GEOS-Chem Reference
3. History modules (netCDF diagnostics)

GEOS-Chem Support Team

10 Jul 2018

Contents

1 History modules

These modules contain routines to archive various diagnostic quantities (concentrations, emissions, chemical production and loss, etc.) from GEOS-Chem into netCDF diagnostic output

1.1 Fortran: Module Interface histitem_mod.F90

Contains types and methods to create a HISTORY ITEM object. A HISTORY ITEM represents a single GEOS-Chem diagnostic quantity that will be archived to netCDF file output.

INTERFACE:

```
MODULE HistItem_Mod
```

USES:

```
  USE Precision_Mod

  IMPLICIT NONE
  PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
  PUBLIC :: HistItem_Create
  PUBLIC :: HistItem_Print
  PUBLIC :: HistItem_Destroy
```

PUBLIC TYPES:

```
  !=========================================================================
  ! This is the derived type for a SINGLE HISTORY ITEM OBJECT, which will
  ! hold a quantity from GEOS-Chem that we want to save to netCDF output.
  !=========================================================================
  TYPE, PUBLIC :: HistItem

     !----------------------------------------------------------------------
     ! Identifying information
     !----------------------------------------------------------------------
     CHARACTER(LEN=255) :: Name                  ! Item name
     INTEGER            :: Id                    ! Item Id
     INTEGER            :: ContainerId           ! Container Id

     !----------------------------------------------------------------------
     ! netCDF variable attributes (for COARDS-compliance)
     !----------------------------------------------------------------------
     INTEGER            :: NcXDimId              ! Id of netCDF X (lon  ) dim
     INTEGER            :: NcYdimId              ! Id of netCDF Y (lat  ) dim
     INTEGER            :: NcZDimId              ! Id of netCDF Z (lev C) dim
     INTEGER            :: NcIDimId              ! ID of netCDF I (lev E) dim
     INTEGER            :: NcTdimId              ! Id of netCDF T (time ) dim
```

```
  INTEGER            :: NcVarId               ! netCDF variable ID
  CHARACTER(LEN=255) :: LongName              ! Item description
  CHARACTER(LEN=255) :: Units                 ! Units of data
  REAL(f4)           :: AddOffset             ! Offset and scale factor
  REAL(f4)           :: ScaleFactor           !  for packed data
  REAL(f4)           :: MissingValue          ! Missing value
  CHARACTER(LEN=255) :: AvgMethod             ! Averaging method

  !-----------------------------------------------------------------
  ! Pointers to the data in State_Chm, State_Diag, or State_Met
  !-----------------------------------------------------------------
  INTEGER            :: Source_KindVal        ! Identifies the source type

  REAL(f8), POINTER  :: Source_0d_8           ! Ptr to 0D 8-byte    data

  REAL(fp), POINTER  :: Source_1d  (:    )    ! Ptr to 1D flex-prec data
  REAL(f8), POINTER  :: Source_1d_8(:    )    ! Ptr to 1D 8-byte    data
  REAL(f4), POINTER  :: Source_1d_4(:    )    ! Ptr to 1D 4-byte    data
  INTEGER,  POINTER  :: Source_1d_I(:    )    ! Ptr to 1D integer   data

  REAL(fp), POINTER  :: Source_2d  (:,:  )    ! Ptr to 2D flex-prec data
  REAL(f8), POINTER  :: Source_2d_8(:,:  )    ! Ptr to 2D 8-byte    data
  REAL(f4), POINTER  :: Source_2d_4(:,:  )    ! Ptr to 2D 4-byte    data
  INTEGER,  POINTER  :: Source_2d_I(:,:  )    ! Ptr to 2D integer   data

  REAL(fp), POINTER  :: Source_3d  (:,:,:)    ! Ptr to 3D flex-prec data
  REAL(f8), POINTER  :: Source_3d_8(:,:,:)    ! Ptr to 3D 8-byte    data
  REAL(f4), POINTER  :: Source_3d_4(:,:,:)    ! Ptr to 3D 4-byte    data
  INTEGER,  POINTER  :: Source_3d_I(:,:,:)    ! Ptr to 3D integer   data

  !-----------------------------------------------------------------
  ! Data arrays
  !-----------------------------------------------------------------
  INTEGER            :: SpaceDim              ! # of dims (0-3)
  REAL(f8), POINTER  :: Data_0d               ! 0D scalar
  REAL(f8), POINTER  :: Data_1d(:    )        ! 1D vector
  REAL(f8), POINTER  :: Data_2d(:,:  )        ! 2D array
  REAL(f8), POINTER  :: Data_3d(:,:,:)        ! 3D array
  CHARACTER(LEN=3)   :: DimNames              ! Used to specify if data is
                                              !  "xyz", "yz", "x", "y" etc.
  INTEGER, POINTER   :: NcChunkSizes(:)       ! Chunk sizes for netCDF
  LOGICAL            :: OnLevelEdges          ! =T if data is defined on
                                              !    vertical level edges;
                                              ! =F if on level centers

  !-----------------------------------------------------------------
  ! Data archival
  !-----------------------------------------------------------------
```

```
    REAL(f8)           :: nUpdates             ! # of times updated
    INTEGER            :: Operation            ! Operation code
                                               !  0=copy from source
                                               !  1=accumulate from source
  END TYPE HistItem
```

REMARKS:

```
Linked list routines taken from original code (linkedlist.f90)
by Arjen Markus; http://flibs.sourceforge.net/linked_list.html
```

REVISION HISTORY:

```
13 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
06 Jul 2017 - R. Yantosca - Add source pointers to 4-byte and integer data
04 Aug 2017 - R. Yantosca - Declare Data_* accumulator arrays as REAL(fp),
                            which should avoid roundoff for long runs
11 Aug 2017 - R. Yantosca - Remove 0d pointers and data arrays
25 Aug 2017 - R. Yantosca - Added Source_0d_8 and Source_1d_8 pointers
25 Aug 2017 - R. Yantosca - Added Source_2d_8 and Source_3d_8 pointers
```

1.1.1 HistItem_Create

Initializes a single history item that will be archived via History (and eventually sent to netCDF output).

INTERFACE:

```
SUBROUTINE HistItem_Create( am_I_Root,    Item,           Id,             &
                            ContainerId,  Name,           RC,             &
                            LongName,     Units,          SpaceDim,       &
                            OnLevelEdges, AddOffset,      MissingValue,   &
                            ScaleFactor,  Source_KindVal, Operation,      &
                            DimNames,     Dimensions,     Source_0d_8,    &
                            Source_1d,    Source_1d_8,    Source_1d_4,    &
                            Source_1d_I,  Source_2d,      Source_2d_8,    &
                            Source_2d_4,  Source_2d_I,    Source_3d,      &
                            Source_3d_8,  Source_3d_4,    Source_3d_I    )
```

USES:

```
USE CharPak_Mod,         ONLY : TranLc
USE ErrCode_Mod
USE History_Util_Mod
USE Registry_Params_Mod
```

INPUT PARAMETERS:

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

```
INTEGER,           INTENT(IN)  :: Id                ! History item Id #
INTEGER,           INTENT(IN)  :: ContainerId       ! Container Id #
CHARACTER(LEN=*),  INTENT(IN)  :: Name              ! Item's short name
CHARACTER(LEN=*),  INTENT(IN)  :: LongName          ! Item's long name
CHARACTER(LEN=*),  INTENT(IN)  :: Units             ! Units of the data
INTEGER,           INTENT(IN)  :: SpaceDim          ! Dimension of data

! Optional arguments
LOGICAL,           OPTIONAL    :: OnLevelEdges      ! =T if data defined
                                                    !  on level edges;
                                                    ! =F if on centers
REAL(f4),          OPTIONAL    :: AddOffset         ! COARDS-compliant
REAL(f4),          OPTIONAL    :: MissingValue      !  attributes for
REAL(f4),          OPTIONAL    :: ScaleFactor       !  netCDF output
INTEGER,           OPTIONAL    :: Operation         ! Operation code
                                                    !  0=copy  from source
                                                    !  1=accum from source
CHARACTER(LEN=*),  OPTIONAL    :: DimNames          ! Use this to specify
                                                    !  dimensions of data
                                                    !  ("yz", "z", etc.)

! Optional pointers to data targets
INTEGER,           OPTIONAL    :: Source_KindVal    ! Type of source data
REAL(fp), POINTER, OPTIONAL    :: Source_0d_8       ! 0D 8-byte    data
REAL(fp), POINTER, OPTIONAL    :: Source_1d  (:    ) ! 1D flex-prec data
REAL(fp), POINTER, OPTIONAL    :: Source_1d_8(:    ) ! 1D 8-byte    data
REAL(f4), POINTER, OPTIONAL    :: Source_1d_4(:    ) ! 1D 4-byte    data
INTEGER,  POINTER, OPTIONAL    :: Source_1d_I(:    ) ! 1D integer   data
REAL(fp), POINTER, OPTIONAL    :: Source_2d  (:,:  ) ! 2D flex-prec data
REAL(f8), POINTER, OPTIONAL    :: Source_2d_8(:,:  ) ! 2D 8-byte    data
REAL(f4), POINTER, OPTIONAL    :: Source_2d_4(:,:  ) ! 2D 4-byte    data
INTEGER,  POINTER, OPTIONAL    :: Source_2d_I(:,:  ) ! 2D integer   data
REAL(fp), POINTER, OPTIONAL    :: Source_3d  (:,:,:) ! 3D flex-prec data
REAL(f8), POINTER, OPTIONAL    :: Source_3d_8(:,:,:) ! 3D 8-byte    data
REAL(f4), POINTER, OPTIONAL    :: Source_3d_4(:,:,:) ! 3D 4-byte    data
INTEGER,  POINTER, OPTIONAL    :: Source_3d_I(:,:,:) ! 3D integer   data
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HistItem),    POINTER     :: Item              ! HISTORY ITEM object
```

OUTPUT PARAMETERS:

```
INTEGER,           OPTIONAL    :: Dimensions(3)     ! Spatial dims of data
INTEGER,           INTENT(OUT) :: RC                ! Success or failure
```

REMARKS:

(1) We need to copy string data to a temporary string of length 255 characters, or else Gfortran will choke.

REVISION HISTORY:

```
13 Jun 2017 - R. Yantosca - Initial version
03 Aug 2017 - R. Yantosca - Add OPERATION as an optional argument
08 Aug 2017 - R. Yantosca - Now assign NcVarId a default value
11 Aug 2017 - R. Yantosca - Remove 0d pointers and data arrays
11 Aug 2017 - R. Yantosca - Added optional DimNames argument
24 Aug 2017 - R. Yantosca - Now size the data accumulator array from
                            the size of the data pointer
24 Aug 2017 - R. Yantosca - Set the NcILevDim field to undefined
25 Aug 2017 - R. Yantosca - Added Source_0d_8 and Source_1d_8 arguments
28 Aug 2017 - R. Yantosca - Now define the NcChunkSizes field
25 Aug 2017 - R. Yantosca - Added Source_2d_8 and Source_3d_8 arguments
```

1.1.2 HistItem_Print

Prints information contained within a single history item.

INTERFACE:

```
SUBROUTINE HistItem_Print( am_I_Root, Item, RC, ShortFormat )
```

USES:

```
USE ErrCode_Mod
USE History_Util_Mod
```

INPUT PARAMETERS:

```
LOGICAL,        INTENT(IN)  :: am_I_Root    ! Are we on the root CPU?
TYPE(HistItem), POINTER     :: Item         ! History Item
LOGICAL,        OPTIONAL    :: ShortFormat  ! Print truncated format
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

```
13 Jun 2017 - R. Yantosca - Initial version
06 Jul 2017 - R. Yantosca - Add option to print truncated output format
11 Aug 2017 - R. Yantosca - Remove 0d pointers and data arrays
24 Aug 2017 - R. Yantosca - Now print OnLevelEdges for the full format
24 Aug 2017 - R. Yantosca - Now print NcIDimId for the full format
25 Aug 2017 - R. Yantosca - Now print the vertical cell position: C or E
28 Aug 2017 - R. Yantosca - Now prints the netCDF chunksizes (full format)
```

1.1.3 HistItem_Destroy(Item)

Deallocates all pointer-based array fields of the history item, then destroys the history item itself.

INTERFACE:

```
SUBROUTINE HistItem_Destroy( am_I_Root, Item, RC )
```

USES:

```
USE ErrCode_Mod
USE History_Util_Mod
```

INPUT PARAMETERS:

```
LOGICAL,        INTENT(IN)  :: am_I_Root   ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HistItem), POINTER     :: Item        ! History item
```

OUTPUT PARAMETERS:

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

REMARKS:

REVISION HISTORY:

```
13 Jun 2017 - R. Yantosca - Initial version
06 Jul 2017 - R. Yantosca - Nullify source pointers to 4-byte & integer data
11 Aug 2017 - R. Yantosca - Remove 0d pointers and data arrays
28 Aug 2017 - R. Yantosca - Deallocate Data_0d and NcChunkSizes fields
06 Oct 2017 - R. Yantosca - Nullify Source_2d_8 and Source_3d_8 pointers
```

1.2 Fortran: Module Interface metahistitem_mod.F90

Contains types and methods to create a METAHISTORY ITEM object, which is a container for a HISTORY ITEM. In other words, a METAHISTORY ITEM represents a single node of a linked list that is used to contain HISTORY ITEMS.

In practice, we can think of a METAHISTORY ITEM as a list of HISTORY ITEMS that will be archived to netCDF output at a specified frequency (e.g. instantaneous, daily, hourly, etc.)

INTERFACE:

```
MODULE MetaHistItem_Mod
```

USES:

```
USE HistItem_Mod, ONLY : HistItem
USE Precision_Mod

IMPLICIT NONE
PRIVATE
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: MetaHistItem_Create
PRIVATE :: MetaHistItem_Insert
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: MetaHistItem_AddNew
PUBLIC  :: MetaHistItem_Count
PUBLIC  :: MetaHistItem_Destroy
PUBLIC  :: MetaHistItem_Print
```

PUBLIC TYPES:

```
!=============================================================================
! This is the derived type for a METAHISTORY ITEM object, which represents
! a SINGLE NODE OF A LINKED LIST consisting of HISTORY ITEMS.
! As such, the METAHISTORY ITEM does not contain any data itself,
! but is a wrapper for a single HISTORY ITEM object, plus a pointer
! to another METAHISTORY ITEM (i.e. the next node in the list).
!=============================================================================
TYPE, PUBLIC ::  MetaHistItem

   ! Pointer to the next METAHISTORY ITEM object
   ! (i.e. the next node in the linked list)
   TYPE(MetaHistItem), POINTER :: Next => NULL()

   ! The HISTORY ITEM object (which represents a diagnostic
   ! quantity that will be archived to netCDF file format)
   TYPE(HistItem),     POINTER :: Item => NULL()

END TYPE MetaHistItem
```

REMARKS:

As described above, a METAHISTORY ITEM can be thought of as a SINGLE NODE OF A LINKED LIST INTENDED TO HOLD HISTORY ITEMS. It looks like this:

```
   +-------------------------+   +-------------------------+
   | METAHISTORY ITEM n      |   | METAHISTORY ITEM n+1    |
   | (aka NODE n of list)    |   | (aka NODE n+1 of list)  |
   |                         |   |                         |
   | Contains:               |   | Contains:               |
   |                         |   |                         |
   |   HISTORY ITEM n        |   |   HISTORY ITEM n+1      |
```

```
     |                          |   |                          |
=======> Pointer to next    =========> Pointer to next    ========> etc ...
     |    METAHISTORY ITEM      |   |    METAHISTORY ITEM      |
     +--------------------------+   +--------------------------+
 Linked list routines taken from original code (linkedlist.f90)
 by Arjen Markus; http://flibs.sourceforge.net/linked_list.html
```

REVISION HISTORY:

```
14 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.2.1 MetaHistItem_AddNew

Wrapper for methods MetaHistItem_Create and MetaHistItem_Insert. Will create a METAHISTORY ITEM (containing a HISTORY ITEM) and (1) set it as the head node of a new linked list, or (2) append it to an existing linked list.

INTERFACE:

```
SUBROUTINE MetaHistItem_AddNew( am_I_Root, Node, Item, RC )
```

USES:

```
USE ErrCode_Mod
USE HistItem_Mod, ONLY : HistItem
```

INPUT PARAMETERS:

```
LOGICAL,            INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
TYPE(HistItem),     POINTER     :: Item       ! HISTORY ITEM object
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistItem), POINTER     :: Node       ! METAHISTORY ITEM object
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

```
13 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.2.2 MetaHistItem_Create

This method creates a new METAHISTORY ITEM (to contain the supplied HISTORY ITEM) and sets it as the head node of a linked list.

INTERFACE:

```
SUBROUTINE MetaHistItem_Create( am_I_Root, Node, Item, RC )
```

USES:

```
USE ErrCode_Mod
USE HistItem_Mod, ONLY : HistItem
```

INPUT PARAMETERS:

```
LOGICAL,            INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
TYPE(HistItem),     POINTER     :: Item       ! HISTORY ITEM object
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistItem), POINTER     :: Node       ! METAHISTORY ITEM object
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This method is not intended to be called directly, but is rather
wrapped by the MetaHistItem_AddNew method.
```

REVISION HISTORY:

```
13 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.2.3 MetaHistItem_Insert

Creates a new METAHISTORY ITEM (to contain the supplied HISTORY ITEM), and pops it into an existing linked list, immediately following the head node.

INTERFACE:

```
SUBROUTINE MetaHistItem_Insert( am_I_Root, Node, Item, RC )
```

USES:

```
USE ErrCode_Mod
USE HistItem_Mod, ONLY : HistItem
```

INPUT PARAMETERS:

```
LOGICAL,            INTENT(IN)  :: am_I_Root ! Are we on the root CPU?
TYPE(HistItem),     POINTER     :: Item      ! HISTORY ITEM object
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistItem), POINTER     :: Node      ! METAHISTORY ITEM object
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This method is not intended to be called directly, but is rather
wrapped by the MetaHistItem_AddNew method.
```

REVISION HISTORY:

```
13 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
06 Oct 2017 - R. Yantosca - Now insert new node at the head of the list
```

1.2.4 MetaHistItem_Count

Counts the number of METAHISTORY ITEMS stored in a linked list. By extension, this is also the number of HISTORY ITEMS stored in the list, because each METAHISTORY ITEM contains only one HISTORY ITEM.

INTERFACE:

```
FUNCTION MetaHistItem_Count( List ) RESULT( nNodes )
```

INPUT PARAMETERS:

```
TYPE(MetaHistItem), POINTER :: List    ! Linked list of METAHISTORY ITEMS
```

RETURN VALUE:

```
INTEGER                     :: nNodes  ! Number of METAHISTORY ITEMS
```

REVISION HISTORY:

```
14 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.2.5 MetaHistItem_Print

This method will print information about the HISTORY ITEM belonging to each METAHISTORY ITEM (aka node) of a linked list.

INTERFACE:

```
SUBROUTINE MetaHistItem_Print( am_I_Root, List, RC )
```

USES:

```
USE ErrCode_Mod
USE HistItem_Mod, ONLY : HistItem, HistItem_Print
```

INPUT PARAMETERS:

```
LOGICAL,            INTENT(IN)  :: am_I_Root   ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistItem), POINTER     :: List        ! List of history items
```

OUTPUT PARAMETERS:

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

REMARKS:

REVISION HISTORY:

14 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus

---

1.2.6 MetaHistItem_Destroy

This method will destroy the HISTORY ITEM belonging to each METAHISTORY ITEM (aka node) of a linked list. It will then destroy each METAHISTORY ITEM itself.

INTERFACE:

```
SUBROUTINE MetaHistItem_Destroy( am_I_Root, List, RC )
```

USES:

```
USE ErrCode_Mod
USE HistItem_Mod, ONLY : HistItem, HistItem_Destroy
```

INPUT PARAMETERS:

```
LOGICAL,            INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistItem), POINTER     :: List       ! List of METAHISTORY ITEMS
```

OUTPUT PARAMETERS:

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

REMARKS:

REVISION HISTORY:

14 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus

---

1.3 Fortran: Module Interface histcontainer_mod.F90

Contains methods to create a HISTORY CONTAINER object. A HISTORY CONTAINER represents a collection of HISTORY ITEMS that will be archived to a netCDF file at a specific temporal frequencly (e.g. instantaneous, hourly, daily, monthly, end-of-run, etc.)

In other words, the HISTORY CONTAINER provides metadata for the netCDF file, and the HISTORY ITEMS belonging to the HISTORY CONTAINER contains the data and attributes for each variable that will be saved to the netCDF file. **INTERFACE:**

```
MODULE HistContainer_Mod
```

USES:

```
USE MetaHistItem_Mod,  ONLY: MetaHistItem
USE Precision_Mod

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC :: HistContainer_Create
PUBLIC :: HistContainer_Print
PUBLIC :: HistContainer_Destroy
PUBLIC :: HistContainer_SetTime
PUBLIC :: HistContainer_UpdateIvalSet
PUBLIC :: HistContainer_FileCloseIvalSet
PUBLIC :: HistContainer_FileWriteIvalSet
```

PUBLIC TYPES:

```
!=============================================================================
! This is the derived type for a single HISTORY CONTAINER OBJECT, which
! contains several HISTORY ITEMS that will be archived at a specified
! frequency (e.g. instantaneous, hourly, daily, etc) to netCDF output.
!=============================================================================
TYPE, PUBLIC :: HistContainer

   !---------------------------------------------------------------------
   ! Identifying information
   !---------------------------------------------------------------------
   CHARACTER(LEN=255)          :: Name                ! Container name
   INTEGER                     :: Id                  !  and ID number
   INTEGER                     :: nX                  ! X (or lon) dim size
   INTEGER                     :: nY                  ! Y (or lat) dim size
   INTEGER                     :: nZ                  ! Z (or lev) dim size
   LOGICAL                     :: OnLevelEdges        ! =T if data is defined
                                                      !    on level edges;
                                                      ! =F if on centers
```

```
!-------------------------------------------------------------------------
! List of history items in this collection
!-------------------------------------------------------------------------
TYPE(MetaHistItem), POINTER :: HistItems => NULL() ! List and # of
INTEGER                     :: nHistItems          !  HISTORY ITEMS
                                                   !  in this container

!-------------------------------------------------------------------------
! Time quantities measured since start of simulation
!-------------------------------------------------------------------------
REAL(f8)                    :: EpochJd             ! Astronomical Julian
                                                   !  date @ start of sim
                                                   !  1=accum from source
INTEGER                     :: CurrentYmd          ! Current YMD date
INTEGER                     :: CurrentHms          ! Current hms time
REAL(f8)                    :: CurrentJd           ! Astronomical Julian
                                                   !  date @ current time
REAL(f8)                    :: ElapsedSec          ! Elapsed seconds
                                                   !  since start of sim
REAL(f8)                    :: UpdateAlarm         ! Alarm (elapsed sec)
                                                   !  for data updating
REAL(f8)                    :: FileCloseAlarm      ! Alarm (elapsed sec)
                                                   !  for file close/open
REAL(f8)                    :: FileWriteAlarm      ! Alarm (elapsed sec)
                                                   !  for file write

!-------------------------------------------------------------------------
! Time quantities measured since the time of netCDF file creation
!-------------------------------------------------------------------------
INTEGER                     :: ReferenceYmd        ! Reference YMD & hms
INTEGER                     :: ReferenceHms        !  for the "time" dim
REAL(f8)                    :: ReferenceJD         ! Julian Date at the
                                                   !  reference YMD & hms
INTEGER                     :: CurrTimeSlice       ! Current time slice
                                                   !  for the "time" dim
REAL(f8)                    :: TimeStamp           ! Elapsed minutes w/r/t
                                                   !  reference YMD & hms

!-------------------------------------------------------------------------
! Quantities that govern the updating/time averaging of data
!-------------------------------------------------------------------------
CHARACTER(LEN=255)          :: UpdateMode          ! e.g. inst or time-avg
INTEGER                     :: UpdateYmd           ! Update frequency
INTEGER                     :: UpdateHms           !  in YMD and hms
REAL(f8)                    :: UpdateIvalSec       ! Update interval [sec]
INTEGER                     :: Operation           ! Operation code
                                                   !  0=copy from source
REAL(f8)                    :: HeartBeatDtSec      ! The "heartbeat"
```

```
                                                !  timestep [sec]
    REAL(f8)                    :: HeartBeatDtDays     ! The "heartbeat"
                                                !  timestep [days]

    !-------------------------------------------------------------------
    ! Quantities for file creation, writing, and I/O status
    !-------------------------------------------------------------------
    INTEGER                     :: FileWriteYmd        ! File write frequency
    INTEGER                     :: FileWriteHms        !  in YMD and hms
    REAL(f8)                    :: FileWriteIvalSec    ! File write interval
                                                !  in seconds

    INTEGER                     :: FileCloseYmd        ! File closing time
    INTEGER                     :: FileCloseHms        !  in YMD and hms
    REAL(f8)                    :: FileCloseIvalSec    ! File close interval
                                                !  in seconds

    LOGICAL                     :: IsFileDefined       ! Have we done netCDF
                                                !  define mode yet?
    LOGICAL                     :: IsFileOpen          ! Is the netCDF file
                                                !  currently open?

    !-------------------------------------------------------------------
    ! netCDF file identifiers and attributes
    !-------------------------------------------------------------------
    LOGICAL                     :: FirstInst           ! 1st inst file write?
    INTEGER                     :: FileId              ! netCDF file ID
    INTEGER                     :: xDimId              ! X (or lon ) dim ID
    INTEGER                     :: yDimId              ! Y (or lat ) dim ID
    INTEGER                     :: zDimId              ! Z (or lev ) dim ID
    INTEGER                     :: iDimId              ! I (or ilev) dim ID
    INTEGER                     :: tDimId              ! T (or time) dim ID
    CHARACTER(LEN=20)           :: StartTimeStamp      ! Timestamps at start
    CHARACTER(LEN=20)           :: EndTimeStamp        !  and end of sim
    CHARACTER(LEN=20)           :: Spc_Units           ! Units of SC%Species
    CHARACTER(LEN=255)          :: FileExpId           ! Filename ExpId
    CHARACTER(LEN=255)          :: FilePrefix          ! Filename prefix
    CHARACTER(LEN=255)          :: FileTemplate        ! YMDhms template
    CHARACTER(LEN=255)          :: FileName            ! Name of nc file
    CHARACTER(LEN=255)          :: Conventions         ! e.g. "COARDS"
    CHARACTER(LEN=255)          :: NcFormat            ! e.g. "netCDF-4"
    CHARACTER(LEN=255)          :: History             ! History
    CHARACTER(LEN=255)          :: ProdDateTime        ! When produced
    CHARACTER(LEN=255)          :: Reference           ! Reference string
    CHARACTER(LEN=255)          :: Contact             ! Contact string
    CHARACTER(LEN=255)          :: Title               ! Title string

END TYPE HistContainer
```

REMARKS:

```
Linked list routines taken from original code (linkedlist.f90)
by Arjen Markus; http://flibs.sourceforge.net/linked_list.html
```

REVISION HISTORY:

```
12 Jun 2017 - R. Yantosca - Initial version, based on history_list_mod.F90
07 Aug 2017 - R. Yantosca - Add FileWriteYmd, FileWriteHms
08 Aug 2017 - R. Yantosca - Add IsFileDefined, IsFileOpen, nX, nY, nZ and
                            the ouptuts xDimId, yIDimd, zDimId, tDimId
16 Aug 2017 - R. Yantosca - Rename Archival* variables to Update*
17 Aug 2017 - R. Yantosca - Added the *Alarm variables
18 Aug 2017 - R. Yantosca - Added EpochJd so that we can compute Julian
                            dates as relative to the start of the run
18 Aug 2017 - R. Yantosca - Added ElapsedMin
18 Aug 2017 - R. Yantosca - Add HistContainer_ElapsedTime routine
21 Aug 2017 - R. Yantosca - Removed *_AlarmCheck, *_AlarmSet routines
24 Aug 2017 - R. Yantosca - Added iDimId as the dimension ID for ilev,
                             which is the vertical dimension on interfaces
28 Aug 2017 - R. Yantosca - Added SpcUnits, FirstInst to type HistContainer
06 Sep 2017 - R. Yantosca - Split HistContainer_AlarmIntervalSet into 3
                             separate routines, now made public
18 Sep 2017 - R. Yantosca - Elapsed time and alarms are now in seconds
```

1.3.1 HistContainer_Create

Initializes a single HISTORY CONTAINER object, which will hold a METAHISTORY ITEM (which is a list of HISTORY ITEMS), to archive to netCDF output.

INTERFACE:

```
SUBROUTINE HistContainer_Create( am_I_Root,      Container,              &
                                 Id,             Name,                   &
                                 RC,             EpochJd,                &
                                 CurrentYmd,     CurrentHms,             &
                                 UpdateMode,     UpdateYmd,              &
                                 UpdateHms,      UpdateAlarm,            &
                                 Operation,      HeartBeatDtSec,         &
                                 FileWriteYmd,   FileWriteHms,           &
                                 FileWriteAlarm, FileCloseYmd,           &
                                 FileCloseHms,   FileCloseAlarm,         &
                                 FileId,         FileExpId,              &
                                 FilePrefix,     FileName,               &
                                 FileTemplate,   Conventions,            &
                                 NcFormat,       History,                &
                                 ProdDateTime,   Reference,              &
                                 Title,          Contact,                &
                                 StartTimeStamp, EndTimeStamp            )
```

USES:

```
USE ErrCode_Mod
USE History_Util_Mod
USE MetaHistItem_Mod, ONLY : MetaHistItem
```

INPUT PARAMETERS:

```
!-------------------------------------------------------------------------
! REQUIRED INPUTS
!-------------------------------------------------------------------------
LOGICAL,             INTENT(IN)  :: am_I_Root      ! Root CPU?
INTEGER,             INTENT(IN)  :: Id             ! Container Id #
CHARACTER(LEN=*),    INTENT(IN)  :: Name           ! Container name

!-------------------------------------------------------------------------
! OPTIONAL INPUTS: Time and date quantities
!-------------------------------------------------------------------------
REAL(f8),            OPTIONAL    :: EpochJd        ! Astronomical Julian
                                                   !  date @ start of sim
INTEGER,             OPTIONAL    :: CurrentYmd     ! Current YMD date
INTEGER,             OPTIONAL    :: CurrentHms     ! Current hms time

!-------------------------------------------------------------------------
! OPTIONAL INPUTS: quantities controlling data updates
!-------------------------------------------------------------------------
CHARACTER(LEN=*),    OPTIONAL    :: UpdateMode     ! e.g. inst or time-avg
INTEGER,             OPTIONAL    :: UpdateYmd      ! Update frequency
INTEGER,             OPTIONAL    :: UpdateHms      !  in both YMD and hms
REAL(f8),            OPTIONAL    :: UpdateAlarm    ! JD for data update
INTEGER,             OPTIONAL    :: Operation      ! Operation code:
                                                   !  0=copy  from source
                                                   !  1=accum from source
REAL(f8),            OPTIONAL    :: HeartBeatDtSec ! Model "heartbeat"
                                                   !  timestep [sec]

!-------------------------------------------------------------------------
! OPTIONAL INPUTS: quantities controlling file write and close/reopen
!-------------------------------------------------------------------------
INTEGER,             OPTIONAL    :: FileWriteYmd   ! File write frequency
INTEGER,             OPTIONAL    :: FileWriteHms   !  in both YMD and hms
REAL(f8),            OPTIONAL    :: FileWriteAlarm ! JD for file write

INTEGER,             OPTIONAL    :: FileCloseYmd   ! File close/open freq
INTEGER,             OPTIONAL    :: FileCloseHms   !  in both YMD and hm
REAL(f8),            OPTIONAL    :: FileCloseAlarm ! JD for file close

!-------------------------------------------------------------------------
! OPTIONAL INPUTS: netCDF file identifiers and metadata
```

```
!-----------------------------------------------------------------------
INTEGER,             OPTIONAL    :: FileId         ! netCDF file ID
CHARACTER(LEN=*),    OPTIONAL    :: FileExpId      ! Dir name + file string
CHARACTER(LEN=*),    OPTIONAL    :: FilePrefix     ! Filename prefix
CHARACTER(LEN=*),    OPTIONAL    :: FileTemplate   ! YMDhms template
CHARACTER(LEN=*),    OPTIONAL    :: Conventions    ! e.g. "COARDS"
CHARACTER(LEN=*),    OPTIONAL    :: Filename       ! Name of nc file
CHARACTER(LEN=*),    OPTIONAL    :: NcFormat       ! e.g. "netCDF-4"
CHARACTER(LEN=*),    OPTIONAL    :: History        ! History
CHARACTER(LEN=*),    OPTIONAL    :: ProdDateTime   ! When produced
CHARACTER(LEN=*),    OPTIONAL    :: Reference      ! Reference string
CHARACTER(LEN=*),    OPTIONAL    :: Title          ! Title string
CHARACTER(LEN=*),    OPTIONAL    :: Contact        ! Contact string
CHARACTER(LEN=*),    OPTIONAL    :: StartTimeStamp ! Timestamps at start
CHARACTER(LEN=*),    OPTIONAL    :: EndTimeStamp   !  & end of simulation
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HistContainer), POINTER     :: Container      ! Collection object
```

OUTPUT PARAMETERS:

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

REMARKS:

```
(1) We need to copy string data to a temporary string of length 255
    characters, or else Gfortran will choke.
```

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version, based on history_list_mod.F90
07 Aug 2017 - R. Yantosca - Add FileWriteYmd and FileWriteHms
09 Aug 2017 - R. Yantosca - Add nX, ny, and, nZ
11 Aug 2017 - R. Yantosca - Add FileCloseYmd, FileCloseHms, ReferenceYmd,
                            ReferenceHms, and CurrTimeSlice
14 Aug 2017 - R. Yantosca - Add FileCloseYmd and FileCloseHms arguments
16 Aug 2017 - R. Yantosca - Renamed Archival* variables to Update*
17 Aug 2017 - R. Yantosca - Add *Alarm and Reference* arguments
18 Aug 2017 - R. Yantosca - Now initialize CurrentJd with EpochJd
21 Aug 2017 - R. Yantosca - Reorganize arguments, now define several time
                            fields from EpochJd, CurrentYmd, CurrentHms
21 Aug 2017 - R. Yantosca - Now define initial alarm intervals and alarms
28 Aug 2017 - R. Yantosca - Now initialize Container%Spc_Units to null str
29 Aug 2017 - R. Yantosca - Reset NcFormat if netCDF compression is off
                            for GEOS-Chem "Classic" simulations.
29 Aug 2017 - R. Yantosca - Now define the heartbeat timestep fields
30 Aug 2017 - R. Yantosca - Subtract the heartbeat timestep from the
                             UpdateAlarm value so as to update collections
                             at the same times w/r/t the "legacy" diags
18 Sep 2017 - R. Yantosca - Now accept heartbeat dt in seconds
02 Jan 2018 - R. Yantosca - Fix construction of default file template
```

1.3.2 HistContainer_Print

Prints information stored in a single HISTORY CONTAINER object.

INTERFACE:

```
SUBROUTINE HistContainer_Print( am_I_Root, Container, RC )
```

USES:

```
USE ErrCode_Mod
```

INPUT PARAMETERS:

```
LOGICAL,             INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
TYPE(HistContainer), POINTER     :: Container  ! HISTORY CONTAINER object
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version
16 Aug 2017 - R. Yantosca - Renamed Archival* variables to Update*
17 Aug 2017 - R. Yantosca - Now print *Alarm variables.  Also use the
                            Item%DimNames field to print the data dims
18 Sep 2017 - R. Yantosca - Updated for elapsed time in seconds
```

1.3.3 HistContainer_Destroy

This method will destroy the METAHISTORY ITEM belonging to a HISTORY CONTAINER. It will then destroy the HISTORY CONTAINER itself.

INTERFACE:

```
SUBROUTINE HistContainer_Destroy( am_I_Root, Container, RC )
```

USES:

```
USE ErrCode_Mod
USE MetaHistItem_Mod, ONLY : MetaHistItem_Destroy
```

INPUT PARAMETERS:

```
LOGICAL,             INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HistContainer), POINTER     :: Container  ! HISTORY CONTAINER object
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.3.4 HistContainer_UpdateIvalSet

Defines the alarm interval for the UPDATE operation.

INTERFACE:

```
SUBROUTINE HistContainer_UpdateIvalSet( am_I_root, Container, RC )
```

USES:

```
USE ErrCode_Mod
USE History_Util_Mod
USE Time_Mod,         ONLY : Its_A_Leapyear, Ymd_Extract
```

INPUT PARAMETERS:

```
LOGICAL,             INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HistContainer), POINTER     :: Container  ! HISTORY CONTAINER object
```

OUTPUT PARAMETERS:

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

REMARKS:

```
Assume that we will always update data more frequently than 1 month.
This means that we only have to compute this interval at initialization.
```

REVISION HISTORY:

```
06 Sep 2017 - R. Yantosca - Initial version
18 Sep 2017 - R. Yantosca - Now return update interval in seconds
26 Oct 2017 - R. Yantosca - Now allow update interval to be greater
                            than one month (similar to write, close)
08 Mar 2018 - R. Yantosca - Fixed logic bug that was causing incorrect
                            computation of UpdateIvalSec for simulations
                            longer than a day.
```

1.3.5 HistContainer_FileCloseIvalSet

Defines the alarm interval for the FILE CLOSE/REOPEN operation.

INTERFACE:

```
SUBROUTINE HistContainer_FileCloseIvalSet( am_I_Root, Container, RC )
```

USES:

```
USE ErrCode_Mod
USE History_Util_Mod
USE Time_Mod,         ONLY : Its_A_Leapyear, Ymd_Extract
```

INPUT PARAMETERS:

```
LOGICAL,             INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HistContainer), POINTER     :: Container  ! HISTORY CONTAINER object
```

OUTPUT PARAMETERS:

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

REMARKS:

The algorithm may not be as robust when straddling leap-year months, so we would recommend selecting an interval of 1 month or 1 year at a time.

REVISION HISTORY:

06 Sep 2017 - R. Yantosca - Initial version
18 Sep 2017 - R. Yantosca - Now return file close interval in seconds
26 Oct 2017 - R. Yantosca - Add modifications for a little more efficiency

---

1.3.6 HistContainer_FileWriteIvalSet

Defines the alarm intervals for the FILE WRITE operation.

INTERFACE:

```
SUBROUTINE HistContainer_FileWriteIvalSet( am_I_Root, Container, RC )
```

USES:

```
USE ErrCode_Mod
USE History_Util_Mod
USE Time_Mod,         ONLY : Its_A_Leapyear, Ymd_Extract
```

INPUT PARAMETERS:

```
LOGICAL,             INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HistContainer), POINTER     :: Container  ! HISTORY CONTAINER object
```

OUTPUT PARAMETERS:

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

REMARKS:

The algorithm may not be as robust when straddling leap-year months, so we would recommend selecting an interval of 1 month or 1 year at a time.

REVISION HISTORY:

06 Jan 2015 - R. Yantosca - Initial version
18 Sep 2017 - R. Yantosca - Now return file write interval in seconds
26 Oct 2017 - R. Yantosca - Add modifications for a little more efficiency

---

1.3.7 HistContainer_SetTime

Increments the current astronomical Julian Date of a HISTORY CONTAINER object by the HeartBeat interval (in fractional days). Then it recomputes the corresponding date/time and elapsed minutes.

INTERFACE:

```
SUBROUTINE HistContainer_SetTime( am_I_Root, Container, HeartBeatDt, RC )
```

USES:

```
USE ErrCode_Mod
USE History_Util_Mod
USE Julday_Mod,      ONLY :CALDATE
```

INPUT PARAMETERS:

```
LOGICAL,             INTENT(IN)  :: am_I_Root   ! Are we on the root CPU?
TYPE(HistContainer), POINTER     :: Container   ! HISTORY CONTAINER object
REAL(f8),            OPTIONAL    :: HeartBeatDt ! Heartbeat increment for
                                                !  for timestepping [days]
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This routine is called after the initial creation of the HISTORY
CONTAINER object.  It is also called from History_SetTime, which is
placed after the call to History_Update but before History_Write.
```

REVISION HISTORY:

```
21 Aug 2017 - R. Yantosca - Initial version
29 Aug 2017 - R. Yantosca - Now make HeartBeatDt an optional field; if not
                            specified, use Container%HeartBeatDtDays
```

1.4 Fortran: Module Interface metahistcontainer_mod.F90

Contains types and methods to create a METAHISTORY CONTAINER object, which is a container for a HISTORY CONTAINER. In other words, a METAHISTORY CONTAINER represents a single node of a linked list that is used to contain HISTORY CONTAINERS.

In practice, we can think of a METAHISTORY CONTAINER as a list of diagnostic collections, each of which contains a list of HISTORY ITEMS to be archived to netCDF output at a specified frequency (e.g. instantaneous, daily, hourly, etc.) !

INTERFACE:

```
MODULE MetaHistContainer_Mod
```

USES:

```
USE HistContainer_Mod, ONLY : HistContainer
USE Precision_Mod

IMPLICIT NONE
PRIVATE
```

PRIVATE MEMBER FUNCTIONS:

```
PRIVATE :: MetaHistContainer_Create
PRIVATE :: MetaHistContainer_Insert
```

PUBLIC MEMBER FUNCTIONS:

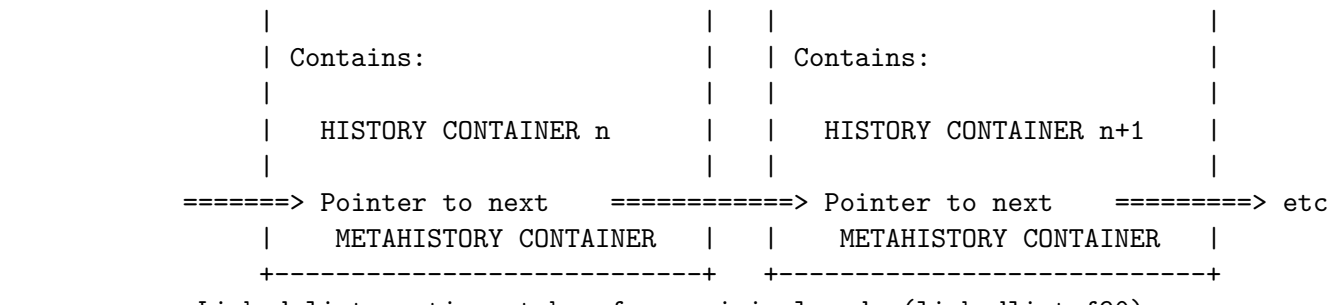
```
PUBLIC  :: MetaHistContainer_AddNew
PUBLIC  :: MetaHistContainer_Count
PUBLIC  :: MetaHistContainer_Destroy
PUBLIC  :: MetaHistContainer_Print
```

PUBLIC TYPES:

```
!=============================================================================
! This is the derived type for a METAHISTORY CONTAINER object, which
! represents a SINGLE NODE OF A LINKED LIST consisting of HISTORY
! CONTAINERS.
! As such, the METAHISTORY CONTAINER does not contain any data itself,
! but is a wrapper for a single HISTORY CONTAINER object, plus a pointer
! to another METAHISTORY CONTAINER (i.e. the next node in the list).
!=============================================================================
TYPE, PUBLIC ::  MetaHistContainer

   ! Pointer to the next METAHISTORY CONTAINER object
   ! (i.e. the next node in the linked list)
   TYPE(MetaHistContainer), POINTER :: Next      => NULL()

   ! The HISTORY CONTAINER object (which represents a diagnostic
   ! quantity that will be archived to netCDF file format)
   TYPE(HistContainer),     POINTER :: Container => NULL()

END TYPE MetaHistContainer
```

REMARKS:

```
 As described above, a METAHISTORY CONTAINER can be thought of as a SINGLE
 NODE OF A LINKED LIST INTENDED TO HOLD HISTORY CONTAINERS.  It looks like
 this:

     +---------------------------+   +---------------------------+
     | METAHISTORY CONTAINER n    |   | METAHISTORY CONTAINER n+1  |
     | (aka NODE n of list)       |   | (aka NODE n+1 of list)     |
```



Linked list routines taken from original code (linkedlist.f90)
by Arjen Markus; http://flibs.sourceforge.net/linked_list.html

REVISION HISTORY:

16 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus

---

1.4.1 MetaHistContainer_AddNew

Wrapper for methods MetaHistContainer_Create and MetaHistContainer_Insert. Will create a METAHISTORY CONTAINER (containing a HISTORY CONTAINER) and (1) set it as the head node of a new linked list, or (2) append it to an existing linked list.

INTERFACE:

```
SUBROUTINE MetaHistContainer_AddNew( am_I_Root, Node, Container, RC )
```

USES:

```
USE ErrCode_Mod
USE HistContainer_Mod, ONLY : HistContainer
```

INPUT PARAMETERS:

```
LOGICAL,                INTENT(IN)  :: am_I_Root ! Are we on the root CPU?
TYPE(HistContainer),    POINTER     :: Container ! HISTORY CONTAINER
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistContainer), POINTER    :: Node      ! METAHISTORY CONTAINER
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

16 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus

---

1.4.2 MetaHistContainer_Create

This method creates a new METAHISTORY CONTAINER (to contain the supplied HISTORY CONTAINER) and sets it as the head node of a linked list.

INTERFACE:

```
SUBROUTINE MetaHistContainer_Create( am_I_Root, Node, Container, RC )
```

USES:

```
USE ErrCode_Mod
USE HistContainer_Mod, ONLY : HistContainer
```

INPUT PARAMETERS:

```
LOGICAL,                INTENT(IN)  :: am_I_Root ! Are we on the root CPU?
TYPE(HistContainer),    POINTER     :: Container ! HISTORY CONTAINER
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistContainer), POINTER    :: Node      ! METAHISTORY CONTAINER
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This method is not intended to be called directly, but is rather
wrapped by the MetaHistContainer_AddNew method.
```

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.4.3 MetaHistContainer_Insert

Creates a new METAHISTORY CONTAINER (to contain the supplied HISTORY CONTAINER), and pops it into an existing linked list, immediately following the head node.

INTERFACE:

```
SUBROUTINE MetaHistContainer_Insert( am_I_Root, Node, Container, RC )
```

USES:

```
USE ErrCode_Mod
USE HistContainer_Mod, ONLY : HistContainer
```

INPUT PARAMETERS:

```
LOGICAL,                INTENT(IN)  :: am_I_Root ! Are we on the root CPU?
TYPE(HistContainer),    POINTER     :: Container ! HISTORY CONTAINER
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistContainer), POINTER     :: Node      ! METAHISTORY CONTAINER
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This method is not intended to be called directly, but is rather
wrapped by the MetaHistContainer_AddNew method.
```

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
06 Oct 2017 - R. Yantosca - Now insert new node at the head of the list
```

1.4.4 MetaHistContainer_Count

Counts the number of METAHISTORY CONTAINERS stored in a linked list. By extension, this is also the number of HISTORY CONTAINERS stored in the list, because each METAHISTORY CONTAINER contains only one HISTORY ! CONTAINER.

INTERFACE:

```
FUNCTION MetaHistContainer_Count( List ) RESULT( nNodes )
```

INPUT PARAMETERS:

```
TYPE(MetaHistContainer), POINTER :: List   ! List of METAHISTORY CONTAINERS
```

RETURN VALUE:

```
INTEGER                          :: nNodes ! # of METAHISTORY CONTAINERS
                                           ! (aka nodes) in the list
```

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.4.5 MetaHistContainer_Print

This method will print information about the HISTORY CONTAINER belonging to each METAHISTORY CONTAINER (aka node) of a linked list.

INTERFACE:

```
SUBROUTINE MetaHistContainer_Print( am_I_Root, List, RC )
```

USES:

```
USE ErrCode_Mod
USE HistContainer_Mod, ONLY : HistContainer, HistContainer_Print
```

INPUT PARAMETERS:

```
LOGICAL,                INTENT(IN)  :: am_I_Root ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistContainer), POINTER    :: List      ! List of METAHISTORY
                                                 !  CONTAINERS
```

OUTPUT PARAMETERS:

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

REMARKS:

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.4.6 MetaHistContainer_Destroy

This method will destroy the HISTORY CONTAINER belonging to each METAHISTORY CONTAINER (aka node) of a linked list. It will then destroy each METAHISTORY CONTAINER in the list.

INTERFACE:

```
SUBROUTINE MetaHistContainer_Destroy( am_I_Root, List, RC )
```

USES:

```
USE ErrCode_Mod
USE HistContainer_Mod, ONLY : HistContainer, HistContainer_Destroy
```

INPUT PARAMETERS:

```
LOGICAL,                INTENT(IN)  :: am_I_Root ! Are we on the root CPU?
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(MetaHistContainer), POINTER    :: List      ! List of METAHISTORY
                                                 !  CONTAINERS
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version, based on code by Arjen Markus
```

1.5 Fortran: Module Interface history_mod.F90

Driver module for GEOS-Chem's netCDF diagnostics package, aka the "History Component".

INTERFACE:

```
MODULE History_Mod
```

USES:

```
USE Precision_Mod
USE HistContainer_Mod,     ONLY : HistContainer
USE MetaHistContainer_Mod, ONLY : MetaHistContainer

IMPLICIT NONE
PRIVATE
```

PUBLIC MEMBER FUNCTIONS:

```
PUBLIC  :: History_Init
PUBLIC  :: History_SetTime
PUBLIC  :: History_Update
PUBLIC  :: History_Write
PUBLIC  :: History_Cleanup
PRIVATE MEMBER FUNCTIONS:
PRIVATE :: History_ReadCollectionNames
PRIVATE :: History_ReadCollectionData
PRIVATE :: History_AddItemToCollection
PRIVATE :: History_Close_AllFiles
```

REMARKS:

REVISION HISTORY:

```
06 Jan 2015 - R. Yantosca - Initial version
02 Aug 2017 - R. Yantosca - Added History_Update routine
14 Aug 2017 - R. Yantosca - Now read the "acc_interval" field for
                            time-averaged data collections
16 Aug 2017 - R. Yantosca - Add subroutine TestTimeForAction to avoid
                            duplicating similar code
16 Aug 2017 - R. Yantosca - Now close all netCDF files in routine
                            History_Close_AllFiles
18 Aug 2017 - R. Yantosca - Added routine History_SetTime
02 Oct 2017 - R. Yantosca - Added CollectionFileName
01 Nov 2017 - R. Yantosca - Moved ReadOneLine, CleanText to charpak_mod.F90
```

1.5.1 History_Init

Reads the HISTORY.rc file and creates the linked list of collections (i.e. netCDF diagnostic files containing several data fields with a specified update frequency). The list of fields belonging to each collection is also determined.

Each collection is described by a HISTORY CONTAINER object, which also contains a linked list of diagnostic quantities (i.e. a METAHISTORY ITEM) that will be archived to netCDF format. The list of diagnostic quantities is determined here by parsing the HISTORY.rc file.

NOTE: The HISTORY.rc file is read twice. The first (done by method History_ReadCollectionNames) reads the list of all collections. Then, for each defined collection, the list of diagnostic quantities belonging to that collection is determined by routine History_ReadCollectionData.

INTERFACE:

```
SUBROUTINE History_Init( am_I_root, Input_Opt,  State_Met,                &
                         State_Chm, State_Diag, RC                        )
```

USES:

```
USE ErrCode_Mod
USE History_Netcdf_Mod, ONLY : History_Netcdf_Init
USE History_Util_Mod
USE Input_Opt_Mod,      ONLY : OptInput
USE State_Chm_Mod ,     ONLY : ChmState
USE State_Diag_Mod,     ONLY : DgnState
USE State_Met_Mod,      ONLY : MetState
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)  :: am_I_Root
TYPE(OptInput),   INTENT(IN)  :: Input_Opt
TYPE(ChmState),   INTENT(IN)  :: State_Chm
TYPE(DgnState),   INTENT(IN)  :: State_Diag
TYPE(MetState),   INTENT(IN)  :: State_Met
```

OUTPUT PARAMETERS:

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

REMARKS:

```
Calls internal routines History_ReadCollectionNames and
History_Read_CollectionData
```

REVISION HISTORY:

```
06 Jan 2015 - R. Yantosca - Initial version
06 Nov 2017 - R. Yantosca - Reorder arguments for consistency (Input_Opt,
                            then State_Met, State_Chm, State_Diag).
```

1.5.2 History_Read_Collection_Names

Reads the History input file (e.g. HISTORY.rc) and determines the names of each individual diagnostic collection. It stores this information in module variables for use in the next step.

INTERFACE:

```
SUBROUTINE History_ReadCollectionNames( am_I_Root,  Input_Opt, State_Chm,  &
                                        State_Diag, State_Met, RC         )
```

USES:

```
USE Charpak_Mod
USe DiagList_Mod,      ONLY : CollList, ColItem
USE ErrCode_Mod
USE History_Util_Mod
USE Input_Opt_Mod,     ONLY : OptInput
USE InquireMod,        ONLY : FindFreeLun
USE State_Chm_Mod ,    ONLY : ChmState
USE State_Diag_Mod,    ONLY : DgnState
USE State_Met_Mod,     ONLY : MetState
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)  :: am_I_Root    ! Are we on the root CPU?
TYPE(OptInput),   INTENT(IN)  :: Input_Opt    ! Input Options object
TYPE(ChmState),   INTENT(IN)  :: State_Chm    ! Chemistry State object
TYPE(DgnState),   INTENT(IN)  :: State_Diag   ! Diagnostic State object
TYPE(MetState),   INTENT(IN)  :: State_Met    ! Meteorology State object
```

OUTPUT PARAMETERS:

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

REMARKS:

```
Private routine, called from routine History_Init.
```

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version
15 Aug 2017 - R. Yantosca - Now initialize string arrays to UNDEFINED_STR
02 Oct 2017 - R. Yantosca - Now initialize CollectionFileName
28 Feb 2018 - R. Yantosca - Now use the CollList object from diaglist_mod
                            to get the collection names
```

1.5.3 History_Read_Collection_Data

Parses the History input file (e.g. HISTORY.rc) and compiles the list of diagnostic quantities belonging to each collection. In other words, this is the list of individual fields that will be archived to a particular netCDF file with a given update frequency.

INTERFACE:

```
SUBROUTINE History_ReadCollectionData( am_I_Root,  Input_Opt, State_Chm,   &
                                       State_Diag, State_Met, RC          )
```

USES:

```
USE Charpak_Mod
USE DiagList_Mod,          ONLY : CollList, Search_CollList
USE CMN_Size_Mod,          ONLY : IIPAR, JJPAR, LLPAR
USE ErrCode_Mod
USE HistContainer_Mod
USE HistItem_Mod
USE History_Util_Mod
USE Input_Opt_Mod,         ONLY : OptInput
USE InquireMod,            ONLY : FindFreeLun
USE MetaHistContainer_Mod
USE MetaHistItem_Mod
USE Species_Mod,           ONLY : Species
USE State_Chm_Mod
USE State_Diag_Mod
USE State_Met_Mod
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)  :: am_I_Root    ! Are we on the root CPU?
TYPE(OptInput),   INTENT(IN)  :: Input_Opt    ! Input Options object
TYPE(ChmState),   INTENT(IN)  :: State_Chm    ! Chemistry State object
TYPE(DgnState),   INTENT(IN)  :: State_Diag   ! Diagnostic State object
TYPE(MetState),   INTENT(IN)  :: State_Met    ! Meteorology State object
```

OUTPUT PARAMETERS:

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

REMARKS:

```
Private routine, called from History_Init.
```

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version
03 Aug 2017 - R. Yantosca - Pass OPERATION to History_AddItemToCollection
14 Aug 2017 - R. Yantosca - FileWrite{Ymd,Hms} and FileClose{Ymd,Hms} are
                            now computed properly, w/r/t acc_interval
30 Aug 2017 - R. Yantosca - Now write collection info only on the root CPU
18 Sep 2017 - R. Yantosca - Don't allow acc_interval for inst collections
29 Sep 2017 - R. Yantosca - Now get the starting and ending date/time info
                            from the Input_Opt object
24 Jan 2018 - E. Lundgren - Allow diagnostic names to include input params
06 Feb 2018 - E. Lundgren - Change TS_DYN units from minutes to seconds
 9 Mar 2018 - R. Yantosca - Now accept "YYYYMMDD hhmmss" as the long format
                            for collection frequency and duration attrs
```

1.5.4 History_AddItemToCollection

Creates a HISTORY ITEM object for a given diagnostic quantity, and then attaches it to a given diagnostic collection. Given the name of the diagnostic quantity, it will obtain metadata (and pointers to the data array) via the appropriate state registry.

INTERFACE:

```
SUBROUTINE History_AddItemToCollection( am_I_Root,    Input_Opt,          &
                                        State_Chm,    State_Diag,         &
                                        State_Met,    Collection,         &
                                        CollectionId, ItemName,           &
                                        ItemCount,    SubsetDims,         &
                                        RC                               )
```

USES:

```
USE Charpak_Mod,           ONLY : To_UpperCase
USE ErrCode_Mod
USE HistContainer_Mod
USE HistItem_Mod
USE History_Util_Mod,      ONLY : UNDEFINED_INT
USE Input_Opt_Mod,         ONLY : OptInput
USE MetaHistContainer_Mod
USE MetaHistItem_Mod
USE Registry_Mod,          ONLY : Registry_Lookup
USE State_Chm_Mod
USE State_Diag_Mod
USE State_Met_Mod
```

INPUT PARAMETERS:

```
! Required arguments
LOGICAL,             INTENT(IN)  :: am_I_Root      ! Are we on the root CPU?
TYPE(OptInput),      INTENT(IN)  :: Input_Opt      ! Input Options State
TYPE(ChmState),      INTENT(IN)  :: State_Chm      ! Chemistry State
TYPE(DgnState),      INTENT(IN)  :: State_Diag     ! Diagnostic State
TYPE(MetState),      INTENT(IN)  :: State_Met      ! Meteorology State
INTEGER,             INTENT(IN)  :: CollectionID   ! Collection ID number
CHARACTER(LEN=255),  INTENT(IN)  :: ItemName       ! Name of HISTORY ITEM
INTEGER,             INTENT(IN)  :: ItemCount      ! Index of HISTORY ITEM

! Optional arguments
INTEGER,             OPTIONAL    :: SubsetDims(3)  ! Dimensions specified
                                                   !  by the collection
```

INPUT/OUTPUT PARAMETERS:

```
TYPE(HistContainer), POINTER     :: Collection     ! Diagnostic Collection
```

OUTPUT PARAMETERS:

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

REMARKS:

```
Private routine, called from History_Init.
```

REVISION HISTORY:

```
06 Jan 2015 - R. Yantosca - Initial version
03 Aug 2017 - R. Yantosca - Inherit operation code from the Collection
26 Sep 2017 - E. Lundgren - Replace Lookup_State_xx calls with direct
                            calls to Registry_Lookup
01 Nov 2017 - R. Yantosca - Make the registry lookup case-insensitive
```

1.5.5 History_SetTime

Sets the time values for each HISTORY CONTAINER object that specifies a diagnostic collection.

INTERFACE:

```
SUBROUTINE History_SetTime( am_I_Root, RC )
```

USES:

```
USE ErrCode_Mod
USE HistContainer_Mod,     ONLY : HistContainer_SetTime
USE History_Util_Mod
USE MetaHistContainer_Mod, ONLY : MetaHistContainer
```

INPUT PARAMETERS:

```
LOGICAL,  INTENT(IN)  :: am_I_Root        ! Are we on the root CPU?
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This routine is meant to be called after History_Update() but before
History_Write().
```

REVISION HISTORY:

```
18 Aug 2017 - R. Yantosca - Initial version
29 Aug 2017 - R. Yantosca - Remove HeartBeatDtMin as an argument; now the
                            Container object contains heartbeat timesteps
```

1.5.6 History_Update

For each HISTORY ITEM belonging to a diagnostic COLLECTION, the data from the target variable is copied or accumulated into the HISTORY ITEM's data field for further analysis.

INTERFACE:

```
SUBROUTINE History_Update( am_I_Root, RC )
```

USES:

```
USE ErrCode_Mod
USE HistItem_Mod,          ONLY : HistItem
USE HistContainer_Mod,     ONLY : HistContainer
USE History_Util_Mod
USE MetaHistContainer_Mod, ONLY : MetaHistContainer
USE MetaHistItem_Mod,      ONLY : MetaHistItem
USE Registry_Params_Mod
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN)  :: am_I_Root  ! Are we on the root CPU?
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This routine is called from the main program at the end of each
"heartbeat" timestep.
```

REVISION HISTORY:

```
03 Aug 2017 - R. Yantosca - Initial version
11 Aug 2017 - R. Yantosca - Remove references to 0d pointers, data arrays
16 Aug 2017 - R. Yantosca - Now call TestTimeForAction to test if it is
                            time to update the diagnostic collection.
21 Aug 2017 - R. Yantosca - Now get yyyymmdd, hhmmss from the container
```

1.5.7 History_Write

For each HISTORY ITEM belonging to a diagnostic COLLECTION, the data from the target variable is copied or accumulated into the HISTORY ITEM's data field for further analysis.

INTERFACE:

```
SUBROUTINE History_Write( am_I_Root, Spc_Units, RC )
```

USES:

```
USE ErrCode_Mod
USE HistContainer_Mod
USE HistItem_Mod,          ONLY : HistItem
USE History_Netcdf_Mod
USE History_Util_Mod
USE MetaHistContainer_Mod, ONLY : MetaHistContainer
USE MetaHistItem_Mod,      ONLY : MetaHistItem
USE Registry_Params_Mod
```

INPUT PARAMETERS:

```
LOGICAL,          INTENT(IN)  :: am_I_Root   ! Are we on the root CPU?
CHARACTER(LEN=*), INTENT(IN)  :: Spc_Units   ! Units of SC%Species array
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This routine is called from the main program at the end of each
"heartbeat" timestep.
```

REVISION HISTORY:

```
03 Aug 2017 - R. Yantosca - Initial version
21 Aug 2017 - R. Yantosca - Now get yyyymmdd, hhmmss from the container
28 Aug 2017 - R. Yantosca - Now save the species units to the container
06 Sep 2017 - R. Yantosca - Now recompute the file write and file close
                            intervals, if they are 1 month or longer
```

1.5.8 GetCollectionMetaData

Parses a line of the HISTORY.rc file and returns metadata for a given attribute (e.g. "frequency", "template", etc.)

INTERFACE:

```
SUBROUTINE GetCollectionMetaData( Line, Pattern, MetaData, nCollection )
```

USES:

```
USE Charpak_Mod,      ONLY: CleanText, StrSplit
USE DiagList_Mod,     ONLY: CollList,  Search_CollList
USE History_Util_Mod
```

INPUT PARAMETERS:

```
CHARACTER(LEN=*),   INTENT(IN)  :: Line          ! Line to be searched
CHARACTER(LEN=*),   INTENT(IN)  :: Pattern       ! Search pattern
```

OUTPUT PARAMETERS:

```
CHARACTER(LEN=255), INTENT(OUT) :: MetaData      ! Metadata value
INTEGER,            INTENT(OUT) :: nCollection   ! Collection Id
```

REVISION HISTORY:

```
06 Jan 2015 - R. Yantosca - Initial version
03 Aug 2017 - R. Yantosca - Make search algorithm more robust
14 Aug 2017 - R. Yantosca - Initialize MetaData and nCollection
15 Aug 2017 - R. Yantosca - Bug fix: TRIM string arguments to INDEX, and
                            initialize output arguments to undefined values
01 Nov 2017 - R. Yantosca - Now get CleanText from charpak_mod.F90
18 Jan 2018 - R. Yantosca - Bug fix: now DO N = 1, CollectionCount
```

1.5.9 History_Close_AllFiles

Closes the netCDF file described by each HISTORY CONTAINER object in the master list of diagnostic collections.

INTERFACE:

```
SUBROUTINE History_Close_AllFiles( am_I_Root, RC )
```

USES:

```
USE ErrCode_Mod
USE History_Netcdf_Mod,    ONLY : History_Netcdf_Close
USE MetaHistContainer_Mod, ONLY : MetaHistContainer
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN)  :: am_I_Root   ! Are we on the root CPU?
```

OUTPUT PARAMETERS:

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

REMARKS:

```
This is called from History_Cleanup, but may also be called in other
locations (e.g. when processing abnormal exits)
```

REVISION HISTORY:

```
16 Aug 2017 - R. Yantosca - Initial version
```

1.5.10 History_Cleanup

Deallocates all module variables and objects. Also closes any remaining open netCDF files.

INTERFACE:

```
SUBROUTINE History_Cleanup( am_I_Root, RC )
```

USES:

```
USE ErrCode_Mod
USE History_Netcdf_Mod,    ONLY : History_Netcdf_Cleanup
USE MetaHistContainer_Mod, ONLY : MetaHistContainer_Destroy
```

INPUT PARAMETERS:

```
LOGICAL, INTENT(IN)  :: am_I_Root
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

```
16 Jun 2017 - R. Yantosca - Initial version
14 Aug 2017 - R. Yantosca - Call History_Netcdf_Close to close open files
16 Aug 2017 - R. Yantosca - Move netCDF close code to History_Close_AllFiles
26 Sep 2017 - R. Yantosca - Now call MetaHistItem_Destroy to finalize the
                            ContainerList object, instead of DEALLOCATE
```