_STATE_MOD,   ONLY : Hco_State
USE HCOX_STATE_MOD,  ONLY : ExtDat_3S
```

ARGUMENTS:

```
LOGICAL,              INTENT(IN   )   :: am_I_Root
CHARACTER(LEN=*),     INTENT(IN   )   :: FldName
TYPE(HCO_State),      POINTER         :: HcoState
TYPE(ExtDat_3S),      POINTER         :: ExtDat
INTEGER,              INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
08 Feb 2016 - C. Keller - Initial version
11 Apr 2017 - C. Keller - It's now ok if field not found.
```

3.1.6 HCO_Imp2Ext2R

Subroutine HCO_Imp2Ext copies fields from the import state to the HEMCO ExtState object.

INTERFACE:

```
SUBROUTINE HCO_Imp2Ext2R( am_I_Root, HcoState, ExtDat, FldName, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : Hco_State
USE HCOX_STATE_MOD,  ONLY : ExtDat_2R
USE HCO_ARR_MOD,     ONLY : HCO_ArrAssert
```

ARGUMENTS:

```
LOGICAL,              INTENT(IN   )   :: am_I_Root
CHARACTER(LEN=*),     INTENT(IN   )   :: FldName
TYPE(HCO_State),      POINTER         :: HcoState
TYPE(ExtDat_2R),      POINTER         :: ExtDat
INTEGER,              INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
08 Feb 2016 - C. Keller - Initial version
11 Apr 2017 - C. Keller - It's now ok if field not found.
```

3.1.7 HCO_Imp2Ext3R

Subroutine HCO_Imp2Ext copies fields from the import state to the HEMCO ExtState object.

INTERFACE:

```
SUBROUTINE HCO_Imp2Ext3R( am_I_Root, HcoState, ExtDat, FldName, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : Hco_State
USE HCOX_STATE_MOD,  ONLY : ExtDat_3R
USE HCO_ARR_MOD,     ONLY : HCO_ArrAssert
```

ARGUMENTS:

```
LOGICAL,             INTENT(IN   )   :: am_I_Root
CHARACTER(LEN=*),    INTENT(IN   )   :: FldName
TYPE(HCO_State),     POINTER         :: HcoState
TYPE(ExtDat_3R),     POINTER         :: ExtDat
INTEGER,             INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
08 Feb 2016 - C. Keller - Initial version
11 Apr 2017 - C. Keller - It's now ok if field not found.
```

3.1.8 HCO_Imp2Ext2I

Subroutine HCO_Imp2Ext copies fields from the import state to the HEMCO ExtState object.

INTERFACE:

```
SUBROUTINE HCO_Imp2Ext2I( am_I_Root, HcoState, ExtDat, FldName, RC )
```

USES:

```
USE HCO_STATE_MOD,   ONLY : Hco_State
USE HCOX_STATE_MOD,  ONLY : ExtDat_2I
USE HCO_ARR_MOD,     ONLY : HCO_ArrAssert
```

ARGUMENTS:

```
LOGICAL,             INTENT(IN   )   :: am_I_Root
CHARACTER(LEN=*),    INTENT(IN   )   :: FldName
TYPE(HCO_State),     POINTER         :: HcoState
TYPE(ExtDat_2I),     POINTER         :: ExtDat
INTEGER,             INTENT(INOUT)   :: RC
```

REVISION HISTORY:

```
08 Feb 2016 - C. Keller - Initial version
11 Apr 2017 - C. Keller - It's now ok if field not found.
```

3.2 Fortran: Module Interface hemco_standalone.F90

Program HEMCO_StandAlone is the driver for HEMCO in in stand-alone mode. It receives the configuration file name as input argument and then calls the routine HCOI_StandAlone_Run (from module file hcoi_standalone_mod.F90) to repeatedly call HEMCO for one or more timesteps on the predefined grid.

INTERFACE:

```
PROGRAM HEMCO_StandAlone
```

USES:

```
USE HCOI_STANDALONE_MOD, ONLY : HCOI_Standalone_Run

IMPLICIT NONE
INTRINSIC :: TRIM
```

REVISION HISTORY:

```
16 Jul 2014 - R. Yantosca - Initial version
21 Jul 2013 - C. Keller   - Now pass configuration file name as argument
```

3.3 Fortran: Module Interface hcoi_standalone_mod.F90

Module HCOI_StandAlone_Mod contains all wrapper routines to run HEMCO in standalone mode, i.e. without any external model connected to it. All HEMCO input variables (grid, species, times) are read from disk. All meteorological variables needed by the (enabled) HEMCO extensions must be provided through the HEMCO configuration file (see ExtOpt_SetPointers).
Subroutine HCOI_StandAlone_Run will execute the standalone version of HEMCO. The following input files are needed for a standalone run:

- HEMCO_sa_Config: the HEMCO configuration file. Must be passed as argument to HCO_StandAlone_Run.
- HEMCO_sa_Spec: contains the HEMCO species definitions. The first row must contain the total number of species. For each species, the following parameter need to be specified (separated by at least one space character): species ID, name, molecular weight [g/mol], emitted molecular weight [g/mol], the molecule emission ratio, the liq. over gas Henry constant [M/atm], the temperature dependency of the Henry constant (K0, in [K]), and the pKa (for correction of the Henry constant).
- HEMCO_sa_Grid: contains the definition of the emission grid. Must contain the grid dimensions (NX, NY, NZ) in the first three rows (e.g. NX: 72), followed by the horizontal grid spaces (DX and DY) in rows four and five, respectively. DX and DY can be only one value (applied to all grid boxes), or a vector of length NX or NY, respectively. For now, no vertical regridding is supported, e.g. all emissions input file need to be either 2D fields or already remapped onto the correct model levels.

- HEMCO_sa_Time: contains the time definitions. The first two rows must contain the start and end date of the simulation, in format Start/End: YYYY-MM-DD HH:MM:SS (e.g. 2013-07-01 00:00:00). The third row must contain the emission time step (e.g. TS_EMIS: 3600.0).

The file names of the species, grid, and time input files can be provided in the settings section of the HEMCO configuration file. For instance, to set the species file to 'mySpecFile', add the following line to the con- figuration file: 'SpecFile: mySpecFile'. The same applies to grid and time definitions (GridFile and TimeFile, respectively). If no file names are provided in the configuration file, the default file names (HEMCO_sa_Spec, HEMCO_sa_Grid, HEMCO_sa_Time) will be used. **INTERFACE:**

```
MODULE HCOI_StandAlone_Mod
```

USES:

```
USE HCO_Error_Mod
USE HCO_Diagn_Mod
USE HCO_CharTools_Mod
USE HCO_Types_Mod
USE HCOX_State_Mod,      ONLY : Ext_State
USE HCO_State_Mod,       ONLY : HCO_State

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: HCOI_StandAlone_Run
PUBLIC  :: HCOI_SA_Init
PUBLIC  :: HCOI_SA_Run
PUBLIC  :: HCOI_SA_Final
PUBLIC  :: HCOI_SA_InitCleanup
PUBLIC  :: Get_nnMatch
PUBLIC  :: Register_Species
PUBLIC  :: Define_Diagnostics
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: Model_GetSpecies
PRIVATE :: Set_Grid
PRIVATE :: Read_Time
PRIVATE :: ExtState_SetFields
PRIVATE :: ExtState_UpdateFields
```

REVISION HISTORY:

```
 20 Aug 2013 - C. Keller   - Initial version.
 14 Jul 2014 - R. Yantosca - Now use F90 free-format indentation
 14 Jul 2014 - R. Yantosca - Cosmetic changes in ProTeX headers
 09 Apr 2015 - C. Keller   - Now accept comments and empty lines in
                             all input files.
 15 Feb 2015 - C. Keller   - Update to v2.0
```

3.3.1 HCOI_StandAlone_Run

Subroutine HCOI_StandAlone_Run runs the standalone version of HEMCO. All input variables are taken from input files.

INTERFACE:

```
SUBROUTINE HCOI_StandAlone_Run( ConfigFile )
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*), INTENT(IN)  :: ConfigFile
```

REVISION HISTORY:

```
12 Sep 2013 - C. Keller    - Initial version
```

3.3.2 HCOI_SA_Init

Subroutine HCOI_SA_Init initializes the HEMCO derived types and arrays.

INTERFACE:

```
SUBROUTINE HCOI_SA_Init( am_I_Root, ConfigFile, RC )
```

USES:

```
USE HCO_Config_Mod,    ONLY : Config_ReadFile
USE HCO_State_Mod,     ONLY : HcoState_Init
USE HCO_Driver_Mod,    ONLY : HCO_Init
USE HCOX_Driver_Mod,   ONLY : HCOX_Init
USE HCO_EXTLIST_Mod,   ONLY : GetExtOpt, CoreNr
```

INPUT PARAMETERS:

```
LOGICAL,            INTENT(IN   )           :: am_I_Root  ! root CPU?
CHARACTER(LEN=*),   INTENT(IN   )           :: ConfigFile ! Configuration file
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,            INTENT(INOUT)           :: RC         ! Failure or success
```

REVISION HISTORY:

```
12 Sep 2013 - C. Keller    - Initial version
```

3.3.3 HCOI_SA_Run

Subroutine HCOI_SA_RUN runs HCO from GEOS-Chem.

INTERFACE:

```
SUBROUTINE HCOI_SA_RUN( am_I_Root, RC )
```

USES:

```
USE HCO_FluxArr_Mod,       ONLY : HCO_FluxarrReset
USE HCO_Clock_Mod,         ONLY : HcoClock_Set
USE HCO_Clock_Mod,         ONLY : HcoClock_Get
USE HCO_Clock_Mod,         ONLY : HcoClock_Increase
USE HCO_Driver_Mod,        ONLY : HCO_RUN
USE HCOX_Driver_Mod,       ONLY : HCOX_RUN
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN   ) :: am_I_Root  ! root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER, INTENT(INOUT) :: RC         ! Failure or success
```

REVISION HISTORY:

```
12 Sep 2013 - C. Keller    - Initial version
```

3.3.4 HCOI_SA_Final

Subroutine HCOI_SA_Final cleans up HEMCO.

INTERFACE:

```
SUBROUTINE HCOI_SA_Final( am_I_Root )
```

USES:

```
USE HCO_Driver_Mod,    ONLY : HCO_Final
USE HCOX_Driver_Mod,   ONLY : HCOX_Final
USE HCO_State_Mod,     ONLY : HcoState_Final
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN) :: am_I_Root
```

REVISION HISTORY:

```
12 Sep 2013 - C. Keller    - Initial version
```

3.3.5 Model_GetSpecies

SUBROUTINE Model_GetSpecies returns 'model' species information from the HEMCO standalone input file.

INTERFACE:

```
SUBROUTINE Model_GetSpecies( am_I_Root,      HcoConfig,          &
                             nModelSpec,     ModelSpecNames,     &
                             ModelSpecIDs,   ModelSpecMW,        &
                             ModelSpecEmMW,  ModelSpecMolecRatio,&
                             ModelSpecK0,    ModelSpecCR,        &
                             ModelSpecPKA,   RC                   )
```

USES:

```
USE inquireMod,      ONLY : findfreeLUN
USE HCO_EXTLIST_Mod, ONLY : GetExtOpt, CoreNr
```

OUTPUT PARAMETERS:

```
LOGICAL,            INTENT(IN ) :: am_I_Root
TYPE(ConfigObj),    POINTER     :: HcoConfig
INTEGER,            INTENT(OUT) :: nModelSpec
CHARACTER(LEN= 31), POINTER     :: ModelSpecNames(:)
INTEGER,            POINTER     :: ModelSpecIDs  (:)
REAL(hp),           POINTER     :: ModelSpecMW   (:)
REAL(hp),           POINTER     :: ModelSpecEmMW (:)
REAL(hp),           POINTER     :: ModelSpecMolecRatio(:)
REAL(hp),           POINTER     :: ModelSpecK0   (:)
REAL(hp),           POINTER     :: ModelSpecCR   (:)
REAL(hp),           POINTER     :: ModelSpecPKA  (:)
INTEGER,            INTENT(OUT) :: RC
```

REVISION HISTORY:

```
13 Sep 2013 - C. Keller - Initial Version
```

3.3.6 Set_Grid

SUBROUTINE SET_GRID reads the grid information from the HEMCO standalone grid file and sets all HEMCO grid arrays accordingly. The grid file is expected to contain information on the grid edge lon/lat range, as well as the number of grid cells in longitude and latitude direction.

INTERFACE:

```
SUBROUTINE SET_Grid( am_I_Root, HcoState, RC )
```

USES:

```
 USE Grid_Mod,         ONLY : DoGridComputation
USE inquireMod,       ONLY  : findFreeLUN
USE HCO_ExtList_Mod, ONLY  : HCO_GetOpt, GetExtOpt, CoreNr
USE HCO_VertGrid_Mod, ONLY : HCO_VertGrid_Define
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   ) :: am_I_Root
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_STATE), POINTER       :: HcoState
INTEGER,         INTENT(INOUT) :: RC
```

REVISION HISTORY:

```
13 Sep 2013 - C. Keller - Initial Version
11 May 2015 - C. Keller - Now provide lon/lat edges instead of assuming
                          global grid.
10 Sep 2015 - C. Keller - Allow to provide mid-points instead of edges.
```

3.3.7 Get_nnMatch

Subroutine Get_nnMatch returns the number of species found in both the HEMCO configuration and the species input file.

INTERFACE:

```
SUBROUTINE Get_nnMatch( am_I_Root, HcoConfig, nnMatch, RC )
```

USES:

```
USE HCO_Config_Mod, ONLY : Config_GetnSpecies
USE HCO_Config_Mod, ONLY : Config_GetSpecNames
```

OUTPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )  :: am_I_Root ! Root CPU?
INTEGER,          INTENT(  OUT)  :: nnMatch   ! Number of HEMCO species that are
                                                ! also species in the atm. model
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(ConfigObj), POINTER        :: HcoConfig ! Config object
INTEGER,         INTENT(INOUT)  :: RC        ! Success or failure?
```

REVISION HISTORY:

```
13 Sep 2013 - C. Keller - Initial Version
```

3.3.8 Register_Species

Subroutine Register_Species registers all species in the HEMCO state object.

INTERFACE:

```
SUBROUTINE Register_Species( am_I_Root, HcoState, RC )
```

USES:

```
USE HCO_LogFile_Mod, ONLY : HCO_SPEC2LOG
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN   )   :: am_I_Root   ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HCO_STATE), POINTER :: HcoState
INTEGER, INTENT(INOUT)   :: RC          ! Success or failure
```

REVISION HISTORY:

```
13 Sep 2013 - C. Keller - Initial Version
```

3.3.9 Define_Diagnostics

Subroutine Define_Diagnostics defines all diagnostics to be used in this simulation.

INTERFACE:

```
SUBROUTINE Define_Diagnostics( am_I_Root, HcoState, RC, SetDefault )
```

USES:

```
USE HCO_EXTLIST_MOD,   ONLY : GetExtNr
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN   )             :: am_I_Root   ! Are we on the root CPU?
TYPE(HCO_STATE),  POINTER                   :: HcoState
LOGICAL,          INTENT(IN   ), OPTIONAL   :: SetDefault  ! Define default diagnostics
```

INPUT/OUTPUT PARAMETERS:

```
INTEGER,          INTENT(INOUT)             :: RC          ! Success or failure
```

REVISION HISTORY:

```
13 Sep 2013 - C. Keller - Initial Version
05 Feb 2015 - C. Keller - Added SetDefault flag
```

3.3.10 Read_Time

Subroutine READ_TIME reads the time information for the HEMCO standalone from an input file.

INTERFACE:

```
SUBROUTINE Read_Time( am_I_Root, HcoState, RC )
```

USES:

```
USE inquireMod,      ONLY : findfreeLUN
USE HCO_Extlist_Mod, ONLY : HCO_GetOpt, GetExtOpt, CoreNr
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
TYPE(HCO_State), POINTER       :: HcoState
!INPUT/OUTPUT PARAMETERS
INTEGER,         INTENT(INOUT) :: RC          ! Success or failure?
```

REVISION HISTORY:

```
13 Sep 2013 - C. Keller - Initial Version
```

3.3.11 ExtState_SetFields

Subroutine ExtState_SetFields fills the ExtState data fields with data read through the HEMCO configuration file.

INTERFACE:

```
SUBROUTINE ExtState_SetFields ( am_I_Root, HcoState, ExtState, RC )
```

USES:

```
USE HCO_ARR_MOD,        ONLY : HCO_ArrAssert
USE HCO_GEOTOOLS_MOD,   ONLY : HCO_GetSUNCOS
USE HCO_GEOTOOLS_MOD,   ONLY : HCO_CalcVertGrid
USE HCOX_STATE_MOD,     ONLY : ExtDat_Set
USE HCO_CLOCK_MOD,      ONLY : HcoClock_First
```

INPUT PARAMETERS:

```
LOGICAL,         INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
!INPUT/OUTPUT PARAMETERS
TYPE(HCO_STATE), POINTER       :: HcoState
TYPE(EXT_STATE), POINTER       :: ExtState
INTEGER,         INTENT(INOUT) :: RC          ! Success or failure?
```

REVISION HISTORY:

```
28 Jul 2014 - C. Keller   - Initial Version
06 Oct 2014 - M. Sulprizio- Remove PCENTER. Now calculate from pressure edges
09 Jul 2015 - E. Lundgren - Add MODIS Chlorophyll-a (CHLR)
26 Oct 2016 - R. Yantosca - Don't nullify local ptrs in declaration stmts
```

3.3.12 ExtState_UpdateFields

Subroutine ExtState_UpdateFields makes sure that all local variables that ExtState is pointing to are up to date. For the moment, this is just a placeholder routine as none of the ExtState fields is filled by local module fields. Content can be added to it if there are variables that need to be updated manually, e.g. not through netCDF input data.

INTERFACE:

```
SUBROUTINE ExtState_UpdateFields ( am_I_Root, HcoState, ExtState, RC )
```

USES:

INPUT PARAMETERS:

```
  LOGICAL,          INTENT(IN   ) :: am_I_Root   ! Are we on the root CPU?
!INPUT/OUTPUT PARAMETERS
  TYPE(HCO_STATE),  POINTER       :: HcoState
  TYPE(EXT_STATE),  POINTER       :: ExtState
  INTEGER,          INTENT(INOUT) :: RC          ! Success or failure?
```

REVISION HISTORY:

```
28 Jul 2014 - C. Keller - Initial Version
```

3.3.13 IsEndOfSimulation

Function IsEndOfSimulation returns true if the passed date is beyond the end of the simulation date.

INTERFACE:

```
FUNCTION IsEndOfSimulation( Yr, Mt, Dy, Hr, Mn, Sc ) RESULT ( IsEnd )
```

USES:

INPUT PARAMETERS:

```
  INTEGER,          INTENT(IN   ) :: YR
  INTEGER,          INTENT(IN   ) :: MT
  INTEGER,          INTENT(IN   ) :: DY
  INTEGER,          INTENT(IN   ) :: HR
  INTEGER,          INTENT(IN   ) :: MN
  INTEGER,          INTENT(IN   ) :: SC
!OUTPUT PARAMETERS
  LOGICAL                         :: IsEnd
```

REVISION HISTORY:

```
08 Sep 2014 - C. Keller - Initial Version
13 Jul 2015 - C. Keller - Bug fix: now save YYYYMMDD and hhmmss in different
                          variables to avoid integer truncation errors.
```

3.3.14 HCOI_Sa_InitCleanup

deallocates all local species arrays used during initialization.

INTERFACE:

```
SUBROUTINE HCOI_SA_InitCleanup ( am_I_Root, RC )
```

USES:

INPUT PARAMETERS:

```
  LOGICAL,          INTENT(IN   )          :: am_I_Root   ! Are we on the root CPU?
!INPUT/OUTPUT PARAMETERS
  INTEGER,          INTENT(INOUT)          :: RC          ! Success or failure?
```

REVISION HISTORY:

```
04 Feb 2016 - C. Keller - Initial Version
```