The Model Coupling Toolkit API Reference Manual: MCT v. 2.11

R. L. Jacob
J. W. Larson
E. Ong
R. Loy
Mathematics and Computer Science Division, Argonne National Laboratory

Revision History

This Technical Note was produced for the Scientific Discovery through Advanced Computing (SciDAC) project.

Version Number	**Version Date**	**Pages Affected/ Extent of Changes**	**Aproval Authority**
Version 1β	December 13, 2000	First draft (before review)	
Version 1β2	February 16, 2001	Add more routines	
Version 1β3	June 6, 2001	Convert to pure API's doc	
Version 1β4	Apr 24, 2002	Update with latest source	
Version 1.0	Nov 14, 2002	1.0 Version	
Version 2.0.0	Apr 23, 2004	2.0.0 Version	
Version 2.0.1	May 18, 2004	2.0.1 Version	
Version 2.1.0	Feb 11, 2005	2.1.0 Version	
Version 2.2.0	Dec 01, 2005	2.2.0 Version	
Version 2.2.1	Apr 22, 2006	2.2.1 Version	
Version 2.2.2	Sep 08, 2006	2.2.2 Version	
Version 2.2.3	Oct 16, 2006	2.2.3 Version	
Version 2.3.0	Jan 10, 2007	2.3.0 Version	
Version 2.4.0	Aug 17, 2007	2.4.0 Version	
Version 2.4.1	Nov 21, 2007	2.4.1 Version	
Version 2.5.0	Jan 28, 2008	2.5.0 Version	
Version 2.5.1	May 20, 2008	2.5.1 Version	
Version 2.6.0	Mar 05, 2009	2.6.0 Version	
Version 2.7.0	Jan 05, 2010	2.7.0 Version	
Version 2.7.1	Feb 28, 2010	2.7.1 Version	
Version 2.7.2	Nov 30, 2010	2.7.2 Version	
Version 2.7.3	Jan 25, 2011	2.7.3 Version	
Version 2.7.4	Mar 07, 2012	2.7.4 Version	
Version 2.8.0	Apr 30, 2012	2.8.0 Version	
Version 2.8.1	Jul 05, 2012	2.8.1 Version	
Version 2.8.2	Sep 12, 2012	2.8.2 Version	
Version 2.8.3	Dec 17, 2012	2.8.3 Version	
Version 2.9.0	Jun 19, 2015	2.9.0 Version	
Version 2.10.0	Apr 19, 2018	2.10.0 Version	
Version 2.11.0	Feb 11, 2021	2.11.0 Version	

This document describes the Application Program Interfaces (APIs) for the Model Coupling Toolkit (MCT).

For functions that take a Fortran90 `real` argument, either a scalar or a vector, MCT provides both double and single precision versions. Only the single precision version are described here denoted by SP. The double precision versions are otherwise identical.

Contents

Part I
Basic API's and associated communication routines

1 MCTWorld

1.1 Module m_MCTWorld – MCTWorld Class (Source File: m_MCTWorld.F90)

MCTWorld is a datatype which acts as a component model registry. All models communicating through MCT must participate in initialization of MCTWorld. The single instance of MCTWorld, `ThisMCTWorld` stores the component id and local and global processor rank of each component. This module contains methods for creating and destroying `ThisMCTWorld` as well as inquiry functions.

INTERFACE:

```
module m_MCTWorld
```

USES:

```
use m_List, only : List   ! Support for List components.

implicit none

private   ! except
```

PUBLIC TYPES:

```
public :: MCTWorld        ! The MCTWorld  class data structure

type MCTWorld
  integer :: MCT_comm                          ! MCT communicator
  integer :: ncomps                            ! Total number of components
  integer :: mygrank                           ! Rank of this processor in
                                               ! global communicator.
  integer,dimension(:),pointer :: nprocspid => null()   ! Number of processes
                                               ! each component is on (e.g. rank of its
! local communicator.
  integer,dimension(:,:),pointer :: idGprocid => null() ! Translate between local component
                                               ! rank in global communicator.
! idGprocid(modelid,localrank)=globalrank
end type MCTWorld
```

PUBLIC DATA MEMBERS:

```
type(MCTWorld) :: ThisMCTWorld   !  declare the MCTWorld
```

PUBLIC MEMBER FUNCTIONS:

```
public :: initialized          ! Determine if MCT is initialized
public :: init                 ! Create a MCTWorld
public :: clean                ! Destroy a MCTWorld
public :: printnp                ! Print contents of a MCTWorld
public :: NumComponents        ! Number of Components in the MCTWorld
public :: ComponentNumProcs    ! Number of processes owned by a given
                               ! component
public :: ComponentToWorldRank ! Given the rank of a process on a
```

```
                                  ! component, return its rank on the
                                  ! world communicator
    public :: ComponentRootRank   ! Return the rank on the world
                                  ! communicator of the root process of
                                  ! a component
    public :: ThisMCTWorld        ! Instantiation of the MCTWorld


  interface initialized ; module procedure &
    initialized_
  end interface
  interface init ; module procedure &
    initd_, &
    initm_, &
    initr_
  end interface
  interface clean ; module procedure clean_ ; end interface
  interface printnp ; module procedure printnp_ ; end interface
  interface NumComponents ; module procedure &
     NumComponents_
  end interface
  interface ComponentNumProcs ; module procedure &
     ComponentNumProcs_
  end interface
  interface ComponentToWorldRank ; module procedure &
     ComponentToWorldRank_
  end interface
  interface ComponentRootRank ; module procedure &
     ComponentRootRank_
  end interface
```

REVISION HISTORY:

```
19Jan01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
05Feb01 - J. Larson <larson@mcs.anl.gov> - added query and
          local-to-global mapping services NumComponents,
          ComponentNumProcs, ComponentToWorldRank, and ComponentRootRank
08Feb01 - R. Jacob <jacob@mcs.anl.gov> - add mylrank and mygrank
          to datatype
20Apr01 - R. Jacob <jacob@mcs.anl.gov> - remove allids from
          MCTWorld datatype.  Not needed because component
          ids are always from 1 to number-of-components.
07Jun01 - R. Jacob <jacob@mcs.anl.gov> - remove myid, mynprocs
          and mylrank from MCTWorld datatype because they are not
          clearly defined in PCM mode.  Add MCT_comm for future use.
03Aug01 - E. Ong <eong@mcs.anl.gov> - explicity specify starting
          address in mpi_irecv
27Nov01 - E. Ong <eong@mcs.anl.gov> - added R. Jacob's version of initd_
          to support PCM mode.
15Feb02 - R. Jacob - elminate use of MP_COMM_WORLD.  Use
          argument globalcomm instead.  Create MCT_comm from
          globalcomm
```

1.1.1 initialized_ - determine if MCTWorld is initialized

This routine may be used to determine whether `MCTWorld::init` has been called. If not, the user must call `init` before performing any other MCT library calls.

INTERFACE:

```
logical function initialized_()
```

USES:

INPUT PARAMETERS:

REVISION HISTORY:

```
01June07 - R. Loy <rloy@mcs.anl.gov> - initial version
```

1.1.2 initm_ - initialize MCTWorld

Do a distributed init of MCTWorld for the case where a set of processors contains more then one model and the models may not span the set of processors. `ncomps` is the total number of components in the entire coupled system. `globalcomm` encompasses all the models (typically this can be MPI_COMM_WORLD). `mycomms` is an array of MPI communicators, each sized for the appropriate model and `myids` is a corresponding array of integers containing the model ids for the models on this particular set of processors.
This routine is called once for the models covered by the set of processors.

INTERFACE:

```
subroutine initm_(ncomps,globalcomm,mycomms,myids)
```

USES:

```
      use m_mpif90
      use m_die
      use m_stdio

      implicit none
```

INPUT PARAMETERS:

```
      integer, intent(in)        :: ncomps          ! number of components
      integer, intent(in)        :: globalcomm      ! global communicator
      integer, dimension(:),pointer    :: mycomms         ! my communicators
      integer, dimension(:),pointer    :: myids           ! component ids
```

REVISION HISTORY:

```
20Sep07 - T. Craig migrated code from initd routine
20Sep07 - T. Craig - made mycomms an array
03Nov19 - J. Edwards - Add barrier to improve peformance on high proc counts
```

1.1.3 initd_ - initialize MCTWorld

Do a distributed init of MCTWorld using the total number of components `ncomps` and either a unique integer component id `myid` or, if more than one model is placed on a processor, an array of integer ids specifying the models `myids`. Also required is the local communicator `mycomm` and global communicator `globalcomm` which encompasses all the models (typically this can be MPI_COMM_WORLD). This routine must be called once by each component (using *myid*) or component group (using *myids*).

INTERFACE:

```
subroutine initd_(ncomps,globalcomm,mycomm,myid,myids)
```

USES:

```
use m_mpif90
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer, intent(in)        :: ncomps          ! number of components
integer, intent(in)        :: globalcomm      ! global communicator
integer, intent(in)        :: mycomm          ! my communicator
integer, intent(in),optional     :: myid            ! my component id
integer, dimension(:),pointer,optional  :: myids    ! component ids
```

REVISION HISTORY:

```
19Jan01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
07Feb01 - R. Jacob <jacob@mcs.anl.gov> - non fatal error
          if init is called a second time.
08Feb01 - R. Jacob <jacob@mcs.anl.gov> - initialize the new
          mygrank and mylrank
20Apr01 - R. Jacob <jacob@mcs.anl.gov> - remove allids from
          MCTWorld datatype.  Not needed because component
          ids are always from 1 to number-of-components.
22Jun01 - R. Jacob <jacob@mcs.anl.gov> - move Bcast and init
          of MCTWorld to initr_
20Sep07 - T. Craig migrated code to new initm routine
```

1.1.4 initr_ - initialize MCTWorld from global root

Initialize MCTWorld using information valid only on the global root. This is called by initm_ but could also be called by the user for very complex model–processor geometries.

INTERFACE:

```
subroutine initr_(ncomps,globalcomm,rnprocspid,ridGprocid)
```

USES:

```
use m_mpif90
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer, intent(in)                :: ncomps     ! total number of components
integer, intent(in)                :: globalcomm ! the global communicator
integer, dimension(:),intent(in)   :: rnprocspid ! number of processors for each component
integer, dimension(:,:),intent(in) :: ridGprocid ! an array of size (1:ncomps) x (0:Gsize-
  ! which maps local ranks to global ranks
                                                 ! it's actually 1:Gsize here
```

REVISION HISTORY:

```
22Jun01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
```

1.1.5 clean_ - Destroy a MCTWorld

This routine deallocates the arrays of `ThisMCTWorld` It also zeros out the integer components.

INTERFACE:

```
subroutine clean_()
```

USES:

```
use m_mpif90
use m_die

implicit none
```

REVISION HISTORY:

```
19Jan01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
08Feb01 - R. Jacob <jacob@mcs.anl.gov> - clean the new
          mygrank and mylrank
20Apr01 - R. Jacob <jacob@mcs.anl.gov> - remove allids from
          MCTWorld datatype.  Not needed because component
          ids are always from 1 to number-of-components.
07Jun01 - R. Jacob <jacob@mcs.anl.gov> - remove myid,mynprocs
          and mylrank.
```

1.1.6 NumComponents_ - Determine number of components in World.

The function `NumComponents_` takes an input `MCTWorld` argument `World`, and returns the number of component models present.

INTERFACE:

```
integer function NumComponents_(World)
```

USES:

```
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
type(MCTWorld), intent(in)      :: World
```

REVISION HISTORY:

```
05Feb01 - J. Larson <larson@mcs.anl.gov> - initial version
```

1.1.7 ComponentNumProcs_ - Number of processes a component owns.

The function `ComponentNumProcs_` takes an input `MCTWorld` argument `World`, and a component ID `comp_id`, and returns the number of processes owned by that component.

INTERFACE:

```
integer function ComponentNumProcs_(World, comp_id)
```

USES:

```
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
type(MCTWorld), intent(in)      :: World
integer,        intent(in)      :: comp_id
```

REVISION HISTORY:

```
05Feb01 - J. Larson <larson@mcs.anl.gov> - initial version
07Jun01 - R. Jacob <jacob@mcs.anl.gov> - modify to use
          nprocspid and comp_id instead of World%mynprocs
```

1.1.8 ComponentToWorldRank_ - Determine rank on COMM_WORLD.

The function `ComponentToWorldRank_` takes an input component ID `comp_id` and input rank on that component communicator `comp_rank`, and returns the rank of that process on the world communicator of `MCTWorld`.

INTERFACE:

```
integer function ComponentToWorldRank_(comp_rank, comp_id, World)
```

USES:

```
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer, intent(in)      :: comp_rank ! process rank on the communicator
                                      ! associated with comp_id
integer, intent(in)      :: comp_id   ! component id
type(MCTWorld), intent(in)     :: World     ! World
```

REVISION HISTORY:

```
05Feb01 - J. Larson <larson@mcs.anl.gov> - initial version
14Jul02 - E. Ong <eong@mcs.anl.gov> - made argument checking required
```

1.1.9 ComponentRootRank_ - Rank of component root on COMM_WORLD.

The function `ComponentRootRank_` takes an input component ID `comp_id` and input `MCTWorld` variable `World`, and returns the global rank of the root of this component.

INTERFACE:

```
integer function ComponentRootRank_(comp_id, World)
```

USES:

```
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer, intent(in)      :: comp_id   ! component id
type(MCTWorld), intent(in)     :: World     ! World
```

REVISION HISTORY:

```
05Feb01 - J. Larson <larson@mcs.anl.gov> - initial version
14Jul02 - E. Ong <eong@mcs.anl.gov> - made argument checking required
```

1.1.10 printnp_ - Print number of procs for a component id.

Print out number of MPI processes for the givin component id.

INTERFACE:

```
subroutine printnp_(compid,lun)
```

USES:

```
use m_die
use m_mpif90

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer, intent(in)           :: compid
integer, intent(in)           :: lun
```

REVISION HISTORY:

```
06Jul12 - R. Jacob <jacob@mcs.anl.gov> - initial version
```

2 The Attribute Vector

2.1 Module m_AttrVect - Multi-field Storage (Source File: m_AttrVect.F90)

An *attribute vector* is a scheme for storing bundles of integer and real data vectors, indexed by the names of the fields stored in `List` format (see the mpeu module `m_List` for more information about the `List` datatype). The ordering of the fieldnames in the integer and real attribute `List` components (`AttrVect%iList` and `AttrVect%rList`, respectively) corresponds to the storage order of the attributes in their respective data buffers (the components `AttrVect%iAttr(:,:)` and `AttrVect%rAttr(:,:)`, respectively). The organization of the fieldnames in `List` format, along with the direct mapping between `List` items and locations in the data buffer, allows the user to have *random access* to the field data. This approach also allows the user to set the number and the names of fields stored in an `AttrVect` at run-time.
The `AttrVect` stores field data in a *pointwise* fashion (that is, the data are grouped so that all the integer or real data associated with an individual point are adjacent to each other in memory. This amounts to the having the integer and real field data arrays in the `AttrVect` (the components `AttrVect%iAttr(:,:)` and `AttrVect%rAttr(:,:)`, respectively) having the attribute index as the major (or fastest-varying) index. A prime example of this is observational data input to a data assimilation system. In the Model Coupling Toolkit, this datatype is the fundamental type for storing field data exchanged by component models, and forms a basis for other MCT datatypes that encapsulate time accumulation/averaging buffers (the `Accumulator` datatype defined in the module `m_Accumulator`), coordinate grid information (the `GeneralGrid` datatype defined in the module `m_GeneralGrid`), and sparse interpolation matrices (the `SparseMatrix` datatype defined in the module `m_SparseMatrix`).
The attribute vector is implemented in Fortran 90 using the `AttrVect` derived type. This module contains the definition of the `AttrVect`, and the numerous methods that service it. There are a number of initialization (creation) schemes, and a routine for zeroing out the elements of an `AttrVect`. There is a method to *clean* up allocated memory used by an `AttrVect` (destruction). There are numerous query methods that return: the number of datapoints (or *length*; the numbers of integer and real attributes; the data buffer index of a given real or integer attribute; and return the lists of real and integer attributes. There also exist methods for exporting a given attribute as a one-dimensional array and importing a given attribute from a one-dimensional array. There is a method for copying attributes from one `AttrVect` to another. There is also a method for cross-indexing the attributes in two `AttrVect` variables. In addition, there are methods that return those cross-indexed attributes along with some auxiliary data in a `AVSharedIndicesOneType` or `AVSharedIndices` structure. Finally, there are methods for sorting and permuting `AttrVect` entries using a MergeSort scheme keyed by the attributes of the `AttrVect`.

INTERFACE:

```
module m_AttrVect
```

USES:

```
use m_realkinds,only : SP,DP,FP          ! Real types definitions

use m_List, only : List   ! Support for rList and iList components.

implicit none

private ! except
```

PUBLIC TYPES:

```
public :: AttrVect        ! The class data structure
public :: AVSharedIndicesOneType ! Data structure recording shared indices between
                                 ! two attribute vectors, for a single data type
                                 ! (e.g., shared real attributes)
public :: AVSharedIndices ! Data structure recording shared indices between two
                          ! attribute vectors, for all data types
```

```
    type AttrVect
#ifdef SEQUENCE
      sequence
#endif
      type(List) :: iList
      type(List) :: rList
      integer,dimension(:,:),pointer :: iAttr
      real(FP) ,dimension(:,:),pointer :: rAttr
    end type AttrVect

    type AVSharedIndicesOneType
       integer :: num_indices           ! number of shared items
       logical :: contiguous            ! true if index segments are contiguous in memory
       character*7 :: data_flag         ! data type flag (e.g., 'REAL' or 'INTEGER')

       ! arrays of indices to storage locations of shared attributes between the two
       ! attribute vectors:
       integer, dimension(:), pointer :: aVindices1
       integer, dimension(:), pointer :: aVindices2
    end type AVSharedIndicesOneType

    type AVSharedIndices
       type(AVSharedIndicesOneType) :: shared_real     ! shared indices of type REAL
       type(AVSharedIndicesOneType) :: shared_integer  ! shared indices of type INTEGER
    end type AVSharedIndices
```

PUBLIC MEMBER FUNCTIONS:

```
      public :: init ! create a local vector
      public :: clean ! clean the local vector
      public :: zero ! zero the local vector
      public :: lsize ! size of the local vector
      public :: nIAttr ! number of integer attributes on local
      public :: nRAttr ! number of real attributes on local
      public :: indexIA ! index the integer attributes
      public :: indexRA ! index the real attributes
      public :: getIList        ! return list of integer attributes
      public :: getRList        ! return list of real attributes
      public :: getIListtoChar  ! return list of integer attributes as Char
      public :: getRListtoChar  ! return list of real attributes as Char
      public :: exportIList     ! export INTEGER attibute List
      public :: exportRList     ! export REAL attibute List
      public :: exportIListToChar ! export INTEGER attibute List as Char
      public :: exportRListToChar ! export REAL attibute List as Char
      public :: appendIAttr     ! append INTEGER attribute List
      public :: appendRAttr     ! append REAL attribute List
      public :: exportIAttr     ! export INTEGER attribute to vector
      public :: exportRAttr     ! export REAL attribute to vector
      public :: importIAttr     ! import INTEGER attribute from vector
      public :: importRAttr     ! import REAL attribute from vector
      public :: Copy ! copy attributes from one Av to another
      public :: RCopy ! copy real attributes from one Av to another
      public :: ICopy ! copy integer attributes from one Av to another
      public :: Sort            ! sort entries, and return permutation
      public :: Permute         ! permute entries
      public :: Unpermute       ! Unpermute entries
      public :: SortPermute     ! sort and permute entries
```

```
  public :: SharedAttrIndexList  ! Cross-indices of shared
                                 ! attributes of two AttrVects
  public :: SharedIndices        ! Given two AttrVects, create an AVSharedIndices structure
  public :: SharedIndicesOneType ! Given two AttrVects, create an
                                 ! AVSharedIndicesOneType structure for a single type
  public :: cleanSharedIndices   ! clean a AVSharedIndices structure
  public :: cleanSharedIndicesOneType  ! clean a AVSharedIndicesOneType structure


interface init   ; module procedure &
   init_,  &
   initv_, &
   initl_
end interface
interface clean  ; module procedure clean_  ; end interface
interface zero  ; module procedure zero_  ; end interface
interface lsize  ; module procedure lsize_  ; end interface
interface nIAttr ; module procedure nIAttr_ ; end interface
interface nRAttr ; module procedure nRAttr_ ; end interface
interface indexIA; module procedure indexIA_; end interface
interface indexRA; module procedure indexRA_; end interface
interface getIList; module procedure getIList_; end interface
interface getRList; module procedure getRList_; end interface
interface getIListToChar; module procedure getIListToChar_; end interface
interface getRListToChar; module procedure getRListToChar_; end interface
interface exportIList; module procedure exportIList_; end interface
interface exportRList; module procedure exportRList_; end interface
interface exportIListToChar
   module procedure exportIListToChar_
end interface
interface exportRListToChar
   module procedure exportRListToChar_
end interface
interface appendIAttr  ; module procedure appendIAttr_  ; end interface
interface appendRAttr  ; module procedure appendRAttr_  ; end interface
interface exportIAttr; module procedure exportIAttr_; end interface
interface exportRAttr; module procedure &
   exportRAttrSP_, &
   exportRAttrDP_
end interface
interface importIAttr; module procedure importIAttr_; end interface
interface importRAttr; module procedure &
     importRAttrSP_, &
     importRAttrDP_
end interface
interface Copy    ; module procedure Copy_    ; end interface
interface RCopy   ; module procedure  &
     RCopy_, &
     RCopyL_
end interface
interface ICopy   ; module procedure  &
      ICopy_, &
      ICopyL_
end interface
interface Sort    ; module procedure Sort_    ; end interface
interface Permute ; module procedure Permute_ ; end interface
interface Unpermute ; module procedure Unpermute_ ; end interface
interface SortPermute ; module procedure SortPermute_ ; end interface
interface SharedAttrIndexList ; module procedure &
```

```
    aVaVSharedAttrIndexList_
  end interface
  interface SharedIndices ; module procedure SharedIndices_ ; end interface
  interface SharedIndicesOneType ; module procedure SharedIndicesOneType_ ; end interface
  interface cleanSharedIndices ; module procedure cleanSharedIndices_ ; end interface
  interface cleanSharedIndicesOneType ; module procedure cleanSharedIndicesOneType_ ; end inte
```

REVISION HISTORY:

```
10Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
10Oct00 - J.W. Larson <larson@mcs.anl.gov> - made getIList
          and getRList functions public and added appropriate
          interface definitions
20Oct00 - J.W. Larson <larson@mcs.anl.gov> - added Sort,
          Permute, and SortPermute functions.
09May01 - J.W. Larson <larson@mcs.anl.gov> - added initl_().
19Oct01 - J.W. Larson <larson@mcs.anl.gov> - added routines
          exportIattr(), exportRAttr(), importIAttr(),
          and importRAttr().  Also cleaned up module and
          routine prologues.
13Dec01 - J.W. Larson <larson@mcs.anl.gov> - made importIAttr()
          and importRAttr() public (bug fix).
14Dec01 - J.W. Larson <larson@mcs.anl.gov> - added exportIList()
          and exportRList().
14Feb02 - J.W. Larson <larson@mcs.anl.gov> - added CHARCTER
          functions exportIListToChar() and exportRListToChar()
26Feb02 - J.W. Larson <larson@mcs.anl.gov> - corrected of usage
          of m_die routines throughout this module.
16Apr02 - J.W. Larson <larson@mcs.anl.gov> - added the method
          LocalReduce(), and the public data members AttrVectSUM,
          AttrVectMIN, and AttrVectMAX.
7May02 - J.W. Larson <larson@mcs.anl.gov> - Refactoring.  Moved
         LocalReduce() and the public data members AttrVectSUM,
          AttrVectMIN, and AttrVectMAX to a new module named
          m_AttrVectReduce.
12Jun02 - R.L. Jacob <jacob@mcs.anl.gov> - add Copy function
13Jun02 - R.L. Jacob <jacob@mcs.anl.gov> - move aVavSharedAttrIndexList
          to this module from old m_SharedAttrIndicies
28Apr11 - W.J. Sacks <sacks@ucar.edu> - added AVSharedIndices and
          AVSharedIndicesOneType derived types, and associated
          subroutines
10Apr12 - W.J. Sacks <sacks@ucar.edu> - modified AVSharedIndices code
          to be Fortran-90 compliant
10Jan13 - T.Craig <tcraig@ucar.edu> - add getRListToChar and getIListToChar
```

2.1.1 init_ - Initialize an AttrVect Given Attribute Lists and Length

This routine creates an `AttrVect` (the output argument `aV`) using the optional input `CHARACTER` arguments `iList`, and `rList` to define its integer and real attributes, respectively. The optional input `INTEGER` argument `lsize` defines the number of points for which we are storing attributes, or the *length* of `aV`. The expected form for the arguments `iList` and `rList` are colon-delimited strings where each substring defines an attribute. Suppose we wish to store `N` observations that have the real attributes `'latitude'`, `'longitude'`, `pressure`, `'u-wind'`, and `'v-wind'`. Suppose we also wish to store the integer attributes `'hour'`, `'day'`, `'month'`, `'year'`, and `'data source'`. This can be accomplished by invoking `init_()` as follows:

```
call init_(aV, 'hour:day:month:year:data source', &
```

```
'latitude:longitude:pressure:u-wind:v-wind', N)
```

The resulting `AttrVect aV` will have five integer attributes, five real attributes, and length `N`.

INTERFACE:

```
subroutine init_(aV, iList, rList, lsize)
```

USES:

```
use m_List, only : List
use m_List, only : init,nitem
use m_List, only : List_nullify => nullify
use m_mall
use m_die

implicit none
```

INPUT PARAMETERS:

```
character(len=*), optional, intent(in)  :: iList
character(len=*), optional, intent(in)  :: rList
integer,          optional, intent(in)  :: lsize
```

OUTPUT PARAMETERS:

```
type(AttrVect),             intent(out) :: aV
```

REVISION HISTORY:

```
09Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
09Oct01 - J.W. Larson <larson@mcs.anl.gov> - added feature to
          nullify all pointers before usage.  This was done to
          accomodate behavior of the f90 ASSOCIATED intrinsic
          function on the AIX platform.
07Dec01 - E.T. Ong <eong@mcs.anl.gov> - added support for
          intialization with blank character strings for iList
          and rList
```

2.1.2 initv_ - Initialize One AttrVect from Another

This routine takes an input `AttrVect` argument `bV`, and uses its attribute list information to create an output `AttrVect` variable `aV`. The length of `aV` is defined by the input `INTEGER` argument `lsize`.

INTERFACE:

```
subroutine initv_(aV, bV, lsize)
```

USES:

```
use m_String, only : String,char
use m_String, only : String_clean => clean
use m_List,   only : get
use m_List,   only : List_nullify => nullify
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),intent(in)  :: bV
integer,       intent(in)  :: lsize
```

OUTPUT PARAMETERS:

```
type(AttrVect),intent(out) :: aV
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
17May01 - R. Jacob <jacob@mcs.anl.gov> - add a check to see if
          input argument has been defined.  SGI will dump
          core if its not.
10Oct01 - J. Larson <larson@mcs.anl.gov> - Nullify all pointers
          in ouput AttrVect aV before initializing aV.
19Sep08 - J. Wolfe <jwolfe@ucar.edu> - plug memory leak from not deallocating
          strings.
```

2.1.3 initl_ - Initialize an AttrVect Using the List Type

This routine initializes an `AttrVect` directly from input `List` data type arguments `iList` and `rList` (see the module `m_List` in mpeu for further details), and an input length `lsize`. The resulting `AttrVect` is returned in the argument `aV`.
N.B.: If the user supplies an empty list for the arguments `iList` (`rList`), then `aV` will be created only with `REAL` (`INTEGER`) attributes. If both arguments `iList` and `rList` are empty, the routine will terminate execution and report an error.
N.B.: The outcome of this routine, `aV` represents allocated memory. When this `AttrVect` is no longer needed, it must be deallocated by invoking the routine `AttrVect_clean()`. Failure to do so will spawn a memory leak.

INTERFACE:

```
subroutine initl_(aV, iList, rList, lsize)
```

USES:

```
use m_die
use m_stdio

use m_String, only : String
use m_String, only : String_clean => clean
use m_String, only : String_toChar => toChar

use m_List, only : List
use m_List, only : List_nitem => nitem
use m_List, only : List_exportToChar => exportToChar

implicit none
```

INPUT PARAMETERS:

```
type(List),  intent(in)  :: iList
type(List),  intent(in)  :: rList
integer,     intent(in)  :: lsize
```

OUTPUT PARAMETERS:

```
type(AttrVect), intent(out) :: aV
```

REVISION HISTORY:

```
09May98 - J.W. Larson <larson@mcs.anl.gov> - initial version.
08Aug01 - E.T. Ong <eong@mcs.anl.gov> - change list assignment(=)
          to list copy to avoid compiler errors with pgf90.
10Oct01 - J. Larson <larson@mcs.anl.gov> - Nullify all pointers
          in ouput AttrVect aV before initializing aV.  Also,
          greater caution taken regarding validity of input
          arguments iList and rList.
15May08 - J. Larson <larson@mcs.anl.gov> - Simplify to use
          the init_ routine.  Better argument checking.
```

2.1.4 clean_ - Deallocate Allocated Memory Structures of an AttrVect

This routine deallocates the allocated memory structures of the input/output `AttrVect` argument `aV`. This amounts to cleaning the `List` structures `aV%iList` and `av%rList`, and deallocating the arrays `aV%iAttr(:,:)` and `aV%rAttr(:,:)`. The success (failure) of this operation is signified by a zero (non-zero) value of the optional `INTEGER` output argument `stat`. If `clean_()` is invoked without supplying `stat`, and any of the deallocation operations fail, the routine will terminate with an error message.

INTERFACE:

```
subroutine clean_(aV, stat)
```

USES:

```
use m_mall
use m_stdio
use m_die
use m_List, only : List_clean => clean

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),    intent(inout) :: aV
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
09Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
10Oct01 - J. Larson <larson@mcs.anl.gov> - various fixes to
          prevent deallocation of UNASSOCIATED pointers.
01Mar01 - E.T. Ong <eong@mcs.anl.gov> - removed dies to prevent
          crashes when cleaning uninitialized attrvects. Added
          optional stat argument.
```

2.1.5 lsize_ - Length of an AttrVect

This function returns the number of elements, or *length* of the input `AttrVect` argument `aV`. This function examines the length of the second dimension of the arrays `aV%iAttr(:,:)` and `aV%rAttr(:,:)`. If neither `aV%iAttr(:,:)` nor `aV%rAttr(:,:)` are associated, then `lsize_(aV)` = 0. If `aV%iAttr(:,:)` is associated, but `aV%rAttr(:,:)` is not, then `lsize_(aV)` = `size(aV%iAttr,2)`. If `aV%iAttr(:,:)` is not associated, but `aV%rAttr(:,:)` is, then `lsize_(aV)` = `size(aV%rAttr,2)`. If both `aV%iAttr(:,:)` and `aV%rAttr(:,:)` are associated, the function `lsize_()` will do one of two things: If `size(aV%iAttr,2)` = `size(aV%rAttr,2)`, this equal value will be returned. If `size(aV%iAttr,2)` $\neq$ `size(aV%rAttr,2)`, termination with an error message will occur.

INTERFACE:

```
integer function lsize_(aV)
```

USES:

```
use m_List,  only : List
use m_List,  only : List_allocated => allocated

use m_stdio, only : stderr
use m_die

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect), intent(in) :: aV
```

REVISION HISTORY:

```
09Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
10Oct01 - J. Larson <larson@mcs.anl.gov> - made code more robust
          to handle cases where the length of either aV%iAttr or
          aV%rAttr is zero, but the other is positive.
```

2.1.6 zero_ - Set AttrVect Field Data to Zero

This routine sets all of the point values of the integer and real attributes of an the input/output `AttrVect` argument `aV` to zero. The default action is to set the values of all the real and integer attributes to zero. The user may prevent the zeroing of the real (integer) attributes invoking `zero_()` with the optional `LOGICAL` argument `zeroReals` (`zeroInts`) set with value `.FALSE.`

INTERFACE:

```
subroutine zero_(aV, zeroReals, zeroInts)
```

USES:

```
use m_die,only : die
use m_stdio,only : stderr

use m_List, only : List
use m_List, only : List_allocated => allocated

implicit none
```

INPUT PARAMETERS:

```
logical, optional, intent(IN)    :: zeroReals
logical, optional, intent(IN)    :: zeroInts
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),    intent(INOUT) :: aV
```

REVISION HISTORY:

```
17May01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype/code
15Oct01 - J. Larson <larson@mcs.anl.gov> - switched loop order
          for cache optimization.
03Dec01 - E.T. Ong <eong@mcs.anl.gov> - eliminated looping method of
          of zeroing. "Compiler assignment" of attrvect performs faster
          on the IBM SP with mpxlf90 compiler.
05Jan10 - R. Jacob <jacob@mcs.anl.gov> - zeroing an uninitialized aV is no
          longer a fatal error.
```

2.1.7 nIAttr_ - Return the Number of Integer Attributes

This integer function returns the number of integer attributes present in the input `AttrVect` argument `aV`.

INTERFACE:

```
integer function nIAttr_(aV)
```

USES:

```
use m_List, only : nitem

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),intent(in) :: aV
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
10Oct01 - J. Larson <larson@mcs.anl.gov> - made code more robust
          by checking status of pointers in aV%iList
```

2.1.8 nRAttr_ - Return the Number of Real Attributes

This integer function returns the number of real attributes present in the input `AttrVect` argument `aV`.

INTERFACE:

```
integer function nRAttr_(aV)
```

USES:

```
use m_List, only : nitem

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),intent(in) :: aV
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
10Oct01 - J. Larson <larson@mcs.anl.gov> - made code more robust
          by checking status of pointers in aV%iList
```

2.1.9 getIList_ - Retrieve the Name of a Numbered Integer Attribute

This routine returns the name of the `ith` integer attribute of the input `AttrVect` argument `aVect`. The name is returned in the output `String` argument `item` (see the mpeu module `m_String` for more information regarding the `String` type).

INTERFACE:

```
subroutine getIList_(item, ith, aVect)
```

USES:

```
use m_String, only : String
use m_List,   only : get

implicit none
```

INPUT PARAMETERS:

```
integer,     intent(in)  :: ith
type(AttrVect),intent(in) :: aVect
```

OUTPUT PARAMETERS:

```
type(String),intent(out) :: item
```

REVISION HISTORY:

```
24Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

2.1.10 getRList_ - Retrieve the Name of a Numbered Real Attribute

This routine returns the name of the `ith` real attribute of the input `AttrVect` argument `aVect`. The name is returned in the output `String` argument `item` (see the mpeu module `m_String` for more information regarding the `String` type).

INTERFACE:

```
subroutine getRList_(item, ith, aVect)
```

USES:

```
use m_String, only : String
use m_List,   only : get

implicit none
```

INPUT PARAMETERS:

```
integer,        intent(in)  :: ith
type(AttrVect), intent(in)  :: aVect
```

OUTPUT PARAMETERS:

```
type(String),   intent(out) :: item
```

REVISION HISTORY:

```
24Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

2.1.11 getIListToChar_ - Retrieve the Name of a Numbered Integer Attribute

This routine returns the name of the `ith` integer attribute of the input `AttrVect` argument `aVect`. The name is returned in the function `char` argument.

INTERFACE:

```
function getIListToChar_(ith, aVect)
```

USES:

```
use m_String, only : String
use m_String, only : String_ToChar => ToChar
use m_String, only : String_clean => clean
use m_List,   only : get

implicit none
```

INPUT PARAMETERS:

```
integer,     intent(in)  :: ith
type(AttrVect),intent(in) :: aVect
```

OUTPUT PARAMETERS:

```
character(len=size(aVect%iList%bf,1)) :: getIListToChar_
```

REVISION HISTORY:

```
10Jan13 - T. Craig <tcraig@ucar.edu> - initial prototype/prolog/code
```

2.1.12 getRListToChar_ - Retrieve the Name of a Numbered Integer Attribute

This routine returns the name of the `ith` integer attribute of the input `AttrVect` argument `aVect`. The name is returned in the function `char` argument.

INTERFACE:

```
function getRListToChar_(ith, aVect)
```

USES:

```
use m_String, only : String
use m_String, only : String_ToChar => ToChar
use m_String, only : String_clean => clean
use m_List,   only : get

implicit none
```

INPUT PARAMETERS:

```
integer,     intent(in)  :: ith
type(AttrVect),intent(in) :: aVect
```

OUTPUT PARAMETERS:

```
character(len=size(aVect%rList%bf,1)) :: getRListToChar_
```

REVISION HISTORY:

```
10Jan13 - T. Craig <tcraig@ucar.edu> - initial prototype/prolog/code
```

2.1.13 indexIA_ - Index an Integer Attribute

This function returns an `INTEGER`, corresponding to the location of an integer attribute within the input `AttrVect` argument `aV`. For example, suppose `aV` has the following attributes `'month'`, `'day'`, and `'year'`. The array of integer values for the attribute `'day'` is stored in `aV%iAttr(indexIA_(aV,'day'),:)`. If `indexIA_()` is unable to match `item` to any of the integer attributes in `aV`, the resulting value is zero which is equivalent to an error. The optional input `CHARACTER` arguments `perrWith` and `dieWith` control how such errors are handled.

1. if neither `perrWith` nor `dieWith` are present, `indexIA_()` terminates execution with an internally generated error message;
2. if `perrWith` is present, but `dieWith` is not, an error message is written to `stderr` incorporating user-supplied traceback information stored in the argument `perrWith`;
3. if `perrWith` is present, but `dieWith` is not, and `perrWith` is equal to "quiet", no error message is written.
4. if `dieWith` is present, execution terminates with an error message written to `stderr` that incorporates user-supplied traceback information stored in the argument `dieWith`; and
5. if both `perrWith` and `dieWith` are present, execution terminates with an error message using `dieWith`, and the argument `perrWith` is ignored.

INTERFACE:

```
integer function indexIA_(aV, item, perrWith, dieWith)
```

USES:

```
use m_die,  only : die
use m_stdio,only : stderr

use m_String, only : String
use m_String, only : String_init => init
```

```
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => ToChar

use m_List, only : index

use m_TraceBack, only : GenTraceBackString

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(in) :: aV
character(len=*),            intent(in) :: item
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

REVISION HISTORY:

```
27Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
 2Aug02 - J. Larson - Solidified error handling using perrWith/dieWith
 1Jan05 - R. Jacob - add quiet option for error handling
```

2.1.14 indexRA_ - Index a Real Attribute

This function returns an `INTEGER`, corresponding to the location of a real attribute within the input `AttrVect` argument `aV`. For example, suppose `aV` has the following attributes `'latitude'`, `'longitude'`, and `'pressure'`. The array of real values for the attribute `'longitude'` is stored in `aV%iAttr(indexRA_(aV,'longitude'),:)`. If `indexRA_()` is unable to match `item` to any of the real attributes in `aV`, the resulting value is zero which is equivalent to an error. The optional input `CHARACTER` arguments `perrWith` and `dieWith` control how such errors are handled.

1. if neither `perrWith` nor `dieWith` are present, `indexRA_()` terminates execution with an internally generated error message;
2. if `perrWith` is present, but `dieWith` is not, an error message is written to `stderr` incorporating user-supplied traceback information stored in the argument `perrWith`;
3. if `perrWith` is present, but `dieWith` is not, and `perrWith` is equal to "quiet", no error message is written.
4. if `dieWith` is present, execution terminates with an error message written to `stderr` that incorporates user-supplied traceback information stored in the argument `dieWith`; and
5. if both `perrWith` and `dieWith` are present, execution terminates with an error message using `dieWith`, and the argument `perrWith` is ignored.

INTERFACE:

```
integer function indexRA_(aV, item, perrWith, dieWith)
```

USES:

```
use m_die,  only : die
use m_stdio,only : stderr

use m_List, only : index

use m_String, only : String
```

```
use m_String, only : String_init => init
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => ToChar

use m_TraceBack, only : GenTraceBackString

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),             intent(in) :: aV
character(len=*),           intent(in) :: item
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

REVISION HISTORY:

```
27Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
 2Aug02 - J. Larson - Solidified error handling using perrWith/dieWith
18Jan05 - R. Jacob - add quiet option for error handling
```

2.1.15 appendIAttr_ - Append one or more attributes onto the INTEGER part of an AttrVect.

This routine takes an input `AttrVect` argument `aV`, and an input character string `rList` and Appends `rList` to the INTEGER part of `aV`. The success (failure) of this operation is signified by a zero (nonzero) value for the optional `INTEGER` output argument `status`.

INTERFACE:

```
subroutine appendIAttr_(aV, iList, status)
```

USES:

```
use m_List,   only : List_init => init
use m_List,   only : List_append => append
use m_List,   only : List_clean => clean
use m_List,   only : List_nullify => nullify
use m_List,   only : List_allocated => allocated
use m_List,   only : List_copy => copy
use m_List,   only : List
use m_die
use m_stdio

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),intent(inout)  :: aV
```

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
integer,optional,intent(out)  :: status
```

REVISION HISTORY:

```
08Jul03 - R. Jacob <jacob@mcs.anl.gov> - initial version
```

2.1.16 appendRAttr_ - Append one or more attributes onto the REAL part of an AttrVect.

This routine takes an input `AttrVect` argument `aV`, and an input character string `rList` and Appends `rList` to the REAL part of `aV`. The success (failure) of this operation is signified by a zero (nonzero) value for the optional `INTEGER` output argument `status`.

INTERFACE:

```
subroutine appendRAttr_(aV, rList, status)
```

USES:

```
use m_List,   only : List_init => init
use m_List,   only : List_append => append
use m_List,   only : List_clean => clean
use m_List,   only : List_nullify => nullify
use m_List,   only : List_allocated => allocated
use m_List,   only : List_copy => copy
use m_List,   only : List
use m_die
use m_stdio

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),intent(inout)  :: aV
```

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
integer,optional,intent(out)  :: status
```

REVISION HISTORY:

```
04Jun03 - R. Jacob <jacob@mcs.anl.gov> - initial version
```

2.1.17 exportIList_ - Return INTEGER Attribute List

This routine extracts from the input `AttrVect` argument `aV` the integer attribute list, and returns it as the `List` output argument `outIList`. The success (failure) of this operation is signified by a zero (nonzero) value for the optional `INTEGER` output argument `status`.
N.B.: This routine returns an allocated `List` data structure (`outIList`). The user is responsible for deallocating this structure by invoking `List_clean()` (see the module `m_List` for details) once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportIList_(aV, outIList, status)
```

USES:

```
use m_die ,  only : die
use m_stdio, only : stderr

use m_List,  only : List
use m_List,  only : List_allocated => allocated
use m_List,  only : List_copy => copy
use m_List,  only : List_nullify => nullify

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),             intent(in)  :: aV
```

OUTPUT PARAMETERS:

```
type(List),                 intent(out) :: outIList
integer,          optional, intent(out) :: status
```

REVISION HISTORY:

```
14Dec01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

2.1.18 exportRList_ - Return REAL attribute List

This routine extracts from the input `AttrVect` argument `aV` the real attribute list, and returns it as the `List` output argument `outRList`. The success (failure) of this operation is signified by a zero (nonzero) value for the optional `INTEGER` output argument `status`.
N.B.: This routine returns an allocated `List` data structure (`outRList`). The user is responsible for deallocating this structure by invoking `List_clean()` (see the module `m_List` for details) once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportRList_(aV, outRList, status)
```

USES:

```
use m_die ,  only : die
use m_stdio, only : stderr

use m_List,  only : List
use m_List,  only : List_allocated => allocated
use m_List,  only : List_copy => copy
use m_List,  only : List_nullify => nullify

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),           intent(in)  :: aV
```

OUTPUT PARAMETERS:

```
type(List),               intent(out) :: outRList
integer,        optional, intent(out) :: status
```

REVISION HISTORY:

```
14Dec01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

2.1.19 exportIListToChar_ - Return AttrVect%iList as CHARACTER

This routine extracts from the input `AttrVect` argument `aV` the integer attribute list (see the mpeu module `m_List` for more information regarding the `List` type), and returns it as a `CHARACTER` suitable for printing. An example of its usage is

```
write(stdout,'(1a)') exportIListToChar_(aV)
```

which writes the contents of `aV%iList%bf` to the Fortran device `stdout`.

INTERFACE:

```
function exportIListToChar_(aV)
```

USES:

```
use m_die ,  only : die
use m_stdio, only : stderr

use m_List,  only : List
use m_List,  only : List_allocated => allocated
use m_List,  only : List_copy => copy
use m_List,  only : List_exportToChar => exportToChar
use m_List,  only : List_clean => clean

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),       intent(in) :: aV
```

OUTPUT PARAMETERS:

```
character(len=size(aV%iList%bf,1)) :: exportIListToChar_
```

REVISION HISTORY:

```
13Feb02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
05Jun03 - R. Jacob <jacob@mcs.anl.gov> - return a blank instead of dying
          to avoid I/O errors when this function is used in a write statement.
```

2.1.20 exportRListToChar_ - Return AttrVect%rList as CHARACTER

This routine extracts from the input `AttrVect` argument `aV` the real attribute list (see the mpeu module `m_List` for more information regarding the `List` type), and returns it as a `CHARACTER` suitable for printing. An example of its usage is

```
write(stdout,'(1a)') exportRListToChar_(aV)
```

which writes the contents of `aV%rList%bf` to the Fortran device `stdout`.

INTERFACE:

```
function exportRListToChar_(aV)
```

USES:

```
use m_die ,  only : die
use m_stdio, only : stderr

use m_List,  only : List
use m_List,  only : List_allocated => allocated
use m_List,  only : List_copy => copy
use m_List,  only : List_exportToChar => exportToChar
use m_List,  only : List_clean => clean

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),       intent(in) :: aV
```

OUTPUT PARAMETERS:

```
character(len=size(aV%rList%bf,1)) :: exportRListToChar_
```

REVISION HISTORY:

```
13Feb02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
05Jun03 - R. Jacob <jacob@mcs.anl.gov> - return a blank instead of dying
          to avoid I/O errors when this function is used in a write statement.
```

2.1.21 exportIAttr_ - Return INTEGER Attribute as a Vector

This routine extracts from the input `AttrVect` argument `aV` the integer attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`, and returns it in the `INTEGER` output array `outVect`, and its length in the output `INTEGER` argument `lsize`. The optional input `CHARACTER` arguments `perrWith` and `dieWith` control how errors are handled.

1. if neither `perrWith` nor `dieWith` are present, `exportIAttr_()` terminates execution with an internally generated error message;
2. if `perrWith` is present, but `dieWith` is not, an error message is written to `stderr` incorporating user-supplied traceback information stored in the argument `perrWith`;
3. if `dieWith` is present, execution terminates with an error message written to `stderr` that incorporates user-supplied traceback information stored in the argument `dieWith`; and

4. if both `perrWith` and `dieWith` are present, execution terminates with an error message using `dieWith`, and the argument `perrWith` is ignored.

N.B.: This routine will fail if the `AttrTag` is not in the `AttrVect` `List` component `aV%iList`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `outVect` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `outVect`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) before this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `outVect`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportIAttr_(aV, AttrTag, outVect, lsize, perrWith, dieWith)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

use m_String, only : String
use m_String, only : String_init => init
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => ToChar

use m_TraceBack, only : GenTraceBackString

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(in) :: aV
character(len=*),            intent(in) :: AttrTag
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

OUTPUT PARAMETERS:

```
integer,      dimension(:), pointer     :: outVect
integer,          optional, intent(out) :: lsize
```

REVISION HISTORY:

```
19Oct01 - J.W. Larson <larson@mcs.anl.gov> - initial (slow)
          prototype.
 6May02 - J.W. Larson <larson@mcs.anl.gov> - added capability
          to work with pre-allocated outVect.
```

2.1.22 exportRAttrSP_ - Return REAL Attribute as a Pointer to Array

This routine extracts from the input `AttrVect` argument `aV` the real attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`, and returns it in the `REAL` output array `outVect`, and its length in the output `INTEGER` argument `lsize`. The optional input `CHARACTER` arguments `perrWith` and `dieWith` control how errors are handled.

1. if neither `perrWith` nor `dieWith` are present, `exportRAttr_()` terminates execution with an internally generated error message;
2. if `perrWith` is present, but `dieWith` is not, an error message is written to `stderr` incorporating user-supplied traceback information stored in the argument `perrWith`;
3. if `dieWith` is present, execution terminates with an error message written to `stderr` that incorporates user-supplied traceback information stored in the argument `dieWith`; and
4. if both `perrWith` and `dieWith` are present, execution terminates with an error message using `dieWith`, and the argument `perrWith` is ignored.

N.B.: This routine will fail if the `AttrTag` is not in the `AttrVect List` component `aV%iList`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `outVect` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `outVect`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) before this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `outVect`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportRAttrSP_(aV, AttrTag, outVect, lsize, perrWith, dieWith)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr


use m_String, only : String
use m_String, only : String_init => init
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => ToChar

use m_TraceBack, only : GenTraceBackString

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(in) :: aV
character(len=*),            intent(in) :: AttrTag
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

OUTPUT PARAMETERS:

```
real(SP),        dimension(:),  pointer     :: outVect
integer,          optional,     intent(out) :: lsize
```

REVISION HISTORY:

```
19Oct01 - J.W. Larson <larson@mcs.anl.gov> - initial (slow)
          prototype.
 6May02 - J.W. Larson <larson@mcs.anl.gov> - added capability
          to work with pre-allocated outVect.
```

2.1.23 importIAttr_ - Import INTEGER Vector as an Attribute

This routine imports into the input/output `AttrVect` argument `aV` the integer attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`. The data to be imported is provided in the `INTEGER` input array `inVect`, and the number of entries to be imported in the optional input `INTEGER` argument `lsize`.
N.B.: This routine will fail if the `AttrTag` is not in the `AttrVect` `List` component `aV%iList`.

INTERFACE:

```
subroutine importIAttr_(aV, AttrTag, inVect, lsize)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

implicit none
```

INPUT PARAMETERS:

```
character(len=*),       intent(in)    :: AttrTag
integer,  dimension(:), pointer       :: inVect
integer,  optional,     intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),         intent(inout) :: aV
```

REVISION HISTORY:

```
19Oct01 - J.W. Larson <larson@mcs.anl.gov> - initial (slow)
          prototype.
```

2.1.24 importRAttrSP_ - Import REAL Vector as an Attribute

This routine imports into the input/output `AttrVect` argument `aV` the real attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`. The data to be imported is provided in the `REAL` input array `inVect`, and its length in the optional input `INTEGER` argument `lsize`.
N.B.: This routine will fail if the `AttrTag` is not in the `AttrVect` `List` component `aV%rList`.

INTERFACE:

```
subroutine importRAttrSP_(aV, AttrTag, inVect, lsize)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

implicit none
```

INPUT PARAMETERS:

```
character(len=*),   intent(in)    :: AttrTag
real(SP), dimension(:), pointer   :: inVect
integer, optional,  intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),     intent(inout) :: aV
```

REVISION HISTORY:

```
19Oct01 - J.W. Larson <larson@mcs.anl.gov> - initial (slow)
          prototype.
```

2.1.25 RCopy_ - Copy Real Attributes from One AttrVect to Another

This routine copies from input argment `aVin` into the output `AttrVect` argument `aVout` the shared real attributes.
If the optional argument `Vector` is present and true, the vector architecture-friendly portions of this routine will be invoked.
If the optional argument `sharedIndices` is present, it should be the result of the call `SharedIndicesOneType_(aVin, aVout, 'REAL', sharedIndices)`. Providing this argument speeds up this routine substantially. For example, you can compute a `sharedIndices` structure once for a given pair of `AttrVect`s, then use that same structure for all copies between those two `AttrVect`s (although note that a different `sharedIndices` variable would be needed if `aVin` and `aVout` were reversed).
N.B.: This routine will fail if the `aVout` is not initialized.

INTERFACE:

```
subroutine RCopy_(aVin, aVout, vector, sharedIndices)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(in)    :: aVin
logical, optional,           intent(in)    :: vector
type(AVSharedIndicesOneType), optional, intent(in) :: sharedIndices
```

OUTPUT PARAMETERS:

```
type(AttrVect),              intent(inout) :: aVout
```

REVISION HISTORY:

```
18Aug06 - R. Jacob <jacob@mcs.anl.gov> - initial version.
28Apr11 - W.J. Sacks <sacks@ucar.edu> - added sharedIndices argument
```

2.1.26 RCopyL_ - Copy Specific Real Attributes from One AttrVect to Another

This routine copies from input argment `aVin` into the output `AttrVect` argument `aVout` the real attributes specified in input `CHARACTER` argument `rList`. The attributes can be listed in any order. If any attributes in `aVout` have different names but represent the the same quantity and should still be copied, you must provide a translation argument `TrList`. The translation arguments should be identical in length to the `rList` but with the correct `aVout` name subsititued at the appropriate place.
If the optional argument `Vector` is present and true, the vector architecture-friendly portions of this routine will be invoked.
N.B.: This routine will fail if the `aVout` is not initialized or if any of the specified attributes are not present in either `aVout` or `aVin`.

INTERFACE:

```
subroutine RCopyL_(aVin, aVout, rList, TrList,  vector)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

use m_List,          only : GetIndices => get_indices

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),             intent(in)    :: aVin
character(len=*),           intent(in)    :: rList
character(len=*), optional, intent(in)    :: TrList
logical, optional,          intent(in)    :: vector
```

OUTPUT PARAMETERS:

```
type(AttrVect),             intent(inout) :: aVout
```

REVISION HISTORY:

```
16Aug06 - R. Jacob <jacob@mcs.anl.gov> - initial version from breakup
          of Copy_.
```

2.1.27 ICopy_ - Copy Integer Attributes from One AttrVect to Another

This routine copies from input argment `aVin` into the output `AttrVect` argument `aVout` the shared integer attributes.
If the optional argument `Vector` is present and true, the vector architecture-friendly portions of this routine will be invoked.
If the optional argument `sharedIndices` is present, it should be the result of the call `SharedIndicesOneType_(aVin, aVout, 'INTEGER', sharedIndices)`. Providing this argument speeds up this routine substantially. For example, you can compute a `sharedIndices` structure once for a given pair of `AttrVect`s, then use that same structure for all copies between those two `AttrVect`s (although note that a different `sharedIndices` variable would be needed if `aVin` and `aVout` were reversed).
N.B.: This routine will fail if the `aVout` is not initialized.

INTERFACE:

```
subroutine ICopy_(aVin, aVout, vector, sharedIndices)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),             intent(in)    :: aVin
logical, optional,          intent(in)    :: vector
type(AVSharedIndicesOneType), optional, intent(in) :: sharedIndices
```

OUTPUT PARAMETERS:

```
type(AttrVect),             intent(inout) :: aVout
```

REVISION HISTORY:

```
16Aug06 - R. Jacob <jacob@mcs.anl.gov> - initial version.
28Apr11 - W.J. Sacks <sacks@ucar.edu> - added sharedIndices argument
```

2.1.28 ICopyL_ - Copy Specific Integer Attributes from One AttrVect to Another

This routine copies from input argment `aVin` into the output `AttrVect` argument `aVout` the integer attributes specified in input `CHARACTER` argument `iList`. The attributes can be listed in any order. If any attributes in `aVout` have different names but represent the the same quantity and should still be copied, you must provide a translation argument `TiList`. The translation arguments should be identical in length to the `iList` but with the correct `aVout` name subsititued at the appropriate place.
If the optional argument `Vector` is present and true, the vector architecture-friendly portions of this routine will be invoked.
N.B.: This routine will fail if the `aVout` is not initialized or if any of the specified attributes are not present in either `aVout` or `aVin`.

INTERFACE:

```
subroutine ICopyL_(aVin, aVout, iList, TiList, vector)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

use m_List,          only : GetIndices => get_indices

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),             intent(in)    :: aVin
character(len=*)          , intent(in)    :: iList
character(len=*), optional, intent(in)    :: TiList
logical, optional,          intent(in)    :: vector
```

OUTPUT PARAMETERS:

```
type(AttrVect),             intent(inout) :: aVout
```

REVISION HISTORY:

```
16Aug06 - R. Jacob <jacob@mcs.anl.gov> - initial version from breakup
          of Copy_.
```

2.1.29 Copy_ - Copy Real and Integer Attributes from One AttrVect to Another

This routine copies from input argment `aVin` into the output `AttrVect` argument `aVout` the real and integer attributes specified in input `CHARACTER` argument `iList` and `rList`. The attributes can be listed in any order. If neither `iList` nor `rList` are provided, all attributes shared between `aVin` and `aVout` will be copied.
If any attributes in `aVout` have different names but represent the the same quantity and should still be copied, you must provide a translation argument `TrList` and/or `TiList`. The translation arguments should be identical to the `rList` or `iList` but with the correct `aVout` name subsititued at the appropriate place.
This routines combines the functions of `RCopy_`, `RCopyL_`, `ICopy_` and `ICopyL_`. If you know you only want to copy real attributes, use the `RCopy` functions. If you know you only want to copy integer attributes, use the `ICopy` functions.
If the optional argument `Vector` is present and true, the vector architecture-friendly portions of this routine will be invoked.
If the optional argument `sharedIndices` is present, it should be the result of the call `SharedIndices_(aVin, aVout, sharedIndices)`. Providing this argument speeds up this routine substantially. For example, you can compute a `sharedIndices` structure once for a given pair of `AttrVect`s, then use that same structure for all copies between those two `AttrVect`s (although note that a different `sharedIndices` variable would be needed if `aVin` and `aVout` were reversed). Note, however, that `sharedIndices` is ignored if either `rList` or `iList` are given.
N.B.: This routine will fail if the `aVout` is not initialized or if any of the specified attributes are not present in either `aVout` or `aVin`.

INTERFACE:

```
subroutine Copy_(aVin, aVout, rList, TrList, iList, TiList, vector, sharedIndices)
```

USES:

```
use m_die ,          only : die, warn
use m_stdio ,        only : stderr

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),             intent(in)    :: aVin
character(len=*), optional, intent(in)    :: iList
character(len=*), optional, intent(in)    :: rList
character(len=*), optional, intent(in)    :: TiList
character(len=*), optional, intent(in)    :: TrList
logical, optional,          intent(in)    :: vector
type(AVSharedIndices), optional, intent(in) :: sharedIndices
```

OUTPUT PARAMETERS:

```
type(AttrVect),             intent(inout) :: aVout
```

REVISION HISTORY:

```
12Jun02 - R. Jacob <jacob@mcs.anl.gov> - initial version.
13Jun02 - R. Jacob <jacob@mcs.anl.gov> - copy shared attributes
          if no attribute lists are specified.
30Sep02 - R. Jacob <jacob@mcs.anl.gov> - new argument order with all
          optional arguments last
19Feb02 - E. Ong <eong@mcs.anl.gov> - new implementation using
          new list function get_indices and faster memory copy
28Oct03 - R. Jacob <jacob@mcs.anl.gov> - add optional vector
          argument to use vector-friendly code provided by Fujitsu
16Aug06 - R. Jacob <jacob@mcs.anl.gov> - split into 4 routines:
          RCopy_,RCopyL_,ICopy_,ICopyL_
28Apr11 - W.J. Sacks <sacks@ucar.edu> - added sharedIndices argument
```

2.1.30 Sort_ - Use Attributes as Keys to Generate an Index Permutation

The subroutine `Sort_()` uses a list of keys defined by the `List` `key_list`, searches for the appropriate integer or real attributes referenced by the items in `key_list` (that is, it identifies the appropriate entries in aV%iList and `aV%rList`), and then uses these keys to generate a permutation `perm` that will put the entries of the attribute vector `aV` in lexicographic order as defined by `key_list` (the ordering in `key_list` being from left to right.
N.B.: This routine will fail if `aV%iList` and `aV%rList` share one or more common entries.
N.B.: This routine will fail if one of the sorting keys presented is not present in `aV%iList` nor `aV%rList`.

INTERFACE:

```
subroutine Sort_(aV, key_list, perm, descend, perrWith, dieWith)
```

USES:

```
use m_String,        only : String
use m_String,        only : String_tochar => tochar
use m_String,        only : String_clean => clean
use m_List ,         only : List_allocated => allocated
use m_List ,         only : List_index => index
use m_List ,         only : List_nitem => nitem
use m_List ,         only : List_get   => get
use m_die ,          only : die
use m_stdio ,        only : stderr
use m_SortingTools , only : IndexSet
use m_SortingTools , only : IndexSort

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),                 intent(in) :: aV
type(List),                     intent(in) :: key_list
logical, dimension(:), optional, intent(in) :: descend
character(len=*),      optional, intent(in) :: perrWith
character(len=*),      optional, intent(in) :: dieWith
```

OUTPUT PARAMETERS:

```
integer, dimension(:),           pointer    :: perm
```

REVISION HISTORY:

```
20Oct00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
25Apr01 - R.L. Jacob <jacob@mcs.anl.gov> - add -1 to make a
          backwards loop go backwards
14Jun01 - J. Larson / E. Ong -- Fixed logic bug in REAL attribute
          sort (discovered by E. Ong), and cleaned up error /
          shutdown logic.
```

2.1.31 Permute_ - Permute AttrVect Elements

The subroutine `Permute_()` uses a a permutation `perm` (which can be generated by the routine `Sort_()` in this module) to rearrange the entries in the attribute integer and real storage areas of the input attribute vector `aV`–`aV%iAttr` and `aV%rAttr`, respectively.

INTERFACE:

```
subroutine Permute_(aV, perm, perrWith, dieWith)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr
use m_SortingTools , only : Permute

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:),           intent(in)    :: perm
character(len=*),      optional, intent(in)    :: perrWith
character(len=*),      optional, intent(in)    :: dieWith
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),                  intent(inout) :: aV
```

REVISION HISTORY:

```
23Oct00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

2.1.32 Unpermute_ - Unpermute AttrVect Elements

The subroutine `Unpermute_()` uses a a permutation `perm` (which can be generated by the routine `Sort_()` in this module) to rearrange the entries in the attribute integer and real storage areas of the input attribute vector `aV`–`aV%iAttr` and `aV%rAttr`, respectively. This is meant to be called on an `aV` that has already been permuted but it could also be used to perform the inverse operation implied by `perm` on an unpermuted `aV`.

INTERFACE:

```
subroutine Unpermute_(aV, perm, perrWith, dieWith)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr
use m_SortingTools , only : Unpermute

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:),           intent(in)    :: perm
character(len=*),      optional, intent(in)    :: perrWith
character(len=*),      optional, intent(in)    :: dieWith
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),                  intent(inout) :: aV
```

REVISION HISTORY:

23Nov05 - R. Jacob <jacob@mcs.anl.gov> - based on Permute

2.1.33 SortPermute_ - In-place Lexicographic Sort of an AttrVect

The subroutine `SortPermute_()` uses the routine `Sort_()` to create an index permutation `perm` that will place the AttrVect entries in the lexicographic order defined by the keys in the List variable `key_list`. This permutation is then used by the routine `Permute_()` to place the AttreVect entries in lexicographic order.

INTERFACE:

```
subroutine SortPermute_(aV, key_list, descend, perrWith, dieWith)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

implicit none
```

INPUT PARAMETERS:

```
type(List),                       intent(in)    :: key_list
logical , dimension(:), optional, intent(in)    :: descend
character(len=*),       optional, intent(in)    :: perrWith
character(len=*),       optional, intent(in)    :: dieWith
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),                   intent(inout) :: aV
```

REVISION HISTORY:

24Oct00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype

---

2.1.34 aVaVSharedAttrIndexList_ - AttrVect shared attributes.

aVaVSharedAttrIndexList_() takes a pair of user-supplied AttrVect variables aV1 and aV2, and for choice of either REAL or INTEGER attributes (as specified literally in the input CHARACTER argument attrib) returns the number of shared attributes NumShared, and arrays of indices Indices1 and Indices2 to their storage locations in aV1 and aV2, respectively.
N.B.: This routine returns two allocated arrays—Indices1(:) and Indices2(:)—which must be deallocated once the user no longer needs them. Failure to do this will create a memory leak.

INTERFACE:

```
subroutine aVaVSharedAttrIndexList_(aV1, aV2, attrib, NumShared, &
                                    Indices1, Indices2)
```

USES:

```
use m_stdio
use m_die,      only : MP_perr_die, die, warn

use m_List,     only : GetSharedListIndices

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),        intent(in)  :: aV1
type(AttrVect),        intent(in)  :: aV2
character(len=*),      intent(in)  :: attrib
```

OUTPUT PARAMETERS:

```
integer,               intent(out) :: NumShared
integer, dimension(:), pointer     :: Indices1
integer, dimension(:), pointer     :: Indices2
```

REVISION HISTORY:

```
07Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

2.1.35 SharedIndices_ - AttrVect shared attributes and auxiliary information

SharedIndices_() takes a pair of user-supplied AttrVect variables aV1 and aV2, and returns a structure of type AVSharedIndices (sharedIndices). This structure contains arrays of indices to the locations of the shared attributes, as well as auxiliary information. The structure contains information on both the REAL and INTEGER attributes. See documentation for the SharedIndicesOneType_ subroutine for some additional details, as much of the work is done there.
N.B.: The returned structure, sharedIndices, contains allocated arrays that must be deallocated once the user no longer needs them. This should be done through a call to cleanSharedIndices_.

INTERFACE:

```
subroutine SharedIndices_(aV1, aV2, sharedIndices)

   implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),        intent(in)  :: aV1
type(AttrVect),        intent(in)  :: aV2
```

INPUT/OUTPUT PARAMETERS:

```
type(AVSharedIndices), intent(inout) :: sharedIndices
```

REVISION HISTORY:

28Apr11 - W.J. Sacks <sacks@ucar.edu> - initial version

2.1.36 SharedIndicesOneType_ - AttrVect shared attributes and auxiliary information, for one data type

`SharedIndicesOneType_()` takes a pair of user-supplied `AttrVect` variables `aV1` and `aV2`, and for choice of either `REAL` or `INTEGER` attributes (as specified literally in the input `CHARACTER` argument `attrib`) returns a structure of type `AVSharedIndicesOneType` (`sharedIndices`). This structure contains arrays of indices to the locations of the shared attributes of the given type, as well as auxiliary information.
The `aVindices1` and `aVindices2` components of `sharedIndices` will be indices into `aV1` and `aV2`, respectively.
N.B.: The returned structure, `sharedIndices`, contains allocated arrays that must be deallocated once the user no longer needs them. This should be done through a call to `cleanSharedIndicesOneType_`.
Even if there are no attributes in common between `aV1` and `aV2`, `sharedIndices` will still be initialized, and memory will still be allocated. Furthermore, if an already-initialized `sharedIndices` variable is to be given new values, `cleanSharedIndicesOneType_` must be called before `SharedIndicesOneType_` is called a second time, in order to prevent a memory leak.

INTERFACE:

```
subroutine SharedIndicesOneType_(aV1, aV2, attrib, sharedIndices)

   implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),        intent(in)  :: aV1
type(AttrVect),        intent(in)  :: aV2
character(len=*),      intent(in)  :: attrib
```

INPUT/OUTPUT PARAMETERS:

```
type(AVSharedIndicesOneType), intent(inout) :: sharedIndices
```

REVISION HISTORY:

28Apr11 - W.J. Sacks <sacks@ucar.edu> - initial version

2.1.37 cleanSharedIndices_ - Deallocate allocated memory structures of an AVSharedIndices structure

This routine deallocates the allocated memory structures of the input/output `AVSharedIndicesOneType` argument `sharedIndices`, if they are currently associated. It also resets other components of this structure to a default state. The success (failure) of this operation is signified by a zero (non-zero)

value of the optional INTEGER output argument stat. If clean_() is invoked without supplying stat, and any of the deallocation operations fail, the routine will terminate with an error message. If multiple errors occur, stat will give the error condition for the last error.

INTERFACE:

```
subroutine cleanSharedIndices_(sharedIndices, stat)

   implicit none
```

INPUT/OUTPUT PARAMETERS:

```
      type(AVSharedIndices), intent(inout) :: sharedIndices
```

OUTPUT PARAMETERS:

```
      integer, optional, intent(out)       :: stat
```

REVISION HISTORY:

```
28Apr11 - W.J. Sacks <sacks@ucar.edu> - initial version
```

2.1.38 cleanSharedIndicesOneType_ - Deallocate allocated memory structures of an AVSharedIndicesOneType structure

This routine deallocates the allocated memory structures of the input/output AVSharedIndices argument sharedIndices, if they are currently associated. It also resets other components of this structure to a default state. The success (failure) of this operation is signified by a zero (non-zero) value of the optional INTEGER output argument stat. If clean_() is invoked without supplying stat, and any of the deallocation operations fail, the routine will terminate with an error message. If multiple errors occur, stat will give the error condition for the last error.

INTERFACE:

```
subroutine cleanSharedIndicesOneType_(sharedIndices, stat)
```

USES:

```
      use m_die,      only : die

    implicit none
```

INPUT/OUTPUT PARAMETERS:

```
      type(AVSharedIndicesOneType), intent(inout) :: sharedIndices
```

OUTPUT PARAMETERS:

```
      integer, optional, intent(out)       :: stat
```

REVISION HISTORY:

```
28Apr11 - W.J. Sacks <sacks@ucar.edu> - initial version
```

2.2 Module m_AttrVectComms - MPI Communications Methods for the AttrVect (Source File: m_AttrVectComms.F90)

This module defines the communications methods for the `AttrVect` datatype (see the module `m_AttrVect` for more information about this class and its methods). MCT's communications are implemented in terms of the Message Passing Interface (MPI) standard, and we have as best as possible, made the interfaces to these routines appear as similar as possible to the corresponding MPI routines. For the `AttrVect`, we supply *blocking* point-to-point send and receive operations. We also supply the following collective operations: broadcast, gather, and scatter. The gather and scatter operations rely on domain decomposition descriptors that are defined elsewhere in MCT: the `GlobalMap`, which is a one-dimensional decomposition (see the MCT module `m_GlobalMap` for more details); and the `GlobalSegMap`, which is a segmented decomposition capable of supporting multidimensional domain decompositions (see the MCT module `m_GlobalSegMap` for more details).

INTERFACE:

```
module m_AttrVectComms
```

USES:

```
      use m_AttrVect ! AttrVect class and its methods

      implicit none

      private ! except

      public :: gather ! gather all local vectors to the root
      public :: scatter ! scatter from the root to all PEs
      public :: bcast ! bcast from root to all PEs
      public :: send ! send an AttrVect
      public :: recv ! receive an AttrVect

    interface gather ; module procedure &
       GM_gather_, &
       GSM_gather_
    end interface
    interface scatter ; module procedure &
       GM_scatter_, &
       GSM_scatter_
    end interface
    interface bcast  ; module procedure bcast_  ; end interface
    interface send  ; module procedure send_  ; end interface
    interface recv  ; module procedure recv_  ; end interface
```

REVISION HISTORY:

```
  27Oct00 - J.W. Larson <larson@mcs.anl.gov> - relocated routines
            from m_AttrVect to create this module.
  15Jan01 - J.W. Larson <larson@mcs.anl.gov> - Added APIs for
            GSM_gather_() and GSM_scatter_().
   9May01 - J.W. Larson <larson@mcs.anl.gov> - Modified GM_scatter_
            so its communication model agrees with MPI_scatter().
            Also tidied up prologues in all module routines.
   7Jun01 - J.W. Larson <larson@mcs.anl.gov> - Added send()
            and recv().
   3Aug01 - E.T. Ong <eong@mcs.anl.gov> - in GSM_scatter, call
            GlobalMap_init with actual shaped array to satisfy
            Fortran 90 standard. See comment in subroutine.
  23Aug01 - E.T. Ong <eong@mcs.anl.gov> - replaced assignment(=)
            with copy for list type to avoid compiler bugs in pgf90.
```

```
          Added more error checking in gsm scatter. Fixed minor bugs
         in gsm and gm gather.
13Dec01 - E.T. Ong <eong@mcs.anl.gov> - GSM_scatter, allow users
          to scatter with a haloed GSMap. Fixed some bugs in
          GM_scatter.
19Dec01 - E.T. Ong <eong@mcs.anl.gov> - allow bcast of an AttrVect
          with only an integer or real attribute.
27Mar02 - J.W. Larson <larson@mcs.anl.gov> - Corrected usage of
          m_die routines throughout this module.
```

2.2.1 send_ - Point-to-point Send of an AttrVect

This routine takes an input `AttrVect` argument `inAV` and sends it to processor `dest` on the communicator associated with the Fortran `INTEGER` MPI communicator handle `comm`. The overalll message is tagged by the input `INTEGER` argument `TagBase`. The success (failure) of this operation is reported in the zero (nonzero) optional output argument `status`.
N.B.: One must avoid assigning elsewhere the MPI tag values between `TagBase` and `TagBase+7`, inclusive. This is because `send_()` performs the send of the `AttrVect` as a series of eight send operations.

INTERFACE:

```
subroutine send_(inAV, dest, TagBase, comm, status)
```

USES:

```
use m_stdio
use m_mpif90
use m_die

use m_List, only : List
use m_List, only : List_allocated => allocated
use m_List, only : List_nitem => nitem
use m_List, only : List_send => send

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),     intent(in)  :: inAV
integer,            intent(in)  :: dest
integer,            intent(in)  :: TagBase
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
integer, optional,  intent(out) :: status
```

REVISION HISTORY:

```
 7Jun01 - J.W. Larson - initial version.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize status
          (if present).
```

2.2.2 recv_ - Point-to-point Receive of an AttrVect

This routine receives the output `AttrVect` argument `outAV` from processor `source` on the communicator associated with the Fortran `INTEGER` MPI communicator handle `comm`. The overall message is tagged by the input `INTEGER` argument `TagBase`. The success (failure) of this operation is reported in the zero (nonzero) optional output argument `status`.
N.B.: One must avoid assigning elsewhere the MPI tag values between `TagBase` and `TagBase+7`, inclusive. This is because `recv_()` performs the receive of the `AttrVect` as a series of eight receive operations.

INTERFACE:

```
subroutine recv_(outAV, dest, TagBase, comm, status)
```

USES:

```
use m_stdio
use m_mpif90
use m_die

use m_List, only : List
use m_List, only : List_nitem => nitem
use m_List, only : List_recv => recv

use m_AttrVect, only : AttrVect

implicit none
```

INPUT PARAMETERS:

```
integer,            intent(in)  :: dest
integer,            intent(in)  :: TagBase
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),     intent(out) :: outAV
integer, optional,  intent(out) :: status
```

REVISION HISTORY:

```
 7Jun01 - J.W. Larson - initial working version.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize status
          (if present).
```

2.2.3 GM_gather_ - Gather an AttrVect Distributed by a GlobalMap

This routine gathers a *distributed* `AttrVect iV` to the `root` process, and returns it in the output `AttrVect` argument `oV`. The decomposition of `iV` is described by the input `GlobalMap` argument `GMap`. The input `INTEGER` argument `comm` is the Fortran integer MPI communicator handle. The success (failure) of this operation corresponds to a zero (nonzero) value of the optional output `INTEGER` argument `stat`.

INTERFACE:

```
subroutine GM_gather_(iV, oV, GMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90
use m_realkinds, only : FP
use m_GlobalMap, only : GlobalMap
use m_GlobalMap, only : GlobalMap_lsize => lsize
use m_GlobalMap, only : GlobalMap_gsize => gsize
use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nIAttr => nIAttr
use m_AttrVect, only : AttrVect_nRAttr => nRAttr
use m_AttrVect, only : AttrVect_clean => clean
use m_FcComms,  only : fc_gatherv_int, fc_gatherv_fp

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),           intent(in)  :: iV
type(GlobalMap),          intent(in)  :: GMap
integer,                  intent(in)  :: root
integer,                  intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),           intent(out) :: oV
integer,        optional, intent(out) :: stat
```

REVISION HISTORY:

```
15Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
27Oct00 - J.W. Larson <larson@mcs.anl.gov> - relocated from
          m_AttrVect
15Jan01 - J.W. Larson <larson@mcs.anl.gov> - renamed GM_gather_
 9May01 - J.W. Larson <larson@mcs.anl.gov> - tidied up prologue
18May01 - R.L. Jacob <jacob@mcs.anl.gov> - use MP_Type function
          to determine type for mpi_gatherv
31Jan09 - P.H. Worley <worleyph@ornl.gov> - replaced call to
          MPI_gatherv with call to flow controlled gather routines
```

2.2.4 GSM_gather_ - Gather an AttrVect Distributed by a GlobalSegMap

The routine `GSM_gather_()` takes a distributed input `AttrVect` argument `iV`, whose decomposition is described by the input `GlobalSegMap` argument `GSMap`, and gathers it to the output `AttrVect` argument `oV`. The gathered `AttrVect oV` is valid only on the root process specified by the input argument `root`. The communicator used to gather the data is specified by the argument `comm`. The success (failure) is reported in the zero (non-zero) value of the output argument `stat`.
`GSM_gather_()` converts the problem of gathering data according to a `GlobalSegMap` into the simpler problem of gathering data as specified by a `GlobalMap`. The `GlobalMap` variable `GMap` is created based on the local storage requirements for each distributed piece of `iV`. On the root, a complete (including halo points) gathered copy of `iV` is collected into the temporary `AttrVect` variable `workV` (the length of `workV` is the larger of `GlobalSegMap_GlobalStorage(GSMap)` or `GlobalSegMap_GlobalSize(GSMap)`). The variable `workV` is segmented by process, and segments are copied into it by process, but ordered in the same order the segments appear in `GSMap`. Once

workV is loaded, the data are copied segment-by-segment to their appropriate locations in the output AttrVect oV.

INTERFACE:

```
subroutine GSM_gather_(iV, oV, GSMap, root, comm, stat, rdefault, idefault)
```

USES:

```
Message-passing environment utilities (mpeu) modules:
    use m_stdio
    use m_die
    use m_mpif90
    use m_realkinds, only: FP
GlobalSegMap and associated services:
    use m_GlobalSegMap, only : GlobalSegMap
    use m_GlobalSegMap, only : GlobalSegMap_comp_id => comp_id
    use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg
    use m_GlobalSegMap, only : GlobalSegMap_lsize => lsize
    use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize
    use m_GlobalSegMap, only : GlobalSegMap_haloed => haloed
    use m_GlobalSegMap, only : GlobalSegMap_GlobalStorage => GlobalStorage
AttrVect and associated services:
    use m_AttrVect, only : AttrVect
    use m_AttrVect, only : AttrVect_init => init
    use m_AttrVect, only : AttrVect_zero => zero
    use m_AttrVect, only : AttrVect_lsize => lsize
    use m_AttrVect, only : AttrVect_nIAttr => nIAttr
    use m_AttrVect, only : AttrVect_nRAttr => nRAttr
    use m_AttrVect, only : AttrVect_clean => clean
GlobalMap and associated services:
    use m_GlobalMap, only : GlobalMap
    use m_GlobalMap, only : GlobalMap_init => init
    use m_GlobalMap, only : GlobalMap_clean => clean

    implicit none
```

INPUT PARAMETERS:

```
    type(AttrVect),            intent(in)  :: iV
    type(GlobalSegMap),        intent(in)  :: GSMap
    integer,                   intent(in)  :: root
    integer,                   intent(in)  :: comm
    real(FP), optional,        intent(in)  :: rdefault
    integer, optional,         intent(in)  :: idefault
```

OUTPUT PARAMETERS:

```
    type(AttrVect),            intent(out) :: oV
    integer,        optional,  intent(out) :: stat
```

REVISION HISTORY:

```
15Jan01 - J.W. Larson <larson@mcs.anl.gov> - API specification.
25Feb01 - J.W. Larson <larson@mcs.anl.gov> - Prototype code.
26Apr01 - R.L. Jacob <jacob@mcs.anl.gov> - add use statement for
          AttVect_clean
 9May01 - J.W. Larson <larson@mcs.anl.gov> - tidied up prologue
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize stat
```

```
          (if present).
  20Aug01 - E.T. Ong <eong@mcs.anl.gov> - Added error checking for
          matching processors in gsmap and comm. Corrected
          current_pos assignment.
  23Nov01 - R. Jacob <jacob@mcs.anl.gov> - zero the oV before copying in
          gathered data.
  27Jul07 - R. Loy <rloy@mcs.anl.gov> - add Tony's suggested improvement
          for a default value in the output AV
  11Aug08 - R. Jacob <jacob@mcs.anl.gov> - add Pat Worley's faster way
          to initialize lns
```

2.2.5 GM_scatter_ - Scatter an AttrVect Using a GlobalMap

The routine `GM_scatter_` takes an input `AttrVect` type `iV` (valid only on the root), and scatters it to a distributed `AttrVect oV`. The input `GlobalMap` argument `GMap` dictates how `iV` is scattered to `oV`. The success (failure) of this routine is reported in the zero (non-zero) value of the output argument `stat`.
N.B.: The output `AttrVect` argument `oV` represents dynamically allocated memory. When it is no longer needed, it should be deallocated by invoking `AttrVect_clean()` (see the module `m_AttrVect` for more details).

INTERFACE:

```
subroutine GM_scatter_(iV, oV, GMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90
use m_realkinds, only : FP

use m_List, only : List
use m_List, only : List_copy => copy
use m_List, only : List_bcast => bcast
use m_List, only : List_clean => clean
use m_List, only : List_nullify => nullify
use m_List, only : List_nitem => nitem

use m_GlobalMap, only : GlobalMap
use m_GlobalMap, only : GlobalMap_lsize => lsize
use m_GlobalMap, only : GlobalMap_gsize => gsize

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nIAttr => nIAttr
use m_AttrVect, only : AttrVect_nRAttr => nRAttr
use m_AttrVect, only : AttrVect_clean => clean

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),           intent(in)  :: iV
type(GlobalMap),          intent(in)  :: GMap
integer,                  intent(in)  :: root
```

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

OUTPUT PARAMETERS:

```
type(AttrVect),           intent(out) :: oV
integer,        optional, intent(out) :: stat
```

REVISION HISTORY:

```
21Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
27Oct00 - J.W. Larson <larson@mcs.anl.gov> - relocated from
          m_AttrVect
15Jan01 - J.W. Larson <larson@mcs.anl.gov> - renamed  GM_scatter_
 8Feb01 - J.W. Larson <larson@mcs.anl.gov> - add logic to prevent
          empty calls (i.e. no data in buffer) to MPI_SCATTERV()
27Apr01 - R.L. Jacob <jacob@mcs.anl.gov> - small bug fix to
          integer attribute scatter
 9May01 - J.W. Larson <larson@mcs.anl.gov> - Re-vamped comms model
          to reflect MPI comms model for the scatter.  Tidied up
          the prologue, too.
18May01 - R.L. Jacob <jacob@mcs.anl.gov> - use MP_Type function
          to determine type for mpi_scatterv
 8Aug01 - E.T. Ong <eong@mcs.anl.gov> - replace list assignment(=)
          with list copy to avoid compiler errors in pgf90.
13Dec01 - E.T. Ong <eong@mcs.anl.gov> - allow scatter with an
          AttrVect containing only an iList or rList.
```

2.2.6 GSM_scatter_ - Scatter an AttrVect using a GlobalSegMap

The routine `GSM_scatter_` takes an input `AttrVect` type `iV` (valid only on the root), and scatters it to a distributed `AttrVect` `oV`. The input `GlobalSegMap` argument `GSMap` dictates how `iV` is scattered to `oV`. The success (failure) of this routine is reported in the zero (non-zero) value of the output argument `stat`.
`GSM_scatter_()` converts the problem of scattering data according to a `GlobalSegMap` into the simpler problem of scattering data as specified by a `GlobalMap`. The `GlobalMap` variable `GMap` is created based on the local storage requirements for each distributed piece of `iV`. On the root, a complete (including halo points) copy of `iV` is stored in the temporary `AttrVect` variable `workV` (the length of `workV` is `GlobalSegMap_GlobalStorage(GSMap)`). The variable `workV` is segmented by process, and segments are copied into it by process, but ordered in the same order the segments appear in `GSMap`. Once `workV` is loaded, the data are scattered to the output `AttrVect` `oV` by a call to the routine `GM_scatter_()` defined in this module, with `workV` and `GMap` as the input arguments.
N.B.: This algorithm assumes that memory access times are much shorter than message-passing transmission times.
N.B.: The output `AttrVect` argument `oV` represents dynamically allocated memory. When it is no longer needed, it should be deallocated by invoking `AttrVect_clean()` (see the module `m_AttrVect` for more details).

INTERFACE:

```
subroutine GSM_scatter_(iV, oV, GSMap, root, comm, stat)
```

USES:

```
Environment utilities from mpeu:

    use m_stdio
    use m_die
```

```
    use m_mpif90

    use m_List, only : List_nullify => nullify

  GlobalSegMap and associated services:
    use m_GlobalSegMap, only : GlobalSegMap
    use m_GlobalSegMap, only : GlobalSegMap_comp_id => comp_id
    use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg
    use m_GlobalSegMap, only : GlobalSegMap_lsize => lsize
    use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize
    use m_GlobalSegMap, only : GlobalSegMap_GlobalStorage => GlobalStorage
  AttrVect and associated services:
    use m_AttrVect, only : AttrVect
    use m_AttrVect, only : AttrVect_init => init
    use m_AttrVect, only : AttrVect_zero => zero
    use m_AttrVect,  only : AttrVect_lsize => lsize
    use m_AttrVect, only : AttrVect_nIAttr => nIAttr
    use m_AttrVect, only : AttrVect_nRAttr => nRAttr
    use m_AttrVect, only : AttrVect_clean => clean
  GlobalMap and associated services:
    use m_GlobalMap, only : GlobalMap
    use m_GlobalMap, only : GlobalMap_init => init
    use m_GlobalMap, only : GlobalMap_clean => clean

    implicit none
```

INPUT PARAMETERS:

```
    type(AttrVect),            intent(in)  :: iV
    type(GlobalSegMap),        intent(in)  :: GSMap
    integer,                   intent(in)  :: root
    integer,                   intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
    type(AttrVect),            intent(out) :: oV
    integer,        optional,  intent(out) :: stat
```

REVISION HISTORY:

```
  15Jan01 - J.W. Larson <larson@mcs.anl.gov> - API specification.
   8Feb01 - J.W. Larson <larson@mcs.anl.gov> - Initial code.
  25Feb01 - J.W. Larson <larson@mcs.anl.gov> - Bug fix--replaced
            call to GlobalSegMap_lsize with call to the new fcn.
            GlobalSegMap_ProcessStorage().
  26Apr01 - R.L. Jacob <jacob@mcs.anl.gov> - add use statement for
            AttVect_clean
  26Apr01 - J.W. Larson <larson@mcs.anl.gov> - bug fixes--data
            misalignment in use of the GlobalMap to compute the
            memory map into workV, and initialization of workV
            on all processes.
   9May01 - J.W. Larson <larson@mcs.anl.gov> - tidied up prologue
  15May01 - Larson / Jacob <larson@mcs.anl.gov> - stopped initializing
            workV on off-root processes (no longer necessary).
  13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize stat
            (if present).
  20Jun01 - J.W. Larson <larson@mcs.anl.gov> - Fixed a subtle bug
            appearing on AIX regarding the fact workV is uninitial-
```

```
          ized on non-root processes.  This is fixed by nullifying
          all the pointers in workV for non-root processes.
  20Aug01 - E.T. Ong <eong@mcs.anl.gov> - Added argument check
          for matching processors in gsmap and comm.
  13Dec01 - E.T. Ong <eong@mcs.anl.gov> - got rid of restriction
          GlobalStorage(GSMap)==AttrVect_lsize(AV) to allow for
          GSMap to be haloed.
  11Aug08 - R. Jacob <jacob@mcs.anl.gov> - remove call to ProcessStorage
          and replace with faster algorithm provided by Pat Worley
```

2.2.7 bcast_ - Broadcast an AttrVect

This routine takes an `AttrVect` argument `aV` (at input, valid on the root only), and broadcasts it to all the processes associated with the communicator handle `comm`. The success (failure) of this routine is reported in the zero (non-zero) value of the output argument `stat`.
N.B.: The output (on non-root processes) `AttrVect` argument `aV` represents dynamically allocated memory. When it is no longer needed, it should be deallocated by invoking `AttrVect_clean()` (see the module `m_AttrVect` for details).

INTERFACE:

```
subroutine bcast_(aV, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90
use m_String, only : String,bcast,char,String_clean
use m_String, only : String_bcast => bcast
use m_List, only : List_get => get
use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nIAttr => nIAttr
use m_AttrVect, only : AttrVect_nRAttr => nRAttr

implicit none
```

INPUT PARAMETERS:

```
integer,                  intent(in)    :: root
integer,                  intent(in)    :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),           intent(inout) :: aV ! (IN) on the root,
                                              ! (OUT) elsewhere
```

OUTPUT PARAMETERS:

```
integer,        optional, intent(out)   :: stat
```

REVISION HISTORY:

```
27Apr98 - Jing Guo <guo@thunder> - initial prototype/prologue/code
27Oct00 - J.W. Larson <larson@mcs.anl.gov> - relocated from
          m_AttrVect
 9May01 - J.W. Larson <larson@mcs.anl.gov> - tidied up prologue
18May01 - R.L. Jacob <jacob@mcs.anl.gov> - use MP_Type function
          to determine type for bcast
19Dec01 - E.T. Ong <eong@mcs.anl.gov> - adjusted for case of AV with
          only integer or real attribute
```

2.3 Module m_AttrVectReduce - Local/Distributed AttrVect Reduction Ops. (Source File: m_AttrVectReduce.F90)

This module provides routines to perform reductions on the `AttrVect` datatype. These reductions can either be the types of operations supported by MPI (currently, summation, minimum and maximum are available) that are applied either to all the attributes (both integer and real), or specific reductions applicable only to the real attributes of an `AttrVect`. This module provides services for both local (i.e., one address space) and global (distributed) reductions. The type of reduction is defined through use of one of the public data members of this module:

Value	Action
`AttrVectSUM`	Sum
`AttrVectMIN`	Minimum
`AttrVectMAX`	Maximum

INTERFACE:

```
module m_AttrVectReduce
```

USES:

```
No modules are used in the declaration section of this module.

implicit none

private ! except
```

PUBLIC MEMBER FUNCTIONS:

```
public :: LocalReduce            ! Local reduction of all attributes
public :: LocalReduceRAttr       ! Local reduction of REAL attributes
public :: AllReduce              ! AllReduce for distributed AttrVect
public :: GlobalReduce           ! Local Reduce followed by AllReduce
public :: LocalWeightedSumRAttr  ! Local weighted sum of
                                 ! REAL attributes
public :: GlobalWeightedSumRAttr ! Global weighted sum of REAL
                                 ! attributes for a distrubuted
                                 ! AttrVect

interface LocalReduce ; module procedure LocalReduce_ ; end interface
interface LocalReduceRAttr
   module procedure LocalReduceRAttr_
end interface
interface AllReduce
   module procedure AllReduce_
end interface
interface GlobalReduce
   module procedure GlobalReduce_
end interface
interface LocalWeightedSumRAttr; module procedure &
   LocalWeightedSumRAttrSP_, &
   LocalWeightedSumRAttrDP_
end interface
interface GlobalWeightedSumRAttr; module procedure &
   GlobalWeightedSumRAttrSP_, &
   GlobalWeightedSumRAttrDP_
end interface
```

PUBLIC DATA MEMBERS:

```
public :: AttrVectSUM
public :: AttrVectMIN
public :: AttrVectMAX

integer, parameter :: AttrVectSUM = 1
integer, parameter :: AttrVectMIN = 2
integer, parameter :: AttrVectMAX = 3
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - Created module
         using routines originally prototyped in m_AttrVect.
```

2.3.1 LocalReduce_ - Local Reduction of INTEGER and REAL Attributes

The subroutine `LocalReduce_()` takes the input `AttrVect` argument `inAV`, and reduces each of its integer and real attributes, returning them in the output `AttrVect` argument `outAV` (which is created by this routine). The type of reduction is defined by the input `INTEGER` argument `action`. Allowed values for action are defined as public data members to this module, and are summarized below:

Value	Action
`AttrVectSUM`	Sum
`AttrVectMIN`	Minimum
`AttrVectMAX`	Maximum

N.B.: The output `AttrVect` argument `outAV` is allocated memory, and must be destroyed by invoking the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine LocalReduce_(inAV, outAV, action)
```

USES:

```
use m_realkinds,     only : FP
use m_die ,          only : die
use m_stdio ,        only : stderr
use m_AttrVect,      only : AttrVect
use m_AttrVect,      only : AttrVect_init => init
use m_AttrVect,      only : AttrVect_zero => zero
use m_AttrVect,      only : AttrVect_nIAttr => nIAttr
use m_AttrVect,      only : AttrVect_nRAttr => nRAttr
use m_AttrVect,      only : AttrVect_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),  intent(IN)  :: inAV
integer,         intent(IN)  :: action
```

OUTPUT PARAMETERS:

```
type(AttrVect),  intent(OUT) :: outAV
```

REVISION HISTORY:

```
16Apr02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

2.3.2 LocalReduceRAttr_ - Local Reduction of REAL Attributes

The subroutine `LocalReduceRAttr_()` takes the input `AttrVect` argument `inAV`, and reduces each of its `REAL` attributes, returning them in the output `AttrVect` argument `outAV` (which is created by this routine). The type of reduction is defined by the input `INTEGER` argument `action`. Allowed values for action are defined as public data members to this module (see the declaration section of `m_AttrVect`, and are summarized below:

Value	Action
`AttrVectSUM`	Sum
`AttrVectMIN`	Minimum
`AttrVectMAX`	Maximum

N.B.: The output `AttrVect` argument `outAV` is allocated memory, and must be destroyed by invoking the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine LocalReduceRAttr_(inAV, outAV, action)
```

USES:

```
use m_realkinds,     only : FP

use m_die ,          only : die
use m_stdio ,        only : stderr

use m_List,          only : List
use m_List,          only : List_copy => copy
use m_List,          only : List_exportToChar => exportToChar
use m_List,          only : List_clean => clean

use m_AttrVect,      only : AttrVect
use m_AttrVect,      only : AttrVect_init => init
use m_AttrVect,      only : AttrVect_zero => zero
use m_AttrVect,      only : AttrVect_nIAttr => nIAttr
use m_AttrVect,      only : AttrVect_nRAttr => nRAttr
use m_AttrVect,      only : AttrVect_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(IN)  :: inAV
integer,                     intent(IN)  :: action
```

OUTPUT PARAMETERS:

```
    type(AttrVect),               intent(OUT) :: outAV
```

REVISION HISTORY:

```
 16Apr02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
  6May02 - J.W. Larson <larson@mcs.anl.gov> - added optional
           argument weights(:)
  8May02 - J.W. Larson <larson@mcs.anl.gov> - modified interface
           to return it to being a pure reduction operation.
  9May02 - J.W. Larson <larson@mcs.anl.gov> - renamed from
           LocalReduceReals_() to LocalReduceRAttr_() to make
           the name more consistent with other module procedure
           names in this module.
```

2.3.3 AllReduce_ - Reduction of INTEGER and REAL Attributes

The subroutine `AllReduce_()` takes the distributed input `AttrVect` argument `inAV`, and performs a global reduction of all its attributes across the MPI communicator associated with the Fortran90 `INTEGER` handle `comm`, and returns these reduced values to all processes in the `AttrVect` argument `outAV` (which is created by this routine). The reduction operation is specified by the user, and must have one of the values listed in the table below:

Value	Action
`AttrVectSUM`	Sum
`AttrVectMIN`	Minimum
`AttrVectMAX`	Maximum

N.B.: The output `AttrVect` argument `outAV` is allocated memory, and must be destroyed by invoking the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
  subroutine AllReduce_(inAV, outAV, ReductionOp, comm, ierr)
```

USES:

```
    use m_die
    use m_stdio ,        only : stderr
    use m_mpif90

    use m_List,          only : List
    use m_List,          only : List_exportToChar => exportToChar
    use m_List,          only : List_allocated => allocated

    use m_AttrVect,      only : AttrVect
    use m_AttrVect,      only : AttrVect_init => init
    use m_AttrVect,      only : AttrVect_zero => zero
    use m_AttrVect,      only : AttrVect_lsize => lsize
    use m_AttrVect,      only : AttrVect_nIAttr => nIAttr
    use m_AttrVect,      only : AttrVect_nRAttr => nRAttr

    implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(IN)  :: inAV
integer,                     intent(IN)  :: ReductionOp
integer,                     intent(IN)  :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),              intent(OUT) :: outAV
integer,        optional,    intent(OUT) :: ierr
```

REVISION HISTORY:

```
8May02 - J.W. Larson <larson@mcs.anl.gov> - initial version.
9Jul02 - J.W. Larson <larson@mcs.anl.gov> - slight modification;
         use List_allocated() to determine if there is attribute
         data to be reduced (this patch is to support the Sun
         F90 compiler).
```

2.3.4 GlobalReduce_ - Reduction of INTEGER and REAL Attributes

The subroutine `GlobalReduce_()` takes the distributed input `AttrVect` argument `inAV`, and performs a local reduction of all its integer and real attributes, followed by a an `AllReduce` of all the result of the local reduction across the MPI communicator associated with the Fortran90 `INTEGER` handle `comm`, and returns these reduced values to all processes in the `AttrVect` argument `outAV` (which is created by this routine). The reduction operation is specified by the user, and must have one of the values listed in the table below:

Value	Action
`AttrVectSUM`	Sum
`AttrVectMIN`	Minimum
`AttrVectMAX`	Maximum

N.B.: The output `AttrVect` argument `outAV` is allocated memory, and must be destroyed by invoking the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine GlobalReduce_(inAV, outAV, ReductionOp, comm, ierr)
```

USES:

```
use m_die
use m_stdio ,        only : stderr
use m_mpif90

use m_AttrVect,      only : AttrVect
use m_AttrVect,      only : AttrVect_clean => clean

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(IN)  :: inAV
integer,                     intent(IN)  :: ReductionOp
integer,                     intent(IN)  :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),               intent(OUT) :: outAV
integer,        optional,     intent(OUT) :: ierr
```

REVISION HISTORY:

6May03 - J.W. Larson <larson@mcs.anl.gov> - initial version.

2.3.5 LocalWeightedSumRAttrSP_ - Local Weighted Sum of REAL Attributes

The subroutine `LocalWeightedSumRAttr_()` takes the input `AttrVect` argument `inAV`, and performs a weighted sum of each of its `REAL` attributes, returning them in the output `AttrVect` argument `outAV` (which is created by this routine and will contain *no* integer attributes). The weights used for the summation are provided by the user in the input argument `Weights(:)`. If the sum of the weights is desired, this can be returned as an attribute in `outAV` if the optional `CHARACTER` argument `WeightSumAttr` is provided (which will be concatenated onto the list of real attributes in `inAV`).
N.B.: The argument `WeightSumAttr` must not be identical to any of the real attribute names in `inAV`.
N.B.: The output `AttrVect` argument `outAV` is allocated memory, and must be destroyed by invoking the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine LocalWeightedSumRAttrSP_(inAV, outAV, Weights, WeightSumAttr)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr
use m_realkinds,     only : SP, FP

use m_List,          only : List
use m_List,          only : List_init => init
use m_List,          only : List_clean => clean
use m_List,          only : List_exportToChar => exportToChar
use m_List,          only : List_concatenate => concatenate

use m_AttrVect,      only : AttrVect
use m_AttrVect,      only : AttrVect_init => init
use m_AttrVect,      only : AttrVect_zero => zero
use m_AttrVect,      only : AttrVect_nIAttr => nIAttr
use m_AttrVect,      only : AttrVect_nRAttr => nRAttr
use m_AttrVect,      only : AttrVect_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),               intent(IN)  :: inAV
real(SP), dimension(:),       pointer     :: Weights
character(len=*),   optional, intent(IN)  :: WeightSumAttr
```

OUTPUT PARAMETERS:

```
type(AttrVect),              intent(OUT) :: outAV
```

REVISION HISTORY:

```
 8May02 - J.W. Larson <larson@mcs.anl.gov> - initial version.
14Jun02 - J.W. Larson <larson@mcs.anl.gov> - bug fix regarding
          accumulation of weights when invoked with argument
          weightSumAttr.  Now works in MCT unit tester.
```

2.3.6 GlobalWeightedSumRAttrSP_ - Global Weighted Sum of REAL Attributes

The subroutine `GlobalWeightedSumRAttr_()` takes the distributed input `AttrVect` argument `inAV`, and performs a weighted global sum across the MPI communicator associated with the Fortran90 `INTEGER` handle `comm` of each of its `REAL` attributes, returning the sums to each process in the `AttrVect` argument `outAV` (which is created by this routine and will contain *no* integer attributes). The weights used for the summation are provided by the user in the input argument `weights(:)`. If the sum of the weights is desired, this can be returned as an attribute in `outAV` if the optional `CHARACTER` argument `WeightSumAttr` is provided (which will be concatenated onto the list of real attributes in `inAV` to form the list of real attributes for `outAV`).
N.B.: The argument `WeightSumAttr` must not be identical to any of the real attribute names in `inAV`.
N.B.: The output `AttrVect` argument `outAV` is allocated memory, and must be destroyed by invoking the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine GlobalWeightedSumRAttrSP_(inAV, outAV, Weights, comm, &
                                     WeightSumAttr)
```

USES:

```
use m_die
use m_stdio ,        only : stderr
use m_mpif90
use m_realkinds,     only : SP

use m_List,          only : List
use m_List,          only : List_exportToChar => exportToChar

use m_AttrVect,      only : AttrVect
use m_AttrVect,      only : AttrVect_clean => clean
use m_AttrVect,      only : AttrVect_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(IN)  :: inAV
real(SP), dimension(:),      pointer     :: Weights
integer,                     intent(IN)  :: comm
character(len=*),   optional, intent(IN)  :: WeightSumAttr
```

OUTPUT PARAMETERS:

```
type(AttrVect),              intent(OUT) :: outAV
```

REVISION HISTORY:

8May02 - J.W. Larson <larson@mcs.anl.gov> - initial version.

3 Global Segment Map

3.1 Module m_GlobalSegMap - a nontrivial 1-D decomposition of an array. (Source File: m_GlobalSegMap.F90)

Consider the problem of the 1-dimensional decomposition of an array across multiple processes. If each process owns only one contiguous segment, then the `GlobalMap` (see `m_GlobalMap` or details) is sufficient to describe the decomposition. If, however, each process owns multiple, non-adjacent segments of the array, a more sophisticated approach is needed. The `GlobalSegMap` data type allows one to describe a one-dimensional decomposition of an array with each process owning multiple, non-adjacent segments of the array.
In the current implementation of the `GlobalSegMap`, there is no santity check to guarantee that

$$\texttt{GlobalSegMap\%gsize} = \sum_{\texttt{i}=1}^{\texttt{ngseg}} \texttt{GlobalSegMap\%length(i)}.$$

The reason we have not implemented such a check is to allow the user to use the `GlobalSegMap` type to support decompositions of both *haloed* and *masked* data.

INTERFACE:

```
module m_GlobalSegMap

      implicit none

      private ! except
```

PUBLIC MEMBER FUNCTIONS:

```
      public :: GlobalSegMap    ! The class data structure
      public :: init            ! Create
      public :: clean           ! Destroy
      public :: comp_id         ! Return component ID number
      public :: gsize           ! Return global vector size (excl. halos)
      public :: GlobalStorage   ! Return total number of points in map,
                                ! including halo points (if present).
      public :: ProcessStorage  ! Return local storage on a given process.
      public :: OrderedPoints   ! Return grid points of a given process in
                                ! MCT-assumed order.
      public :: lsize           ! Return local--that is, on-process--storage
                                ! size (incl. halos)
      public :: ngseg           ! Return global number of segments
      public :: nlseg           ! Return local number of segments
      public :: max_nlseg       ! Return max local number of segments
      public :: active_pes      ! Return number of pes with at least 1
                                ! datum, and if requested, a list of them.
      public :: peLocs          ! Given an input list of point indices,
                                ! return its (unique) process ID.
      public :: haloed          ! Is the input GlobalSegMap haloed?
      public :: rank            ! Rank which process owns a datum
      public :: Sort            ! compute index permutation to re-order
                                ! GlobalSegMap%start, GlobalSegMap%length,
                                ! and GlobalSegMap%pe_loc
      public :: Permute         ! apply index permutation to re-order
                                ! GlobalSegMap%start, GlobalSegMap%length,
                                ! and GlobalSegMap%pe_loc
      public :: SortPermute     ! compute index permutation and apply it to
                                ! re-order the GlobalSegMap components
                                ! GlobalSegMap%start, GlobalSegMap%length,
```

```
                                ! and GlobalSegMap%pe_loc
      public :: increasing      ! Are the indices for each pe strictly
                                ! increasing?
      public :: copy            ! Copy the gsmap
      public :: print           ! Print the contents of the GSMap
```

PUBLIC TYPES:

```
    type GlobalSegMap
#ifdef SEQUENCE
      sequence
#endif
      integer :: comp_id ! Component ID number
      integer :: ngseg ! No. of Global segments
      integer :: gsize ! No. of Global elements
      integer,dimension(:),pointer :: start ! global seg. start index
      integer,dimension(:),pointer :: length ! segment lengths
      integer,dimension(:),pointer :: pe_loc ! PE locations
    end type GlobalSegMap

    interface init ; module procedure &
        initd_,&       ! initialize from all PEs
        initr_, &       ! initialize from the root
        initp_,&       ! initialize in parallel from replicated arrays
        initp1_, &      ! initialize in parallel from 1 replicated array
        initp0_, &      ! null constructor using replicated data
        init_index_     ! initialize from local index arrays
    end interface

    interface clean   ; module procedure clean_   ; end interface
    interface comp_id ; module procedure comp_id_ ; end interface
    interface gsize   ; module procedure gsize_   ; end interface
    interface GlobalStorage ; module procedure &
       GlobalStorage_
    end interface
    interface ProcessStorage ; module procedure &
       ProcessStorage_
    end interface
    interface OrderedPoints ; module procedure &
       OrderedPoints_
    end interface
    interface lsize ; module procedure lsize_ ; end interface
    interface ngseg ; module procedure ngseg_ ; end interface
    interface nlseg ; module procedure nlseg_ ; end interface
    interface max_nlseg ; module procedure max_nlseg_ ; end interface
    interface active_pes ; module procedure active_pes_ ; end interface
    interface peLocs ; module procedure peLocs_ ; end interface
    interface haloed ; module procedure haloed_ ; end interface
    interface rank  ; module procedure &
 rank1_ , &      ! single rank case
 rankm_         ! degenerate (multiple) ranks for halo case
    end interface
    interface Sort ; module procedure Sort_ ; end interface
    interface Permute ; module procedure &
 PermuteInPlace_
    end interface
    interface SortPermute ; module procedure &
 SortPermuteInPlace_
    end interface
```

```
 interface increasing ; module procedure increasing_ ; end interface
 interface copy ; module procedure copy_ ; end interface
 interface print ; module procedure &
       print_ ,&
printFromRootnp_
 end interface
```

REVISION HISTORY:

```
28Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
26Jan01 - J.W. Larson <larson@mcs.anl.gov> - replaced the component
          GlobalSegMap%comm with GlobalSegMap%comp_id.
06Feb01 - J.W. Larson <larson@mcs.anl.gov> - removed the
          GlobalSegMap%lsize component.  Also, added the
          GlobalStorage query function.
24Feb01 - J.W. Larson <larson@mcs.anl.gov> - Added the replicated
          initialization routines initp_() and initp1().
25Feb01 - J.W. Larson <larson@mcs.anl.gov> - Added the routine
          ProcessStorage_().
18Apr01 - J.W. Larson <larson@mcs.anl.gov> - Added the routine
          peLocs().
26Apr01 - R. Jacob <jacob@mcs.anl.gov> - Added the routine
          OrderedPoints_().
     03Aug01 - E. Ong <eong@mcs.anl.gov> - In initd_, call initr_
          with actual shaped arguments on non-root processes to satisfy
          F90 standard. See comments in initd.
18Oct01 - J.W. Larson <larson@mcs.anl.gov> - Added the routine
          bcast(), and also cleaned up prologues.
```

3.1.1 initd_ - define the map from distributed data

This routine takes the *scattered* input `INTEGER` arrays `start`, `length`, and `pe_loc`, gathers these data to the `root` process, and from them creates a *global* set of segment information for the output `GlobalSegMap` argument `GSMap`. The input `INTEGER` arguments `comp_id`, `gsize` provide the `GlobalSegMap` component ID number and global grid size, respectively. The input argument `my_comm` is the F90 `INTEGER` handle for the MPI communicator. If the input arrays are overdimensioned, optional argument *numel* can be used to specify how many elements should be used.

INTERFACE:

```
subroutine initd_(GSMap, start, length, root, my_comm, &
                  comp_id, pe_loc, gsize, numel)
```

USES:

```
use m_mpif90
use m_die
use m_stdio
use m_FcComms, only : fc_gather_int, fc_gatherv_int

implicit none
```

INPUT PARAMETERS:

```
integer,dimension(:),intent(in) :: start          ! segment local start
```

```
                                                    ! indices
      integer,dimension(:),intent(in) :: length     ! segment local lengths
      integer,intent(in)              :: root       ! root on my_com
      integer,intent(in)              :: my_comm    ! local communicatior
      integer,intent(in)              :: comp_id    ! component model ID
      integer,dimension(:), pointer, optional :: pe_loc ! process location
      integer,intent(in), optional    :: gsize      ! global vector size
                                                    ! (optional).  It can
                                                    ! be computed by this
                                                    ! routine if no haloing
                                                    ! is assumed.
      integer,intent(in), optional    :: numel      ! specify number of elements
! to use in start, length
```

OUTPUT PARAMETERS:

```
      type(GlobalSegMap),intent(out)  :: GSMap   ! Output GlobalSegMap
```

REVISION HISTORY:

```
    29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
    14Nov00 - J.W. Larson <larson@mcs.anl.gov> - final working version
    09Jan01 - J.W. Larson <larson@mcs.anl.gov> - repaired:  a subtle
                  bug concerning the usage of the argument pe_loc (result
                  was the new pointer variable my_pe_loc); a mistake in
                  the tag arguments to MPI_IRECV; a bug in the declaration
                  of the array status used by MPI_WAITALL.
    26Jan01 - J.W. Larson <larson@mcs.anl.gov> - replaced optional
                  argument gsm_comm with required argument comp_id.
         23Sep02 - Add optional argument numel to allow start, length
                   arrays to be overdimensioned.
         31Jan09 - P.H. Worley <worleyph@ornl.gov> - replaced irecv/send/waitall
                   logic with calls to flow controlled gather routines
```

3.1.2 initr_ initialize the map from the root

This routine takes the input `INTEGER` arrays `start`, `length`, and `pe_loc` (all valid only on the `root` process), and from them creates a *global* set of segment information for the output `GlobalSegMap` argument `GSMap`. The input `INTEGER` arguments `ngseg`, `comp_id`, `gsize` (again, valid only on the `root` process) provide the `GlobalSegMap` global segment count, component ID number, and global grid size, respectively. The input argument `my_comm` is the F90 `INTEGER` handle for the MPI communicator.

INTERFACE:

```
  subroutine initr_(GSMap, ngseg, start, length, pe_loc, root,  &
                    my_comm, comp_id, gsize)
```

USES:

```
      use m_mpif90
      use m_die
      use m_stdio

      implicit none
```

INPUT PARAMETERS:

```
integer, intent(in)             :: ngseg   ! no. of global segments
integer,dimension(:),intent(in) :: start   ! segment local start index
integer,dimension(:),intent(in) :: length  ! the distributed sizes
integer,dimension(:),intent(in) :: pe_loc  ! process location
integer,intent(in)              :: root    ! root on my_com
integer,intent(in)              :: my_comm ! local communicatior
integer,intent(in)              :: comp_id ! component id number
integer,intent(in), optional    :: gsize   ! global vector size
                                           ! (optional).  It can
                                           ! be computed by this
                                           ! routine if no haloing
                                           ! is assumed.
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap),intent(out)  :: GSMap   ! Output GlobalSegMap
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
09Nov00 - J.W. Larson <larson@mcs.anl.gov> - final working version
10Jan01 - J.W. Larson <larson@mcs.anl.gov> - minor bug fix
12Jan01 - J.W. Larson <larson@mcs.anl.gov> - minor bug fix regarding
                                             disparities in ngseg on
                                             the root and other
                                             processes
26Jan01 - J.W. Larson <larson@mcs.anl.gov> - replaced optional
               argument gsm_comm with required argument comp_id.
```

3.1.3 initp_ - define the map from replicated data.

The routine `initp_()` takes the input *replicated* arguments `comp_id`, `ngseg`, `gsize`, `start(:)`, `length(:)`, and `pe_loc(:)`, and uses them to initialize an output `GlobalSegMap` `GSMap`. This routine operates on the assumption that these data are replicated across the communicator on which the `GlobalSegMap` is being created.

INTERFACE:

```
subroutine initp_(GSMap, comp_id, ngseg, gsize, start, length, pe_loc)
```

USES:

```
use m_mpif90
use m_die, only : die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer,intent(in)              :: comp_id ! component model ID
integer,intent(in)              :: ngseg   ! global number of segments
integer,intent(in)              :: gsize   ! global vector size
integer,dimension(:),intent(in) :: start   ! segment local start index
integer,dimension(:),intent(in) :: length  ! the distributed sizes
integer,dimension(:),intent(in) :: pe_loc  ! process location
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap),intent(out)  :: GSMap   ! Output GlobalSegMap
```

REVISION HISTORY:

```
24Feb01 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
```

3.1.4 initp1_ - define the map from replicated data using 1 array.

The routine `initp1_()` takes the input *replicated* arguments `comp_id`, `ngseg`, `gsize`, and `all_arrays(:)`, and uses them to initialize an output `GlobalSegMap` `GSMap`. This routine operates on the assumption that these data are replicated across the communicator on which the `GlobalSegMap` is being created. The input array `all_arrays(:)` should be of length `2 * ngseg`, and is packed so that

$$\texttt{all\_arrays}(1:\texttt{ngseg}) = \texttt{GSMap\%start}(1:\texttt{ngseg})$$

$$\texttt{all\_arrays}(\texttt{ngseg}+1:2*\texttt{ngseg}) = \texttt{GSMap\%length}(1:\texttt{ngseg})$$

$$\texttt{all\_arrays}(2*\texttt{ngseg}+1:3*\texttt{ngseg}) = \texttt{GSMap\%pe\_loc}(1:\texttt{ngseg}).$$

INTERFACE:

```
subroutine initp1_(GSMap, comp_id, ngseg, gsize, all_arrays)
```

USES:

```
use m_mpif90
use m_die, only : die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer,intent(in)              :: comp_id    ! component model ID
integer,intent(in)              :: ngseg      ! global no. of segments
integer,intent(in)              :: gsize      ! global vector size
integer,dimension(:),intent(in) :: all_arrays ! packed array of length
                                              ! 3*ngseg containing (in
                                              ! this order):  start(:),
                                              ! length(:), and pe_loc(:)
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap),intent(out)  :: GSMap      ! Output GlobalSegMap
```

REVISION HISTORY:

```
24Feb01 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
```

3.1.5 initp0_ - Null Constructor Using Replicated Data

The routine `initp0_()` takes the input *replicated* arguments `comp_id`, `ngseg`, `gsize`, and uses them perform null construction of the output `GlobalSegMap` `GSMap`. This is a null constructor in the sense that we are not filling in the segment information arrays. This routine operates on the assumption that these data are replicated across the communicator on which the `GlobalSegMap` is being created.

INTERFACE:

```
subroutine initp0_(GSMap, comp_id, ngseg, gsize)
```

USES:

```
use m_die, only : die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer,intent(in)              :: comp_id ! component model ID
integer,intent(in)              :: ngseg   ! global number of segments
integer,intent(in)              :: gsize   ! global vector size
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap),intent(out)  :: GSMap   ! Output GlobalSegMap
```

REVISION HISTORY:

```
13Aug03 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
```

3.1.6 init_index_ - initialize GSM from local index arrays

The routine `init_index_()` takes a local array of indices `lindx` and uses them to create a `GlobalSegMap`. `lindx` is parsed to determine the lengths of the runs, and then a call is made to `initd_`. The optional argument `lsize` can be used if only the first `lsize` number of elements of `lindx` are valid. The optional argument `gsize` is used to specify the global number of unique points if this can not be determined from the collective `lindx`.

INTERFACE:

```
subroutine init_index_(GSMap, lindx, my_comm, comp_id, lsize, gsize)
```

USES:

```
use m_GlobalSegMap,only: GlobalSegMap
use m_GlobalSegMap,only: MCT_GSMap_init => init

use shr_sys_mod

use m_die
implicit none
```

INPUT PARAMETERS:

```
integer , dimension(:),intent(in) :: lindx   ! index buffer
integer , intent(in) :: my_comm         ! mpi communicator group (mine)
integer , intent(in) :: comp_id         ! component id (mine)

integer , intent(in),optional :: lsize  ! size of index buffer
integer , intent(in),optional :: gsize  ! global vector size
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap),intent(out) :: GSMap ! Output GlobalSegMap
```

REVISION HISTORY:

```
30Jul02 - T. Craig - initial version in cpl6.
17Nov05 - R. Loy <rloy@mcs.anl.gov> - install into MCT
18Nov05 - R. Loy <rloy@mcs.anl.gov> - make lsize optional
25Jul06 - R. Loy <rloy@mcs.anl.gov> - error check on lindex/alloc/dealloc
```

3.1.7 clean_ - clean the map

This routine deallocates the array components of the `GlobalSegMap` argument `GSMap`: `GSMap%start`, `GSMap%length`, and `GSMap%pe_loc`. It also zeroes out the values of the integer components `GSMap%ngseg`, `GSMap%comp_id`, and `GSMap%gsize`.

INTERFACE:

```
subroutine clean_(GSMap,stat)
```

USES:

```
use m_die

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(GlobalSegMap), intent(inout) :: GSMap
integer, optional,  intent(out)   :: stat
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
     01Mar02 - E.T. Ong <eong@mcs.anl.gov> - added stat argument.
               Removed dies to prevent crashing.
```

3.1.8 ngseg_ - Return the global number of segments from the map

The function `ngseg_()` returns the global number of vector segments in the `GlobalSegMap` argument `GSMap`. This is merely the value of `GSMap%ngseg`.

INTERFACE:

```
integer function ngseg_(GSMap)

     implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),intent(in) :: GSMap
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

3.1.9 nlseg_ - Return the local number of segments from the map

The function `nlseg_()` returns the number of vector segments in the `GlobalSegMap` argument `GSMap` that reside on the process specified by the input argument `pID`. This is the number of entries `GSMap%pe_loc` whose value equals `pID`.

INTERFACE:

```
integer function nlseg_(GSMap, pID)

     implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),intent(in) :: GSMap
integer,           intent(in) :: pID
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
14Jun01 - J.W. Larson <larson@mcs.anl.gov> - Bug fix in lower
               limit of loop over elements of GSMap%pe_loc(:).  The
               original code had this lower limit set to 0, which
               was out-of-bounds (but uncaught).  The correct lower
               index is 1.  This bug was discovered by Everest Ong.
```

3.1.10 max_nlseg_ - Return the max number of segments over all procs

The function `max_nlseg_()` returns the maximum number over all processors of the vector segments in the `GlobalSegMap` argument `gsap` E.g. max_p(nlseg(gsmap,p)) but computed more efficiently

INTERFACE:

```
integer function max_nlseg_(gsmap)
```

USES:

```
use m_MCTWorld,      only :ThisMCTWorld
use m_mpif90
use m_die

use m_stdio    ! rml

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in) :: gsmap
```

REVISION HISTORY:

```
17Jan07 - R. Loy <rloy@mcs.anl.gov> - initial prototype
```

3.1.11 comp_id_ - Return the commponent ID from the GlobalSegMap.

The function `comp_id_()` returns component ID number stored in `GSMap%comp_id`.

INTERFACE:

```
integer function comp_id_(GSMap)
```

USES:

```
use m_die,only: die
use m_stdio, only :stderr

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),intent(in) :: GSMap
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
26Jan01 - J.W. Larson <larson@mcs.anl.gov> - renamed comp_id_
          to fit within MCT_World component ID context.
  01May01 - R.L. Jacob  <jacob@mcs.anl.gov> - make sure GSMap
          is defined.
```

3.1.12 gsize_ - Return the global vector size from the GlobalSegMap.

The function `gsize_()` takes the input `GlobalSegMap` arguement `GSMap` and returns the global vector length stored in `GlobalSegMap%gsize`.

INTERFACE:

```
integer function gsize_(GSMap)

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),intent(in) :: GSMap
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

3.1.13 GlobalStorage_ - Return global storage space required.

The function `GlobalStorage_()` takes the input `GlobalSegMap` arguement `GSMap` and returns the global storage space required (*i.e.*, the vector length) to hold all the data specified by `GSMap`.
N.B.: If `GSMap` contains halo or masked points, the value by `GlobalStorage_()` may differ from `GSMap%gsize`.

INTERFACE:

```
integer function GlobalStorage_(GSMap)

      implicit none
```

INPUT PARAMETERS:

```
      type(GlobalSegMap),intent(in) :: GSMap
```

REVISION HISTORY:

```
    06Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

3.1.14 ProcessStorage_ - Number of points on a given process.

The function `ProcessStorage_()` takes the input `GlobalSegMap` arguement `GSMap` and returns the storage space required by process `PEno` (*i.e.*, the vector length) to hold all the data specified by `GSMap`.

INTERFACE:

```
integer function ProcessStorage_(GSMap, PEno)

      implicit none
```

INPUT PARAMETERS:

```
      type(GlobalSegMap),intent(in) :: GSMap
      integer,           intent(in) :: PEno
```

REVISION HISTORY:

```
    06Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

3.1.15 OrderedPoints_ - The grid points on a given process

returned in the assumed MCT order.
The function `OrderedPoints_()` takes the input `GlobalSegMap` arguement `GSMap` and returns a vector of the points owned by `PEno`. `Points` is allocated here. The calling process is responsible for deallocating the space.

INTERFACE:

```
     subroutine OrderedPoints_(GSMap, PEno, Points)
```

USES:

```
use m_die,only: die

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in) :: GSMap   ! input GlobalSegMap
integer,            intent(in) :: PEno ! input process number
integer,dimension(:),pointer   :: Points  ! the vector of points
```

REVISION HISTORY:

```
25Apr01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
```

3.1.16 lsize_ - find the local storage size from the map

This function returns the number of points owned by the local process, as defined by the input `GlobalSegMap` argument `GSMap`. The local process ID is determined through use of the input `INTEGER` argument `comm`, which is the Fortran handle for the MPI communicator.

INTERFACE:

```
integer function lsize_(GSMap, comm)
```

USES:

```
use m_mpif90
use m_die ,          only : MP_perr_die

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in) :: GSMap
integer,            intent(in) :: comm
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
06Feb01 - J.W. Larson <larson@mcs.anl.gov> - Computed directly
               from the GlobalSegMap, rather than returning a hard-
               wired local attribute. This required the addition of
               the communicator argument.
```

3.1.17 rank1_ - rank which process owns a datum with given global

index.
This routine assumes that there is one process that owns the datum with a given global index. It should not be used when the input `GlobalSegMap` argument `GSMap` has been built to incorporate halo points.

INTERFACE:

```
subroutine rank1_(GSMap, i_g, rank)

  implicit none
```

INPUT PARAMETERS:

```
  type(GlobalSegMap), intent(in)  :: GSMap   ! input GlobalSegMap
  integer,            intent(in)  :: i_g  ! a global index
```

OUTPUT PARAMETERS:

```
  integer,            intent(out) :: rank    ! the pe on which this
                                             ! element resides
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

3.1.18 rankm_ - rank which processes own a datum with given global

index.
This routine assumes that there may be more than one process that owns the datum with a given global index. This routine should be used when the input `GlobalSegMap` argument `GSMap` has been built to incorporate ! halo points. *Nota Bene*: The output array `rank` is allocated in this routine and must be deallocated by the routine calling `rankm_()`. Failure to do so could result in a memory leak.

INTERFACE:

```
subroutine rankm_(GSMap, i_g, num_loc, rank)

  implicit none
```

INPUT PARAMETERS:

```
  type(GlobalSegMap), intent(in)  :: GSMap   ! input GlobalSegMap
  integer,            intent(in)  :: i_g  ! a global index
```

OUTPUT PARAMETERS:

```
  integer,            intent(out) :: num_loc ! the number of processes
                                             ! which own element i_g
  integer, dimension(:), pointer  :: rank    ! the process(es) on which
                                             ! element i_g resides
```

REVISION HISTORY:

```
29Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

3.1.19 active_pes_ - number of processes that own data.

index.
This routine scans the pe location list of the input `GlobalSegMap GSMap%pe_loc(:)`, and counts the number of pe locations that own at least one datum. This value is returned in the `INTEGER` argument `n_active`. If the optional `INTEGER` array argument `list` is included in the call, a sorted list (in ascending order) of the active processes will be returned.

N.B.: If `active_pes_()` is invoked with the optional argument `pe_list` included, this routine will allocate and return this array. The user must deallocate this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine active_pes_(GSMap, n_active, pe_list)
```

USES:

```
use m_die ,          only : die

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),    intent(in)        :: GSMap
```

OUTPUT PARAMETERS:

```
integer,               intent(out)       :: n_active
integer, dimension(:), pointer, optional :: pe_list
```

REVISION HISTORY:

```
03Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

3.1.20 peLocs_ - process ID locations for distributed points.

index.
This routine takes an input `INTEGER` array of point indices `points(:)`, compares them with an input `GlobalSegMap` `pointGSMap`, and returns the *unique* process ID location for each point. Note the emphasize on unique. The assumption here (which is tested) is that `pointGSMap` is not haloed. The process ID locations for the points is returned in the array `pe_locs(:)`.
N.B.: The test of `pointGSMap` for halo points, and the subsequent search for the process ID for each point is very slow. This first version of the routine is serial. A parallel version of this routine will need to be developed.

INTERFACE:

```
subroutine peLocs_(pointGSMap, npoints, points, pe_locs)
```

USES:

```
use m_die ,          only : die

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),    intent(in)   :: pointGSMap
integer,               intent(in)   :: npoints
integer, dimension(:), intent(in)   :: points
```

OUTPUT PARAMETERS:

```
integer, dimension(:), intent(out)  :: pe_locs
```

REVISION HISTORY:

```
18Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial version.
     18Oct16 - P. Worley <worleyph@gmail.com> - added algorithm options:
               new default changes complexity from O(npoints*ngseg) to
               O(gsize + ngseg) (worst case), and much better in current
               usage. Worst case memory requirements are O(gsize), but
               not seen in current usage. Other new algorithm is a little
               slower in practice, and worst case memory requirement is
               O(ngseg), which is also not seen in current usage.
               Original algorithm is recovered if compiled with
               LOW_MEMORY_PELOCS defined. Otherwise nondefault new
               algorithm is enabled if compiled with MEDIUM_MEMORY_PELOCS
               defined.
```

3.1.21 haloed_ - test GlobalSegMap for presence of halo points.

index.
This LOGICAL function tests the input GlobalSegMap GSMap for the presence of halo points. Halo points are points that appear in more than one segment of a GlobalSegMap. If *any* halo point is found, the function haloed_() returns immediately with value .TRUE. If, after an exhaustive search of the map has been completed, no halo points are found, the function haloed_() returns with value .FALSE.
The search algorithm is:

1. Extract the segment start and length information from GSMap%start and GSMap%length into the temporary arrays start(:) and length(:).
2. Sort these arrays in *ascending order* keyed by start.
3. Scan the arrays start andlength. A halo point is present if for at least one value of the index $1 \leq \mathtt{n} \leq \mathtt{GSMap\%ngseg}$

$$\mathtt{start(n)} + \mathtt{length(n)} - 1 \geq \mathtt{start(n+1)}$$

.

N.B.: Beware that the search for halo points is potentially expensive.

INTERFACE:

```
logical function haloed_(GSMap)
```

USES:

```
use m_die ,          only : die
use m_SortingTools , only : IndexSet
use m_SortingTools , only : IndexSort
use m_SortingTools , only : Permute

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)           :: GSMap
```

REVISION HISTORY:

```
08Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version.
26Apr01 - J.W. Larson <larson@mcs.anl.gov> - Bug fix.
```

3.1.22 Sort_ - generate index permutation for GlobalSegMap.

Sort_() uses the supplied keys key1 and key2 to generate a permutation perm that will put the entries of the components GlobalSegMap%start, GlobalSegMap%length and GlobalSegMap%pe_loc in *ascending* lexicographic order.
N.B.: Sort_() returns an allocated array perm(:). It the user must deallocate this array once it is no longer needed. Failure to do so could create a memory leak.

INTERFACE:

```
subroutine Sort_(GSMap, key1, key2, perm)
```

USES:

```
use m_die ,          only : die
use m_SortingTools , only : IndexSet
use m_SortingTools , only : IndexSort

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),    intent(in)           :: GSMap ! input GlobalSegMap
integer, dimension(:), intent(in)           :: key1  ! first sort key
integer, dimension(:), intent(in), optional :: key2  ! second sort key
```

OUTPUT PARAMETERS:

```
integer, dimension(:), pointer :: perm    ! output index permutation
```

REVISION HISTORY:

```
02Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

3.1.23 PermuteInPlace_ - apply index permutation to GlobalSegMap.

PermuteInPlace_() uses a supplied index permutation perm to re-order GlobalSegMap%start, GlobalSegMap%length and GlobalSegMap%pe_loc.

INTERFACE:

```
subroutine PermuteInPlace_(GSMap, perm)
```

USES:

```
use m_die ,          only : die
use m_SortingTools , only : Permute

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:), intent(in) :: perm
```

INPUT/OUTPUT PARAMETERS:

```
type(GlobalSegMap), intent(inout) :: GSMap
```

REVISION HISTORY:

```
02Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

3.1.24 SortPermuteInPlace_ - Sort in-place GlobalSegMap components.

`SortPermuteInPlace_()` uses a the supplied key(s) to generate and apply an index permutation that will place the `GlobalSegMap` components `GlobalSegMap%start`, `GlobalSegMap%length` and `GlobalSegMap%pe_loc` in lexicographic order.

INTERFACE:

```
subroutine SortPermuteInPlace_(GSMap, key1, key2)
```

USES:

```
use m_die ,          only : die

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:), intent(in)           :: key1
integer, dimension(:), intent(in), optional :: key2
```

INPUT/OUTPUT PARAMETERS:

```
type(GlobalSegMap),    intent(inout)        :: GSMap
```

REVISION HISTORY:

```
02Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

3.1.25 increasing_ - Return .TRUE. if GSMap has increasing indices

The function `increasing_()` returns .TRUE. if each proc's indices in the `GlobalSegMap` argument `GSMap` have strictly increasing indices. I.e. the proc's segments have indices in ascending order and are non-overlapping.

INTERFACE:

```
logical function increasing_(gsmap)
```

USES:

```
use m_MCTWorld, only: ThisMCTWorld
use m_die

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),intent(in) :: gsmap
```

REVISION HISTORY:

```
06Jun07 - R. Loy <rloy@mcs.anl.gov> - initial version
```

3.1.26 copy_ - Copy the gsmap to a new gsmap

Make a copy of a gsmap. Note this is a deep copy of all arrays.

INTERFACE:

```
subroutine copy_(src,dest)
```

USES:

```
use m_MCTWorld, only: ThisMCTWorld
use m_die

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),intent(in) :: src
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap),intent(out) :: dest
```

REVISION HISTORY:

```
27Jul07 - R. Loy <rloy@mcs.anl.gov> - initial version
```

3.1.27 print_ - Print GSMap info

Print out contents of GSMAP on unit number 'lun'

INTERFACE:

```
subroutine print_(gsmap,lun)
```

USES:

```
use m_die

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(GlobalSegMap),      intent(in) :: gsmap
integer, intent(in)           :: lun
```

REVISION HISTORY:

```
06Jul12 - R. Jacob <jacob@mcs.anl.gov> - initial version
```

3.1.28 printFromRoot_ - Print GSMap info

Print out contents of GSMAP on unit number 'lun'

INTERFACE:

```
subroutine printFromRootnp_(gsmap,mycomm,lun)
```

USES:

```
use m_MCTWorld,      only : printnp
use m_die
use m_mpif90

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(GlobalSegMap),      intent(in) :: gsmap
integer, intent(in)           :: mycomm
integer, intent(in)           :: lun
```

REVISION HISTORY:

```
06Jul12 - R. Jacob <jacob@mcs.anl.gov> - initial version
```

3.2 Module m_GlobalSegMapComms - GlobalSegMap Communications Support (Source File: m_GlobalSegMapComms.F90)

This module provides communications support for the `GlobalSegMap` datatype. Both blocking and non-blocking point-to-point communications are provided for send (analogues to `MPI_SEND()/MPI_ISEND()`) A receive and broadcast method is also supplied.

INTERFACE:

```
module m_GlobalSegMapComms

      implicit none

      private   ! except
```

PUBLIC MEMBER FUNCTIONS:

```
      public :: send
      public :: recv
      public :: isend
      public :: bcast

      interface bcast ; module procedure bcast_ ; end interface
      interface send  ; module procedure send_  ; end interface
      interface recv  ; module procedure recv_  ; end interface
      interface isend ; module procedure isend_ ; end interface
```

REVISION HISTORY:

```
   11Aug03 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

3.2.1 send_ - Point-to-point blocking Send of a GlobalSegMap

This routine performs a blocking send of a `GlobalSegMap` (the input argument `outgoingGSMap`) to the root processor on component `comp_id`. The input `INTEGER` argument `TagBase` is used to generate tags for the messages associated with this operation; there are six messages involved, so the user should avoid using tag values `TagBase` and `TagBase + 5`. All six messages are blocking. The success (failure) of this operation is reported in the zero (non-zero) value of the optional `INTEGER` output variable `status`.

INTERFACE:

```
  subroutine send_(outgoingGSMap, comp_id, TagBase, status)
```

USES:

```
      use m_mpif90
      use m_die, only : MP_perr_die,die
      use m_stdio

      use m_GlobalSegMap, only : GlobalSegMap
      use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg
      use m_GlobalSegMap, only : GlobalSegMap_comp_id => comp_ID
      use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize

      use m_MCTWorld, only : ComponentToWorldRank
      use m_MCTWorld, only : ThisMCTWorld
```

```
implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),    intent(IN)  :: outgoingGSMap
integer,               intent(IN)  :: comp_id
integer,               intent(IN)  :: TagBase
```

OUTPUT PARAMETERS:

```
integer, optional,     intent(OUT) :: status
```

REVISION HISTORY:

```
13Aug03 - J.W. Larson <larson@mcs.anl.gov> - API and initial version.
26Aug03 - R. Jacob <jacob@mcs.anl.gov> - use same method as isend_
05Mar04 - R. Jacob <jacob@mcs.anl.gov> - match new isend_ method.
```

3.2.2 isend_ - Point-to-point Non-blocking Send of a GlobalSegMap

This routine performs a non-blocking send of a `GlobalSegMap` (the input argument `outgoingGSMap`) to the root processor on component `comp_id` The input `INTEGER` argument `TagBase` is used to generate tags for the messages associated with this operation; there are six messages involved, so the user should avoid using tag values `TagBase` and `TagBase + 5`. All six messages are non- blocking, and the request handles for them are returned in the output `INTEGER` array `reqHandle`, which can be checked for completion using any of MPI's wait functions. The success (failure) of this operation is reported in the zero (non-zero) value of the optional `INTEGER` output variable `status`.
N.B.: Data is sent directly out of `outgoingGSMap` so it must not be deleted until the send has completed.
N.B.: The array `reqHandle` represents allocated memory that must be deallocated when it is no longer needed. Failure to do so will create a memory leak.

INTERFACE:

```
subroutine isend_(outgoingGSMap, comp_id, TagBase, reqHandle, status)
```

USES:

```
use m_mpif90
use m_die, only : MP_perr_die,die
use m_stdio

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg

use m_MCTWorld, only : ComponentToWorldRank
use m_MCTWorld, only : ThisMCTWorld

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),    intent(IN)  :: outgoingGSMap
integer,               intent(IN)  :: comp_id
integer,               intent(IN)  :: TagBase
```

OUTPUT PARAMETERS:

```
integer, dimension(:), pointer     :: reqHandle
integer, optional,     intent(OUT) :: status
```

REVISION HISTORY:

```
13Aug03 - J.W. Larson <larson@mcs.anl.gov> - API and initial version.
05Mar04 - R. Jacob <jacob@mcs.anl.gov> - Send everything directly out
          of input GSMap.  Don't use a SendBuffer.
```

3.2.3 recv_ - Point-to-point blocking Receive of a GlobalSegMap

This routine performs a blocking receive of a `GlobalSegMap` (the input argument `outgoingGSMap`) from the root processor on component `comp_id`. The input `INTEGER` argument `TagBase` is used to generate tags for the messages associated with this operation; there are six messages involved, so the user should avoid using tag values `TagBase` and `TagBase + 5`. The success (failure) of this operation is reported in the zero (non-zero) value of the optional `INTEGER` output variable `status`.

INTERFACE:

```
subroutine recv_(incomingGSMap, comp_id, TagBase, status)
```

USES:

```
      use m_mpif90
      use m_die, only : MP_perr_die, die
      use m_stdio

      use m_GlobalSegMap, only : GlobalSegMap
      use m_GlobalSegMap, only : GlobalSegMap_init => init

      use m_MCTWorld, only : ComponentToWorldRank
      use m_MCTWorld, only : ThisMCTWorld

      implicit none
```

INPUT PARAMETERS:

```
integer,               intent(IN)  :: comp_id
integer,               intent(IN)  :: TagBase
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap),    intent(OUT) :: incomingGSMap
integer, optional,     intent(OUT) :: status
```

REVISION HISTORY:

```
13Aug03 - J.W. Larson <larson@mcs.anl.gov> - API and initial version.
25Aug03 - R.Jacob <larson@mcs.anl.gov> - rename to recv_.
```

3.2.4 bcast_ - broadcast a GlobalSegMap object

The routine `bcast_()` takes the input/output *GlobalSegMap* argument `GSMap` (on input valid only on the `root` process, on output valid on all processes) and broadcasts it to all processes on the communicator associated with the F90 handle `comm`. The success (failure) of this operation is returned as a zero (non-zero) value of the optional output `INTEGER` argument `status`.

INTERFACE:

```
subroutine bcast_(GSMap, root, comm, status)
```

USES:

```
use m_mpif90
use m_die, only : MP_perr_die,die
use m_stdio

use m_GlobalSegMap, only : GlobalSegMap

implicit none
```

INPUT PARAMETERS:

```
integer,            intent(in)     :: root
integer,            intent(in)     :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(GlobalSegMap), intent(inout)  :: GSMap  ! Output GlobalSegMap
```

OUTPUT PARAMETERS:

```
integer, optional,  intent(out)    :: status ! global vector size
```

REVISION HISTORY:

```
17Oct01 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
11Aug03 - J.W. Larson <larson@mcs.anl.gov> - Relocated from original
          location in m_GlobalSegMap.
```

4 The Router

4.1 Module m_Router – Router class (Source File: m_Router.F90)

The Router data type contains all the information needed to send an AttrVect between a component on M MPI-processes and a component on N MPI-processes. This module defines the Router datatype and provides methods to create and destroy one.

INTERFACE:

```
  module m_Router

       use m_realkinds, only : FP
       use m_zeit

       implicit none

       private   ! except

    !declare a private pointer structure for the real data
       type :: rptr
 #ifdef SEQUENCE
         sequence
 #endif
         real(FP),dimension(:),pointer :: pr
       end type

    !declare a private pointer structure for the integer data
       type :: iptr
 #ifdef SEQUENCE
         sequence
 #endif
         integer,dimension(:),pointer :: pi
       end type
```

PUBLIC TYPES:

```
       public :: Router         ! The class data structure

       public :: rptr,iptr       ! pointer types used in Router
```

```
     type Router
 #ifdef SEQUENCE
       sequence
 #endif
       integer :: comp1id                          ! myid
       integer :: comp2id                          ! id of second component
       integer :: nprocs                           ! number of procs to talk to
       integer :: maxsize                          ! maximum amount of data going to a processor
       integer :: lAvsize                          ! The local size of AttrVect which can be
                                                   ! used with this Router in MCT_Send/MCT_Recv
       integer :: numiatt                          ! Number of integer attributes currently in u
       integer :: numratt                          ! Number of real attributes currently in use
       integer,dimension(:),pointer   :: pe_list    ! processor ranks of send/receive in MCT_comm
       integer,dimension(:),pointer   :: pe_list_loc! processor ranks of send/receive in local co
       integer                        :: mpicomm    ! local MPI communicator
       integer,dimension(:),pointer   :: num_segs   ! number of segments to send/receive
```

```
  integer,dimension(:),pointer   :: locsize    ! total of seg_lengths for a proc
  integer,dimension(:),pointer   :: permarr    ! possible permutation array
  integer,dimension(:,:),pointer :: seg_starts ! starting index
  integer,dimension(:,:),pointer :: seg_lengths! total length
  type(rptr),dimension(:),pointer :: rp1       ! buffer to hold real data
  type(iptr),dimension(:),pointer :: ip1       ! buffer to hold integer data
  integer,dimension(:),pointer   :: ireqs,rreqs  ! buffer for MPI_Requests
  integer,dimension(:,:),pointer :: istatus,rstatus  ! buffer for MPI_Status
end type Router
```

PUBLIC MEMBER FUNCTIONS:

```
  public :: init            ! Create a Router
  public :: clean           ! Destroy a Router
  public :: print           ! Print info about a Router


interface init  ; module procedure  &
    initd_, &       ! initialize a Router between two seperate components
    initp_          ! initialize a Router locally with two GSMaps
end interface
interface clean ; module procedure clean_ ; end interface
interface print ; module procedure print_ ; end interface
```

REVISION HISTORY:

```
15Jan01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
08Feb01 - R. Jacob <jacob@mcs.anl.gov> add locsize and maxsize
          to Router type
25Sep02 - R. Jacob <jacob@mcs.anl.gov> Remove type string.  Add lAvsize
23Jul03 - R. Jacob <jacob@mcs.anl.gov> Add status and reqs arrays used
          in send/recv to the Router datatype.
24Jul03 - R. Jacob <jacob@mcs.anl.gov> Add real and integer buffers
          for send/recv to the Router datatype.
22Jan08 - R. Jacob <jacob@mcs.anl.gov> Add ability to handle an unordered
          GSMap by creating a new, ordered one and building Router from
          that.  Save permutation info in Router datatype.
```

4.1.1 initd_ - initialize a Router between two seperate components

The routine `initd_()` exchanges the `GSMap` with the component identified by `othercomp` and then calls `initp_()` to build a Router `Rout` between them.
N.B. The `GSMap` argument must be declared so that the index values on a processor are in ascending order.

INTERFACE:

```
subroutine initd_(othercomp,GSMap,mycomm,Rout,name )
```

USES:

```
  use m_GlobalSegMap, only :GlobalSegMap
  use m_ExchangeMaps,only: MCT_ExGSMap => ExchangeMap
  use m_mpif90
  use m_die

  implicit none
```

INPUT PARAMETERS:

```
integer, intent(in)        :: othercomp
integer, intent(in)        :: mycomm
type(GlobalSegMap),intent(in)    :: GSMap     ! of the calling comp
character(len=*), intent(in),optional     :: name
```

OUTPUT PARAMETERS:

```
type(Router), intent(out)        :: Rout
```

REVISION HISTORY:

```
15Jan01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
06Feb01 - R. Jacob <jacob@mcs.anl.gov> - Finish initialization
          of the Router.  Router now works both ways.
25Apr01 - R. Jacob <jacob@mcs.anl.gov> - Eliminate early
          custom code to exchange GSMap components and instead
          the more general purpose routine in m_ExchangeMaps.
          Use new subroutine OrderedPoints in m_GlobalSegMap
          to construct the vector of local and remote GSMaps.
          Clean-up code a little.
03May01 - R. Jacob <jacob@mcs.anl.gov> - rename to initd and
          move most of code to new initp routine
```

4.1.2 initp_ - initialize a Router from two GlobalSegMaps

Given two GlobalSegmentMaps `GSMap` and `RGSMap`, intialize a Router `Rout` between them. Use local communicator `mycomm`.
N.B. The two `GSMap` arguments must be declared so that the index values on a processor are in ascending order.

INTERFACE:

```
subroutine initp_(inGSMap,inRGSMap,mycomm,Rout,name )
```

USES:

```
use m_GlobalSegMap,  only :GlobalSegMap
use m_GlobalSegMap,  only :ProcessStorage
use m_GlobalSegMap,  only :GSMap_comp_id => comp_id
use m_GlobalSegMap,  only :GSMap_increasing => increasing
use m_GlobalSegMap,  only :GlobalSegMap_copy => copy
use m_GlobalSegMap,  only :GlobalSegMap_init => init
use m_GlobalSegMap,  only :GlobalSegMap_clean => clean
use m_GlobalSegMap,  only :GlobalSegMap_OPoints => OrderedPoints
use m_GlobalSegMap,  only :GlobalSegMap_ngseg => ngseg  ! rml
use m_GlobalSegMap,  only :GlobalSegMap_nlseg => nlseg  ! rml
use m_GlobalSegMap,  only :GlobalSegMap_max_nlseg => max_nlseg  ! rml

use m_GlobalToLocal, only :GlobalToLocalIndex
use m_MCTWorld,      only :MCTWorld
use m_MCTWorld,      only :ThisMCTWorld

use m_Permuter      ,only:Permute
use m_MergeSorts    ,only:IndexSet
use m_MergeSorts    ,only:IndexSort
```

```
use m_mpif90
use m_die

 use m_zeit


use m_stdio    ! rml
 use shr_timer_mod        ! rml timers

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in) :: inGSMap
type(GlobalSegMap), intent(in) :: inRGSMap
integer      ,    intent(in) :: mycomm
character(len=*), intent(in),optional     :: name
```

OUTPUT PARAMETERS:

```
type(Router),      intent(out) :: Rout
```

REVISION HISTORY:

```
03May01 - R.L. Jacob <jacob@mcs.anl.gov> - Initial code brought
          in from old init routine.
31Jul01 - Jace A Mogill <mogill@cray.com>
          Rewrote to reduce number of loops and temp storage
26Apr06 - R. Loy <rloy@mcs.anl.gov> - recode the search through
          the remote GSMap to improve efficiency
05Jan07 - R. Loy <rloy@mcs.anl.gov> - improved bound on size of
          tmpsegcount and tmpsegstart
15May07 - R. Loy <rloy@mcs.anl.gov> - improved bound on size of
          rgs_lb and rgs_ub
25Jan08 - R. Jacob <jacob@mcs.anl.gov> - Dont die if GSMap is not
          increasing.  Instead, permute it to increasing and proceed.
07Sep12 - T. Craig <tcraig@ucar.edu> - Replace a double loop with a single
          to improve speed for large proc and segment counts.
12Nov16 - P. Worley <worleyph@gmail.com> - eliminate iterations in nested
          loop that can be determined to be unnecessary
```

4.1.3 clean_ - Destroy a Router

Deallocate Router internal data structures and set integer parts to zero.

INTERFACE:

```
subroutine clean_(Rout,stat)
```

USES:

```
use m_die

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(Router),      intent(inout) :: Rout
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
15Jan01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
08Feb01 - R. Jacob <jacob@mcs.anl.gov> - add code to clean
          the maxsize and locsize
01Mar02 - E.T. Ong <eong@mcs.anl.gov> removed the die to prevent
          crashes and added stat argument.
```

4.1.4 print_ - Print router info

Print out communication info about router on unit number 'lun' e.g. (source,destination,length)

INTERFACE:

```
subroutine print_(rout,mycomm,lun)
```

USES:

```
use m_die
use m_mpif90

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(Router),      intent(in) :: Rout
integer, intent(in)           :: mycomm
integer, intent(in)           :: lun
```

REVISION HISTORY:

```
27Jul07 - R. Loy <rloy@mcs.anl.gov>  initial version
```

5 The General Grid

5.1 Module m_GeneralGrid – Physical Coordinate Grid Information Storage (Source File: m_GeneralGrid.F90)

The `GeneralGrid` data type is a flexible, generic structure for storing physical coordinate grid information. The `GeneralGrid` may be employed to store coordinate grids of arbitrary dimension, and is also capable of supporting unstructured grids such as meteorological observation data streams. The grid is representated by a literal listing of the gridpoint coordinates, along with other integer and real *attributes* associated with each location. Examples of real non-coordinate attributes are grid cell length, cross-sectional area, and volume elements, projections of local directional unit vectors onto *et cetera* A `GeneralGrid` as at minimum one integer attribute—*the global grid point number*, or `GlobGridNum`, which serves as a unique identifier for each physical grid location.
The real attributes of of the `GeneralGrid` are grouped as `List` components:

- `GGrid%coordinate_list` contains the list of the physical dimension names of the grid. The user initializes a `List` by supplying the items in it as a string with the items delimitted by colons. For example, setting the coordinates for Euclidean 3-space is accomplished by a choice of `'x:y:z'`, cylindrical coordinates by `'rho:theta:z'`, spherical coordinates by `'r:theta:phi'`, *et cetera*.

- `GGrid%weight_list` contains the names of the spatial cell length, area, and volume weights associated with the grid. These are also stored in `List` form, and are set by the user in the same fashion as described above for coordinates. For example, one might wish create cell weight attributes for a cylindrical grid by defining a weight list of `'drho:dphi:rhodphi:dz`.

- `GGrid%other_list` is space for the user to define other real attributes. For example, one might wish to do vector calculus operatons in spherical coordinates. Since the spherical coordinate unit vectors $\hat{r}$, $\hat{\theta}$, and $\hat{\phi}$ vary in space, it is sometimes useful to store their projections on the fixed Euclidean unit vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$. To do this one might set up a list of attributes using the string

  ```
  'rx:ry:rz:thetax:thetay:thetaz:phix:phiy:phyz'
  ```

- `GGrid%index_list` provides space for the user to define integer attributes such as alternative indexing schemes, indices for defining spatial regions, *et cetera*. This attribute list contains all the integer attributes for the `GeneralGrid` save one: the with the ever-present *global gridpoint number attribute* `GlobGridNum`, which is set automatically by MCT.

This module contains the definition of the `GeneralGrid` datatype, various methods for creating and destroying it, query methods, and tools for multiple-key sorting of gridpoints.

INTERFACE:

```
module m_GeneralGrid
```

USES:

```
use m_List, only : List   ! Support for List components.

use m_AttrVect, only : AttrVect ! Support for AttrVect component.

implicit none

private   ! except
```

PUBLIC TYPES:

```
      public :: GeneralGrid      ! The class data structure

    Type GeneralGrid
#ifdef SEQUENCE
      sequence
#endif
      type(List)                      :: coordinate_list
      type(List)                      :: coordinate_sort_order
      logical, dimension(:), pointer :: descend
      type(List)                      :: weight_list
      type(List)                      :: other_list
      type(List)                      :: index_list
      type(AttrVect)                  :: data
    End Type GeneralGrid
```

PUBLIC MEMBER FUNCTIONS:

```
      public :: init             ! Create a GeneralGrid
      public :: initCartesian    !
      public :: initUnstructured !
      public :: clean            ! Destroy a GeneralGrid
      public :: zero             ! Zero data in a GeneralGrid

                              ! Query functions-----------------
      public :: dims          ! Return dimensionality of the GeneralGrid
      public :: indexIA       ! Index integer attribute (indices)
      public :: indexRA       ! Index integer attribute (coords/weights)
      public :: lsize         ! Return local number of points
      public :: exportIAttr   ! Return INTEGER attribute as a vector
      public :: exportRAttr   ! Return REAL attribute as a vector

                              ! Manipulation--------------------
      public :: importIAttr   ! Insert INTEGER vector as attribute
      public :: importRAttr   ! Insert REAL vector as attribute
      public :: Sort          ! Sort point data by coordinates -> permutation
      public :: Permute       ! Rearrange point data using input permutation
      public :: SortPermute   ! Sort and Permute point data

    interface init  ; module procedure &
        init_, &
        initl_, &
        initgg_
    end interface
    interface initCartesian ; module procedure &
        initCartesianSP_, &
 initCartesianDP_
    end interface
    interface initUnstructured ; module procedure &
        initUnstructuredSP_, &
 initUnstructuredDP_
    end interface
    interface clean ; module procedure clean_ ; end interface
    interface zero ; module procedure zero_ ; end interface

    interface dims ; module procedure dims_ ; end interface
    interface indexIA ; module procedure indexIA_ ; end interface
    interface indexRA ; module procedure indexRA_ ; end interface
    interface lsize   ; module procedure lsize_   ; end interface
```

```
interface exportIAttr ; module procedure exportIAttr_ ; end interface
interface exportRAttr ; module procedure &
   exportRAttrSP_, &
   exportRAttrDP_
end interface
interface importIAttr ; module procedure importIAttr_ ; end interface
interface importRAttr ; module procedure &
   importRAttrSP_, &
   importRAttrDP_
end interface

interface Sort    ; module procedure Sort_    ; end interface
interface Permute ; module procedure Permute_ ; end interface
interface SortPermute ; module procedure SortPermute_ ; end interface
```

PUBLIC DATA MEMBERS:

```
CHARACTER Tag for GeneralGrid Global Grid Point Identification Number

character(len=*), parameter :: GlobGridNum='GlobGridNum'
```

SEE ALSO:

```
The MCT module m_AttrVect and the mpeu module m_List.
```

REVISION HISTORY:

```
25Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
31Oct00 - J.W. Larson <larson@mcs.anl.gov> - modified the
          GeneralGrid type to allow inclusion of grid cell
          dimensions (lengths) and area/volume weights.
15Jan01 - J.W. Larson implemented new GeneralGrid type
          definition and added numerous APIs.
17Jan01 - J.W. Larson fixed minor bug in module header use
          statement.
19Jan01 - J.W. Larson added other_list and coordinate_sort_order
          components to the GeneralGrid type.
21Mar01 - J.W. Larson - deleted the initv_ API (more study
          needed before implementation.
 2May01 - J.W. Larson - added initgg_ API (replaces old initv_).
13Dec01 - J.W. Larson - added import and export methods.
27Mar02 - J.W. Larson <larson@mcs.anl.gov> - Corrected usage of
          m_die routines throughout this module.
 5Aug02 - E. Ong <eong@mcs.anl.gov> - Modified GeneralGrid usage
          to allow user-defined grid numbering schemes.
```

5.1.1 init_ - Create an Empty GeneralGrid

The routine `init_()` creates the storage space for grid point coordinates, area/volume weights, and other coordinate data (*e.g.*, local cell dimensions). These data are referenced by `List` components that are also created by this routine (see the documentation of the declaration section of this module for more details about setting list information). Each of the input `CHARACTER` arguments is a colon-delimited string of attribute names, each corrsponding to a `List` element of the output `GeneralGrid` argument `GGrid`, and are summarized in the table below:
The input `INTEGER` argument `lsize` defines the number of grid points to be stored in `GGrid`.
If a set of sorting keys is supplied in the argument `CoordSortOrder`, the user can control whether the sorting by each key is in descending or ascending order by supplying the input `LOGICAL` array

Argument	Component of GGrid	Significance	Required?
CoordChars	GGrid%coordinate_list	Dimension Names	Yes
CoordSortOrder	GGrid%coordinate_sort_order	Grid Point Sorting Keys	No
WeightChars	GGrid%weight_list	Grid Cell Length, Area, and Volume Weights	No
OtherChars	GGrid%other_list	All Other Real Attributes	No
IndexChars	GGrid%index_list	All Other Integer Attributes	No

descend(:). By default, all sorting is in *ascending* order for each key if the argument descend is not provided.
N.B.: The output GeneralGrid GGrid is dynamically allocated memory. When one no longer needs GGrid, one should release this space by invoking clean() for the GeneralGrid.

INTERFACE:

```
subroutine init_(GGrid, CoordChars, CoordSortOrder, descend, WeightChars, &
                 OtherChars, IndexChars, lsize )
```

USES:

```
use m_stdio
use m_die

use m_List,     only : List
use m_List,     only : List_init => init
use m_List,     only : List_nitem => nitem
use m_List,     only : List_shared => GetSharedListIndices
use m_List,     only : List_append => append
use m_List,     only : List_copy => copy
use m_List,     only : List_nullify => nullify
use m_List,     only : List_clean => clean

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init

implicit none
```

INPUT PARAMETERS:

```
character(len=*),                intent(in) :: CoordChars
character(len=*),      optional, intent(in) :: CoordSortOrder
character(len=*),      optional, intent(in) :: WeightChars
logical, dimension(:), optional, pointer    :: descend
character(len=*),      optional, intent(in) :: OtherChars
character(len=*),      optional, intent(in) :: IndexChars
integer,               optional, intent(in) :: lsize
```

OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(out)   :: GGrid
```

REVISION HISTORY:

```
25Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
15Jan01 - Jay Larson <larson@mcs.anl.gov> - modified to fit
          new GeneralGrid definition.
19Mar01 - Jay Larson <larson@mcs.anl.gov> - added OtherChars
25Apr01 - Jay Larson <larson@mcs.anl.gov> - added GlobGridNum
          as a mandatory integer attribute.
13Jun01 - Jay Larson <larson@mcs.anl.gov> - No longer define
          blank List attributes of the GeneralGrid.  Previous
          versions of this routine had this feature, and this
          caused problems with the GeneralGrid Send and Receive
          operations on the AIX platform.
13Jun01 - R. Jacob <jacob@mcs.anl.gov> - nullify any pointers
          for lists not declared.
15Feb02 - Jay Larson <larson@mcs.anl.gov> - made the input
          argument CoordSortOrder mandatory (rather than
          optional).
18Jul02 - E. Ong <eong@mcs.anl.gov> - replaced this version of
          init with one that calls initl_.
 5Aug02 - E. Ong <eong@mcs.anl.gov> - made the input argument
          CoordSortOrder optional to allow user-defined grid
          numbering schemes.
```

5.1.2 initl_ - Create an Empty GeneralGrid from Lists

The routine `initl_()` creates the storage space for grid point coordinates, area/volume weights, and other coordinate data (*e.g.*, local cell dimensions). These data are referenced by `List` components that are also created by this routine (see the documentation of the declaration section of this module for more details about setting list information). Each of the input `List` arguments is used directly to create the corresponding `List` element of the output `GeneralGrid` argument `GGrid`, and are summarized in the table below:

Argument	**Component of** `GGrid`	**Significance**	**Required?**
`CoordList`	`GGrid%coordinate_list`	Dimension Names	Yes
`CoordSortOrder`	`GGrid%coordinate_sort_order`	Grid Point Sorting Keys	No
`WeightList`	`GGrid%weight_list`	Grid Cell Length, Area, and Volume Weights	No
`OtherList`	`GGrid%other_list`	All Other Real Attributes	No
`IndexList`	`GGrid%index_list`	All Other Integer Attributes	No

The input `INTEGER` argument `lsize` defines the number of grid points to be stored in `GGrid`.
If a set of sorting keys is supplied in the argument `CoordSortOrder`, the user can control whether the sorting by each key is in descending or ascending order by supplying the input `LOGICAL` array `descend(:)`. By default, all sorting is in *ascending* order for each key if the argument `descend` is not provided.
N.B.: The output `GeneralGrid` `GGrid` is dynamically allocated memory. When one no longer needs `GGrid`, one should release this space by invoking `clean()` for the `GeneralGrid`.

INTERFACE:

```
subroutine initl_(GGrid, CoordList, CoordSortOrder, descend, WeightList, &
                  OtherList, IndexList, lsize )
```

USES:

```
use m_stdio
use m_die

use m_List,     only : List
use m_List,     only : List_init => init
use m_List,     only : List_allocated => allocated
use m_List,     only : List_nitem => nitem
use m_List,     only : List_shared => GetSharedListIndices
use m_List,     only : List_append => append
use m_List,     only : List_copy => copy
use m_List,     only : List_nullify => nullify
use m_List,     only : List_clean => clean

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init

implicit none
```

INPUT PARAMETERS:

```
Type(List),                     intent(in)  :: CoordList
Type(List),           optional, intent(in)  :: CoordSortOrder
Type(List),           optional, intent(in)  :: WeightList
logical, dimension(:), optional, pointer    :: descend
Type(List),           optional, intent(in)  :: OtherList
Type(List),           optional, intent(in)  :: IndexList
integer,              optional, intent(in)  :: lsize
```

OUTPUT PARAMETERS:

```
type(GeneralGrid),              intent(out) :: GGrid
```

REVISION HISTORY:

```
10May01 - Jay Larson <larson@mcs.anl.gov> - initial version
 8Aug01 - E.T. Ong <eong@mcs.anl.gov> - changed list assignment(=)
          to list copy to avoid compiler bugs with pgf90
17Jul02 - E. Ong <eong@mcs.anl.gov> - general revision;
          added error checks
 5Aug02 - E. Ong <eong@mcs.anl.gov> - made input argument
          CoordSortOrder optional to allow for user-defined
          grid numbering schemes
```

5.1.3 initgg_ - Create a GeneralGrid from Another

The routine `initgg_()` creates the storage space for grid point coordinates, area/volume weights, and other coordinate data (*e.g.*, nearest-neighbor coordinates). These data are all copied from the already initialized input `GeneralGrid` argument `iGGrid`. This routine initializes the output `GeneralGrid` argument `oGGrid` with the same `List` data as `iGGrid`, but with storage space for `lsize` gridpoints.
N.B.: Though the attribute lists and gridpoint sorting strategy of `iGGrid` is copied to `oGGrid`, the actual values of the attributes are not.
N.B.: It is assumed that `iGGrid` has been initialized.
N.B.: The output `GeneralGrid oGGrid` is dynamically allocated memory. When one no longer needs `oGGrid`, one should release this space by invoking `GeneralGrid_clean()`.

INTERFACE:

```
subroutine initgg_(oGGrid, iGGrid, lsize)
```

USES:

```
use m_stdio
use m_die

use m_List, only : List
use m_List, only : List_allocated => allocated
use m_List, only : List_copy => copy
use m_List, only : List_nitems => nitem
use m_List, only : List_nullify => nullify

use m_AttrVect, only:  AttrVect
use m_AttrVect, only:  AttrVect_init => init

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid), intent(in)  :: iGGrid
integer, optional, intent(in)  :: lsize
```

OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(out) :: oGGrid
```

REVISION HISTORY:

```
 2May01 - Jay Larson <larson@mcs.anl.gov> - Initial version.
13Jun01 - Jay Larson <larson@mcs.anl.gov> - Now, undefined List
          components of the GeneralGrid iGGrid are no longer
          copied to oGGrid.
 8Aug01 - E.T. Ong <eong@mcs.anl.gov> - changed list assignment(=)
          to list copy to avoid compiler bugs with pgf90
24Jul02 - E.T. Ong <eong@mcs.anl.gov> - updated this init version
          to correspond with initl_
 5Aug02 - E. Ong <eong@mcs.anl.gov> - made input argument
          CoordSortOrder optional to allow for user-defined
          grid numbering schemes
```

5.1.4 initCartesianSP_ - Initialize a Cartesian GeneralGrid

The routine `initCartesian_()` creates the storage space for grid point coordinates, area and volume weights, and other coordinate data (*e.g.*, cell area and volume weights). The names of the Cartesian axes are supplied by the user as a colon-delimitted string in the input `CHARACTER` argument `CoordChars`. For example, a Cartesian grid for Euclidian 3-space would have `CoordChars` = $'$`x:y:z`$'$. The user can define named real attributes for spatial weighting data in the input `CHARACTER` argument `WeightChars`. For example, one could define attributes for Euclidean 3-space length elements by setting `WeightChars` = $'$`dx:dy:dz`$'$. The input `CHARCTER` argument `OtherChars` provides space for defining other real attributes (again as a colon-delimited string of attribute names). One can define integer attributes by supplying a colon-delimitted string of names in the input `CHARACTER` argument `IndexChars`. For example, on could set aside storage space for the `x`-, `y`-, and `z`-indices by setting `IndexChars` = $'$`xIndex:yIndex:zIndex`$'$.
Once the storage space in `GGrid` is initialized, The gridpoint coordinates are evaluated using the input arguments `Dims` (the number of points on each coordinate axis) and `AxisData` (the coordinate values on all of the points of all of the axes). The user presents the axes with each axis stored

in a column of AxisData, and the axes are laid out in the same order as the ordering of the axis names in CoordChars. The number of points on each axis is defined by the entries of the input INTEGER array Dims(:). Continuing with the Euclidean 3-space example given above, setting $\texttt{Dims}(1:3) = (256, 256, 128)$ will result in a Cartesian grid with 256 points in the x- and y-directions, and 128 points in the z-direction. Thus the appropriate dimensions of AxisData are 256 rows (the maximum number of axis points among all the axes) by 3 columns (the number of physical dimensions). The x-axis points are stored in AxisData(1:256,1), the y-axis points are stored in AxisData(1:256,2), and the z-axis points are stored in AxisData(1:128,3).
The sorting order of the gridpoints can be either user-defined, or set automatically by MCT. If the latter is desired, the user must supply the argument CoordSortOrder, which defines the lexicographic ordering (by coordinate). The entries optional input LOGICAL array descend(:) stipulates whether the ordering with respect to the corresponding key in CoordChars is to be *descending*. If CoordChars is supplied, but descend(:) is not, the gridpoint information is placed in *ascending* order for each key. Returning to our Euclidian 3-space example, a choice of $\texttt{CoordSortOrder} = \texttt{y} : \texttt{x} : \texttt{z}$ and $\texttt{descend}(1:3) = (\texttt{.TRUE.}, \texttt{.FALSE.}, \texttt{.FALSE.})$ will result in the entries of GGrid being orderd lexicographically by y (in descending order), x (in ascending order), and z (in ascending order). Regardless of the gridpoint sorting strategy, MCT will number each of the gridpoints in GGrid, storing this information in the integer attribute named 'GlobGridNum'.

INTERFACE:

```
subroutine initCartesianSP_(GGrid, CoordChars, CoordSortOrder, descend, &
                          WeightChars, OtherChars, IndexChars, Dims, &
                          AxisData)
```

USES:

```
use m_stdio
use m_die
use m_realkinds,  only : SP

use m_String,     only : String
use m_String,     only : String_ToChar => ToChar
use m_String,     only : String_clean => clean

use m_List,     only : List
use m_List,     only : List_init => init
use m_List,     only : List_clean => clean
use m_List,     only : List_nullify => nullify
use m_List,     only : List_append => append
use m_List,     only : List_nitem => nitem
use m_List,     only : List_get => get
use m_List,     only : List_shared => GetSharedListIndices

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero

implicit none
```

INPUT PARAMETERS:

```
character(len=*),                 intent(in)  :: CoordChars
character(len=*),        optional, intent(in)  :: CoordSortOrder
character(len=*),        optional, intent(in)  :: WeightChars
logical, dimension(:),   optional, pointer     :: descend
character(len=*),        optional, intent(in)  :: OtherChars
character(len=*),        optional, intent(in)  :: IndexChars
integer, dimension(:),             pointer     :: Dims
real(SP), dimension(:,:),          pointer     :: AxisData
```

OUTPUT PARAMETERS:

```
type(GeneralGrid),                intent(out) :: GGrid
```

REVISION HISTORY:

```
 7Jun01 - Jay Larson <larson@mcs.anl.gov> - API Specification.
12Aug02 - Jay Larson <larson@mcs.anl.gov> - Implementation.
```

5.1.5 initUnstructuredSP_ - Initialize an Unstructured GeneralGrid

This routine creates the storage space for grid point coordinates, area/volume weights, and other coordinate data (*e.g.*, local cell dimensions), and fills in user-supplied values for the grid point coordinates. These data are referenced by `List` components that are also created by this routine (see the documentation of the declaration section of this module for more details about setting list information). Each of the input `CHARACTER` arguments is a colon-delimited string of attribute names, each corrsponding to a `List` element of the output `GeneralGrid` argument `GGrid`, and are summarized in the table below:

Argument	**Component of `GGrid`**	**Significance**	**Required?**
`CoordChars`	`GGrid%coordinate_list`	Dimension Names	Yes
`CoordSortOrder`	`GGrid%coordinate_sort_order`	Grid Point Sorting Keys	No
`WeightChars`	`GGrid%weight_list`	Grid Cell Length, Area, and Volume Weights	No
`OtherChars`	`GGrid%other_list`	All Other Real Attributes	No
`IndexChars`	`GGrid%index_list`	All Other Integer Attributes	No

The number of physical dimensions of the grid is set by the user in the input `INTEGER` argument `nDims`, and the number of grid points stored in `GGrid` is set using the input `INTEGER` argument `nPoints`. The grid point coordinates are input via the `REAL` array `PointData(:)`. The number of entries in `PointData` must equal the product of `nDims` and `nPoints`. The grid points are grouped in `nPoints` consecutive groups of `nDims` entries, with the coordinate values for each point set in the same order as the dimensions are named in the list `CoordChars`.
If a set of sorting keys is supplied in the argument `CoordSortOrder`, the user can control whether the sorting by each key is in descending or ascending order by supplying the input `LOGICAL` array `descend(:)`. By default, all sorting is in *ascending* order for each key if the argument `descend` is not provided.
N.B.: The output `GeneralGrid GGrid` is dynamically allocated memory. When one no longer needs `GGrid`, one should release this space by invoking `clean()` for the `GeneralGrid`.

INTERFACE:

```
subroutine initUnstructuredSP_(GGrid, CoordChars, CoordSortOrder, descend, &
                               WeightChars, OtherChars, IndexChars, nDims, &
                               nPoints, PointData)
```

USES:

```
use m_stdio
use m_die
use m_realkinds,only : SP
```

```
use m_String,   only : String, char
use m_List,     only : List
use m_List,     only : List_init => init
use m_List,     only : List_clean => clean
use m_List,     only : List_nitem => nitem
use m_List,     only : List_nullify => nullify
use m_List,     only : List_copy => copy
use m_List,     only : List_append => append
use m_List,     only : List_shared => GetSharedListIndices
use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero

implicit none
```

INPUT PARAMETERS:

```
character(len=*),              intent(in) :: CoordChars
character(len=*), optional,    intent(in) :: CoordSortOrder
character(len=*), optional,    intent(in) :: WeightChars
logical, dimension(:), optional, pointer :: descend
character(len=*), optional,    intent(in) :: OtherChars
character(len=*), optional,    intent(in) :: IndexChars
integer,                       intent(in) :: nDims
integer,                       intent(in) :: nPoints
real(SP), dimension(:),        pointer    :: PointData
```

OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(out)   :: GGrid
```

REVISION HISTORY:

```
 7Jun01 - Jay Larson <larson@mcs.anl.gov> - API specification.
22Aug02 - J. Larson <larson@mcs.anl.gov> - Implementation.
```

5.1.6 clean_ - Destroy a GeneralGrid

This routine deallocates all attribute storage space for the input/output `GeneralGrid` argument `GGrid`, and destroys all of its `List` components and sorting flags. The success (failure) of this operation is signified by the zero (non-zero) value of the optional `INTEGER` output argument `stat`.

INTERFACE:

```
subroutine clean_(GGrid, stat)
```

USES:

```
use m_stdio
use m_die

use m_List,     only : List_clean => clean
use m_List,     only : List_allocated => allocated
use m_AttrVect, only : AttrVect_clean => clean

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(inout) :: GGrid
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
25Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
20Mar01 - J.W. Larson <larson@mcs.anl.gov> - complete version.
 1Mar01 - E.T. Ong <eong@mcs.anl.gov> - removed dies to prevent
          crashes when cleaning uninitialized attrvects. Added
          optional stat argument.
 5Aug02 - E. Ong <eong@mcs.anl.gov> - a more rigorous revision
```

5.1.7 zero_ - Set GeneralGrid Data to Zero

This routine sets all of the point values of the integer and real attributes of an the input/output `GeneralGrid` argument `GGrid` to zero. The default action is to set the values of all the real and integer attributes to zero.

INTERFACE:

```
subroutine zero_(GGrid, zeroReals, zeroInts)
```

USES:

```
use m_die,only     : die
use m_stdio,only   : stderr

use m_AttrVect, only : AttrVect_zero => zero

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(GeneralGrid),    intent(INOUT) :: GGrid
```

INPUT PARAMETERS:

```
logical, optional, intent(IN)    :: zeroReals
logical, optional, intent(IN)    :: zeroInts
```

REVISION HISTORY:

```
11May08 - R. Jacob <jacob@mcs.anl.gov> - initial prototype/code
```

5.1.8 dims_ - Return the Dimensionality of a GeneralGrid

This `INTEGER` function returns the number of physical dimensions of the input `GeneralGrid` argument `GGrid`.

INTERFACE:

```
integer function dims_(GGrid)
```

USES:

```
use m_stdio
use m_die

use m_List,     only : List_nitem => nitem

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid), intent(in)  :: GGrid
```

REVISION HISTORY:

15Jan01 - Jay Larson <larson@mcs.anl.gov> - initial version

5.1.9 indexIA - Index an Integer Attribute

This function returns an `INTEGER`, corresponding to the location of an integer attribute within the input `GeneralGrid` argument `GGrid`. For example, every `GGrid` has at least one integer attribute (namely the global gridpoint index `'GlobGridNum'`). The array of integer values for the attribute `'GlobGridNum'` is stored in

{\tt GGrid%data%iAttr(indexIA_(GGrid,'GlobGridNum'),:)}.

If `indexIA_()` is unable to match `item` to any of the integer attributes present in `GGrid`, the resulting value is zero which is equivalent to an error. The optional input `CHARACTER` arguments `perrWith` and `dieWith` control how such errors are handled. Below are the rules how error handling is controlled by using `perrWith` and `dieWith`:

1. if neither `perrWith` nor `dieWith` are present, `indexIA_()` terminates execution with an internally generated error message;
2. if `perrWith` is present, but `dieWith` is not, an error message is written to `stderr` incorporating user-supplied traceback information stored in the argument `perrWith`;
3. if `dieWith` is present, execution terminates with an error message written to `stderr` that incorporates user-supplied traceback information stored in the argument `dieWith`; and
4. if both `perrWith` and `dieWith` are present, execution terminates with an error message using `dieWith`, and the argument `perrWith` is ignored.

INTERFACE:

```
integer function indexIA_(GGrid, item, perrWith, dieWith)
```

USES:

```
use m_die
use m_stdio

use m_String, only : String
use m_String, only : String_init => init
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => ToChar
```

```
use m_TraceBack, only : GenTraceBackString

use m_AttrVect,     only : AttrVect_indexIA => indexIA

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid),          intent(in) :: GGrid
character(len=*),           intent(in) :: item
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - Initial version.
27Mar02 - Jay Larson <larson@mcs.anl.gov> - Cleaned up error
          handling logic.
 2Aug02 - Jay Larson <larson@mcs.anl.gov> - Further refinement
          of error handling.
```

5.1.10 indexRA - Index a Real Attribute

This function returns an `INTEGER`, corresponding to the location of an integer attribute within the input `GeneralGrid` argument `GGrid`. For example, every `GGrid` has at least one integer attribute (namely the global gridpoint index `'GlobGridNum'`). The array of integer values for the attribute `'GlobGridNum'` is stored in

{\tt GGrid%data%iAttr(indexRA_(GGrid,'GlobGridNum'),:)}.

If `indexRA_()` is unable to match `item` to any of the integer attributes present in `GGrid`, the resulting value is zero which is equivalent to an error. The optional input `CHARACTER` arguments `perrWith` and `dieWith` control how such errors are handled. Below are the rules how error handling is controlled by using `perrWith` and `dieWith`:

1. if neither `perrWith` nor `dieWith` are present, `indexRA_()` terminates execution with an internally generated error message;
2. if `perrWith` is present, but `dieWith` is not, an error message is written to `stderr` incorporating user-supplied traceback information stored in the argument `perrWith`;
3. if `dieWith` is present, execution terminates with an error message written to `stderr` that incorporates user-supplied traceback information stored in the argument `dieWith`; and
4. if both `perrWith` and `dieWith` are present, execution terminates with an error message using `dieWith`, and the argument `perrWith` is ignored.

INTERFACE:

```
integer function indexRA_(GGrid, item, perrWith, dieWith)
```

USES:

```
use m_stdio
use m_die

use m_String, only : String
use m_String, only : String_init => init
```

```
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => ToChar

use m_TraceBack, only : GenTraceBackString

use m_AttrVect,     only : AttrVect_indexRA => indexRA

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid),          intent(in)  :: GGrid
character(len=*),           intent(in)  :: item
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - Initial version.
27Mar02 - Jay Larson <larson@mcs.anl.gov> - Cleaned up error
          handling logic.
```

5.1.11 lsize - Number of Grid Points

This `INTEGER` function returns the number of grid points stored in the input `GeneralGrid` argument `GGrid`. Note that the value returned will be the number of points stored on a local process in the case of a distributed `GeneralGrid`.

INTERFACE:

```
integer function lsize_(GGrid)
```

USES:

```
use m_List,     only : List
use m_List,     only : List_allocated => allocated
use m_AttrVect, only : AttrVect_lsize => lsize
use m_die,      only : die


implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid), intent(in)  :: GGrid
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - Initial version.
27Mar02 - Jay Larson <larson@mcs.anl.gov> - slight logic change.
27Mar02 - Jay Larson <larson@mcs.anl.gov> - Bug fix and use of
          List_allocated() function to check for existence of
          attributes.
 5Aug02 - E. Ong <eong@mcs.anl.gov> - more rigorous revision
```

5.1.12 exportIAttr_ - Return GeneralGrid INTEGER Attribute as a Vector

This routine extracts from the input `GeneralGrid` argument `GGrid` the integer attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`, and returns it in the `INTEGER` output array `outVect`, and its length in the output `INTEGER` argument `lsize`.
N.B.: This routine will fail if the `AttrTag` is not in the `GeneralGrid List` component `GGrid%data%iList`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `outVect` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `outVect`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) before this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `outVect`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportIAttr_(GGrid, AttrTag, outVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_exportIAttr => exportIAttr

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid),      intent(in)  :: GGrid
character(len=*),       intent(in)  :: AttrTag
```

OUTPUT PARAMETERS:

```
integer,  dimension(:), pointer     :: outVect
integer,  optional,     intent(out) :: lsize
```

REVISION HISTORY:

```
13Dec01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

5.1.13 exportRAttrSP_ - Return GeneralGrid REAL Attribute as a Vector

This routine extracts from the input `GeneralGrid` argument `GGrid` the real attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`, and returns it in the `REAL` output array `outVect`, and its length in the output `INTEGER` argument `lsize`.
N.B.: This routine will fail if the `AttrTag` is not in the `GeneralGrid List` component `GGrid%data%rList`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `outVect` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `outVect`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) before this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `outVect`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportRAttrSP_(GGrid, AttrTag, outVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_realkinds,  only : SP

use m_AttrVect,   only : AttrVect_exportRAttr => exportRAttr

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid),          intent(in)  :: GGrid
character(len=*),           intent(in)  :: AttrTag
```

OUTPUT PARAMETERS:

```
real(SP),  dimension(:),    pointer     :: outVect
integer,   optional,        intent(out) :: lsize
```

REVISION HISTORY:

```
13Dec01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

5.1.14 importIAttr_ - Import GeneralGrid INTEGER Attribute

This routine imports data provided in the input `INTEGER` vector `inVect` into the `GeneralGrid` argument `GGrid`, storing it as the integer attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`. The input `INTEGER` argument `lsize` is used to ensure there is sufficient space in the `GeneralGrid` to store the data.
N.B.: This routine will fail if the `AttrTag` is not in the `GeneralGrid` `List` component `GGrid%data%iList`.

INTERFACE:

```
subroutine importIAttr_(GGrid, AttrTag, inVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_importIAttr => importIAttr

implicit none
```

INPUT PARAMETERS:

```
character(len=*),       intent(in)    :: AttrTag
integer,  dimension(:), pointer       :: inVect
integer,                intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(GeneralGrid),      intent(inout) :: GGrid
```

REVISION HISTORY:

```
13Dec01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
27Mar02 - Jay Larson <larson@mcs.anl.gov> - improved error handling.
```

5.1.15 importRAttrSP_ - Import GeneralGrid REAL Attribute

This routine imports data provided in the input `REAL` vector `inVect` into the `GeneralGrid` argument `GGrid`, storing it as the real attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`. The input `INTEGER` argument `lsize` is used to ensure there is sufficient space in the `GeneralGrid` to store the data.
N.B.: This routine will fail if the `AttrTag` is not in the `GeneralGrid` `List` component `GGrid%data%rList`.

INTERFACE:

```
subroutine importRAttrSP_(GGrid, AttrTag, inVect, lsize)
```

USES:

```
use m_die ,          only : die
use m_die ,          only : MP_perr_die
use m_stdio ,        only : stderr

use m_realkinds,     only : SP

use m_AttrVect,      only : AttrVect_importRAttr => importRAttr

implicit none
```

INPUT PARAMETERS:

```
character(len=*),           intent(in)    :: AttrTag
real(SP), dimension(:),     pointer       :: inVect
integer,                    intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(GeneralGrid),          intent(inout) :: GGrid
```

REVISION HISTORY:

```
13Dec01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
27Mar02 - Jay Larson <larson@mcs.anl.gov> - improved error handling.
```

5.1.16 Sort_ - Generate Sort Permutation Defined by Arbitrary Keys.

The subroutine `Sort_()` uses the list of keys present in the input `List` variable `key_List`. This list of keys is checked to ensure that *only* coordinate attributes are present in the sorting keys, and that there are no redundant keys. Once checked, this list is used to find the appropriate real attributes referenced by the items in `key_list` (that is, it identifies the appropriate entries in `GGrid%data%rList`), and then uses these keys to generate a an output permutation `perm` that will put the entries of the attribute vector `GGrid%data` in lexicographic order as defined by `key_list` (the ordering in `key_list` being from left to right.

INTERFACE:

```
subroutine Sort_(GGrid, key_List, perm, descend)
```

USES:

```
use m_stdio
use m_die

use m_AttrVect,     only : AttrVect_Sort => Sort
use m_List,        only : List_nitem => nitem

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid),               intent(in) :: GGrid
type(List),                      intent(in) :: key_list
logical, dimension(:), optional, intent(in) :: descend
```

OUTPUT PARAMETERS:

```
integer, dimension(:), pointer              :: perm
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - Initial version.
20Mar01 - Jay Larson <larson@mcs.anl.gov> - Final working version.
```

5.1.17 Sortg_ - Generate Sort Permutation Based on GeneralGrid Keys.

The subroutine `Sortg_()` uses the list of sorting keys present in the input `GeneralGrid` variable `GGrid%coordinate_sort_order` to create a sort permutation `perm(:)`. Sorting is either in ascending or descending order based on the entries of `GGrid%descend(:)`. The output index permutation is stored in the array `perm(:)` that will put the entries of the attribute vector `GGrid%data` in lexicographic order as defined by `GGrid%coordinate_sort_order`. The ordering in `GGrid%coordinate_sort_order` being from left to right.
N.B.: This routine returnss an allocatable array perm(:). This allocated array must be deallocated when the user no longer needs it. Failure to do so will cause a memory leak.
N.B.: This routine will fail if `GGrid` has not been initialized with sort keys in the `List` component `GGrid%coordinate_sort_order`.

INTERFACE:

```
subroutine Sortg_(GGrid, perm)
```

USES:

```
use m_List, only : List_allocated => allocated
use m_die,  only : die

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid),     intent(in) :: GGrid
```

OUTPUT PARAMETERS:

```
integer, dimension(:), pointer    :: perm
```

REVISION HISTORY:

```
22Mar01 - Jay Larson <larson@mcs.anl.gov> - Initial version.
 5Aug02 - E. Ong <eong@mcs.anl.gov> - revise with more error checking.
```

5.1.18 Permute_ - Permute GeneralGrid Attributes Using Supplied Index Permutation

The subroutine `Permute_()` uses an input index permutation `perm` to re-order the coordinate data stored in the `GeneralGrid` argument `GGrid`. This permutation can be generated by either of the routines `Sort_()` or `Sortg_()` contained in this module.

INTERFACE:

```
subroutine Permute_(GGrid, perm)
```

USES:

```
use m_stdio
use m_die

use m_AttrVect,     only : AttrVect
use m_AttrVect, only : AttrVect_Permute => Permute

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:), intent(in)    :: perm
```

INPUT/OUTPUT PARAMETERS:

```
type(GeneralGrid),     intent(inout) :: GGrid
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
10Apr01 - Jay Larson <larson@mcs.anl.gov> - API modified, working
          code.
```

5.1.19 SortPermute_ - Sort and Permute GeneralGrid Attributes

The subroutine `SortPermute_()` uses the list of keys defined in `GGrid%coordinate_sort_order` to create an index permutation `perm`, which is then applied to re-order the coordinate data stored in the `GeneralGrid` argument `GGrid` (more specifically, the gridpoint data stored in `GGrid%data`. This permutation is generated by the routine `Sortg_()` contained in this module. The permutation is carried out by the routine `Permute_()` contained in this module.
N.B.: This routine will fail if `GGrid` has not been initialized with sort keys in the `List` component `GGrid%coordinate_sort_order`.

INTERFACE:

```
subroutine SortPermute_(GGrid)
```

USES:

```
use m_stdio
use m_die

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(GeneralGrid),     intent(inout)   :: GGrid
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
10Apr01 - Jay Larson <larson@mcs.anl.gov> - API modified, working
          code.
13Apr01 - Jay Larson <larson@mcs.anl.gov> - Simplified API and
          code (Thanks to Tony Craig of NCAR for detecting the
          bug that inspired these changes).
```

5.2 Module m_GeneralGridComms - Communications for the GeneralGrid type. (Source File: m_GeneralGridComms.F90)

In this module, we define communications methods specific to the `GeneralGrid` class (see the module `m_GeneralGrid` for more information about this class and its methods).

INTERFACE:

```
module m_GeneralGridComms
```

USES:

```
use m_GeneralGrid ! GeneralGrid class and its methods


implicit none

private   ! except

public :: gather          ! gather all local vectors to the root
public :: scatter         ! scatter from the root to all PEs
public :: bcast           ! bcast from root to all PEs
public :: send            ! Blocking SEND
public :: recv            ! Blocking RECEIVE

interface gather ; module procedure &
          GM_gather_, &
          GSM_gather_
end interface
interface scatter ; module procedure &
          GM_scatter_, &
          GSM_scatter_
end interface
interface bcast ; module procedure bcast_ ; end interface
interface send  ; module procedure send_  ; end interface
interface recv  ; module procedure recv_  ; end interface
```

REVISION HISTORY:

```
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - Initial module/APIs
07Jun01 - J.W. Larson <larson@mcs.anl.gov> - Added point-to-point
27Mar02 - J.W. Larson <larson@mcs.anl.gov> - Overhaul of error
          handling calls throughout this module.
05Aug02 - E. Ong <eong@mcs.anl.gov> - Added buffer association
          error checks to avoid making bad MPI calls
```

5.2.1 send_ - Point-to-point blocking send for the GeneralGrid.

The point-to-point send routine `send_()` sends the input `GeneralGrid` argument `iGGrid` to component `comp_id`. The message is identified by the tag defined by the `INTEGER` argument `TagBase`. The value of `TagBase` must match the value used in the call to `recv_()` on process `dest`. The success (failure) of this operation corresponds to a zero (nonzero) value for the output `INTEGER` flag `status`. The argument will be sent to the local root of the component.
N.B.: One must avoid assigning elsewhere the MPI tag values between `TagBase` and `TagBase+20`, inclusive. This is because `send_()` performs one send operation set up the header transfer, up to five `List_send` operations (two `MPI_SEND` calls in each), two send operations to transfer `iGGrid%descend(:)`, and finally the send of the `AttrVect` component `iGGrid%data` (which comprises eight `MPI_SEND` operations).

INTERFACE:

```
subroutine send_(iGGrid, comp_id, TagBase, status)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_init => init
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

use m_MCTWorld, only : ComponentToWorldRank
use m_MCTWorld, only : ThisMCTWorld

use m_AttrVectComms,only : AttrVect_send => send

use m_List, only : List_send => send
use m_List, only : List_allocated => allocated

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid), intent(in) :: iGGrid
integer,           intent(in) :: comp_id
integer,           intent(in) :: TagBase
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out) :: status
```

REVISION HISTORY:

```
04Jun01 - J.W. Larson <larson@mcs.anl.gov> - API Specification.
07Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
10Jun01 - J.W. Larson <larson@mcs.anl.gov> - Bug fixes--now works.
11Jun01 - R. Jacob <jacob@mcs.anl.gov> use component id as input
          argument.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize status
          (if present).
15Feb02 - J.W. Larson <larson@mcs.anl.gov> - Made input argument
          comm optional.
13Jun02 - J.W. Larson <larson@mcs.anl.gov> - Removed the argument
          comm.  This routine is now explicitly for intercomponent
          communications only.
```

5.2.2 recv_ - Point-to-point blocking recv for the GeneralGrid.

The point-to-point receive routine `recv_()` receives the output `GeneralGrid` argument `oGGrid` from component `comp_id`. The message is identified by the tag defined by the `INTEGER` argument `TagBase`. The value of `TagBase` must match the value used in the call to `send_()` on the other component. The success (failure) of this operation corresponds to a zero (nonzero) value for the output `INTEGER` flag `status`.
N.B.: This routine assumes that the `GeneralGrid` argument `oGGrid` is uninitialized on input; that is, all the `List` components are blank, the `LOGICAL` array `oGGrid%descend` is unallocated,

and the AttrVect component oGGrid%data is uninitialized. The GeneralGrid oGGrid represents allocated memory. When the user no longer needs oGGrid, it should be deallocated by invoking GeneralGrid_clean() (see m_GeneralGrid for further details).
N.B.: One must avoid assigning elsewhere the MPI tag values between TagBase and TagBase+20, inclusive. This is because recv_() performs one receive operation set up the header transfer, up to five List_recv operations (two MPI_RECV calls in each), two receive operations to transfer iGGrid%descend(:), and finally the receive of the AttrVect component iGGrid%data (which comprises eight MPI_RECV operations).

INTERFACE:

```
subroutine recv_(oGGrid, comp_id, TagBase, status)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_init => init
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

use m_MCTWorld, only : ComponentToWorldRank
use m_MCTWorld, only : ThisMCTWorld

use m_AttrVectComms,only : AttrVect_recv => recv

use m_List,only : List_recv => recv
use m_List,only : List_nullify => nullify

implicit none
```

INPUT PARAMETERS:

```
integer,           intent(in) :: comp_id
integer,           intent(in) :: TagBase
```

OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(out) :: oGGrid
integer, optional, intent(out) :: status
```

REVISION HISTORY:

```
04Jun01 - J.W. Larson <larson@mcs.anl.gov> - API Specification.
07Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
10Jun01 - J.W. Larson <larson@mcs.anl.gov> - Bug fixes--now works.
11Jun01 - R. Jacob <jacob@mcs.anl.gov> use component id as input
          argument.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize status
          (if present).
13Jun02 - J.W. Larson <larson@mcs.anl.gov> - Removed the argument
          comm.  This routine is now explicitly for intercomponent
          communications only.
```

5.2.3 GM_gather_ - gather a GeneralGrid using input GlobalMap.

GM_gather_() takes an input GeneralGrid argument iG whose decomposition on the communicator associated with the F90 handle comm is described by the GlobalMap argument GMap, and gathers it to the GeneralGrid output argument oG on the root. The success (failure) of this operation is reported as a zero (nonzero) value in the optional INTEGER output argument stat.
N.B.: An important assumption made here is that the distributed GeneralGrid iG has been initialized with the same coordinate system, sort order, other real attributes, and the same indexing attributes for all processes on comm.
N.B.: Once the gridpoint data of the GeneralGrid are assembled on the root, they are stored in the order determined by the input GlobalMap GMap. The user may need to sorted these gathered data to order them in accordance with the coordinate_sort_order attribute of iG.
N.B.: The output GeneralGrid oG represents allocated memory on the root. When the user no longer needs oG it should be deallocated using GeneralGrid_clean() to avoid a memory leak

INTERFACE:

```
subroutine GM_gather_(iG, oG, GMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalMap, only : GlobalMap
use m_GlobalMap, only : GlobalMap_gsize => gsize

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_init => init

use m_AttrVectComms,only : AttrVect_Gather => gather

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid), intent(in)  :: iG
type(GlobalMap),   intent(in)  :: GMap
integer,           intent(in)  :: root
integer,           intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(out) :: oG
integer, optional, intent(out) :: stat
```

REVISION HISTORY:

```
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - API Specification.
02May01 - J.W. Larson <larson@mcs.anl.gov> - Initial code.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize stat
          (if present).
```

5.2.4 GSM_gather_ - gather a GeneralGrid using input GlobalSegMap.

GMS_gather_() takes an input GeneralGrid argument iG whose decomposition on the communicator associated with the F90 handle comm is described by the GlobalSegMap argument GSMap, and gathers

it to the GeneralGrid output argument oG on the root. The success (failure) of this operation is reported as a zero (nonzero) value in the optional INTEGER output argument stat.
N.B.: An important assumption made here is that the distributed GeneralGrid iG has been initialized with the same coordinate system, sort order, other real attributes, and the same indexing attributes for all processes on comm.
N.B.: Once the gridpoint data of the GeneralGrid are assembled on the root, they are stored in the order determined by the input GlobalSegMap GSMap. The user may need to sorted these gathered data to order them in accordance with the coordinate_sort_order attribute of iG.
N.B.: The output GeneralGrid oG represents allocated memory on the root. When the user no longer needs oG it should be deallocated using GeneralGrid_clean() to avoid a memory leak

INTERFACE:

```
subroutine GSM_gather_(iG, oG, GSMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_lsize => lsize
use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_init => init
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

use m_AttrVectComms,only : AttrVect_Gather => gather

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid),  intent(in)  :: iG
type(GlobalSegMap), intent(in)  :: GSMap
integer,            intent(in)  :: root
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(GeneralGrid),  intent(out) :: oG
integer, optional,  intent(out) :: stat
```

REVISION HISTORY:

```
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - API Specification.
01May01 - J.W. Larson <larson@mcs.anl.gov> - Working Version.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize stat
          (if present).
```

5.2.5 GM_scatter_ - scatter a GeneralGrid using input GlobalMap.

GM_scatter_() takes an input GeneralGrid argument iG (valid only on the root process), and scatters it to the distributed GeneralGrid variable oG. The GeneralGrid oG is distributed on the communicator associated with the F90 handle comm using the domain decomposition described by

the GlobalMap argument GMap. The success (failure) of this operation is reported as a zero (nonzero) value in the optional INTEGER output argument stat.
N.B.: The output GeneralGrid oG represents allocated memory on the root. When the user no longer needs oG it should be deallocated using GeneralGrid_clean() to avoid a memory leak.

INTERFACE:

```
subroutine GM_scatter_(iG, oG, GMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalMap, only : GlobalMap
use m_GlobalMap, only : GlobalMap_lsize => lsize
use m_GlobalMap, only : GlobalMap_gsize => gsize

use m_AttrVectComms, only : AttrVect_scatter => scatter

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_init => init
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid), intent(in)  :: iG
type(GlobalMap),   intent(in)  :: GMap
integer,           intent(in)  :: root
integer,           intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(out) :: oG
integer, optional, intent(out) :: stat
```

REVISION HISTORY:

```
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - API Specification.
04Jun01 - J.W. Larson <larson@mcs.anl.gov> - Changed comms model
          to MPI-style (i.e. iG valid on root only).
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize stat
          (if present).
```

5.2.6 GSM_scatter_ - scatter a GeneralGrid using input GlobalSegMap.

GM_scatter_() takes an input GeneralGrid argument iG (valid only on the root process), and scatters it to the distributed GeneralGrid variable oG. The GeneralGrid oG is distributed on the communicator associated with the F90 handle comm using the domain decomposition described by the GlobalSegMap argument GSMap. The success (failure) of this operation is reported as a zero (nonzero) value in the optional INTEGER output argument stat.
N.B.: The output GeneralGrid oG represents allocated memory on the root. When the user no longer needs oG it should be deallocated using GeneralGrid_clean() to avoid a memory leak.

INTERFACE:

```
subroutine GSM_scatter_(iG, oG, GSMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_lsize => lsize
use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize

use m_AttrVectComms, only : AttrVect_scatter => scatter

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_init => init
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid),  intent(in)  :: iG
type(GlobalSegMap), intent(in)  :: GSMap
integer,            intent(in)  :: root
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(GeneralGrid),  intent(out) :: oG
integer, optional,  intent(out) :: stat
```

REVISION HISTORY:

```
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - API Specification.
04Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initial code.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize stat
          (if present).
```

5.2.7 bcast_ - Broadcast a GeneralGrid.

`bcast_()` takes an input `GeneralGrid` argument `ioG` (valid only on the `root` process), and broadcasts it to all processes on the communicator associated with the F90 handle `comm`. The success (failure) of this operation is reported as a zero (nonzero) value in the optional `INTEGER` output argument `stat`.
N.B.: On the non-root processes, the output `GeneralGrid ioG` represents allocated memory. When the user no longer needs `ioG` it should be deallocated by invoking `GeneralGrid_clean()`. Failure to do so risks a memory leak.

INTERFACE:

```
subroutine bcast_(ioG, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90
```

```
use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_lsize => lsize
use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_init => init
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

use m_AttrVectComms,only : AttrVect_bcast => bcast

implicit none
```

INPUT PARAMETERS:

```
integer,           intent(in)    :: root
integer,           intent(in)    :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(inout) :: ioG
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - API Specification.
02May01 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize stat
          (if present).
```

5.2.8 bcastGeneralGridHeader_ - Broadcast the GeneralGrid Header.

This routine broadcasts the header information from the input `GeneralGrid` argument `ioGGrid` (on input valid on the `root` only). This broadcast is from the `root` to all processes on the communicator associated with the fortran 90 `INTEGER` handle `comm`. The success (failure) of this operation corresponds to a zero (nonzero) value for the output `INTEGER` flag `stat`.
The *header information* in a `GeneralGrid` variable comprises all the non-`AttrVect` components of the `GeneralGrid`; that is, everything except the gridpoint coordinate, geometry, and index data stored in `iGGrid%data`. This information includes:

1. The coordinates in `iGGrid%coordinate_list`
2. The coordinate sort order in `iGGrid%coordinate_sort_order`
3. The area/volume weights in `iGGrid%weight_list`
4. Other `REAL` geometric information in `iGGrid%other_list`
5. Indexing information in `iGGrid%index_list`
6. The `LOGICAL` descending/ascending order sort flags in `iGGrid%descend(:)`.

INTERFACE:

```
subroutine bcastGeneralGridHeader_(ioGGrid, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_lsize => lsize
use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_init => init
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

use m_List, only : List
use m_List, only : List_allocated => allocated
use m_List, only : List_nullify => nullify
use m_List, only : List_bcast => bcast

implicit none
```

INPUT PARAMETERS:

```
integer,           intent(in)    :: root
integer,           intent(in)    :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(inout) :: ioGGrid
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
05Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initial code.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize stat
          (if present).
05Aug02 - E. Ong <eong@mcs.anl.gov> - added association checking
```

5.2.9 copyGeneralGridHeader_ - Copy the GeneralGrid Header.

This routine copies the header information from the input `GeneralGrid` argument `iGGrid` to the output `GeneralGrid` argument `oGGrid`. The *header information* in a `GeneralGrid` variable comprises all the non-`AttrVect` components of the `GeneralGrid`; that is, everything except the gridpoint coordinate, geometry, and index data stored in `iGGrid%data`. This information includes:

1. The coordinates in `iGGrid%coordinate_list`
2. The coordinate sort order in `iGGrid%coordinate_sort_order`
3. The area/volume weights in `iGGrid%weight_list`
4. Other `REAL` geometric information in `iGGrid%other_list`

5. Indexing information in `iGGrid%index_list`
6. The `LOGICAL` descending/ascending order sort flags in `iGGrid%descend(:)`.

INTERFACE:

```
subroutine copyGeneralGridHeader_(iGGrid, oGGrid)
```

USES:

```
use m_stdio
use m_die

use m_List, only : List
use m_List, only : List_copy => copy
use m_List, only : List_allocated => allocated
use m_List, only : List_nullify => nullify

use m_GeneralGrid, only : GeneralGrid

implicit none
```

INPUT PARAMETERS:

```
type(GeneralGrid), intent(in)  :: iGGrid
```

OUTPUT PARAMETERS:

```
type(GeneralGrid), intent(out) :: oGGrid
```

REVISION HISTORY:

```
05Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initial code.
08Aug01 - E.T. Ong <eong@mcs.anl.gov> - changed list assignments(=)
          to list copy.
05Aug02 - E. Ong <eong@mcs.anl.gov> - added association checking
```

6 The Navigator

6.1 Module m_Navigator - An Object for Indexing Segments of a Vector (Source File: m_Navigator.F90)

A *Navigator* is a table used to *index* or *Navigate* segments of a vector, or segments of a dimension of a higher-dimensional array. In MCT, this concept is embodied in the `Navigator` datatype, which contains the following components:

- The *number* of segments;
- The *displacement* of the starting index of each segment from the vector's first element (i.e. the starting index minus 1);
- The *length* of each segment; and
- The *total length* of the vector or array dimension for which segments are defined. This last item is optional, but if defined provides the ability for the `Navigator` to check for erroneous segment entries (i.e., segments that are out-of-bounds).

This module defines the `Navigator` datatype, creation and destruction methods, a variety of query methods, and a method for resizing the `Navigator`.

INTERFACE:

```
module m_Navigator
```

USES:

```
No external modules are used in the declaration section of this module.

    implicit none

    private ! except
```

PUBLIC TYPES:

```
    public :: Navigator ! The class data structure

  Type Navigator
    integer :: NumSegments ! Number of defined Segments
    integer :: VectorLength ! Length of the Vector being indexed
    integer,pointer,dimension(:) :: displs ! Segment start displacements
    integer,pointer,dimension(:) :: counts ! Segment lengths
  End Type Navigator
```

PUBLIC MEMBER FUNCTIONS:

```
    public :: Navigator_init,init ! initialize an object
    public :: clean               ! clean an object
    public :: NumSegments         ! number of vector segments
    public :: VectorLength        ! indexed vector's total length
    public :: msize               ! the maximum size
    public :: resize              ! adjust the true size
    public :: get                 ! get an entry
    public :: ptr_displs          ! referencing %displs(:)
    public :: ptr_counts          ! referencing %counts(:)

  interface Navigator_init; module procedure &
     init_
```

```
end interface
interface init  ; module procedure init_  ; end interface
interface clean ; module procedure clean_ ; end interface
interface NumSegments ; module procedure  &
   NumSegments_
end interface
interface VectorLength ; module procedure  &
   VectorLength_
end interface
interface msize ; module procedure msize_ ; end interface
interface resize; module procedure resize_; end interface
interface get   ; module procedure get_   ; end interface
interface ptr_displs; module procedure &
   ptr_displs_
end interface
interface ptr_counts; module procedure &
   ptr_counts_
end interface
```

REVISION HISTORY:

```
22May00 - Jing Guo <guo@dao.gsfc.nasa.gov> - initial prototype/prolog/code
26Aug02 - J. Larson <larson@mcs.anl.gov> - expanded datatype to inlcude
          VectorLength component.
```

6.1.1 init_ - Create a Navigator

This routine creates a `Navigator` `Nav` capable of storing information about `NumSegments` segments. The user can supply the length of the vector (or array subspace) being indexed by supplying the optional input `INTEGER` argument `VectorLength` (if it is not supplied, this component of `Nav` will be set to zero, signifying to other `Navigator` routines that vector length information is unavailable). The success (failure) of this operation is signified by the zero (non-zero) value of the optional output `INTEGER` argument `stat`.

INTERFACE:

```
subroutine init_(Nav, NumSegments, VectorLength, stat)
```

USES:

```
use m_mall,only : mall_ison,mall_mci
use m_die ,only : die,perr
use m_stdio, only : stderr

implicit none
```

INPUT PARAMETERS:

```
integer,                   intent(in)  :: NumSegments
integer,         optional, intent(in)  :: VectorLength
```

OUTPUT PARAMETERS:

```
type(Navigator),           intent(out) :: Nav
integer,         optional, intent(out) :: stat
```

REVISION HISTORY:

```
22May00 - Jing Guo <guo@dao.gsfc.nasa.gov> - initial prototype/prolog/code
```

6.1.2 clean_ - Destroy a Navigator

This routine deallocates allocated memory associated with the input/output `Navigator` argument `Nav`, and clears the vector length and number of segments components The success (failure) of this operation is signified by the zero (non-zero) value of the optional output `INTEGER` argument `stat`.

INTERFACE:

```
subroutine clean_(Nav, stat)
```

USES:

```
use m_mall, only : mall_ison,mall_mco
use m_die,  only : warn

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(Navigator),intent(inout) :: Nav
```

OUTPUT PARAMETERS:

```
integer,optional,intent(out) :: stat
```

REVISION HISTORY:

```
22May00 - Jing Guo <guo@dao.gsfc.nasa.gov> initial prototype/prolog/code
```

6.1.3 NumSegments_ - Return the Number of Segments

This `INTEGER` query function returns the number of segments in the input `Navigator` argument `Nav` for which segment start and length information are defined .

INTERFACE:

```
integer function NumSegments_(Nav)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(Navigator), intent(in) :: Nav
```

REVISION HISTORY:

```
22May00 - Jing Guo <guo@dao.gsfc.nasa.gov> initial prototype/prolog/code
 1Mar02 - E.T. Ong <eong@mcs.anl.gov> - removed die to prevent crashes.
```

6.1.4 msize_ - Return the Maximum Capacity for Segment Storage

This INTEGER query function returns the maximum number of segments for which start and length information can be stored in the input Navigator argument Nav.

INTERFACE:

```
integer function msize_(Nav)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(Navigator),intent(in) :: Nav
```

REVISION HISTORY:

22May00 - Jing Guo <guo@dao.gsfc.nasa.gov> initial prototype/prolog/code

---

6.1.5 VectorLength_ - Return the Navigated Vector's Length

This INTEGER query function returns the total length of the vector navigated by the input Navigator argument Nav. Note that the vector length is a quantity the user must have set when Nav was initialized. If it has not been set, the return value will be zero.

INTERFACE:

```
integer function VectorLength_(Nav)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(Navigator), intent(in) :: Nav
```

REVISION HISTORY:

26Aug02 - J. Larson <larson@mcs.anl.gov> - initial implementation

---

6.1.6 resize_ - Reset the Number of Segments

This routine resets the number of segments stored in the input/output Navigator argument Nav. It behaves in one of two modes: If the optional INTEGER input argument NumSegments is provided, then this value is taken to be the new number of segments. If this routine is invoked without NumSegments provided, then the new number of segments is set as per the result of the Fortran size() function applied to the segment table arrays.

INTERFACE:

```
subroutine resize_(Nav, NumSegments)
```

USES:

```
use m_stdio, only : stderr
use m_die,  only : die

implicit none
```

INPUT PARAMETERS:

```
integer,optional,intent(in) :: NumSegments
```

INPUT/OUTPUT PARAMETERS:

```
type(Navigator),intent(inout) :: Nav
```

REVISION HISTORY:

```
22May00 - Jing Guo <guo@dao.gsfc.nasa.gov> initial prototype/prolog/code
```

6.1.7 get_ - Retrieve Characteristics of a Segment

This multi-purpose query routine can be used to retrieve various characteristics of a given segment (identified by the input `INTEGER` argument `iSeg`) stored in the input `Navigator` argument `Nav`:

1. The *displacement* of the first element in this segment from the first element of the vector. This quantity is returned in the optional output `INTEGER` argument `displ`
2. The *number of elements* in this segment. This quantity is returned in the optional output `INTEGER` argument `displ`
3. The *index* of the first element in this segment This quantity is returned in the optional output `INTEGER` argument `lc`.
4. The *index* of the final element in this segment This quantity is returned in the optional output `INTEGER` argument `le`.

Any combination of the above characteristics may be obtained by invoking this routine with the corresponding optional arguments.

INTERFACE:

```
subroutine get_(Nav, iSeg, displ, count, lc, le)
```

USES:

```
use m_stdio, only : stderr
use m_die,  only : die

implicit none
```

INPUT PARAMETERS:

```
type(Navigator),          intent(in)  :: Nav
integer,                  intent(in)  :: iSeg
```

OUTPUT PARAMETERS:

```
integer,         optional, intent(out) :: displ
integer,         optional, intent(out) :: count
integer,         optional, intent(out) :: lc
integer,         optional, intent(out) :: le
```

REVISION HISTORY:

```
22May00 - Jing Guo <guo@dao.gsfc.nasa.gov>  initial prototype/prolog/code
```

6.1.8 ptr_displs_ - Returns Pointer to the displs(:) Component

This pointer-valued query function returns a pointer to the *displacements* information (the displacement of the first element of each segment from the beginning of the vector) contained in the input `Navigator` argument `Nav`. It has four basic modes of behavior, depending on which (if any) of the optional input `INTEGER` arguments `lbnd` and `ubnd` are supplied.

1. If neither `lbnd` nor `ubnd` is supplied, then `ptr_displs_` returns a pointer to *all* the elements in the array `Nav%displs(:)`.
2. If both `lbnd` and `ubnd` are supplied, then `ptr_displs_` returns a pointer to the segment of the array `Nav%displs(lbnd:ubnd)`.
3. If `lbnd` is supplied but `ubnd` is not, then `ptr_displs_` returns a pointer to the segment of the array `Nav%displs(lbnd:msize)`, where `msize` is the length of the array `Nav%displs(:)`.
4. If `lbnd` is not supplied but `ubnd` is, then `ptr_displs_` returns a pointer to the segment of the array `Nav%displs(1:ubnd)`.

INTERFACE:

```
function ptr_displs_(Nav, lbnd, ubnd)
```

USES:

```
use m_stdio, only : stderr
use m_die,  only : die

implicit none
```

INPUT PARAMETERS:

```
type(Navigator),           intent(in) :: Nav
integer,         optional, intent(in) :: lbnd
integer,         optional, intent(in) :: ubnd
```

OUTPUT PARAMETERS:

```
integer,     dimension(:), pointer    :: ptr_displs_
```

REVISION HISTORY:

```
22May00 - Jing Guo <guo@dao.gsfc.nasa.gov> - initial prototype/prolog/code
```

6.1.9 ptr_counts_ - Returns Pointer to counts(:) Component

This pointer-valued query function returns a pointer to the *counts* information (that is, the number of elements in each of each segment the vector being navigated) contained in the input `Navigator` argument `Nav`. It has four basic modes of behavior, depending on which (if any) of the optional input `INTEGER` arguments `lbnd` and `ubnd` are supplied.

1. If neither `lbnd` nor `ubnd` is supplied, then `ptr_counts_` returns a pointer to *all* the elements in the array `Nav%counts(:)`.
2. If both `lbnd` and `ubnd` are supplied, then `ptr_counts_` returns a pointer to the segment of the array `Nav%counts(lbnd:ubnd)`.
3. If `lbnd` is supplied but `ubnd` is not, then `ptr_counts_` returns a pointer to the segment of the array `Nav%counts(lbnd:msize)`, where `msize` is the length of the array `Nav%counts(:)`.
4. If `lbnd` is not supplied but `ubnd` is, then `ptr_counts_` returns a pointer to the segment of the array `Nav%counts(1:ubnd)`.

INTERFACE:

```
function ptr_counts_(Nav, lbnd, ubnd)
```

USES:

```
use m_stdio, only : stderr
use m_die,  only : die

implicit none
```

INPUT PARAMETERS:

```
type(Navigator),           intent(in) :: Nav
integer,         optional, intent(in) :: lbnd
integer,         optional, intent(in) :: ubnd
```

OUTPUT PARAMETERS:

```
integer, dimension(:),     pointer    :: ptr_counts_
```

REVISION HISTORY:

```
22May00 - Jing Guo <guo@dao.gsfc.nasa.gov>- initial prototype/prolog/code
```

7 The Global Map

7.1 Module m_GlobalMap - One-Dimensional Domain Decomposition Descriptor (Source File: m_GlobalMap.F90)

The `GlobalMap` is a datatype used to store descriptors of a one-dimensional domain decomposition for a vector on an MPI communicator. It is defined with three assumptions:

1. Each process ID owns only one segment;
2. No two segments in the decomposition overlap; and
3. The segments are laid out in identical order to the MPI rank of each process participating in the decomposition.

per process ID). It is the simpler of the two domain decomposition descriptors offerd by MCT (the other being the `GlobalSegMap`). It consists of the following components:

- The MCT component identification number (see the module `m_MCTWorld` for more information about MCT's component model registry);
- The *global* number of elements in the distributed vector;
- The number of elements *stored locally*;
- The number of elements *stored on each process* on the communicator over which the vector is distributed; and
- The index of the elemnent *immediately before* the starting element of each local segment (this choice allows for direct use of this information with MPI's scatter and gather operations). We refer to this quantity as the *displacement* of the segment, a term used both here and in the definition of the MCT `Navigator` datatype.

Both the segment displacement and length data are stored in arrays whose indices run from zero to $N-1$, where N is the number of MPI processes on the communicator on which the `GlobalMap` is defined. This is done so this information corresponds directly to the MPI process ID's on whihc the segments reside.
This module contains the definition of the `GlobalMap` datatype, all-processor and an on-root creation methods (both of which can be used to create a `GlobalMap` on the local communicator), a creation method to create/propagate a `GlobalMap` native to a remote communicator, a destruction method, and a variety of query methods.

INTERFACE:

```
module m_GlobalMap

 !USES
 No external modules are used in the declaration section of this module.

     implicit none

     private ! except
```

PUBLIC TYPES:

```
     public :: GlobalMap ! The class data structure

   Type GlobalMap
     integer :: comp_id                        ! Component ID number
     integer :: gsize ! the Global size
     integer :: lsize ! my local size
     integer,dimension(:),pointer :: counts ! all local sizes
```

```
  integer,dimension(:),pointer :: displs ! PE ordered locations
End Type GlobalMap
```

PUBLIC MEMBER FUNCTIONS:

```
  public :: gsize
  public :: lsize
  public :: init
  public :: init_remote
  public :: clean
  public :: rank
  public :: bounds
  public :: comp_id

interface gsize; module procedure gsize_; end interface
interface lsize; module procedure lsize_; end interface
interface init ; module procedure &
   initd_,& ! initialize from all PEs
   initr_ ! initialize from the root
end interface
interface init_remote; module procedure init_remote_; end interface
interface clean; module procedure clean_; end interface
interface rank ; module procedure rank_ ; end interface
interface bounds; module procedure bounds_; end interface
interface comp_id ; module procedure comp_id_ ; end interface
```

SEE ALSO:

```
The MCT module m_MCTWorld for more information regarding component
ID numbers.
```

REVISION HISTORY:

```
21Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
 9Nov00 - J.W. Larson <larson@mcs.anl.gov> - added init_remote
          interface.
26Jan01 - J.W. Larson <larson@mcs.anl.gov> - added storage for
          component ID number GlobalMap%comp_id, and associated
          method comp_id_()
```

7.1.1 initd_ - Collective Creation on the Local Communicator

This routine creates the `GlobalMap` `GMap` from distributed data spread across the MPI communicatior associated with the input `INTEGER` handle `comm`. The `INTEGER` input argument `comp_id` is used to define the MCT component ID for `GMap`. The input `INTEGER` argument `ln` is the number of elements in the local vector segment.

INTERFACE:

```
subroutine initd_(GMap, comp_id, ln, comm)
```

USES:

```
use m_mpif90
use m_die

implicit none
```

INPUT PARAMETERS:

```
integer,         intent(in)  :: comp_id ! Component ID
integer,         intent(in)  :: ln      ! the local size
integer,         intent(in)  :: comm    ! f90 MPI communicator
                                        ! handle
```

OUTPUT PARAMETERS:

```
type(GlobalMap), intent(out) :: GMap
```

SEE ALSO:

```
The MCT module m_MCTWorld for more information regarding component
ID numbers.
```

REVISION HISTORY:

```
21Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

7.1.2 initr_ Create a GlobalMap from the Root Process

This routine creates the `GlobalMap` `GMap`, and propagates it to all processes on the communicator associated with the MPI `INTEGER` handle `comm`. The input `INTEGER` arguments `comp_id` (the MCT component ID number) and `lns(:)` need only be valid on the process whose rank is equal to `root` on `comm`. The array `lns(:)` should have length equal to the number of processes on `comm`, and contains the length of each local segment.

INTERFACE:

```
subroutine initr_(GMap, comp_id, lns, root, comm)
```

USES:

```
use m_mpif90
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer,               intent(in)  :: comp_id ! component ID number
integer, dimension(:), intent(in)  :: lns     ! segment lengths
integer,               intent(in)  :: root    ! root process ID
integer,               intent(in)  :: comm    ! communicator ID
```

OUTPUT PARAMETERS:

```
type(GlobalMap),       intent(out) :: GMap
```

SEE ALSO:

```
The MCT module m_MCTWorld for more information regarding component
ID numbers.
```

REVISION HISTORY:

```
29May98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

7.1.3 init_remote_ Initialize Remote GlobalMap from the Root

This routine creates and propagates across the local communicator a `GlobalMap` associated with a remote component. The controlling process in this operation has MPI process ID defined by the input `INTEGER` argument `my_root`, and its MPI communinicator is defined by the input `INTEGER` argument `my_comm`. The input `INTEGER` argument `remote_npes` is the number of MPI processes on the remote component's communicator (which need be valid only on the process `my_root`). The input the `INTEGER` array `remote_lns(:)`, and the `INTEGER` argument `remote_comp_id` need only be valid on the process whose rank on the communicator `my_comm` is `my_root`. The argument `remote_lns(:)` defines the vector segment length on each process of the remote component's communicator, and the argument `remote_comp_id` defines the remote component's ID number in the MCT component registry `MCTWorld`.

INTERFACE:

```
subroutine init_remote_(GMap, remote_lns, remote_npes, my_root, &
                        my_comm, remote_comp_id)
```

USES:

```
use m_mpif90
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:), intent(in)  :: remote_lns
integer,               intent(in)  :: remote_npes
integer,               intent(in)  :: my_root
integer,               intent(in)  :: my_comm
integer,               intent(in)  :: remote_comp_id
```

OUTPUT PARAMETERS:

```
type(GlobalMap),       intent(out) :: GMap
```

SEE ALSO:

```
The MCT module m_MCTWorld for more information regarding component
ID numbers.
```

REVISION HISTORY:

```
 8Nov00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
26Jan01 - J.W. Larson <larson@mcs.anl.gov> - slight change--remote
          communicator is replaced by remote component ID number
          in argument remote_comp_id.
```

7.1.4 clean_ - Destroy a GlobalMap

This routine deallocates all allocated memory associated with the input/output `GlobalMap` argument `GMap`, and sets to zero all of its statically defined components. The success (failure) of this operation is signified by the zero (non-zero) value of the optional output `INTEGER` argument `stat`.

INTERFACE:

```
subroutine clean_(GMap, stat)
```

USES:

```
use m_die

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(GlobalMap),           intent(inout) :: GMap
```

OUTPUT PARAMETERS:

```
integer,         optional, intent(out)   :: stat
```

REVISION HISTORY:

```
21Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
26Jan01 - J. Larson <larson@mcs.anl.gov> incorporated comp_id.
 1Mar02 - E.T. Ong <eong@mcs.anl.gov> removed the die to prevent
          crashes and added stat argument.
```

7.1.5 lsize_ - Return Local Segment Length

This `INTEGER` function returns the length of the local vector segment as defined by the input `GlobalMap` argument `GMap`.

INTERFACE:

```
integer function lsize_(GMap)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(GlobalMap), intent(in) :: GMap
```

REVISION HISTORY:

```
21Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

7.1.6 gsize_ - Return Global Vector Length

This `INTEGER` function returns the global length of a vector that is decomposed according to the input `GlobalMap` argument `GMap`.

INTERFACE:

```
integer function gsize_(GMap)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(GlobalMap), intent(in) :: GMap
```

REVISION HISTORY:

21Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code

---

7.1.7 rank_ - Process ID Location of a Given Vector Element

This routine uses the input `GlobalMap` argument `GMap` to determine the process ID (on the communicator on which `GMap` was defined) of the vector element with global index `i_g`. This process ID is returned in the output `INTEGER` argument `rank`.

INTERFACE:

```
subroutine rank_(GMap, i_g, rank)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(GlobalMap), intent(in)  :: GMap
integer,         intent(in)  :: i_g
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

5May98 - Jing Guo <guo@thunder> - initial prototype/prolog/code

---

7.1.8 bounds_ - First/Last Global Indicies for a Process' Segment

This routine takes as input a process ID (defined by the input `INTEGER` argument `pe_no`), examines the input `GlobalMap` argument `GMap`, and returns the global indices for the first and last elements of the segment owned by this process in the output `INTEGER` arguments `lbnd` and `ubnd`, respectively.

INTERFACE:

```
subroutine bounds_(GMap, pe_no, lbnd, ubnd)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(GlobalMap), intent(in)  :: GMap
integer,         intent(in)  :: pe_no
```

OUTPUT PARAMETERS:

```
integer,         intent(out) :: lbnd
integer,         intent(out) :: ubnd
```

REVISION HISTORY:

```
30Jan01 - J. Larson <larson@mcs.anl.gov> - initial code
```

7.1.9 comp_id_ - Return the Component ID Number

This `INTEGER` query function returns the MCT component ID number stored in the input `GlobalMap` argument `GMap`.

INTERFACE:

```
integer function comp_id_(GMap)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(GlobalMap), intent(in) :: GMap
```

SEE ALSO:

```
The MCT module m_MCTWorld for more information regarding component
ID numbers.
```

REVISION HISTORY:

```
25Jan02 - J. Larson <larson@mcs.anl.gov> - initial version
```

Part II
High Level API's

8 Sending and Receiving Attribute Vectors

8.1 Module m_Transfer - Routines for the MxN transfer of Attribute Vectors (Source File: m_Transfer.F90)

This module provides routines for doing MxN transfer of data in an Attribute Vector between two components on separate sets of MPI processes. Uses the Router datatype.

SEE ALSO:

```
m_Rearranger
```

INTERFACE:

```
module m_Transfer
```

USES:

```
use m_MCTWorld, only : MCTWorld
use m_MCTWorld, only : ThisMCTWorld
use m_AttrVect, only : AttrVect
use m_AttrVect, only : nIAttr,nRAttr
use m_AttrVect, only : Permute, Unpermute
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_copy => copy
use m_AttrVect, only : AttrVect_clean => clean
use m_AttrVect, only : lsize
use m_Router,   only : Router

use m_mpif90
use m_die
use m_stdio

implicit none

private ! except
```

PUBLIC MEMBER FUNCTIONS:

```
public  :: isend
public  :: send
public  :: waitsend
public  :: irecv
public  :: recv
public  :: waitrecv


interface isend    ; module procedure isend_    ; end interface
interface send     ; module procedure send_     ; end interface
interface waitsend ; module procedure waitsend_ ; end interface
interface irecv    ; module procedure irecv_    ; end interface
interface recv     ; module procedure recv_     ; end interface
interface waitrecv ; module procedure waitrecv_ ; end interface
```

DEFINED PARAMETERS:

```
integer,parameter        :: DefaultTag = 600
```

REVISION HISTORY:

```
08Nov02 - R. Jacob <jacob@mcs.anl.gov> - make new module by combining
          MCT_Send, MCT_Recv and MCT_Recvsum
11Nov02 - R. Jacob <jacob@mcs.anl.gov> - Remove MCT_Recvsum and use
          optional argument in recv_ to do the same thing.
23Jul03 - R. Jacob <jacob@mcs.anl.gov> - Move buffers for data and
          MPI_Reqest and MPI_Status arrays to Router.  Use them.
24Jul03 - R. Jacob <jacob@mcs.anl.gov> - Split send_ into isend_ and
          waitsend_.  Redefine send_.
22Jan08 - R. Jacob <jacob@mcs.anl.gov> - Handle unordered GSMaps
```

8.1.1 isend_ - Distributed non-blocking send of an Attribute Vector

Send the the data in the `AttrVect` `aV` to the component specified in the `Router` `Rout`. An error will result if the size of the attribute vector does not match the size parameter stored in the `Router`. Requires a corresponding `recv_` or `irecv_` to be called on the other component.
The optional argument `Tag` can be used to set the tag value used in the data transfer. DefaultTag will be used otherwise. `Tag` must be the same in the matching `recv_` or `irecv_`.
N.B.: The `AttrVect` argument in the corresponding `recv_` call is assumed to have exactly the same attributes in exactly the same order as `aV`.

INTERFACE:

```
subroutine isend_(aVin, Rout, Tag)
```

USES:

```
      implicit none
```

INPUT PARAMETERS:

```
      Type(AttrVect),target,intent(in)    :: aVin
      Type(Router),         intent(inout) :: Rout
      integer,optional,     intent(in) :: Tag
```

REVISION HISTORY:

```
07Feb01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
08Feb01 - R. Jacob <jacob@mcs.anl.gov> - First working code
18May01 - R. Jacob <jacob@mcs.anl.gov> - use MP_Type to determine type in mpi_send
07Jun01 - R. Jacob <jacob@mcs.anl.gov> - remove logic to check "direction" of Router.
          remove references to ThisMCTWorld%mylrank
03Aug01 - E. Ong <eong@mcs.anl.gov> - Explicitly specify the starting address in mpi_send.
15Feb02 - R. Jacob <jacob@mcs.anl.gov> - Use MCT_comm
26Mar02 - E. Ong <eong@mcs.anl.gov> - Apply faster copy order
26Sep02 - R. Jacob <jacob@mcs.anl.gov> - Check Av against Router lAvsize
05Nov02 - R. Jacob <jacob@mcs.anl.gov> - Remove iList, rList arguments.
08Nov02 - R. Jacob <jacob@mcs.anl.gov> - MCT_Send is now send_ in m_Transfer
11Nov02 - R. Jacob <jacob@mcs.anl.gov> - Use DefaultTag and add optional Tag argument
25Jul03 - R. Jacob <jacob@mcs.anl.gov> - Split into isend_ and waitsend_
22Jan08 - R. Jacob <jacob@mcs.anl.gov> - Handle unordered GSMaps by permuting before send.
          remove special case for sending one segment directly from Av which probably
          wasn't safe.
```

8.1.2 waitsend_ - Wait for a distributed non-blocking send to complete

Wait for the data being sent with the `Router Rout` to complete.

INTERFACE:

```
subroutine waitsend_(Rout)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
Type(Router),         intent(inout) :: Rout
```

REVISION HISTORY:

```
24Jul03 - R. Jacob <jacob@mcs.anl.gov> - First working version is
          the wait part of original send_
```

8.1.3 send_ - Distributed blocking send of an Attribute Vector

Send the the data in the `AttrVect aV` to the component specified in the `Router Rout`. An error will result if the size of the attribute vector does not match the size parameter stored in the `Router`. Requires a corresponding `recv_` or `irecv_` to be called on the other component.
The optional argument `Tag` can be used to set the tag value used in the data transfer. DefaultTag will be used otherwise. `Tag` must be the same in the matching `recv_` or `irecv_`.
N.B.: The `AttrVect` argument in the corresponding `recv` call is assumed to have exactly the same attributes in exactly the same order as `aV`.

INTERFACE:

```
subroutine send_(aV, Rout, Tag)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
Type(AttrVect),       intent(in)    :: aV
Type(Router),         intent(inout) :: Rout
integer,optional,     intent(in)    :: Tag
```

REVISION HISTORY:

```
24Jul03 - R. Jacob <jacob@mcs.anl.gov> - New version uses isend and waitsend
```

8.1.4 irecv_ - Distributed receive of an Attribute Vector

Recieve into the `AttrVect aV` the data coming from the component specified in the `Router Rout`. An error will result if the size of the attribute vector does not match the size parameter stored in the `Router`.
Requires a corresponding `send_` or `isend_` to be called on the other component.
The optional argument `Tag` can be used to set the tag value used in the data transfer. DefaultTag will be used otherwise. `Tag` must be the same in the matching `send_` or `isend_`.
If data for a grid point is coming from more than one process, `recv_` will overwrite the duplicate values leaving the last received value in the output aV. If the optional argument `Sum` is invoked, the output will contain the sum of any duplicate values received for the same grid point.
Will return as soon as MPI_IRECV's are posted. Call `waitrecv_` to complete the receive operation.
N.B.: The `AttrVect` argument in the corresponding `send_` call is assumed to have exactly the same attributes in exactly the same order as `aV`.

INTERFACE:

```
subroutine irecv_(aV, Rout, Tag, Sum)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
Type(AttrVect),       intent(inout) :: aV
```

INPUT PARAMETERS:

```
Type(Router),         intent(inout) :: Rout
integer,optional,     intent(in)    :: Tag
logical,optional,     intent(in)    :: Sum
```

REVISION HISTORY:

```
07Feb01 - R. Jacob <jacob@mcs.anl.gov> - initial prototype
07Jun01 - R. Jacob <jacob@mcs.anl.gov> - remove logic to
          check "direction" of Router.  remove references
          to ThisMCTWorld%mylrank
03Aug01 - E.T. Ong <eong@mcs.anl.gov> - explicity specify starting
          address in MPI_RECV
27Nov01 - E.T. Ong <eong@mcs.anl.gov> - deallocated to prevent
          memory leaks
15Feb02 - R. Jacob <jacob@mcs.anl.gov> - Use MCT_comm
26Mar02 - E. Ong <eong@mcs.anl.gov> - Apply faster copy order.
26Sep02 - R. Jacob <jacob@mcs.anl.gov> - Check Av against Router lAvsize
08Nov02 - R. Jacob <jacob@mcs.anl.gov> - MCT_Recv is now recv_ in m_Transfer
11Nov02 - R. Jacob <jacob@mcs.anl.gov> - Add optional Sum argument to
          tell recv_ to sum data for the same point received from multiple
          processors.  Replaces recvsum_ which had replaced MCT_Recvsum.
          Use DefaultTag and add optional Tag argument
25Jul03 - R. Jacob <jacob@mcs.anl.gov> - break into irecv_ and waitrecv_
```

8.1.5 waitrecv_ - Wait for a distributed non-blocking recv to complete

Wait for the data being received with the `Router Rout` to complete. When done, copy the data into the `AttrVect aV`.

INTERFACE:

```
subroutine waitrecv_(aV, Rout, Sum)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
Type(AttrVect),       intent(inout) :: aV
Type(Router),         intent(inout) :: Rout
```

INPUT PARAMETERS:

```
logical,optional,     intent(in)    :: Sum
```

REVISION HISTORY:

```
25Jul03 - R. Jacob <jacob@mcs.anl.gov> - First working version is the wait
          and copy parts from old recv_.
25Jan08 - R. Jacob <jacob@mcs.anl.gov> - Handle unordered GSMaps by
          applying permutation to received array.
```

8.1.6 recv_ - Distributed receive of an Attribute Vector

Recieve into the `AttrVect aV` the data coming from the component specified in the `Router Rout`. An error will result if the size of the attribute vector does not match the size parameter stored in the `Router`.
Requires a corresponding `send_` or `isend_`to be called on the other component.
The optional argument `Tag` can be used to set the tag value used in the data transfer. DefaultTag will be used otherwise. `Tag` must be the same in the matching `send_`
If data for a grid point is coming from more than one process, `recv_` will overwrite the duplicate values leaving the last received value in the output aV. If the optional argument `Sum` is invoked, the output will contain the sum of any duplicate values received for the same grid point.
Will not return until all data has been received.
N.B.: The `AttrVect` argument in the corresponding `send_` call is assumed to have exactly the same attributes in exactly the same order as `aV`.

INTERFACE:

```
subroutine recv_(aV, Rout, Tag, Sum)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
Type(AttrVect),       intent(inout) :: aV
```

INPUT PARAMETERS:

```
Type(Router),         intent(inout)    :: Rout
integer,optional,     intent(in)    :: Tag
logical,optional,     intent(in)    :: Sum
```

REVISION HISTORY:

```
25Jul03 - R. Jacob <jacob@mcs.anl.gov> - Rewrite using irecv and waitrecv
```

9 Rearranging Attribute Vectors

9.1 Module m_Rearranger – Remaps an AttrVect within a group of processes (Source File: m_Rearranger.F90)

This module provides routines and datatypes for rearranging data between two `Attribute Vectors` defined on the same grid but with two different `GlobalSegMaps`. "Rearrange" is a generalized form of a parallel matrix transpose. A parallel matrix transpose can take advantage of symmetry in the data movement algorithm. An MCT Rearranger makes no assumptions about symmetry.
When data needs to move between two components and the components share any processors, use m_Rearranger. If the components are on distinct sets of processors, use m_Transfer.

SEE ALSO:

```
m_Transfer
```

INTERFACE:

```
module m_Rearranger
```

USES:

```
use m_Router, only : Router

implicit none

private ! except
```

PUBLIC DATA MEMBERS:

```
public :: Rearranger  ! The class data structure

type :: Rearranger
#ifdef SEQUENCE
   sequence
#endif
   private
   type(Router) :: SendRouter
   type(Router) :: RecvRouter
   integer,dimension(:,:),pointer :: LocalPack
   integer :: LocalSize
end type Rearranger

!PRIVATE DATA MEMBERS:
   integer :: max_nprocs  ! size of MPI_COMM_WORLD used for generation of
                          ! local automatic arrays
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init         ! creation method

public :: rearrange    ! the rearrange routine

public :: clean        ! destruction method
public :: print        ! print out comm info

interface init      ; module procedure init_      ; end interface
interface Rearrange ; module procedure Rearrange_ ; end interface
interface clean     ; module procedure clean_     ; end interface
interface print     ; module procedure print_     ; end interface
```

DEFINED PARAMETERS:

```
integer,parameter                    :: DefaultTag = 500
```

REVISION HISTORY:

```
31Jan02 - E.T. Ong <eong@mcs.anl.gov> - initial prototype
04Jun02 - E.T. Ong <eong@mcs.anl.gov> - changed local copy structure to
          LocalSize. Made myPid a global process in MCTWorld.
27Sep02 - R. Jacob <jacob@mcs.anl.gov> - Remove SrcAVsize and TrgAVsize
          and use Router%lAvsize instead for sanity check.
25Jan08 - R. Jacob <jacob@mcs.anl.gov> - Add ability to handle unordered
          gsmaps.
```

9.1.1 Init_ - Initialize a Rearranger

This routine takes two `GlobalSegMap` inputs, `SourceGSMap` and `TargetGSMap` and build a Rearranger `OutRearranger` between them. `myComm` is used for the internal communication.
N.B. The two `GlolbalSegMap` inputs must be initialized so that the index values on a processor are in ascending order.

INTERFACE:

```
subroutine init_(SourceGSMap,TargetGSMap,myComm,OutRearranger)
```

USES:

```
use m_MCTWorld,     only : ThisMCTWorld
use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GSMap_lsize => lsize
use m_GlobalSegMap, only : GSMap_increasing => increasing
use m_Router,       only : Router
use m_Router,       only : Router_init => init
use m_mpif90
use m_die
use m_stdio

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)           :: SourceGSMap, TargetGSMap
integer,            intent(in)           :: myComm
```

OUTPUT PARAMETERS:

```
type(Rearranger),   intent(out)          :: OutRearranger
```

REVISION HISTORY:

```
31Jan02 - E.T. Ong <eong@mcs.anl.gov> - initial prototype
20Mar02 - E.T. Ong <eong@mcs.anl.gov> - working code
05Jun02 - E.T. Ong <eong@mcs.anl.gov> - Use LocalPack
30Mar06 - P. Worley <worleyph@ornl.gov> - added max_nprocs,
          used in communication optimizations in rearrange
```

9.1.2 clean_ - Clean a Rearranger

This routine deallocates allocated memory associated with the input/output `Rearranger` argument `ReArr`. The success (failure) of this operation is reported in the zero (nonzero) value of the optional output `INTEGER` argument `status`.

INTERFACE:

```
subroutine clean_(ReArr, status)
```

USES:

```
use m_Router,only : Router
use m_Router,only : Router_clean => clean
use m_mpif90
use m_die
use m_stdio

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(Rearranger),    intent(inout)           :: ReArr
```

OUTPUT PARAMETERS:

```
integer, optional,   intent(out)             :: status
```

REVISION HISTORY:

```
31Jan02 - E.T. Ong <eong@mcs.anl.gov> - initial prototype
20Mar02 - E.T. Ong <eong@mcs.anl.gov> - working code
```

9.1.3 rearrange_ - Rearrange data between two Attribute Vectors

This subroutine will take data in the `SourceAv` Attribute Vector and rearrange it to match the GlobalSegMap used to define the `TargetAv` Attribute Vector using the Rearrnger `InRearranger`.
The optional argument `Tag` can be used to set the tag value used in the rearrangement. DefaultTag will be used otherwise.
If the optional argument `Sum` is present and true, data for the same physical point coming from two or more processes will be summed. Otherwise, data is overwritten.
If the optional argument `Vector` is present and true, vector architecture-friendly parts of this routine will be invoked.
If the optional argument `AlltoAll` is present and true, the communication will be done with an alltoall call instead of individual sends and receives.
The size of the `SourceAv` and `TargetAv` argument must match those stored in the `InRearranger` or and error will result.
N.B.: `SourceAv` and `TargetAv` are assumed to have exactly the same attributes in exactly the same order.

INTERFACE:

```
subroutine rearrange_(SourceAVin,TargetAV,InRearranger,Tag,Sum,&
                      Vector,AlltoAll,HandShake,ISend,MaxReq)
```

USES:

```
use m_MCTWorld,only :MCTWorld
use m_MCTWorld,only :ThisMCTWorld
use m_AttrVect,  only : AttrVect
use m_AttrVect,  only : AttrVect_init => init
use m_AttrVect,  only : AttrVect_lsize => lsize
use m_AttrVect,  only : AttrVect_copy => copy
use m_AttrVect,  only : AttrVect_clean => clean
use m_AttrVect,  only : AttrVect_zero => zero
use m_AttrVect,  only : nIAttr,nRAttr
use m_AttrVect,  only : Permute,Unpermute
use m_Router,    only : Router
use m_SPMDutils, only : m_swapm_int, m_swapm_FP
use m_realkinds, only : FP
use m_mpif90
use m_die
use m_stdio

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),             intent(inout)   :: TargetAV
```

INPUT PARAMETERS:

```
type(AttrVect),   target,   intent(in)      :: SourceAVin
type(Rearranger), target,   intent(in)      :: InRearranger
integer,          optional, intent(in)      :: Tag
logical,          optional, intent(in)      :: Sum
logical,          optional, intent(in)      :: Vector
logical,          optional, intent(in)      :: AlltoAll
logical,          optional, intent(in)      :: HandShake
logical,          optional, intent(in)      :: ISend
integer,          optional, intent(in)      :: MaxReq
```

REVISION HISTORY:

```
31Jan02 - E.T. Ong <eong@mcs.anl.gov> - initial prototype
20Mar02 - E.T. Ong <eong@mcs.anl.gov> - working code
08Jul02 - E.T. Ong <eong@mcs.anl.gov> - change intent of Target,Source
29Oct03 - R. Jacob <jacob@mcs.anl.gov> - add optional argument vector
          to control use of vector-friendly mods provided by Fujitsu.
30Mar06 - P. Worley <worleyph@ornl.gov> - added alltoall option and
          reordered send/receive order to improve communication
          performance.  Also remove replace allocated arrays with
          automatic.
14Oct06 - R. Jacob <jacob@mcs.anl.gov> - check value of Sum argument.
25Jan08 - R. Jacob <jacob@mcs.anl.gov> - Permute/unpermute if the internal
          routers permarr is defined.
29Sep16 - P. Worley <worleyph@gmail.com> - added swapm variant of
          alltoall option
```

9.1.4 print_ - Print rearranger communication info

Print out communication info for both routers in a rearranger. Print out on unit number 'lun' e.g. (source,destination,length)

INTERFACE:

```
subroutine print_(rearr,mycomm,lun)
```

USES:

```
use m_die
use m_Router, only: router_print => print

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(Rearranger),      intent(in) :: rearr
integer, intent(in)           :: mycomm
integer, intent(in)           :: lun
```

REVISION HISTORY:

```
27Jul07 - R. Loy <rloy@mcs.anl.gov>  initial version
```

10 Sprase Matrix Support

10.1 Module m_SparseMatrix – Sparse Matrix Object (Source File: m_SparseMatrix.F

The `SparseMatrix` data type is MCT's object for storing sparse matrices. In MCT, intergrid interpolation is implemented as a sparse matrix-vector multiplication, with the `AttrVect` type playing the roles of the input and output vectors. The interpolation matrices tend to be *extremely* sparse. For $\mathbf{x} \in \Re^{N_x}$, and $\mathbf{y} \in \Re^{N_y}$, the interpolation matrix $\mathbf{M}$ used to effect $\mathbf{y} = \mathbf{Mx}$ will typically have $\mathcal{O}(N_y)$ non-zero elements. For that reason, the `SparseMatrix` type stores *only* information about non-zero matrix elements, along with the number of rows and columns in the full matrix. The nonzero matrix elements are stored in `AttrVect` form (see the module `m_AttrVect` for more details), and the set of attributes are listed below:

Attribute Name	**Significance**	`Type`
`grow`	Global Row Index	`INTEGER`
`gcol`	Global Column Index	`INTEGER`
`lrow`	Local Row Index	`INTEGER`
`lcol`	Local Column Index	`INTEGER`
`weight`	Matrix Element M_{ij}	`REAL`

The provision of both local and global column and row indices is made because this datatype can be used in either shared-memory or distributed-memory parallel matrix-vector products.
This module contains the definition of the `SparseMatrix` type, creation and destruction methods, a variety of accessor methods, routines for testing the suitability of the matrix for interpolation (i.e. the sum of each row is either zero or unity), and methods for sorting and permuting matrix entries. For better performance of the Matrix-Vector multiply on vector architectures, the `SparseMatrix` object also contains arrays for holding the sparse matrix data in a more vector-friendly form.

INTERFACE:

```
  module m_SparseMatrix
```

USES:

```
       use m_realkinds, only : FP
       use m_AttrVect, only : AttrVect


       private   ! except
```

PUBLIC TYPES:

```
       public :: SparseMatrix      ! The class data structure

       Type SparseMatrix
 #ifdef SEQUENCE
      sequence
 #endif
      integer :: nrows
   integer :: ncols
   type(AttrVect) :: data
          logical :: vecinit       ! additional data for the vectorized sMat
          integer,dimension(:),pointer :: row_s, row_e
          integer, dimension(:,:), pointer :: tcol
          real(FP), dimension(:,:), pointer :: twgt
          integer :: row_max, row_min
          integer :: tbl_end
       End Type SparseMatrix
```

PUBLIC MEMBER FUNCTIONS:

```
  public :: init              ! Create a SparseMatrix
  public :: vecinit           ! Initialize the vector parts
  public :: clean             ! Destroy a SparseMatrix
  public :: lsize             ! Local number of elements
  public :: indexIA           ! Index integer attribute
  public :: indexRA           ! Index real attribute
  public :: nRows             ! Total number of rows
  public :: nCols             ! Total number of columns

  public :: exportGlobalRowIndices    ! Return global row indices
                                      ! for matrix elements
  public :: exportGlobalColumnIndices ! Return global column indices
                                      ! for matrix elements
  public :: exportLocalRowIndices     ! Return local row indices
                                      ! for matrix elements
  public :: exportLocalColumnIndices  ! Return local column indices
                                      ! for matrix elements
  public :: exportMatrixElements      ! Return matrix elements

  public :: importGlobalRowIndices    ! Set global row indices
                                      ! using
  public :: importGlobalColumnIndices ! Return global column indices
                                      ! for matrix elements
  public :: importLocalRowIndices     ! Return local row indices
                                      ! for matrix elements
  public :: importLocalColumnIndices  ! Return local column indices
                                      ! for matrix elements
  public :: importMatrixElements      ! Return matrix elements
  public :: Copy                      ! Copy a SparseMatrix

  public :: GlobalNumElements ! Total number of nonzero elements
  public :: ComputeSparsity   ! Fraction of matrix that is nonzero
  public :: local_row_range   ! Local (on-process) row range
  public :: global_row_range  ! Local (on-process) row range
  public :: local_col_range   ! Local (on-process) column range
  public :: global_col_range  ! Local (on-process) column range
  public :: CheckBounds       ! Check row and column values
                              ! for out-of-bounds values
  public :: row_sum           ! Return SparseMatrix row sums
  public :: row_sum_check     ! Check SparseMatrix row sums against
                              ! input "valid" values
  public :: Sort              ! Sort matrix entries to generate an
                              ! index permutation (to be used by
                              ! Permute()
  public :: Permute           ! Permute matrix entries using index
                              ! permutation gernerated by Sort()
  public :: SortPermute       ! Sort/Permute matrix entries

interface init  ; module procedure init_  ; end interface
interface vecinit  ; module procedure vecinit_  ; end interface
interface clean ; module procedure clean_ ; end interface
interface lsize ; module procedure lsize_ ; end interface
interface indexIA ; module procedure indexIA_ ; end interface
interface indexRA ; module procedure indexRA_ ; end interface
interface nRows ; module procedure nRows_ ; end interface
interface nCols ; module procedure nCols_ ; end interface
```

```
  interface exportGlobalRowIndices ; module procedure &
exportGlobalRowIndices_
  end interface

  interface exportGlobalColumnIndices ; module procedure &
exportGlobalColumnIndices_
  end interface

  interface exportLocalRowIndices ; module procedure &
exportLocalRowIndices_
  end interface

  interface exportLocalColumnIndices ; module procedure &
exportLocalColumnIndices_
  end interface

  interface exportMatrixElements ; module procedure &
exportMatrixElementsSP_, &
exportMatrixElementsDP_
  end interface

  interface importGlobalRowIndices ; module procedure &
importGlobalRowIndices_
  end interface

  interface importGlobalColumnIndices ; module procedure &
importGlobalColumnIndices_
  end interface

  interface importLocalRowIndices ; module procedure &
importLocalRowIndices_
  end interface

  interface importLocalColumnIndices ; module procedure &
importLocalColumnIndices_
  end interface

  interface importMatrixElements ; module procedure &
importMatrixElementsSP_, &
importMatrixElementsDP_
  end interface

  interface Copy ; module procedure Copy_ ; end interface

  interface GlobalNumElements ; module procedure &
GlobalNumElements_
  end interface

  interface ComputeSparsity ; module procedure &
ComputeSparsitySP_,  &
ComputeSparsityDP_
  end interface

  interface local_row_range ; module procedure &
local_row_range_
  end interface

  interface global_row_range ; module procedure &
global_row_range_
```

```
  end interface

  interface local_col_range ; module procedure &
local_col_range_
  end interface

  interface global_col_range ; module procedure &
global_col_range_
  end interface

  interface CheckBounds; module procedure &
CheckBounds_
  end interface

  interface row_sum ; module procedure &
row_sumSP_, &
row_sumDP_
  end interface

  interface row_sum_check ; module procedure &
row_sum_checkSP_, &
row_sum_checkDP_
  end interface

  interface Sort ; module procedure Sort_ ; end interface
  interface Permute ; module procedure Permute_ ; end interface
  interface SortPermute ; module procedure SortPermute_ ; end interface
```

REVISION HISTORY:

```
19Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
15Jan01 - J.W. Larson <larson@mcs.anl.gov> - added numerous APIs
25Feb01 - J.W. Larson <larson@mcs.anl.gov> - changed from row/column
          attributes to global and local row and column attributes
23Apr01 - J.W. Larson <larson@mcs.anl.gov> - added number of rows
          and columns to the SparseMatrix type.  This means the
          SparseMatrix is no longer a straight AttrVect type.  This
          also made necessary the addition of lsize(), indexIA(),
          and indexRA().
29Oct03 - R. Jacob <jacob@mcs.anl.gov> - extend the SparseMatrix type
          to include mods from Fujitsu for a vector-friendly MatVecMul
```

10.1.1 init_ - Initialize an Empty SparseMatrix

This routine creates the storage space for the entries of a `SparseMatrix`, and sets the number of rows and columns in it. The input `INTEGER` arguments `nrows` and `ncols` specify the number of rows and columns respectively. The optional input argument `lsize` specifies the number of nonzero entries in the `SparseMatrix`. The initialized `SparseMatrix` is returned in the output argument `sMat`.
N.B.: This routine is allocating dynamical memory in the form of a `SparseMatrix`. The user must deallocate this space when the `SparseMatrix` is no longer needed by invoking the routine `clean_()`.

INTERFACE:

```
subroutine init_(sMat, nrows, ncols, lsize)
```

USES:

```
use m_AttrVect, only : AttrVect_init => init
```

```
use m_die

implicit none
```

INPUT PARAMETERS:

```
integer,            intent(in)   :: nrows
integer,            intent(in)   :: ncols
integer, optional,  intent(in)   :: lsize
```

OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(out)  :: sMat
```

REVISION HISTORY:

```
19Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
23Apr01 - Jay Larson <larson@mcs.anl.gov> - added arguments
          nrows and ncols--number of rows and columns in the
          SparseMatrix
```

10.1.2 vecinit_ - Initialize vector parts of a SparseMatrix

This routine creates the storage space for and intializes the vector parts of a `SparseMatrix`.
N.B.: This routine assumes the locally indexed parts of a `SparseMatrix` have been initialized. This is accomplished by either importing the values directly with `importLocalRowIndices` and `importLocalColIndices` or by importing the Global Row and Col Indices and making two calls to `GlobalToLocalMatrix`.
N.B.: The vector portion can use a large amount of memory so it is highly recommended that this routine only be called on a `SparseMatrix` that has been scattered or otherwise sized locally.

INTERFACE:

```
subroutine vecinit_(sMat)
```

USES:

```
use m_die
use m_stdio

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout)  :: sMat
```

REVISION HISTORY:

```
27Oct03 - R. Jacob <jacob@mcs.anl.gov> - initial version
          using code provided by Yoshi et. al.
```

10.1.3 clean_ - Destroy a SparseMatrix.

This routine deallocates dynamical memory held by the input `SparseMatrix` argument `sMat`. It also sets the number of rows and columns in the `SparseMatrix` to zero.

INTERFACE:

```
subroutine clean_(sMat,stat)
```

USES:

```
use m_AttrVect,only : AttrVect_clean => clean
use m_die

implicit none

!INPUT/OUTPTU PARAMETERS:

type(SparseMatrix), intent(inout) :: sMat
```

OUTPUT PARAMETERS:

```
integer, optional,  intent(out)   :: stat
```

REVISION HISTORY:

```
19Sep00 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
23Apr00 - J.W. Larson <larson@mcs.anl.gov> - added changes to
          accomodate clearing nrows and ncols.
01Mar02 - E.T. Ong <eong@mcs.anl.gov> Added stat argument.
03Oct03 - R. Jacob <jacob@mcs.anl.gov> - clean vector parts
```

10.1.4 lsize_ - Local Number Non-zero Elements

This `INTEGER` function reports on-processor storage of the number of nonzero elements in the input `SparseMatrix` argument `sMat`.

INTERFACE:

```
integer function lsize_(sMat)
```

USES:

```
use m_AttrVect,only : AttrVect_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in) :: sMat
```

REVISION HISTORY:

```
23Apr00 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

10.1.5 GlobalNumElements_ - Global Number of Non-zero Elements

This routine computes the number of nonzero elements in a distributed `SparseMatrix` variable `sMat`. The input `SparseMatrix` argument `sMat` is examined on each process to determine the number of nonzero elements it holds, and this value is summed across the communicator associated with the input `INTEGER` handle `comm`, with the total returned *on each process on the communicator*.

INTERFACE:

```
integer function GlobalNumElements_(sMat, comm)
```

USES:

```
use m_die
use m_mpif90

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
integer, optional,  intent(in)  :: comm
```

REVISION HISTORY:

```
24Apr01 - Jay Larson <larson@mcs.anl.gov> - New routine.
```

10.1.6 indexIA_ - Index an Integer Attribute

This `INTEGER` function reports the row index for a given `INTEGER` attribute of the input `SparseMatrix` argument `sMat`. The attribute requested is represented by the input `CHARACTER` variable `attribute`. The list of integer attributes one can request is defined in the description block of the header of this module (`m_SparseMatrix`).
Here is how `indexIA_` provides access to integer attribute data in a `SparseMatrix` variable `sMat`. Suppose we wish to access global row information. This attribute has associated with it the string tag `grow`. The corresponding index returned (`igrow`) is determined by invoking `indexIA_`:

```
igrow = indexIA_(sMat, 'grow')
```

Access to the global row index data in `sMat` is thus obtained by referencing `sMat%data%iAttr(igrow,:)`.

INTERFACE:

```
integer function indexIA_(sMat, item, perrWith, dieWith)
```

USES:

```
use m_String, only : String
use m_String, only : String_init => init
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => ToChar

use m_TraceBack, only : GenTraceBackString

use m_AttrVect,only : AttrVect_indexIA => indexIA

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix),         intent(in) :: sMat
character(len=*),           intent(in) :: item
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

REVISION HISTORY:

```
23Apr00 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

10.1.7 indexRA_ - Index a Real Attribute

This `INTEGER` function reports the row index for a given `REAL` attribute of the input `SparseMatrix` argument `sMat`. The attribute requested is represented by the input `CHARACTER` variable `attribute`. The list of real attributes one can request is defined in the description block of the header of this module (`m_SparseMatrix`).
Here is how `indexRA_` provides access to integer attribute data in a `SparseMatrix` variable `sMat`. Suppose we wish to access matrix element values. This attribute has associated with it the string tag `weight`. The corresponding index returned (`iweight`) is determined by invoking `indexRA_`:

```
iweight = indexRA_(sMat, 'weight')
```

Access to the matrix element data in `sMat` is thus obtained by referencing `sMat%data%rAttr(iweight,:)`.

INTERFACE:

```
integer function indexRA_(sMat, item, perrWith, dieWith)
```

USES:

```
use m_String, only : String
use m_String, only : String_init => init
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => ToChar

use m_TraceBack, only : GenTraceBackString

use m_AttrVect,only : AttrVect_indexRA => indexRA

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix),         intent(in) :: sMat
character(len=*),           intent(in) :: item
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

REVISION HISTORY:

```
24Apr00 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

10.1.8 nRows_ - Return the Number of Rows

This routine returns the *total* number of rows in the input `SparseMatrix` argument `sMat`. This number of rows is a constant, and not dependent on the decomposition of the `SparseMatrix`.

INTERFACE:

```
integer function nRows_(sMat)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in) :: sMat
```

REVISION HISTORY:

```
19Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

10.1.9 nCols_ - Return the Number of Columns

This routine returns the *total* number of columns in the input `SparseMatrix` argument `sMat`. This number of columns is a constant, and not dependent on the decomposition of the `SparseMatrix`.

INTERFACE:

```
integer function nCols_(sMat)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in) :: sMat
```

REVISION HISTORY:

```
19Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

10.1.10 exportGlobalRowIndices_ - Return Global Row Indices

This routine extracts from the input `SparseMatrix` argument `sMat` its global row indices, and returns them in the `INTEGER` output array `GlobalRows`, and its length in the output `INTEGER` argument `length`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `GlobalRows` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `GlobalRows`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) at the time this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `GlobalRows`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportGlobalRowIndices_(sMat, GlobalRows, length)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_exportIAttr => exportIAttr

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix),     intent(in)  :: sMat
```

OUTPUT PARAMETERS:

```
integer,  dimension(:), pointer     :: GlobalRows
integer,  optional,     intent(out) :: length
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

10.1.11 exportGlobalColumnIndices_ - Return Global Column Indices

This routine extracts from the input `SparseMatrix` argument `sMat` its global column indices, and returns them in the `INTEGER` output array `GlobalColumns`, and its length in the output `INTEGER` argument `length`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `GlobalColumns` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `GlobalColumns`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) at the time this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `GlobalColumns`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportGlobalColumnIndices_(sMat, GlobalColumns, length)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_exportIAttr => exportIAttr

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix),     intent(in)  :: sMat
```

OUTPUT PARAMETERS:

```
integer,  dimension(:), pointer     :: GlobalColumns
integer,  optional,     intent(out) :: length
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

10.1.12 exportLocalRowIndices_ - Return Local Row Indices

This routine extracts from the input `SparseMatrix` argument `sMat` its local row indices, and returns them in the `INTEGER` output array `LocalRows`, and its length in the output `INTEGER` argument `length`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `LocalRows` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `LocalRows`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) at the time this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `LocalRows`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportLocalRowIndices_(sMat, LocalRows, length)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_exportIAttr => exportIAttr

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix),     intent(in)  :: sMat
```

OUTPUT PARAMETERS:

```
integer,  dimension(:), pointer     :: LocalRows
integer,  optional,     intent(out) :: length
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

10.1.13 exportLocalColumnIndices_ - Return Local Column Indices

This routine extracts from the input `SparseMatrix` argument `sMat` its local column indices, and returns them in the `INTEGER` output array `LocalColumns`, and its length in the output `INTEGER` argument `length`.

N.B.: The flexibility of this routine regarding the pointer association status of the output argument `LocalColumns` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `LocalColumns`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) at the time this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `LocalColumns`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportLocalColumnIndices_(sMat, LocalColumns, length)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_exportIAttr => exportIAttr

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix),     intent(in)  :: sMat
```

OUTPUT PARAMETERS:

```
integer,  dimension(:), pointer     :: LocalColumns
integer,  optional,     intent(out) :: length
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial version.
```

10.1.14 exportMatrixElementsSP_ - Return Matrix Elements as Array

This routine extracts the matrix elements from the input `SparseMatrix` argument `sMat`, and returns them in the `REAL` output array `MatrixElements`, and its length in the output `INTEGER` argument `length`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `MatrixElements` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `MatrixElements`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) at the time this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `MatrixElements`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.
The native precision version is described here. A double precision version is also available.

INTERFACE:

```
subroutine exportMatrixelementsSP_(sMat, MatrixElements, length)
```

USES:

```
use m_die
use m_stdio
use m_realkinds, only : SP

use m_AttrVect,      only : AttrVect_exportRAttr => exportRAttr

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix),     intent(in)  :: sMat
```

OUTPUT PARAMETERS:

```
real(SP),  dimension(:),    pointer     :: MatrixElements
integer,   optional,        intent(out) :: length
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial version.
6Jan04 - R. Jacob <jacob@mcs.anl.gov> - SP and DP versions
```

10.1.15 importGlobalRowIndices_ - Set Global Row Indices of Elements

This routine imports global row index data into the `SparseMatrix` argument `sMat`. The user provides the index data in the input `INTEGER` vector `inVect`. The input `INTEGER` argument `lsize` is used as a consistencey check to ensure the user is sufficient space in the `SparseMatrix` to store the data.

INTERFACE:

```
subroutine importGlobalRowIndices_(sMat, inVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_importIAttr => importIAttr

implicit none
```

INPUT PARAMETERS:

```
integer,  dimension(:), pointer       :: inVect
integer,                intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),     intent(inout) :: sMat
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

10.1.16 importGlobalColumnIndices_ - Set Global Column Indices of Elements

This routine imports global column index data into the `SparseMatrix` argument `sMat`. The user provides the index data in the input `INTEGER` vector `inVect`. The input `INTEGER` argument `lsize` is used as a consistencey check to ensure the user is sufficient space in the `SparseMatrix` to store the data.

INTERFACE:

```
subroutine importGlobalColumnIndices_(sMat, inVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_importIAttr => importIAttr

implicit none
```

INPUT PARAMETERS:

```
integer,  dimension(:), pointer       :: inVect
integer,                intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),     intent(inout) :: sMat
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

10.1.17 importLocalRowIndices_ - Set Local Row Indices of Elements

This routine imports local row index data into the `SparseMatrix` argument `sMat`. The user provides the index data in the input `INTEGER` vector `inVect`. The input `INTEGER` argument `lsize` is used as a consistencey check to ensure the user is sufficient space in the `SparseMatrix` to store the data.

INTERFACE:

```
subroutine importLocalRowIndices_(sMat, inVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_importIAttr => importIAttr

implicit none
```

INPUT PARAMETERS:

```
integer,  dimension(:), pointer       :: inVect
integer,                intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),     intent(inout) :: sMat
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

10.1.18 importLocalColumnIndices_ - Set Local Column Indices of Elements

This routine imports local column index data into the `SparseMatrix` argument `sMat`. The user provides the index data in the input `INTEGER` vector `inVect`. The input `INTEGER` argument `lsize` is used as a consistencey check to ensure the user is sufficient space in the `SparseMatrix` to store the data.

INTERFACE:

```
subroutine importLocalColumnIndices_(sMat, inVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_importIAttr => importIAttr

implicit none
```

INPUT PARAMETERS:

```
integer,  dimension(:), pointer       :: inVect
integer,                intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),     intent(inout) :: sMat
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

10.1.19 importMatrixElementsSP_ - Import Non-zero Matrix Elements

This routine imports matrix elements index data into the `SparseMatrix` argument `sMat`. The user provides the index data in the input `REAL` vector `inVect`. The input `INTEGER` argument `lsize` is used as a consistencey check to ensure the user is sufficient space in the `SparseMatrix` to store the data.

INTERFACE:

```
subroutine importMatrixElementsSP_(sMat, inVect, lsize)
```

USES:

```
use m_die
use m_stdio
use m_realkinds, only : SP

use m_AttrVect,      only : AttrVect_importRAttr => importRAttr

implicit none
```

INPUT PARAMETERS:

```
real(SP),  dimension(:),    pointer       :: inVect
integer,                intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),     intent(inout) :: sMat
```

REVISION HISTORY:

```
7May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
6Jan04 - R. Jacob <jacob@mcs.anl.gov> - Make SP and DP versions.
```

10.1.20 Copy_ - Create a Copy of an Input SparseMatrix

This routine creates a copy of the input `SparseMatrix` argument `sMat`, returning it as the output `SparseMatrix` argument `sMatCopy`.
N.B.: The output argument `sMatCopy` represents allocated memory the user must deallocate when it is no longer needed. The MCT routine to use for this purpose is `clean()` from this module.

INTERFACE:

```
subroutine Copy_(sMat, sMatCopy)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect
use m_AttrVect,      only : AttrVect_init => init
use m_AttrVect,      only : AttrVect_lsize => lsize
use m_AttrVect,      only : AttrVect_Copy => Copy

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in) :: sMat
```

OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(out) :: sMatCopy
```

REVISION HISTORY:

```
27Sep02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

10.1.21 local_row_range_ - Local Row Extent of Non-zero Elements

This routine examines the input distributed `SparseMatrix` variable `sMat`, and returns the range of local row values having nonzero elements. The first local row with nonzero elements is returned in the `INTEGER` argument `start_row`, the last row in `end_row`.

INTERFACE:

```
subroutine local_row_range_(sMat, start_row, end_row)
```

USES:

```
use m_die

use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_indexIA => indexIA

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
```

OUTPUT PARAMETERS:

```
integer,            intent(out) :: start_row
integer,            intent(out) :: end_row
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
25Feb01 - Jay Larson <larson@mcs.anl.gov> - Initial prototype.
23Apr01 - Jay Larson <larson@mcs.anl.gov> - Modified to accomodate
          changes to the SparseMatrix type.
```

10.1.22 global_row_range_ - Global Row Extent of Non-zero Elements

This routine examines the input distributed `SparseMatrix` variable `sMat`, and returns the range of global row values having nonzero elements. The first local row with nonzero elements is returned in the `INTEGER` argument `start_row`, the last row in `end_row`.

INTERFACE:

```
subroutine global_row_range_(sMat, comm, start_row, end_row)
```

USES:

```
use m_die

use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_indexIA => indexIA

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
integer,            intent(out) :: start_row
integer,            intent(out) :: end_row
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
25Feb01 - Jay Larson <larson@mcs.anl.gov> - Initial prototype.
23Apr01 - Jay Larson <larson@mcs.anl.gov> - Modified to accomodate
          changes to the SparseMatrix type.
```

10.1.23 local_col_range_ - Local Column Extent of Non-zero Elements

This routine examines the input distributed `SparseMatrix` variable `sMat`, and returns the range of local column values having nonzero elements. The first local column with nonzero elements is returned in the `INTEGER` argument `start_col`, the last column in `end_col`.

INTERFACE:

```
subroutine local_col_range_(sMat, start_col, end_col)
```

USES:

```
use m_die

use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_indexIA => indexIA

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
```

OUTPUT PARAMETERS:

```
integer,            intent(out) :: start_col
integer,            intent(out) :: end_col
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
25Feb01 - Jay Larson <larson@mcs.anl.gov> - Initial prototype.
23Apr01 - Jay Larson <larson@mcs.anl.gov> - Modified to accomodate
          changes to the SparseMatrix type.
```

10.1.24 global_col_range_ - Global Column Extent of Non-zero Elements

This routine examines the input distributed `SparseMatrix` variable `sMat`, and returns the range of global column values having nonzero elements. The first global column with nonzero elements is returned in the `INTEGER` argument `start_col`, the last column in `end_col`.

INTERFACE:

```
subroutine global_col_range_(sMat, comm, start_col, end_col)
```

USES:

```
use m_die

use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_indexIA => indexIA

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
integer,            intent(out) :: start_col
integer,            intent(out) :: end_col
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
25Feb01 - Jay Larson <larson@mcs.anl.gov> - Initial prototype.
23Apr01 - Jay Larson <larson@mcs.anl.gov> - Modified to accomodate
          changes to the SparseMatrix type.
```

10.1.25 ComputeSparsitySP_ - Compute Matrix Sparsity

This routine computes the sparsity of a consolidated (all on one process) or distributed `SparseMatrix`. The input `SparseMatrix` argument `sMat` is examined to determine the number of nonzero elements it holds, and this value is divided by the product of the number of rows and columns in `sMat`. If the optional input argument `comm` is given, the number of elements is reduced with a SUM operation and the sparsity computed from that. The resulting value of `sparsity` is computed identically on all processors.
The rows and columns in an `sMat` are always the global values so there is no need to do a reduction on those.
The calculation is always done in double precision even with the argument is single precision.

INTERFACE:

```
subroutine ComputeSparsitySP_(sMat, sparsity, comm)
```

USES:

```
use m_die
use m_mpif90
use m_realkinds, only : SP, FP, DP

use m_AttrVect, only : AttrVect_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
integer, optional,  intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
real(SP),           intent(out) :: sparsity
```

REVISION HISTORY:

```
04Aug20 - Robert Jacob <jacob@anl.gov> - new algorithm
23Apr01 - Jay Larson <larson@mcs.anl.gov> - New routine.
```

10.1.26 CheckBounds_ - Check for Out-of-Bounds Row/Column Values

This routine examines the input distributed `SparseMatrix` variable `sMat`, and examines the global row and column index for each element, comparing them with the known maximum values for each (as returned by the routines `nRows_()` and `nCols_()`, respectively). If global row or column entries are non-positive, or greater than the defined maximum values, this routine stops execution with an error message. If no out-of-bounds values are detected, the output `INTEGER` status `ierror` is set to zero.

INTERFACE:

```
subroutine CheckBounds_(sMat, ierror)
```

USES:

```
use m_die

use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_indexIA => indexIA

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
```

OUTPUT PARAMETERS:

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

REVISION HISTORY:

```
24Apr01 - Jay Larson <larson@mcs.anl.gov> - Initial prototype.
```

10.1.27 row_sumSP_ - Sum Elements in Each Row

Given an input `SparseMatrix` argument `sMat`, `row_sum_()` returns the number of the rows `num_rows` in the sparse matrix and the sum of the elements in each row in the array `sums`. The input argument `comm` is the Fortran 90 MPI communicator handle used to determine the number of rows and perform the sums. The output arguments `num_rows` and `sums` are valid on all processes.
N.B.: This routine allocates an array `sums`. The user is responsible for deallocating this array when it is no longer needed. Failure to do so will cause a memory leak.

INTERFACE:

```
subroutine row_sumSP_(sMat, num_rows, sums, comm)
```

USES:

```
use m_die
use m_mpif90
use m_realkinds, only : SP, FP

use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_indexIA => indexIA
use m_AttrVect, only : AttrVect_indexRA => indexRA

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
integer,            intent(out) :: num_rows
real(SP), dimension(:), pointer :: sums
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
25Jan01 - Jay Larson <larson@mcs.anl.gov> - Prototype code.
23Apr01 - Jay Larson <larson@mcs.anl.gov> - Modified to accomodate
          changes to the SparseMatrix type.
18May01 - R. Jacob <jacob@mcs.anl.gov> - Use MP_TYPE function
          to set type in the mpi_allreduce
```

10.1.28 row_sum_checkSP_ - Check Row Sums vs. Valid Values

The routine `row_sum_check()` sums the rows of the input distributed (across the communicator identified by `comm`) `SparseMatrix` variable `sMat`. It then compares these sums with the `num_valid` input "valid" values stored in the array `valid_sums`. If all of the sums are within the absolute tolerence specified by the input argument `abs_tol` of any of the valid values, the output `LOGICAL` flag `valid` is set to `.TRUE`. Otherwise, this flag is returned with value `.FALSE`.

INTERFACE:

```
subroutine row_sum_checkSP_(sMat, comm, num_valid, valid_sums, abs_tol, valid)
```

USES:

```
use m_die
use m_realkinds, only : SP, FP

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in)  :: sMat
integer,            intent(in)  :: comm
```

```
integer,            intent(in)  :: num_valid
real(SP),           intent(in)  :: valid_sums(num_valid)
real(SP),           intent(in)  :: abs_tol
```

OUTPUT PARAMETERS:

```
logical,            intent(out) :: valid
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
25Feb01 - Jay Larson <larson@mcs.anl.gov> - Prototype code.
06Jan03 - R. Jacob <jacob@mcs.anl.gov> - create DP and SP versions
```

10.1.29 Sort_ - Generate Index Permutation

The subroutine `Sort_()` uses a list of sorting keys defined by the input `List` argument `key_list`, searches for the appropriate integer or real attributes referenced by the items in `key_list` (that is, it identifies the appropriate entries in sMat%data%iList and `sMat%data%rList`), and then uses these keys to generate an index permutation `perm` that will put the nonzero matrix entries of stored in `sMat%data` in lexicographic order as defined by `key_ist` (the ordering in `key_list` being from left to right. The optional `LOGICAL` array input argument `descend` specifies whether or not to sort by each key in *descending* order or *ascending* order. Entries in `descend` that have value `.TRUE.` correspond to a sort by the corresponding key in descending order. If the argument `descend` is not present, the sort is performed for all keys in ascending order.

INTERFACE:

```
subroutine Sort_(sMat, key_list, perm, descend)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

use m_List ,         only : List

use m_AttrVect, only: AttrVect_Sort => Sort

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix),              intent(in) :: sMat
type(List),                      intent(in) :: key_list
logical, dimension(:), optional, intent(in) :: descend
```

OUTPUT PARAMETERS:

```
integer, dimension(:), pointer              :: perm
```

REVISION HISTORY:

```
24Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

10.1.30 Permute_ - Permute Matrix Elements using Supplied Index Permutation

The subroutine `Permute_()` uses an input index permutation `perm` to re-order the entries of the `SparseMatrix` argument `sMat`. The index permutation `perm` is generated using the routine `Sort_()` (in this module).

INTERFACE:

```
subroutine Permute_(sMat, perm)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

use m_AttrVect, only: AttrVect_Permute => Permute

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:), pointer               :: perm
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),            intent(inout) :: sMat
```

REVISION HISTORY:

```
24Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

10.1.31 SortPermute_ - Sort and Permute Matrix Elements

The subroutine `SortPermute_()` uses a list of sorting keys defined by the input `List` argument `key_list`, searches for the appropriate integer or real attributes referenced by the items in `key_ist` (that is, it identifies the appropriate entries in sMat%data%iList and `sMat%data%rList`), and then uses these keys to generate an index permutation that will put the nonzero matrix entries of stored in `sMat%data` in lexicographic order as defined by `key_list` (the ordering in `key_list` being from left to right. The optional `LOGICAL` array input argument `descend` specifies whether or not to sort by each key in *descending* order or *ascending* order. Entries in `descend` that have value `.TRUE.` correspond to a sort by the corresponding key in descending order. If the argument `descend` is not present, the sort is performed for all keys in ascending order.
Once this index permutation is created, it is applied to re-order the entries of the `SparseMatrix` argument `sMat` accordingly.

INTERFACE:

```
subroutine SortPermute_(sMat, key_list, descend)
```

USES:

```
use m_die ,          only : die
use m_stdio ,        only : stderr

use m_List ,         only : List

implicit none
```

INPUT PARAMETERS:

```
type(List),                       intent(in)    :: key_list
logical, dimension(:), optional, intent(in)    :: descend
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),               intent(inout) :: sMat
```

REVISION HISTORY:

```
24Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

10.2 Module m_SparseMatrixComms – sparse matrix communications methods. (Source File: m_SparseMatrixComms.F90)

The `SparseMatrix` datatype provides sparse matrix storage for the parallel matrix-vector multiplication $\mathbf{y} = \mathbf{Mx}$. This module provides communications services for the `SparseMatrix` type. These services include scattering matrix elements based on row or column decompositions, gathering of matrix elements to the root, and broadcasting from the root.
N.B.: These routines will not communicate the vector portion of a `SparseMatrix`, if it has been initialized. A WARNING will be issued in most cases. In general, do communication first, then call `vecinit`.

INTERFACE:

```
module m_SparseMatrixComms

      private   ! except
```

PUBLIC MEMBER FUNCTIONS:

```
      public :: ScatterByColumn
      public :: ScatterByRow
      public :: Gather
      public :: Bcast

    interface ScatterByColumn ; module procedure &
         ScatterByColumnGSMap_
    end interface

    interface ScatterByRow ; module procedure &
         ScatterByRowGSMap_
    end interface

    interface Gather ; module procedure &
  GM_gather_, &
  GSM_gather_
    end interface

    interface Bcast ; module procedure Bcast_ ; end interface
```

REVISION HISTORY:

```
  13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
            and API specifications.
  10May01 - J.W. Larson <larson@mcs.anl.gov> - added GM_gather_
            and cleaned up prologues.
```

10.2.1 ScatterByColumnGSMap_ - Column-based scatter for SparseMatrix.

This routine scatters the input `SparseMatrix` argument `GsMat` (valid only on the root) to a distributed `SparseMatrix` variable `LsMat` across all the processes present on the communicator associated with the integer handle `comm`. The decomposition defining the scatter is supplied by the input `GlobalSegMap` argument `columnGSMap`. The optional output `INTEGER` flag `stat` signifies a successful (failed) operation if it is returned with value zero (nonzero).
N.B.: This routine returns an allocated `SparseMatrix` variable `LsMat`. The user must destroy this variable when it is no longer needed by invoking `SparseMatrix_Clean()`.

INTERFACE:

```
subroutine ScatterByColumnGSMap_(columnGSMap, GsMat, LsMat, root, comm, stat)
```

USES:

```
use m_die, only : MP_perr_die,die
use m_stdio
use m_mpif90

use m_List, only: List
use m_List, only: List_init => init
use m_List, only: List_clean => clean

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_clean => clean

use m_SparseMatrix, only : SparseMatrix
use m_SparseMatrix, only : SparseMatrix_nRows => nRows
use m_SparseMatrix, only : SparseMatrix_nCols => nCols
use m_SparseMatrix, only : SparseMatrix_SortPermute => SortPermute

use m_SparseMatrixDecomp, only : SparseMatrixDecompByColumn => ByColumn

use m_AttrVectComms, only : AttrVect_Scatter => scatter

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)    :: columnGSMap
integer,            intent(in)    :: root
integer,            intent(in)    :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout) :: GsMat
```

OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(out) :: LsMat
integer, optional,  intent(out) :: stat
```

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial API spec.
10May01 - J.W. Larson <larson@mcs.anl.gov> - cleaned up prologue.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Made status flag stat
          optional, and itititalaze it to zero if it is present.
09Jul03 - E.T. Ong <eong@mcs.anl.gov> - added sorting to distributed
          matrix elements
```

10.2.2 ScatterByRowGSMap_ -Row-based scatter for SparseMatrix.

This routine scatters the input `SparseMatrix` argument `GsMat` (valid only on the root) to a distributed `SparseMatrix` variable `LsMat` across all the processes present on the communicator associated with the integer handle `comm`. The decomposition defining the scatter is supplied by the input `GlobalSegMap` argument `rowGSMap`. The output integer flag `stat` signifies a successful (failed) operation if it is returned with value zero (nonzero).

N.B.: This routine returns an allocated `SparseMatrix` variable `LsMat`. The user must destroy this variable when it is no longer needed by invoking `SparseMatrix_Clean()`.

INTERFACE:

```
subroutine ScatterByRowGSMap_(rowGSMap, GsMat, LsMat, root, comm, stat)
```

USES:

```
use m_die, only : MP_perr_die,die
use m_stdio
use m_mpif90

use m_List, only: List
use m_List, only: List_init => init
use m_List, only: List_clean => clean

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_clean => clean

use m_SparseMatrix, only : SparseMatrix
use m_SparseMatrix, only : SparseMatrix_nRows => nRows
use m_SparseMatrix, only : SparseMatrix_nCols => nCols
use m_SparseMatrix, only : SparseMatrix_SortPermute => SortPermute

use m_SparseMatrixDecomp, only : SparseMatrixDecompByRow => ByRow

use m_AttrVectComms, only : AttrVect_Scatter => scatter

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)    :: rowGSMap
integer,            intent(in)    :: root
integer,            intent(in)    :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout) :: GsMat
```

OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(out) :: LsMat
integer, optional,  intent(out) :: stat
```

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial API spec.
26Apr01 - R.L. Jacob  <jacob@mcs.anl.gov> - fix use statement
          from SMDecomp so it points to ByRow
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Made status flag stat
          optional, and initialize it to zero if it is present.
09Jul03 - E.T. Ong <eong@mcs.anl.gov> - Added sorting to distributed
          matrix elements.
```

10.2.3 GM_gather_ - Gather a distributed SparseMatrix to the root.

This routine gathers the input distributed `SparseMatrix` argument `LsMat` to the `SparseMatrix` variable `GsMat` on the root. The decomposition defining the gather is supplied by the input `GlobalMap` argument `GMap`. The status flag `stat` has value zero (nonzero) if the operation has succeeded (failed).
N.B.: This routine returns an allocated `SparseMatrix` variable `GsMat`. The user must destroy this variable when it is no longer needed by invoking `SparseMatrix_Clean()`.

INTERFACE:

```
subroutine GM_gather_(LsMat, GsMat, GMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die, only : die

use m_GlobalMap, only: GlobalMap

use m_SparseMatrix, only: SparseMatrix
use m_SparseMatrix, only: SparseMatrix_nRows => nRows
use m_SparseMatrix, only: SparseMatrix_nCols => nCols

use m_AttrVectComms, only : AttrVect_gather => gather

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in) :: LsMat
type(GlobalMap),    intent(in) :: GMap
integer,            intent(in) :: root
integer,            intent(in) :: comm
```

OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(out) :: GsMat
integer, optional,  intent(out) :: stat
```

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial API spec.
10May01 - J.W. Larson <larson@mcs.anl.gov> - initial routine and
          prologue
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Made status flag stat
          optional, and ititilaze it to zero if it is present.
```

10.2.4 GSM_gather_ - Gather a distributed SparseMatrix to the root.

This routine gathers the input distributed `SparseMatrix` argument `LsMat` to the `SparseMatrix` variable `GsMat` on the root. The decomposition defining the gather is supplied by the input `GlobalSegMap` argument `GSMap`. The status flag `stat` has value zero (nonzero) if the operation has succeeded (failed).
N.B.: This routine returns an allocated `SparseMatrix` variable `GsMat`. The user must destroy this variable when it is no longer needed by invoking `SparseMatrix_Clean()`.

INTERFACE:

```
subroutine GSM_gather_(LsMat, GsMat, GSMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die, only : die

use m_GlobalSegMap, only: GlobalSegMap

use m_SparseMatrix, only: SparseMatrix
use m_SparseMatrix, only: SparseMatrix_nRows => nRows
use m_SparseMatrix, only: SparseMatrix_nCols => nCols

use m_AttrVectComms, only : AttrVect_gather => gather

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrix), intent(in) :: LsMat
type(GlobalSegMap), intent(in) :: GSMap
integer,            intent(in) :: root
integer,            intent(in) :: comm
```

OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(out) :: GsMat
integer, optional,  intent(out) :: stat
```

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial API spec.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Made status flag stat
          optional, and itititilaze it to zero if it is present.
```

10.2.5 Bcast_ - Broadcast a SparseMatrix.

This routine broadcasts the `SparseMatrix` argument `sMat` from the root to all processes on the communicator associated with the communicator handle `comm`. The status flag `stat` has value zero if the operation has succeeded.
N.B.: This routine returns an allocated `SparseMatrix` variable `sMat`. The user must destroy this variable when it is no longer needed by invoking `SparseMatrix_Clean()`.
N.B.: This routine will exit with an error if the vector portion of `sMat` has been initialized prior to broadcast.

INTERFACE:

```
subroutine Bcast_(sMat, root, comm, stat)
```

USES:

```
use m_die, only : MP_perr_die,die
use m_stdio
use m_mpif90

use m_GlobalSegMap, only: GlobalSegMap
```

```
use m_AttrVectComms, only : AttrVect_bcast => bcast

use m_SparseMatrix, only: SparseMatrix
use m_SparseMatrix, only: SparseMatrix_nRows => nRows
use m_SparseMatrix, only: SparseMatrix_nCols => nCols

implicit none
```

INPUT PARAMETERS:

```
integer,            intent(in) :: root
integer,            intent(in) :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout) :: sMat
```

OUTPUT PARAMETERS:

```
integer, optional,  intent(out) :: stat
```

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial API spec/code
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Made status flag stat
          optional, and ititilaze it to zero if it is present.
17Jul02 - J.W. Larson <larson@mcs.anl.gov> - Bug fix--local
          process ID myID was uninitialized.
```

10.3 Module m_SparseMatrixDecomp – Parallel sparse matrix decomposition. (Source File: m_SparseMatrixDecomp.F90)

The `SparseMatrix` datatype provides sparse matrix storage for the parallel matrix-vector multiplication $\mathbf{y} = \mathbf{M}\mathbf{x}$. This module provides services to create decompositions for the `SparseMatrix`. The matrix decompositions available are row and column decompositions. They are generated by invoking the appropriate routine in this module, and passing the corresponding *vector* decomposition. For a row (column) decomposition, one invokes the routine `ByRow()` (`ByColumn()`), passing the domain decomposition for the vector $\mathbf{y}$ ($\mathbf{x}$).

INTERFACE:

```
module m_SparseMatrixDecomp

     private   ! except
```

PUBLIC MEMBER FUNCTIONS:

```
     public :: ByColumn
     public :: ByRow


   interface ByColumn ; module procedure &
        ByColumnGSMap_
   end interface

   interface ByRow ; module procedure &
        ByRowGSMap_
   end interface
```

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
          and API specifications.
03Aug01 - E. Ong <eong@mcs.anl.gov> - in ByRowGSMap and ByColumnGSMap,
          call GlobalSegMap_init on non-root processes with actual
          shaped arguments to satisfy Fortran 90 standard. See
          comments in ByRowGSMap/ByColumnGSMap.
```

10.3.1 ByColumnGSMap_ - Generate Row-based GlobalSegMap for SparseMatrix

INTERFACE:

```
subroutine ByColumnGSMap_(xGSMap, sMat, sMGSMap, root, comm)
```

USES:

```
  use m_die,  only: MP_perr_die,die

  use m_List, only: List
  use m_List, only: List_init => init
  use m_List, only: List_clean => clean

  use m_AttrVect, only: AttrVect
  use m_AttrVect, only: AttrVect_init => init
  use m_AttrVect, only: AttrVect_zero => zero
  use m_AttrVect, only: AttrVect_lsize => lsize
```

```
use m_AttrVect, only: AttrVect_indexIA => indexIA
use m_AttrVect, only: AttrVect_copy => copy
use m_AttrVect, only: AttrVect_clean => clean

use m_AttrVectComms, only: AttrVect_scatter => scatter
use m_AttrVectComms, only: AttrVect_gather => gather

use m_GlobalMap, only : GlobalMap
use m_GlobalMap, only : GlobalMap_init => init
use m_GlobalMap, only : GlobalMap_clean => clean

use m_GlobalSegMap, only: GlobalSegMap
use m_GlobalSegMap, only: GlobalSegMap_init => init
use m_GlobalSegMap, only: GlobalSegMap_peLocs => peLocs
use m_GlobalSegMap, only: GlobalSegMap_comp_id => comp_id

use m_SparseMatrix, only: SparseMatrix
use m_SparseMatrix, only: SparseMatrix_lsize => lsize
use m_SparseMatrix, only: SparseMatrix_SortPermute => SortPermute

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)    :: xGSMap
integer,            intent(in)    :: root
integer,            intent(in)    :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout) :: sMat
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap), intent(out) :: sMGSMap
```

DESCRIPTION:

This routine is invoked from all processes on the communicator `comm` to create from an input `SparseMatrix sMat` (valid only on the `root` process) and an input **x**-vector decomposition described by the `GlobalSegMap` argument `xGSMap` (valid at least on the `root`) to create an output `GlobalSegMap` decomposition of the matrix elements `sMGSMap`, which is valid on all processes on the communicator. This matrix `GlobalSegMap` describes the corresponding column decomposition of `sMat`.
N.B.: The argument `sMat` is returned sorted in lexicographic order by column and row.

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial API spec.
26Apr01 - R.L. Jacob <jacob@mcs.anl.gov> - add use statements for
          GlobalSegMap_init and GSMap_peLocs.
          Add gsize argument required to GSMap_peLocs.
          Add underscore to ComputeSegments call so it matches
          the subroutine decleration.
          change attribute on starts,lengths, and pe_locs to
          pointer to match GSMap_init.
          add use m_die statement
```

```
   26Apr01 - J.W. Larson <larson@mcs.anl.gov> - fixed major logic bug
             that had all processes executing some operations that
             should only occur on the root.
   09Jul03 - E.T. Ong <eong@mcs.anl.gov> - call pe_locs in parallel.
             reduce the serial sort from gcol:grow to just gcol.
```

10.3.2 ByRowGSMap_ - Generate Row-based GlobalSegMap for SparseMatrix

INTERFACE:

```
  subroutine ByRowGSMap_(yGSMap, sMat, sMGSMap, root, comm)
```

USES:

```
    use m_die,  only: MP_perr_die,die

    use m_List, only: List
    use m_List, only: List_init => init
    use m_List, only: List_clean => clean

    use m_AttrVect, only: AttrVect
    use m_AttrVect, only: AttrVect_init => init
    use m_AttrVect, only: AttrVect_lsize => lsize
    use m_AttrVect, only: AttrVect_indexIA => indexIA
    use m_AttrVect, only: AttrVect_copy => copy
    use m_AttrVect, only: AttrVect_clean => clean
    use m_AttrVect, only: AttrVect_zero => zero

    use m_AttrVectComms, only: AttrVect_scatter => scatter
    use m_AttrVectComms, only: AttrVect_gather => gather

    use m_GlobalMap, only : GlobalMap
    use m_GlobalMap, only : GlobalMap_init => init
    use m_GlobalMap, only : GlobalMap_clean => clean

    use m_GlobalSegMap, only: GlobalSegMap
    use m_GlobalSegMap, only: GlobalSegMap_init => init
    use m_GlobalSegMap, only: GlobalSegMap_peLocs => peLocs
    use m_GlobalSegMap, only: GlobalSegMap_comp_id => comp_id

    use m_SparseMatrix, only: SparseMatrix
    use m_SparseMatrix, only: SparseMatrix_lsize => lsize
    use m_SparseMatrix, only: SparseMatrix_SortPermute => SortPermute

    implicit none
```

INPUT PARAMETERS:

```
    type(GlobalSegMap), intent(in)    :: yGSMap
    integer,            intent(in)    :: root
    integer,            intent(in)    :: comm
```

INPUT/OUTPUT PARAMETERS:

```
    type(SparseMatrix), intent(inout) :: sMat
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap), intent(out) :: sMGSMap
```

DESCRIPTION:

This routine is invoked from all processes on the communicator `comm` to create from an input `SparseMatrix sMat` (valid only on the `root` process) and an input **y**-vector decomposition described by the `GlobalSegMap` argument `yGSMap` (valid at least on the `root`) to create an output `GlobalSegMap` decomposition of the matrix elements `sMGSMap`, which is valid on all processes on the communicator. This matrix `GlobalSegMap` describes the corresponding row decomposition of `sMat`. **N.B.**: The argument `sMat` is returned sorted in lexicographic order by row and column.

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial API spec.
26Apr01 - R.L. Jacob <jacob@mcs.anl.gov> - add use statements for
          GlobalSegMap_init and GSMap_peLocs.
          Add gsize argument required to GSMap_peLocs.
          Add underscore to ComputeSegments call so it matches
          the subroutine decleration.
          change attribute on starts,lengths, and pe_locs to
           pointer to match GSMap_init.
26Apr01 - J.W. Larson <larson@mcs.anl.gov> - fixed major logic bug
          that had all processes executing some operations that
          should only occur on the root.
09Jun03 - E.T. Ong <eong@mcs.anl.gov> - call peLocs in parallel.
          reduce the serial sort from grow:gcol to just grow.
```

10.3.3 ComputeSegments_ - Create segments from list data.

INTERFACE:

```
subroutine ComputeSegments_(element_pe_locs, elements, num_elements, &
                            nsegs, seg_starts, seg_lengths, seg_pe_locs)
```

USES:

```
use m_die,  only: die

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:), intent(in)  :: element_pe_locs
integer, dimension(:), intent(in)  :: elements
integer,               intent(in)  :: num_elements
```

OUTPUT PARAMETERS:

```
integer,               intent(out) :: nsegs
integer, dimension(:), pointer     :: seg_starts
integer, dimension(:), pointer     :: seg_lengths
integer, dimension(:), pointer     :: seg_pe_locs
```

DESCRIPTION:

This routine examins an input list of `num_elements` process ID locations stored in the array `element_pe_locs`, counts the number of contiguous segments `nsegs`, and returns the segment start index, length, and process ID location in the arrays `seg_starts(:)`, `seg_lengths(:)`, and `seg_pe_locs(:)`, respectively.
N.B.: The argument `sMat` is returned sorted in lexicographic order by row and column.

REVISION HISTORY:

```
18Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial version.
28Aug01 - M.J. Zavislak <zavislak@mcs.anl.gov>
              Changed first sanity check to get size(element_pe_locs)
              instead of size(elements)
```

10.4 Module m_SparseMatrixToMaps – Maps from the Sparse Matrix (Source File: m_SparseMatrixToMaps.F90)

The `SparseMatrix` provides consolidated (on one process) or distributed sparse matrix storage for the operation $\mathbf{y} = \mathbf{M}\mathbf{x}$, where $\mathbf{x}$ and $\mathbf{y}$ are vectors, and $\mathbf{M}$ is a matrix. In performing parallel matrix-vector multiplication, one has numerous options regarding the decomposition of the matrix $\mathbf{M}$, and the vectors $\mathbf{y}$ and $\mathbf{x}$. This module provides services to generate mct mapping components—the `GlobalMap` and `GlobalSegMap` for the vectors $\mathbf{y}$ and/or $\mathbf{x}$ based on the decomposition of the sparse matrix $\mathbf{M}$.

INTERFACE:

```
module m_SparseMatrixToMaps
```

USES:

```
      use m_SparseMatrix, only : SparseMatrix

      implicit none

      private   ! except

      public :: SparseMatrixToXGlobalSegMap
      public :: SparseMatrixToYGlobalSegMap

    interface SparseMatrixToXGlobalSegMap ; module procedure &
  SparseMatrixToXGlobalSegMap_
    end interface

    interface SparseMatrixToYGlobalSegMap ; module procedure &
  SparseMatrixToYGlobalSegMap_
    end interface
```

REVISION HISTORY:

```
  13Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
            and API specifications.
```

10.4.1 SparseMatrixToXGlobalSegMap_ - Generate X GlobalSegmap.

Given an input `SparseMatrix` argument `sMat`, this routine generates an output `GlobalSegMap` variable `xGSMap`, which describes the domain decomposition of the vector $\mathbf{x}$ in the distributed matrix-vector multiplication

$$\mathbf{y} = \mathbf{M}\mathbf{x}.$$

INTERFACE:

```
subroutine SparseMatrixToXGlobalSegMap_(sMat, xGSMap, root, comm, comp_id)
```

USES:

```
      use m_stdio, only : stderr
      use m_die,   only : die
      use m_mpif90

      use m_List, only : List
      use m_List, only : List_init => init
      use m_List, only : List_clean => clean
```

```
use m_SparseMatrix, only : SparseMatrix
use m_SparseMatrix, only : SparseMatrix_nCols => nCols
use m_SparseMatrix, only : SparseMatrix_lsize => lsize
use m_SparseMatrix, only : SparseMatrix_indexIA => indexIA
use m_SparseMatrix, only : SparseMatrix_SortPermute => SortPermute

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_init => init

implicit none
```

INPUT PARAMETERS:

```
integer,            intent(in)    :: root    ! communicator root
integer,            intent(in)    :: comm    ! communicator handle
integer,            intent(in)    :: comp_id ! component id
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout) :: sMat    ! input SparseMatrix
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap), intent(out)   :: xGSMap  ! segmented decomposition
                                             ! for x
```

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - API specification.
25Apr01 - J.W. Larson <larson@mcs.anl.gov> - First version.
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - Bug fix--intent of
          argument sMat changed from (IN) to (INOUT)
27Apr01 - R.L. Jacob <jacob@mcs.anl.gov> - bug fix-- add use
          statement for SortPermute
01May01 - R.L. Jacob <jacob@mcs.anl.gov> - make comp_id an
          input argument
```

10.4.2 SparseMatrixToYGlobalSegMap_ - Generate Y GlobalSegmap.

Given an input `SparseMatrix` argument `sMat`, this routine generates an output `GlobalSegMap` variable `yGSMap`, which describes the domain decomposition of the vector $\mathbf{y}$ in the distributed matrix-vector multiplication $\mathbf{y} = \mathbf{Mx}$.

INTERFACE:

```
subroutine SparseMatrixToYGlobalSegMap_(sMat, yGSMap, root, comm, comp_id)
```

USES:

```
use m_stdio, only : stderr
use m_die,   only : die

use m_List, only : List
use m_List, only : List_init => init
use m_List, only : List_clean => clean

use m_SparseMatrix, only : SparseMatrix
```

```
use m_SparseMatrix, only : SparseMatrix_nRows => nRows
use m_SparseMatrix, only : SparseMatrix_lsize => lsize
use m_SparseMatrix, only : SparseMatrix_indexIA => indexIA
use m_SparseMatrix, only : SparseMatrix_SortPermute => SortPermute

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_init => init

implicit none
```

INPUT PARAMETERS:

```
integer,            intent(in)    :: root    ! communicator root
integer,            intent(in)    :: comm    ! communicator handle
integer,            intent(in)    :: comp_id ! component id
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout) :: sMat    ! input SparseMatrix
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap), intent(out)   :: yGSMap  ! segmented decomposition
                                             ! for y
```

REVISION HISTORY:

```
13Apr01 - J.W. Larson <larson@mcs.anl.gov> - API specification.
25Apr01 - J.W. Larson <larson@mcs.anl.gov> - initial code.
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - Bug fix--intent of
          argument sMat changed from (IN) to (INOUT)
27Apr01 - R.L. Jacob <jacob@mcs.anl.gov> - bug fix-- add use
          statement for SortPermute
01May01 - R.L. Jacob <jacob@mcs.anl.gov> - make comp_id an
          input argument
07May02 - J.W. Larson <larson@mcs.anl.gov> - Changed interface to
          make it consistent with SparseMatrixToXGlobalSegMap_().
```

10.4.3 CreateSegments_ - Generate segment information.

This routine examines an input `INTEGER` list of numbers `indices` (of length `num_indices`), determines the number of segments of consecutive numbers (or runs) `nsegs`. The starting indices for each run, and their lengths are returned in the `INTEGER` arrays `starts(:)` and `lengths(:)`, respectively.

INTERFACE:

```
subroutine ComputeSegments_(indices, num_indices, nsegs, starts, lengths)
```

USES:

```
use m_stdio, only : stderr
use m_die,   only : die

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:), intent(in)  :: indices
integer,               intent(in)  :: num_indices
```

OUTPUT PARAMETERS:

```
integer,               intent(out) :: nsegs
integer, dimension(:), pointer     :: starts
integer, dimension(:), pointer     :: lengths
```

REVISION HISTORY:

```
19Apr01 - J.W. Larson <larson@mcs.anl.gov> - API specification.
25Apr01 - J.W. Larson <larson@mcs.anl.gov> - Initial code.
27Apr01 - J.W. Larson <larson@mcs.anl.gov> - Bug fix--error in
          computation of segment starts/lengths.
27Nov01 - E.T. Ong <eong@mcs.anl.gov> - Bug fix--initialize
          nsegs=0 in case num_indices=0.
```

10.5 Module m_SparseMatrixPlus – Class Parallel for Matrix-Vector Multiplication (Source File: m_SparseMatrixPlus.F90)

Matrix-vector multiplication is one of the MCT's core services, and is used primarily for the interpolation of data fields from one physical grid to another. Let $\mathbf{x} \in \Re^{N_x}$ and $\mathbf{y} \in \Re^{N_y}$ represent data fields on physical grids A and B, respectively. Field data is interpolated from grid A to grid B by

$$\mathbf{y} = \mathbf{M}\mathbf{x},$$

where $\mathbf{M}$ is aa $N_y \times N_x$ matrix.
Within MCT, the `SparseMatrix` data type is MCT's object for storing sparse matrices such as $\mathbf{M}$, and the `AttrVect` data type is MCT's field data storage object. That is, $\mathbf{x}$ and $\mathbf{y}$ are each stored in `AttrVect` form, and $\mathbf{M}$ is stored as a `SparseMatrix`.
For global address spaces (uniprocessor or shared-memory parallel), this picture of matrix-vector multiplication is sufficient. If one wishes to perform *distributed-memory parallel* matrix-vector multiplication, however, in addition to computation, one must consider *communication*.
There are three basic message-passing parallel strategies for computing $\mathbf{y} = \mathbf{M}\mathbf{x}$:

1. Decompose $\mathbf{M}$ based on its *rows*, and corresponding to the decomposition for the vector $\mathbf{y}$. That is, if a given process owns the i^{th} element of $\mathbf{y}$, then all the elements of row i of $\mathbf{M}$ also reside on this process. Then $\mathbf{y} = \mathbf{M}\mathbf{x}$ is implemented as follows:
   (a) Create an *intermediate vector* **x'** that is the pre-image of the elements of $\mathbf{y}$ owned locally.
   (b) Comunnicate with the appropriate processes on the local communicator to gather from $\mathbf{x}$ the elements of **x'**.
   (c) Compute $\mathbf{y} = \mathbf{M}\mathbf{x}'$.
   (d) Destroy the data structure holding **x'**.
2. Decompose $\mathbf{M}$ based on its *columns*, and corresponding to the decomposition for the vector $\mathbf{x}$. That is, if a given process owns the j^{th} element of $\mathbf{x}$, then all the elements of column j of $\mathbf{M}$ also reside on this process. Then $\mathbf{y} = \mathbf{M}\mathbf{x}$ is implemented as follows:
   (a) Create an *intermediate vector* **y'** that holds *partial sums* of elements of $\mathbf{y}$ computed from $\mathbf{x}$ and $\mathbf{M}$.
   (b) Compute $\mathbf{y}' = \mathbf{M}\mathbf{x}$.
   (c) Perform communications to route elements of **y'** to their eventual destinations in $\mathbf{y}$, where they will be summed, resulting in the distributed vector $\mathbf{y}$.
   (d) Destroy the data structure holding **y'**.
3. Decompose $\mathbf{M}$ based on some arbitrary, user-supplied scheme. This will necessitate two intermediate vectors **x'** and **y'**. Then $\mathbf{y} = \mathbf{M}\mathbf{x}$ is implemented as follows:
   (a) Create *intermediate vectors* **x'** and **y'**. The numbers of elements in **x'** and **y'** are based $\mathbf{M}$, specifically its numbers of *distinct* row and column index values, respectively.
   (b) Comunnicate with the appropriate processes on the local communicator to gather from $\mathbf{x}$ the elements of **x'**.
   (c) Compute $\mathbf{y}' = \mathbf{M}\mathbf{x}'$.
   (d) Perform communications to route elements of **y'** to their eventual destinations in $\mathbf{y}$, where they will be summed, resulting in the distributed vector $\mathbf{y}$.
   (e) Destroy the data structures holding **x'** and **y'**.

These operations require information about many aspects of the multiplication process. These data are:

- The matrix-vector parallelization strategy, which is one of the following:
  1. Distributed in $\mathbf{x}$, purely data local in $\mathbf{y}$, labeled by the public data member `Xonly`
  2. Purely data local $\mathbf{x}$, distributed in $\mathbf{y}$, labeled by the public data member `Yonly`

3. Distributed in both **x** and **y**, labeled by the public data member `XandY`

- A communications scheduler to create **x'** from **x**;
- A communications scheduler to deliver partial sums contained in **y'** to **y**.
- Lengths of the intermediate vectors **x'** and **y'**.

In MCT, the above data are stored in a *master* class for `SparseMatrix- AttrVect` multiplication. This master class is called a `SparseMatrixPlus`.
This module contains the definition of the `SparseMatrixPlus`, and a variety of methods to support it. These include initialization, destruction, query, and data import/export.

INTERFACE:

```
module m_SparseMatrixPlus
```

USES:

```
      use m_String, only : String
      use m_SparseMatrix, only : SparseMatrix
      use m_Rearranger, only : Rearranger
```

PUBLIC TYPES:

```
      public :: SparseMatrixPlus

      Type SparseMatrixPlus
#ifdef SEQUENCE
        sequence
#endif
        type(String) :: Strategy
        integer :: XPrimeLength
        type(Rearranger) :: XToXPrime
        integer :: YPrimeLength
        type(Rearranger) :: YPrimeToY
        type(SparseMatrix) :: Matrix
 integer :: Tag
      End Type SparseMatrixPlus
```

PUBLIC MEMBER FUNCTIONS:

```
      public :: init
      public :: vecinit
      public :: clean
      public :: initialized
      public :: exportStrategyToChar

      interface init ; module procedure &
        initFromRoot_, &
        initDistributed_
      end interface
      interface vecinit ; module procedure vecinit_ ; end interface
      interface clean ; module procedure clean_ ; end interface
      interface initialized ; module procedure initialized_ ; end interface
      interface exportStrategyToChar ; module procedure &
        exportStrategyToChar_
      end interface
```

PUBLIC DATA MEMBERS:

```
public :: Xonly ! Matrix decomposed only by ROW (i.e., based
                ! on the decomposition of y); comms x->x'
public :: Yonly ! Matrix decomposed only by COLUMN (i.e., based
                ! on the decomposition of x); comms y'->y
public :: XandY ! Matrix has complex ROW/COLUMN decomposed
```

DEFINED PARAMETERS:

```
integer,parameter                    :: DefaultTag = 700
```

SEE ALSO:

```
The MCT module m_SparseMatrix for more information about Sparse Matrices.
The MCT module m_Rearranger for deatailed information about Communications
scheduling.
The MCT module m_AttrVect for details regarding the Attribute Vector.
The MCT module m_MatAttrVectMult for documentation of API's that use
the SparseMatrixPlus.
```

REVISION HISTORY:

```
29August 2002 - J. Larson <larson@mcs.anl.gov> - API specification.
```

10.5.1 initFromRoot_ - Creation and Initializtion from the Root

This routine creates an `SparseMatrixPlus` `sMatPlus` using the following elements:

- A `SparseMatrix` (the input argument `sMat`), whose elements all reside only on the `root` process of the MPI communicator with an integer handle defined by the input `INTEGER` argument `comm`;
- A `GlobalSegMap` (the input argument `xGSMap`) describing the domain decomposition of the vector `x` on the communicator `comm`;
- A `GlobalSegMap` (the input argument `yGSMap`) describing the domain decomposition of the vector `y` on the communicator `comm`;
- The matrix-vector multiplication parallelization strategy. This is set by the input `CHARACTER` argument `strategy`, which must have value corresponding to one of the following public data members defined in the declaration section of this module. Acceptable values for use in this routine are: `Xonly` and `Yonly`.

The optional argument `Tag` can be used to set the tag value used in the call to `Rearranger`. DefaultTag will be used otherwise.

INTERFACE:

```
subroutine initFromRoot_(sMatPlus, sMat, xGSMap, yGSMap, strategy, &
                         root, comm, ComponentID, Tag)
```

USES:

```
use m_die
use m_stdio
use m_mpif90

use m_String, only : String
```

```
use m_String, only : String_init => init

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize
use m_GlobalSegMap, only : GlobalSegMap_lsize => lsize
use m_GlobalSegMap, only : GlobalSegMap_clean => clean

use m_SparseMatrix, only : SparseMatrix
use m_SparseMatrix, only : SparseMatrix_nRows => nRows
use m_SparseMatrix, only : SparseMatrix_nCols => nCols

use m_SparseMatrixComms, only : SparseMatrix_ScatterByRow => ScatterByRow
use m_SparseMatrixComms, only : SparseMatrix_ScatterByColumn => &
                                                        ScatterByColumn

use m_SparseMatrixToMaps, only : SparseMatrixToXGlobalSegMap
use m_SparseMatrixToMaps, only : SparseMatrixToYGlobalSegMap

use m_GlobalToLocal, only : GlobalToLocalMatrix

use m_Rearranger, only : Rearranger
use m_Rearranger, only : Rearranger_init => init

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),     intent(in)    :: xGSMap
type(GlobalSegMap),     intent(in)    :: yGSMap
character(len=*),       intent(in)    :: strategy
integer,                intent(in)    :: root
integer,                intent(in)    :: comm
integer,                intent(in)    :: ComponentID
integer,optional,       intent(in)    :: Tag
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),     intent(inout) :: sMat
```

OUTPUT PARAMETERS:

```
type(SparseMatrixPlus), intent(out)   :: SMatPlus
```

REVISION HISTORY:

30Aug02 - Jay Larson <larson@mcs.anl.gov> - API Specification

---

10.5.2 initDistributed_ - Distributed Creation and Initializtion

This routine creates an `SparseMatrixPlus` `sMatPlus` using the following elements:

- A `SparseMatrix` (the input argument `sMat`), whose elements have previously been destributed across the MPI communicator with an integer handle defined by the input `INTEGER` argument `comm`;
- A `GlobalSegMap` (the input argument `xGSMap`) describing the domain decomposition of the vector **x** on the communicator `comm`; and

- A `GlobalSegMap` (the input argument `yGSMap`) describing the domain decomposition of the vector `y` on the communicator `comm`;

The other input arguments required by this routine are the `INTEGER` arguments `root` and `ComponentID`, which define the communicator root ID and MCT component ID, respectively.

INTERFACE:

```
subroutine initDistributed_(sMatPlus, sMat, xGSMap, yGSMap, root, comm, &
                            ComponentID, Tag)
```

USES:

```
use m_die
use m_stdio
use m_mpif90

use m_String, only : String
use m_String, only : String_init => init

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize
use m_GlobalSegMap, only : GlobalSegMap_lsize => lsize
use m_GlobalSegMap, only : GlobalSegMap_clean => clean

use m_SparseMatrix, only : SparseMatrix
use m_SparseMatrix, only : SparseMatrix_nRows => nRows
use m_SparseMatrix, only : SparseMatrix_nCols => nCols
use m_SparseMatrix, only : SparseMatrix_Copy => Copy

use m_SparseMatrixComms, only : SparseMatrix_ScatterByRow => ScatterByRow
use m_SparseMatrixComms, only : SparseMatrix_ScatterByColumn => &
                                                        ScatterByColumn

use m_SparseMatrixToMaps, only : SparseMatrixToXGlobalSegMap
use m_SparseMatrixToMaps, only : SparseMatrixToYGlobalSegMap

use m_GlobalToLocal, only : GlobalToLocalMatrix

use m_Rearranger, only : Rearranger
use m_Rearranger, only : Rearranger_init => init

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),     intent(in)    :: xGSMap
type(GlobalSegMap),     intent(in)    :: yGSMap
integer,                intent(in)    :: root
integer,                intent(in)    :: comm
integer,                intent(in)    :: ComponentID
integer,optional,       intent(in)    :: Tag
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix),     intent(inout) :: sMat
```

OUTPUT PARAMETERS:

```
type(SparseMatrixPlus), intent(out)   :: SMatPlus
```

REVISION HISTORY:

30Aug02 - Jay Larson <larson@mcs.anl.gov> - API Specification

---

10.5.3 vecinit_ - Initialize vector parts of a SparseMatrixPlus

This routine will initialize the parts of the SparseMatrix in the SparseMatrixPlus object that are used in the vector-friendly version of the sparse matrix multiply.

INTERFACE:

```
subroutine vecinit_(SMatP)
```

USES:

```
use m_die
use m_SparseMatrix, only : SparseMatrix_vecinit => vecinit

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrixPlus), intent(inout)  :: SMatP
```

REVISION HISTORY:

29Oct03 - R. Jacob <jacob@mcs.anl.gov> - initial prototype

10.5.4 clean_ - Destruction of a SparseMatrixPlus Object

This routine deallocates all allocated memory belonging to the input/output `SparseMatrixPlus` argument `SMatP`, and sets to zero its integer components describing intermediate vector length, and sets its `LOGICAL` flag signifying initialization to `.FALSE.` The success (failure) of this operation is signified by the zero (non-zero) value of the optional `INTEGER` output argument `status`. If the user does supply `status` when invoking this routine, failure of `clean_()` will lead to termination of execution with an error message.

INTERFACE:

```
subroutine clean_(SMatP, status)
```

USES:

```
use m_die
use m_stdio

use m_String, only : String
use m_String, only : String_init => init
use m_String, only : String_ToChar => toChar
use m_String, only : String_clean => clean

use m_SparseMatrix, only : SparseMatrix
use m_SparseMatrix, only : SparseMatrix_clean => clean
```

```
use m_Rearranger, only : Rearranger
use m_Rearranger, only : Rearranger_clean => clean

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrixPlus), intent(inout)  :: SMatP
```

OUTPUT PARAMETERS:

```
integer, optional,  intent(out)   :: status
```

REVISION HISTORY:

```
30Aug02 - Jay Larson <larson@mcs.anl.gov> - API Specification
```

10.5.5 initialized_ - Confirmation of Initialization

This `LOGICAL` query function tells the user if the input `SparseMatrixPlus` argument `sMatPlus` has been initialized. The return value of `initialized_` is `.TRUE.` if `sMatPlus` has been previously initialized, `.FALSE.` if it has not.

INTERFACE:

```
logical function initialized_(sMatPlus)
```

USES:

```
No external modules are used by this function.

use m_String, only : String_len
use m_List,   only : List
use m_List,   only : List_init => init
use m_List,   only : List_identical => identical
use m_List,   only : List_clean => clean

use m_die

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrixPlus), intent(in)  :: sMatPlus
```

REVISION HISTORY:

```
26Sep02 - Jay Larson <larson@mcs.anl.gov> - Implementation
```

10.5.6 exportStrategyToChar - Return Parallelization Strategy

This query subroutine returns the parallelization strategy set in the input `SparseMatrixPlus` argument `sMatPlus`. The result is returned in the output `CHARACTER` argument `StratChars`.

INTERFACE:

```
function exportStrategyToChar_(sMatPlus)
```

USES:

```
use m_stdio
use m_die

use m_String, only : String_ToChar => toChar
use m_String, only : String_init => init
use m_String, only : String_clean => clean
use m_String, only : String

implicit none
```

INPUT PARAMETERS:

```
type(SparseMatrixPlus), intent(in)  :: sMatPlus
```

OUTPUT PARAMETERS:

```
character(len=size(sMatPlus%Strategy%c)) :: exportStrategyToChar_
```

REVISION HISTORY:

```
01Aug07 - Jay Larson <larson@mcs.anl.gov> - Implementation
```

11 Matrix Vector Multiplication

11.1 Module m_MatAttrVectMul - Sparse Matrix AttrVect Multipication. (Source File: m_MatAttrVectMul.F90)

This module contains routines supporting the sparse matrix-vector multiplication

$$\mathbf{y} = \mathbf{M}\mathbf{x},$$

where the vectors $\mathbf{x}$ and $\mathbf{y}$ are stored using the MCT `AttrVect` datatype, and $\mathbf{M}$ is stored using either the MCT `SparseMatrix` or `SparseMatrixPlus` type. The `SparseMatrix` type is used to represent $\mathbf{M}$ if the multiplication process is purely data-local (e.g., in a global address space, or if the process has been rendered embarrasingly parallel by earlier or subsequent vector data redistributions). If the multiplication process is to be explicitly distributed-memory parallel, then the `SparseMatrixPlus` type is used to store the elements of $\mathbf{M}$ and all information needed to coordinate data redistribution and reduction of partial sums.
N.B.: The matrix-vector multiplication routines in this module process only the **real** attributes of the `AttrVect` arguments corresponding to $\mathbf{x}$ and $\mathbf{y}$. They ignore the integer attributes.

INTERFACE:

```
module m_MatAttrVectMul

     private   ! except

     public :: sMatAvMult        ! The master Sparse Matrix -
                                 ! Attribute Vector multipy API

   interface sMatAvMult   ; module procedure &
       sMatAvMult_DataLocal_, &
       sMatAvMult_sMPlus_
   end interface
```

SEE ALSO:

```
The MCT module m_AttrVect for more information about the AttrVect type.
The MCT module m_SparseMatrix for more information about the SparseMatrix
type.
The MCT module m_SparseMatrixPlus for more details about the master class
for parallel sparse matrix-vector multiplication, the SparseMatrixPlus.
```

REVISION HISTORY:

```
12Jan01 - J.W. Larson <larson@mcs.anl.gov> - initial module.
26Sep02 - J.W. Larson <larson@mcs.anl.gov> - added high-level, distributed
          matrix-vector multiply routine using the SparseMatrixPlus class.
```

11.1.1 sMatAvMult_DataLocal – Purely local matrix-vector multiply

The sparse matrix-vector multiplication routine `sMatAvMult_DataLocal_()` operates on the assumption of total data locality, which is equivalent to the following two conditions:

1. The input `AttrVect xAV` contains all the values referenced by the local column indices stored in the input `SparsMatrix` argument `sMat`; and
2. The output `AttrVect yAV` contains all the values referenced by the local row indices stored in the input `SparsMatrix` argument `sMat`.

By default, the multiplication occurs for each of the common `REAL` attributes shared by `xAV` and `yAV`. This routine is capable of cross-indexing the attributes and performing the necessary multiplications. If the optional argument `rList` is present, only the attributes listed will be multiplied. If the attributes have different names in `yAV`, the optional `TrList` argument can be used to provide the translation.
If the optional argument `Vector` is present and true, the vector architecture-friendly portions of this routine will be invoked. It will also cause the vector parts of
sMat to be initialized if they have not been already.

INTERFACE:

```
subroutine sMatAvMult_DataLocal_(xAV, sMat, yAV, Vector, rList, TrList)
```

USES:

```
use m_realkinds, only : FP
use m_stdio,     only : stderr
use m_die,       only : MP_perr_die, die, warn

use m_List, only : List_identical => identical
use m_List, only : List_nitem => nitem
use m_List, only : GetIndices => get_indices

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_zero => zero
use m_AttrVect, only : AttrVect_indexRA => indexRA
use m_AttrVect, only : SharedAttrIndexList

use m_SparseMatrix, only : SparseMatrix
use m_SparseMatrix, only : SparseMatrix_lsize => lsize
use m_SparseMatrix, only : SparseMatrix_indexIA => indexIA
use m_SparseMatrix, only : SparseMatrix_indexRA => indexRA
use m_SparseMatrix, only : SparseMatrix_vecinit => vecinit

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),     intent(in)    :: xAV
logical,optional,   intent(in)    :: Vector
character(len=*),optional, intent(in)    :: rList
character(len=*),optional, intent(in)    :: TrList
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout)    :: sMat
type(AttrVect),     intent(inout) :: yAV
```

REVISION HISTORY:

```
15Jan01 - J.W. Larson <larson@mcs.anl.gov> - API specification.
10Feb01 - J.W. Larson <larson@mcs.anl.gov> - Prototype code.
24Apr01 - J.W. Larson <larson@mcs.anl.gov> - Modified to accomodate
          changes to the SparseMatrix datatype.
25Apr01 - J.W. Larson <larson@mcs.anl.gov> - Reversed loop order
          for cache-friendliness
17May01 - R. Jacob <jacob@mcs.anl.gov> - Zero the output
```

```
          attribute vector
10Oct01 - J. Larson <larson@mcs.anl.gov> - Added optional LOGICAL
          input argument InterpInts to make application of the
          multiply to INTEGER attributes optional
15Oct01 - J. Larson <larson@mcs.anl.gov> - Added feature to
          detect when attribute lists are identical, and cross-
          indexing of attributes is not needed.
29Nov01 - E.T. Ong <eong@mcs.anl.gov> - Removed MP_PERR_DIE if
          there are zero elements in sMat. This allows for
          decompositions where a process may own zero points.
29Oct03 - R. Jacob <jacob@mcs.anl.gov> - add Vector argument to
          optionally use the vector-friendly version provided by
          Fujitsu
21Nov06 - R. Jacob <jacob@mcs.anl.gov> - Allow attributes to be
          to be multiplied to be specified with rList and TrList.
```

11.1.2 sMatAvMult_SMPlus_ - Parallel Multiply Using SparseMatrixPlus

This routine performs distributed parallel sparse matrix-vector multiplication $\mathbf{y} = \mathbf{Mx}$, where $\mathbf{y}$ and $\mathbf{x}$ are represented by the `AttrVect` arguments `yAV` and `xAV`, respectively. The matrix $\mathbf{M}$ is stored in the input `SparseMatrixPlus` argument `sMatPlus`, which also contains all the information needed to coordinate the communications required to gather intermediate vectors used in the multiplication process, and to reduce partial sums as needed. By default, the multiplication occurs for each of the common `REAL` attributes shared by `xAV` and `yAV`. This routine is capable of cross-indexing the attributes and performing the necessary multiplications.
If the optional argument `rList` is present, only the attributes listed will be multiplied. If the attributes have different names in `yAV`, the optional `TrList` argument can be used to provide the translation.
If the optional argument `Vector` is present and true, the vector architecture-friendly portions of this routine will be invoked. It will also cause the vector parts of `sMatPlus` to be initialized if they have not been already.

INTERFACE:

```
subroutine sMatAvMult_SMPlus_(xAV, sMatPlus, yAV, Vector, rList, TrList)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_String, only : String
use m_String, only : String_ToChar => ToChar

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_clean => clean
use m_AttrVect, only : AttrVect_Rcopy => Rcopy
use m_AttrVect, only : AttrVect_zero  => zero

use m_Rearranger, only : Rearranger
use m_Rearranger, only : Rearrange

use m_SparseMatrixPlus, only : SparseMatrixPlus
use m_SparseMatrixPlus, only : Xonly
use m_SparseMatrixPlus, only : Yonly
```

```
use m_SparseMatrixPlus, only : XandY

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),          intent(in)    :: xAV
logical, optional,       intent(in)    :: Vector
character(len=*),optional, intent(in)    :: rList
character(len=*),optional, intent(in)    :: TrList
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),          intent(inout) :: yAV
type(SparseMatrixPlus), intent(inout) :: sMatPlus
```

SEE ALSO:

```
The MCT module m_AttrVect for more information about the AttrVect type.
The MCT module m_SparseMatrixPlus for more information about the
SparseMatrixPlus type.
```

REVISION HISTORY:

```
26Sep02 - J.W. Larson <larson@mcs.anl.gov> - API specification and
          implementation.
29Oct03 - R. Jacob <jacob@mcs.anl.gov> - add vector argument to all
          calls to Rearrange and DataLocal_.  Add optional input
          argument to change value (assumed false)
22Nov06 - R. Jacob <jacob@mcs.anl.gov> - add rList,TrList arguments
10Jan08 - T. Craig <tcraig@ucar.edu> - zero out intermediate aVs before
          they are used
```

12 Spatial Integration and Averaging

12.1 Module m_SpatialIntegral - Spatial Integrals and Averages using a GeneralGrid (Source File: m_SpatialIntegral.F90)

This module provides spatial integration and averaging services for the MCT. For a field Φ sampled at a point $\mathbf{x}$ in some multidimensional domain Ω, the integral I of $\Phi(\mathbf{x})$ is

$$I = \int_{\Omega} \Phi(\mathbf{x}) d\Omega.$$

The spatial average A of $\Phi(\mathbf{x})$ over Ω is

$$A = \frac{\int_{\Omega} \Phi(\mathbf{x}) d\Omega}{\int_{\Omega} d\Omega}.$$

Since the `AttrVect` represents a discretized field, the integrals above are implemented as:

$$I = \sum_{i=1}^{N} \Phi_i \Delta\Omega_i$$

and

$$A = \frac{\sum_{i=1}^{N} \Phi_i \Delta\Omega_i}{\sum_{i=1}^{N} \Delta\Omega_i},$$

where N is the number of physical locations, Φ_i is the value of the field Φ at location i, and $\Delta\Omega_i$ is the spatial weight (lenghth element, cross-sectional area element, volume element, *et cetera*) at location i.
MCT extends the concept of integrals and area/volume averages to include *masked* integrals and averages. MCT recognizes both *integer* and *real* masks. An integer mask M is a vector of integers (one corresponding to each physical location) with each element having value either zero or one. Integer masks are used to include/exclude data from averages or integrals. For example, if one were to compute globally averaged cloud amount over land (but not ocean nor sea-ice), one would assign a 1 to each location on the land and a 0 to each non-land location. A *real* mask F is a vector of real numbers (one corresponding to each physical location) with each element having value within the closed interval $[0, 1]$. .Real masks are used to represent fractional area/volume coverage at a location by a given component model. For example, if one wishes to compute area averages over sea-ice, one must include the ice fraction present at each point. Masked Integrals and averages are represented in the MCT by:

$$I = \sum_{i=1}^{N} \prod_{j=1}^{J} M_i \prod_{k=1}^{K} F_i \Phi_i \Delta\Omega_i$$

and

$$A = \frac{\sum_{i=1}^{N} \left(\prod_{j=1}^{J} M_i \right) \left(\prod_{k=1}^{K} F_i \right) \Phi_i \Delta\Omega_i}{\sum_{i=1}^{N} \left(\prod_{j=1}^{J} M_i \right) \left(\prod_{k=1}^{K} F_i \right) \Delta\Omega_i},$$

where J is the number of integer masks and K is the number of real masks.
All of the routines in this module assume field data is stored in an attribute vector (`AttrVect`), and the integration/averaging is performed only on the `REAL` attributes. Physical coordinate grid and mask information is assumed to be stored as attributes in either a `GeneralGrid`, or pre-combined into a single integer mask and a single real mask.

INTERFACE:

```
module m_SpatialIntegral
```

```
  implicit none

  private ! except
```

PUBLIC MEMBER FUNCTIONS:

```
  public :: SpatialIntegral        ! Spatial Integral
  public :: SpatialAverage         ! Spatial Area Average

  public :: MaskedSpatialIntegral  ! Masked Spatial Integral
  public :: MaskedSpatialAverage   ! MaskedSpatial Area Average

  public :: PairedSpatialIntegrals ! A Pair of Spatial
                                   ! Integrals

  public :: PairedSpatialAverages  ! A Pair of Spatial
                                   ! Area Averages

  public :: PairedMaskedSpatialIntegrals ! A Pair of Masked
                                         ! Spatial Integrals

  public :: PairedMaskedSpatialAverages ! A Pair of Masked
                                        ! Spatial Area Averages

  interface SpatialIntegral ; module procedure &
SpatialIntegralRAttrGG_
  end interface
  interface SpatialAverage ; module procedure &
SpatialAverageRAttrGG_
  end interface
  interface MaskedSpatialIntegral ; module procedure &
MaskedSpatialIntegralRAttrGG_
  end interface
  interface MaskedSpatialAverage ; module procedure &
MaskedSpatialAverageRAttrGG_
  end interface
  interface PairedSpatialIntegrals ; module procedure &
 PairedSpatialIntegralRAttrGG_
  end interface
  interface PairedSpatialAverages ; module procedure &
 PairedSpatialAverageRAttrGG_
  end interface
  interface PairedMaskedSpatialIntegrals ; module procedure &
 PairedMaskedIntegralRAttrGG_
  end interface
  interface PairedMaskedSpatialAverages ; module procedure &
 PairedMaskedAverageRAttrGG_
  end interface
```

REVISION HISTORY:

```
25Oct01 - J.W. Larson <larson@mcs.anl.gov> - Initial version
      9May02 - J.W. Larson <larson@mcs.anl.gov> - Massive Refactoring.
  10-14Jun02 - J.W. Larson <larson@mcs.anl.gov> - Added Masked methods.
  17-18Jun02 - J.W. Larson <larson@mcs.anl.gov> - Added Paired/Masked
               methods.
     18Jun02 - J.W. Larson <larson@mcs.anl.gov> - Renamed module from
               m_GlobalIntegral to m_SpatialIntegral.
```

```
15Jan03 - E.T. Ong <eong@mcs.anl.gov> - Initialized real-only
          AttrVects using nullfied integer lists. This circuitous
          hack was required because the compaq compiler does not
          compile the function AttrVectExportListToChar.
```

12.1.1 SpatialIntegralRAttrGG_ - Compute spatial integral.

This routine computes spatial integrals of the `REAL` attributes of the `REAL` attributes of the input `AttrVect` argument `inAv`. `SpatialIntegralRAttrGG_()` takes the input `AttrVect` argument `inAv` and computes the spatial integral using weights stored in the `GeneralGrid` argument `GGrid` and identified by the `CHARACTER` tag `WeightTag`. The integral of each `REAL` attribute is returned in the output `AttrVect` argument `outAv`. If `SpatialIntegralRAttrGG_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`, then the weights are also summed and stored in `outAv` (and can be referenced with the attribute tag defined by the argument`WeightTag`. If `SpatialIntegralRAttrGG_()` is invoked with the optional `INTEGER` argument `comm` (a Fortran MPI communicator handle), the summation operations for the integral are completed on the local process, then reduced across the communicator, with all processes receiving the result.
N.B.: The local lengths of the `AttrVect` argument `inAv` and the `GeneralGrid GGrid` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv` and the point weights stored in `GGrid`.
N.B.: If `SpatialIntegralRAttrGG_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`, then the value of `WeightTag` must not conflict with any of the `REAL` attribute tags in `inAv`.
N.B.: The output `AttrVect` argument `outAv` is an allocated data structure. The user must deallocate it using the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine SpatialIntegralRAttrGG_(inAv, outAv, GGrid, WeightTag, &
                                   SumWeights, comm)
 ! USES:

     use m_stdio
     use m_die
     use m_mpif90

     use m_realkinds, only : FP

     use m_AttrVect, only : AttrVect
     use m_AttrVect, only : AttrVect_lsize => lsize

     use m_GeneralGrid, only : GeneralGrid
     use m_GeneralGrid, only : GeneralGrid_lsize => lsize
     use m_GeneralGrid, only : GeneralGrid_indexRA => indexRA
     use m_GeneralGrid, only : GeneralGrid_exportRAttr => exportRAttr

     use m_SpatialIntegralV, only: SpatialIntegralV

     implicit none
```

INPUT PARAMETERS:

```
     type(AttrVect),    intent(IN) :: inAv
     type(GeneralGrid), intent(IN) :: GGrid
     character(len=*),  intent(IN) :: WeightTag
     logical, optional, intent(IN) :: SumWeights
```

```
integer, optional, intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),    intent(OUT) :: outAv
```

REVISION HISTORY:

```
06Feb02 - J.W. Larson <larson@mcs.anl.gov> - initial version
09May02 - J.W. Larson <larson@mcs.anl.gov> - Refactored and
          renamed SpatialIntegralRAttrGG_().
  07Jun02 - J.W. Larson <larson@mcs.anl.gov> - Bug fix and further
          refactoring.
```

12.1.2 SpatialAverageRAttrGG_ - Compute spatial average.

This routine computes spatial averages of the `REAL` attributes of the input `AttrVect` argument `inAv`. `SpatialAverageRAttrGG_()` takes the input `AttrVect` argument `inAv` and computes the spatial average using weights stored in the `GeneralGrid` argument `GGrid` and identified by the `CHARACTER` tag `WeightTag`. The average of each `REAL` attribute is returned in the output `AttrVect` argument `outAv`. If `SpatialAverageRAttrGG_()` is invoked with the optional `INTEGER` argument `comm` (a Fortran MPI communicator handle), the summation operations for the average are completed on the local process, then reduced across the communicator, with all processes receiving the result.
N.B.: The local lengths of the `AttrVect` argument `inAv` and the `GeneralGrid GGrid` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv` and the point weights stored in `GGrid`.
N.B.: The output `AttrVect` argument `outAv` is an allocated data structure. The user must deallocate it using the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine SpatialAverageRAttrGG_(inAv, outAv, GGrid, WeightTag, comm)

 ! USES:

    use m_realkinds, only : FP

    use m_stdio
    use m_die
    use m_mpif90

    use m_AttrVect, only : AttrVect
    use m_AttrVect, only : AttrVect_init => init
    use m_AttrVect, only : AttrVect_zero => zero
    use m_AttrVect, only : AttrVect_clean => clean
    use m_AttrVect, only : AttrVect_nRAttr => nRAttr
    use m_AttrVect, only : AttrVect_indexRA => indexRA

    use m_GeneralGrid, only : GeneralGrid

    use m_List, only : List
    use m_List, only : List_nullify => nullify

    implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),    intent(IN) :: inAv
type(GeneralGrid), intent(IN) :: GGrid
character(len=*),  intent(IN) :: WeightTag
integer, optional, intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),    intent(OUT) :: outAv
```

REVISION HISTORY:

```
08Feb02 - J.W. Larson <larson@mcs.anl.gov> - initial version
08May02 - J.W. Larson <larson@mcs.anl.gov> - minor modifications:
               1) renamed the routine to GlobalAverageRAttrGG_
               2) changed calls to reflect new routine name
                  GlobalIntegralRAttrGG_().
     18Jun02 - J.W. Larson <larson@mcs.anl.gov> - Renamed routine to
               SpatialAverageRAttrGG_().
```

12.1.3 MaskedSpatialIntegralRAttrGG_ - Masked spatial integral.

This routine computes masked spatial integrals of the `REAL` attributes of the input `AttrVect` argument `inAv`, returning the masked integrals in the output `AttrVect` `outAv`. All of the masking data are assumed stored in the input `GeneralGrid` argument `GGrid`. If integer masks are to be used, their integer attribute names in `GGrid` are named as a colon-delimited list in the optional `CHARACTER` input argument `iMaskTags`. Real masks (if desired) are referenced by their real attribute names in `GGrid` are named as a colon-delimited list in the optional `CHARACTER` input argument `rMaskTags`. The user specifies a choice of mask combination method with the input `LOGICAL` argument `UseFastMethod`. If `UseFastMethod` = `.FALSE.` this routine checks each mask entry to ensure that the integer masks contain only ones and zeroes, and that entries in the real masks are all in the closed interval $[0, 1]$. If `UseFastMethod` = `.TRUE.`, this routine performs direct products of the masks, assuming that the user has validated them in advance. The optional `LOGICAL` input argument `SumWeights` determines whether the masked sum of the spatial weights is computed and returned in `outAv` with the real attribute name supplied in the optional `CHARACTER` input argument `WeightSumTag`. This integral can either be a local (i.e. a global memory space operation), or a global distributed integral. The latter is the case if the optional input `INTEGER` argument `comm` is supplied (which corresponds to a Fortran MPI communicatior handle).
N.B.: The local lengths of the `AttrVect` argument `inAv` and the input `GeneralGrid` `GGrid` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv` and the point weights stored in `GGrid`.
N.B.: If `SpatialIntegralRAttrV_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`. In this case, the none of `REAL` attribute tags in `inAv` may be named the same as the string contained in `WeightSumTag`, which is an attribute name reserved for the sum of the weights in the output `AttrVect` `outAv`.
N.B.: The output `AttrVect` argument `outAv` is an allocated data structure. The user must deallocate it using the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine MaskedSpatialIntegralRAttrGG_(inAv, outAv, GGrid, SpatialWeightTag, &
                                         iMaskTags, rMaskTags, UseFastMethod,  &
                                         SumWeights, WeightSumTag, comm)

 ! USES:

     use m_stdio
```

```
use m_die
use m_mpif90

use m_realkinds, only : FP

use m_String, only : String
use m_String, only : String_toChar => toChar
use m_String, only : String_clean => clean

use m_List, only : List
use m_List, only : List_init => init
use m_List, only : List_clean => clean
use m_List, only : List_nitem => nitem
use m_List, only : List_get => get

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_lsize => lsize
use m_GeneralGrid, only : GeneralGrid_indexRA => indexRA
use m_GeneralGrid, only : GeneralGrid_exportIAttr => exportIAttr
use m_GeneralGrid, only : GeneralGrid_exportRAttr => exportRAttr

use m_AttrVectReduce, only : AttrVect_GlobalWeightedSumRAttr => &
                                  GlobalWeightedSumRAttr
use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                  LocalWeightedSumRAttr

use m_SpatialIntegralV, only : MaskedSpatialIntegralV

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),                 intent(IN) :: inAv
type(GeneralGrid),              intent(IN) :: GGrid
character(len=*),               intent(IN) :: SpatialWeightTag
character(len=*),      optional, intent(IN) :: iMaskTags
character(len=*),      optional, intent(IN) :: rMaskTags
logical,                        intent(IN) :: UseFastMethod
logical,               optional, intent(IN) :: SumWeights
character(len=*),      optional, intent(IN) :: WeightSumTag
integer,               optional, intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),                 intent(OUT) :: outAv
```

REVISION HISTORY:

```
11Jun02 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

12.1.4 MaskedSpatialAverageRAttrGG_ - Masked spatial average.

This routine computes masked spatial averages of the `REAL` attributes of the input `AttrVect` argument `inAv`, returning the masked averages in the output `AttrVect outAv`. All of the masking data

are assumed stored in the input `GeneralGrid` argument `GGrid`. If integer masks are to be used, their integer attribute names in `GGrid` are named as a colon-delimited list in the optional `CHARACTER` input argument `iMaskTags`. Real masks (if desired) are referenced by their real attribute names in `GGrid` are named as a colon-delimited list in the optional `CHARACTER` input argument `rMaskTags`. The user specifies a choice of mask combination method with the input `LOGICAL` argument `UseFastMethod`. If `UseFastMethod` = `.FALSE.` this routine checks each mask entry to ensure that the integer masks contain only ones and zeroes, and that entries in the real masks are all in the closed interval $[0, 1]$. If `UseFastMethod` = `.TRUE.`, this routine performs direct products of the masks, assuming that the user has validated them in advance. This averaging can either be a local (equivalent to a global memory space operation), or a global distributed integral. The latter is the case if the optional input `INTEGER` argument `comm` is supplied (which corresponds to a Fortran MPI communicatior handle).
N.B.: The local lengths of the `AttrVect` argument `inAv` and the input `GeneralGrid` `GGrid` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv` and the point weights stored in `GGrid`.
N.B.: The output `AttrVect` argument `outAv` is an allocated data structure. The user must deallocate it using the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine MaskedSpatialAverageRAttrGG_(inAv, outAv, GGrid, SpatialWeightTag, &
                                        iMaskTags, rMaskTags, UseFastMethod,  &
          comm)

 ! USES:

      use m_realkinds, only : FP

      use m_stdio
      use m_die
      use m_mpif90

      use m_AttrVect, only : AttrVect
      use m_AttrVect, only : AttrVect_init => init
      use m_AttrVect, only : AttrVect_zero => zero
      use m_AttrVect, only : AttrVect_clean => clean
      use m_AttrVect, only : AttrVect_lsize => lsize
      use m_AttrVect, only : AttrVect_indexRA => indexRA
      use m_AttrVect, only : AttrVect_nRAttr => nRAttr

      use m_GeneralGrid, only : GeneralGrid
      use m_GeneralGrid, only : GeneralGrid_lsize => lsize
      use m_GeneralGrid, only : GeneralGrid_indexRA => indexRA

      use m_List, only : List
      use m_List, only : List_nullify => nullify

      implicit none
```

INPUT PARAMETERS:

```
      type(AttrVect),                 intent(IN) :: inAv
      type(GeneralGrid),              intent(IN) :: GGrid
      character(len=*),               intent(IN) :: SpatialWeightTag
      character(len=*),      optional, intent(IN) :: iMaskTags
      character(len=*),      optional, intent(IN) :: rMaskTags
      logical,                        intent(IN) :: UseFastMethod
      integer,               optional, intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),                  intent(OUT) :: outAv
```

REVISION HISTORY:

12Jun02 - J.W. Larson <larson@mcs.anl.gov> - initial version

12.1.5 PairedSpatialIntegralRAttrGG_ - Do two spatial integrals at once.

This routine computes spatial integrals of the `REAL` attributes of the `REAL` attributes of the input `AttrVect` arguments `inAv1` and `inAv2`, returning the integrals in the output `AttrVect` arguments `outAv1` and `outAv2`, respectively . The integrals of `inAv1` and `inAv2` are computed using spatial weights stored in the input `GeneralGrid` arguments `GGrid1` and `GGrid2`, respectively. The spatial weights in in `GGrid1` and `GGrid2` are identified by the input `CHARACTER` arguments `WeightTag1` and `WeightTag2`, respectively. If `SpatialIntegralRAttrGG_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`, then the weights are also summed and stored in `outAv1` and `outAv2`, and can be referenced with the attribute tags defined by the arguments `WeightTag1` and `WeightTag2`, respectively. This paired integral is implicitly a distributed operation (the whole motivation for pairing the integrals is to reduce communication latency costs), and the Fortran MPI communicator handle is defined by the input `INTEGER` argument `comm`. The summation is an AllReduce operation, with all processes receiving the global sum.
N.B.: The local lengths of the `AttrVect` argument `inAv1` and the `GeneralGrid` `GGrid1` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv1` and the point weights stored in `GGrid1`. The same relationship must apply between `inAv2` and `GGrid2`.
N.B.: If `SpatialIntegralRAttrGG_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`, then the value of `WeightTag1` must not conflict with any of the `REAL` attribute tags in `inAv1` and the value of `WeightTag2` must not conflict with any of the `REAL` attribute tags in `inAv2`.
N.B.: The output `AttrVect` arguments `outAv1` and `outAv2` are allocated data structures. The user must deallocate them using the routine `AttrVect_clean()` when they are no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine PairedSpatialIntegralRAttrGG_(inAv1, outAv1, GGrid1, WeightTag1, &
                                         inAv2, outAv2, GGrid2, WeightTag2, &
 SumWeights, comm)
 ! USES:

     use m_stdio
     use m_die
     use m_mpif90

     use m_realkinds, only : FP

     use m_AttrVect, only : AttrVect
     use m_AttrVect, only : AttrVect_lsize => lsize
     use m_AttrVect, only : AttrVect_nRAttr => nRAttr

     use m_GeneralGrid, only : GeneralGrid
     use m_GeneralGrid, only : GeneralGrid_lsize => lsize
     use m_GeneralGrid, only : GeneralGrid_indexRA => indexRA
     use m_GeneralGrid, only : GeneralGrid_exportRAttr => exportRAttr

     use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                  LocalWeightedSumRAttr
```

```
use m_SpatialIntegralV, only : PairedSpatialIntegralsV

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),    intent(IN) :: inAv1
type(GeneralGrid), intent(IN) :: GGrid1
character(len=*),  intent(IN) :: WeightTag1
type(AttrVect),    intent(IN) :: inAv2
type(GeneralGrid), intent(IN) :: GGrid2
character(len=*),  intent(IN) :: WeightTag2
logical, optional, intent(IN) :: SumWeights
integer,           intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),    intent(OUT) :: outAv1
type(AttrVect),    intent(OUT) :: outAv2
```

REVISION HISTORY:

```
09May02 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
10Jun02 - J.W. Larson <larson@mcs.anl.gov> - Refactored--now
               built on top of PairedIntegralRAttrV_().
```

12.1.6 PairedSpatialAverageRAttrGG_ - Do two spatial averages at once.

This routine computes spatial averages of the `REAL` attributes of the `REAL` attributes of the input `AttrVect` arguments `inAv1` and `inAv2`, returning the integrals in the output `AttrVect` arguments `outAv1` and `outAv2`, respectively . The integrals of `inAv1` and `inAv2` are computed using spatial weights stored in the input `GeneralGrid` arguments `GGrid1` and `GGrid2`, respectively. The spatial weights in in `GGrid1` and `GGrid2` are identified by the input `CHARACTER` arguments `WeightTag1` and `WeightTag2`, respectively. This paired average is implicitly a distributed operation (the whole motivation for pairing the averages is to reduce communication latency costs), and the Fortran MPI communicator handle is defined by the input `INTEGER` argument `comm`. The summation is an AllReduce operation, with all processes receiving the global sum.
N.B.: The local lengths of the `AttrVect` argument `inAv1` and the `GeneralGrid` `GGrid1` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv1` and the point weights stored in `GGrid1`. The same relationship must apply between `inAv2` and `GGrid2`.
N.B.: The output `AttrVect` arguments `outAv1` and `outAv2` are allocated data structures. The user must deallocate them using the routine `AttrVect_clean()` when they are no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine PairedSpatialAverageRAttrGG_(inAv1, outAv1, GGrid1, WeightTag1, &
                                        inAv2, outAv2, GGrid2, WeightTag2, &
                                        comm)
 ! USES:

    use m_realkinds, only : FP

    use m_stdio
```

```
use m_die
use m_mpif90

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero
use m_AttrVect, only : AttrVect_clean => clean
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nRAttr => nRAttr
use m_AttrVect, only : AttrVect_indexRA => indexRA

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_lsize => lsize
use m_GeneralGrid, only : GeneralGrid_indexRA => indexRA
use m_GeneralGrid, only : GeneralGrid_exportRAttr => exportRAttr

use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                    LocalWeightedSumRAttr

use m_List, only : List
use m_List, only : List_nullify => nullify

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),    intent(IN) :: inAv1
type(GeneralGrid), intent(IN) :: GGrid1
character(len=*),  intent(IN) :: WeightTag1
type(AttrVect),    intent(IN) :: inAv2
type(GeneralGrid), intent(IN) :: GGrid2
character(len=*),  intent(IN) :: WeightTag2
integer,           intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),    intent(OUT) :: outAv1
type(AttrVect),    intent(OUT) :: outAv2
```

REVISION HISTORY:

```
09May02 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
14Jun02 - J.W. Larson <larson@mcs.anl.gov> - Bug fix to reflect
          new interface to PairedSpatialIntegralRAttrGG_().
```

12.1.7 PairedMaskedIntegralRAttrGG_ - Do two masked integrals at once.

This routine computes a pair of masked spatial integrals of the `REAL` attributes of the input `AttrVect` arguments `inAv` and `inAv2`, returning the masked integrals in the output `AttrVect outAv1` and `outAv2`, respectively. All of the spatial weighting and masking data for each set of integrals are assumed stored in the input `GeneralGrid` arguments `GGrid` and `GGrid2`. If integer masks are to be used, their integer attribute names in `GGrid1` and `GGrid2` are named as a colon-delimited lists in the optional `CHARACTER` input arguments `iMaskTags1` and `iMaskTags2`, respectively. Real masks (if desired) are referenced by their real attribute names in `GGrid1` and `GGrid2` are named as colon-delimited lists in the optional `CHARACTER` input arguments `rMaskTags1` and `rMaskTags2`, respectively. The user specifies a choice of mask combination method with the input `LOGICAL`

argument UseFastMethod. If UseFastMethod = .FALSE. this routine checks each mask entry to ensure that the integer masks contain only ones and zeroes, and that entries in the real masks are all in the closed interval $[0, 1]$. If UseFastMethod = .TRUE., this routine performs direct products of the masks, assuming that the user has validated them in advance. The optional LOGICAL input argument SumWeights determines whether the masked sum of the spatial weights is computed and returned in outAv1 and outAv2 with the real attribute names supplied in the CHARACTER input arguments SpatialWeightTag1, and SpatialWeightTag2, respectively. This paired integral is implicitly a distributed operation (the whole motivation for pairing the averages is to reduce communication latency costs), and the Fortran MPI communicator handle is defined by the input INTEGER argument comm. The summation is an AllReduce operation, with all processes receiving the global sum.
N.B.: The local lengths of the AttrVect argument inAv1 and the GeneralGrid GGrid1 must be equal. That is, there must be a one-to-one correspondence between the field point values stored in inAv1 and the point weights stored in GGrid1. The same relationship must apply between inAv2 and GGrid2.
N.B.: If PairedMaskedIntegralRAttrGG_() is invoked with the optional LOGICAL input argument SumWeights set as .TRUE., then the value of SpatialWeightTag1 must not conflict with any of the REAL attribute tags in inAv1 and the value of SpatialWeightTag2 must not conflict with any of the REAL attribute tags in inAv2.
N.B.: The output AttrVect arguments outAv1 and outAv2 are allocated data structures. The user must deallocate them using the routine AttrVect_clean() when they are no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
  subroutine PairedMaskedIntegralRAttrGG_(inAv1, outAv1, GGrid1, &
                                          SpatialWeightTag1, rMaskTags1, &
                                          iMaskTags1, inAv2, outAv2, GGrid2, &
                                          SpatialWeightTag2, rMaskTags2, &
                                          iMaskTags2, UseFastMethod, &
                                          SumWeights, comm)
   ! USES:

       use m_stdio
       use m_die
       use m_mpif90

       use m_realkinds, only : FP

       use m_AttrVect, only : AttrVect
       use m_AttrVect, only : AttrVect_lsize => lsize
       use m_AttrVect, only : AttrVect_nRAttr => nRAttr

       use m_GeneralGrid, only : GeneralGrid
       use m_GeneralGrid, only : GeneralGrid_lsize => lsize
       use m_GeneralGrid, only : GeneralGrid_indexRA => indexRA
       use m_GeneralGrid, only : GeneralGrid_exportRAttr => exportRAttr

       use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                    LocalWeightedSumRAttr

       implicit none
```

INPUT PARAMETERS:

```
       type(AttrVect),              intent(IN) :: inAv1
       type(GeneralGrid),           intent(IN) :: GGrid1
       character(len=*),            intent(IN) :: SpatialWeightTag1
       character(len=*),  optional, intent(IN) :: iMaskTags1
       character(len=*),  optional, intent(IN) :: rMaskTags1
```

```
type(AttrVect),              intent(IN) :: inAv2
type(GeneralGrid),           intent(IN) :: GGrid2
character(len=*),            intent(IN) :: SpatialWeightTag2
character(len=*),  optional, intent(IN) :: iMaskTags2
character(len=*),  optional, intent(IN) :: rMaskTags2
logical,                     intent(IN) :: UseFastMethod
logical,           optional, intent(IN) :: SumWeights
integer,                     intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),    intent(OUT) :: outAv1
type(AttrVect),    intent(OUT) :: outAv2
```

REVISION HISTORY:

```
17Jun02 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
19Jun02 - J.W. Larson <larson@mcs.anl.gov> - Shortened the name
               for compatibility with the Portland Group f90 compiler
```

12.1.8 PairedMaskedAverageRAttrGG_ - Do two masked averages at once.

This routine computes a pair of masked spatial averages of the `REAL` attributes of the input `AttrVect` arguments `inAv` and `inAv2`, returning the masked averagess in the output `AttrVect` `outAv1` and `outAv2`, respectively. All of the spatial weighting and masking data for each set of averages are assumed stored in the input `GeneralGrid` arguments `GGrid` and `GGrid2`. If integer masks are to be used, their integer attribute names in `GGrid1` and `GGrid2` are named as a colon-delimited lists in the optional `CHARACTER` input arguments `iMaskTags1` and `iMaskTags2`, respectively. Real masks (if desired) are referenced by their real attribute names in `GGrid1` and `GGrid2` are named as colon-delimited lists in the optional `CHARACTER` input arguments `rMaskTags1` and `rMaskTags2`, respectively. The user specifies a choice of mask combination method with the input `LOGICAL` argument `UseFastMethod`. If `UseFastMethod` = .`FALSE`. this routine checks each mask entry to ensure that the integer masks contain only ones and zeroes, and that entries in the real masks are all in the closed interval $[0, 1]$. If `UseFastMethod` = .`TRUE`., this routine performs direct products of the masks, assuming that the user has validated them in advance. This paired average is implicitly a distributed operation (the whole motivation for pairing the averages is to reduce communication latency costs), and the Fortran MPI communicator handle is defined by the input `INTEGER` argument `comm`. The summation is an AllReduce operation, with all processes receiving the global sum.
N.B.: The local lengths of the `AttrVect` argument `inAv1` and the `GeneralGrid` `GGrid1` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv1` and the point weights stored in `GGrid1`. The same relationship must apply between `inAv2` and `GGrid2`.
N.B.: The output `AttrVect` arguments `outAv1` and `outAv2` are allocated data structures. The user must deallocate them using the routine `AttrVect_clean()` when they are no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine PairedMaskedAverageRAttrGG_(inAv1, outAv1, GGrid1, &
                                       SpatialWeightTag1, rMaskTags1, &
                                       iMaskTags1, inAv2, outAv2, GGrid2, &
                                       SpatialWeightTag2, rMaskTags2, &
                                       iMaskTags2, UseFastMethod, &
                                       comm)
 ! USES:

     use m_stdio
```

```
use m_die
use m_mpif90

use m_realkinds, only : FP

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero
use m_AttrVect, only : AttrVect_clean => clean
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nRAttr => nRAttr

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_lsize => lsize
use m_GeneralGrid, only : GeneralGrid_indexRA => indexRA
use m_GeneralGrid, only : GeneralGrid_exportRAttr => exportRAttr

use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                    LocalWeightedSumRAttr

use m_List, only : List
use m_List, only : List_nullify => nullify

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),              intent(IN) :: inAv1
type(GeneralGrid),           intent(IN) :: GGrid1
character(len=*),            intent(IN) :: SpatialWeightTag1
character(len=*),  optional, intent(IN) :: iMaskTags1
character(len=*),  optional, intent(IN) :: rMaskTags1
type(AttrVect),              intent(IN) :: inAv2
type(GeneralGrid),           intent(IN) :: GGrid2
character(len=*),            intent(IN) :: SpatialWeightTag2
character(len=*),  optional, intent(IN) :: iMaskTags2
character(len=*),  optional, intent(IN) :: rMaskTags2
logical,                     intent(IN) :: UseFastMethod
integer,                     intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),    intent(OUT) :: outAv1
type(AttrVect),    intent(OUT) :: outAv2
```

REVISION HISTORY:

```
17Jun02 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
19Jun02 - J.W. Larson <larson@mcs.anl.gov> - Shortened the name
               for compatibility with the Portland Group f90 compiler
25Jul02 - J.W. Larson E.T. Ong - Bug fix.  This routine was
               previously doing integrals rather than area averages.
```

12.2 Module m_SpatialIntegralV - Spatial Integrals and Averages using vectors of weights (Source File: m_SpatialIntegralV.F90)

This module provides spatial integration and averaging services for the MCT similar to those in `m_SpatialIntegral` except the weights are provided by an input vector instead of through a `GeneralGrid`. See the description for `m_SpatialIntegral` for more information
Paired masked spatial integrals and averages have not yet been implemented in vector form.

INTERFACE:

```
module m_SpatialIntegralV

      implicit none

      private ! except
```

PUBLIC MEMBER FUNCTIONS:

```
      public :: SpatialIntegralV        ! Spatial Integral
      public :: SpatialAverageV         ! Spatial Area Average

      public :: MaskedSpatialIntegralV  ! Masked Spatial Integral
      public :: MaskedSpatialAverageV   ! MaskedSpatial Area Average

      public :: PairedSpatialIntegralsV ! A Pair of Spatial
                                        ! Integrals

      public :: PairedSpatialAveragesV  ! A Pair of Spatial
                                        ! Area Averages

      interface SpatialIntegralV ; module procedure &
    SpatialIntegralRAttrVSP_, &
    SpatialIntegralRAttrVDP_
      end interface
      interface SpatialAverageV ; module procedure &
    SpatialAverageRAttrVSP_, &
    SpatialAverageRAttrVDP_
      end interface
      interface MaskedSpatialIntegralV ; module procedure &
    MaskedSpatialIntegralRAttrVSP_, &
    MaskedSpatialIntegralRAttrVDP_
      end interface
      interface MaskedSpatialAverageV ; module procedure &
    MaskedSpatialAverageRAttrVSP_, &
    MaskedSpatialAverageRAttrVDP_
      end interface
      interface PairedSpatialIntegralsV ; module procedure &
     PairedSpatialIntegralRAttrVSP_, &
     PairedSpatialIntegralRAttrVDP_
      end interface
      interface PairedSpatialAveragesV ; module procedure &
     PairedSpatialAverageRAttrVSP_, &
     PairedSpatialAverageRAttrVDP_
      end interface
```

REVISION HISTORY:

```
    4Jan04 - R.Jacob <jacob@mcs.anl.gov> - move Vector versions of routines
                  from m_SpatialIntegral to this file.
```

12.2.1 SpatialIntegralRAttrVSP_ - Compute spatial integral.

This routine computes spatial integrals of the REAL attributes of the REAL attributes of the input AttrVect argument inAv. SpatialIntegralRAttrV_() takes the input AttrVect argument inAv and computes the spatial integral using weights stored in the input REAL array argument Weights. The integral of each REAL attribute is returned in the output AttrVect argument outAv. If SpatialIntegralRAttrV_() is invoked with the optional LOGICAL input argument SumWeights set as .TRUE., then the weights are also summed and stored in outAv (and can be referenced with the attribute name WeightTag. If SpatialIntegralRAttrV_() is invoked with the optional INTEGER argument comm (a Fortran MPI communicator handle), the summation operations for the integral are completed on the local process, then reduced across the communicator, with all processes receiving the result.
N.B.: The local lengths of the AttrVect argument inAv and the input array Weights must be equal. That is, there must be a one-to-one correspondence between the field point values stored in inAv and the point weights stored in Weights.
N.B.: If SpatialIntegralRAttrV_() is invoked with the optional LOGICAL input argument SumWeights set as .TRUE.. In this case, the none of REAL attribute tags in inAv may be named the same as the string contained in WeightTag, which is an attribute name reserved for the sum of the weights in the output AttrVect outAv.
N.B.: The output AttrVect argument outAv is an allocated data structure. The user must deallocate it using the routine AttrVect_clean() when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine SpatialIntegralRAttrVSP_(inAv, outAv, Weights, SumWeights, &
                                    WeightTag, comm)

 ! USES:

     use m_stdio
     use m_die
     use m_mpif90
     use m_realkinds, only : SP

     use m_AttrVect, only : AttrVect
     use m_AttrVect, only : AttrVect_lsize => lsize

     use m_AttrVectReduce, only : AttrVect_GlobalWeightedSumRAttr => &
                                          GlobalWeightedSumRAttr
     use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                          LocalWeightedSumRAttr

     implicit none
```

INPUT PARAMETERS:

```
     type(AttrVect),               intent(IN) :: inAv
     real(SP), dimension(:),       pointer    :: Weights
     logical,            optional, intent(IN) :: SumWeights
     character(len=*),   optional, intent(IN) :: WeightTag
     integer,            optional, intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
     type(AttrVect),    intent(OUT) :: outAv
```

REVISION HISTORY:

07Jun02 - J.W. Larson <larson@mcs.anl.gov> - initial version

12.2.2 SpatialAverageRAttrVSP_ - Compute spatial average.

This routine computes spatial averages of the REAL attributes of the input AttrVect argument inAv. SpatialAverageRAttrV_() takes the input AttrVect argument inAv and computes the spatial average using weights stored in the REAL array Weights. The average of each REAL attribute is returned in the output AttrVect argument outAv. If SpatialAverageRAttrV_() is invoked with the optional INTEGER argument comm (a Fortran MPI communicator handle), the summation operations for the average are completed on the local process, then reduced across the communicator, with all processes receiving the result.
N.B.: The local lengths of the AttrVect argument inAv and the input array Weights must be equal. That is, there must be a one-to-one correspondence between the field point values stored in inAv and the point weights stored in Weights.
N.B.: The output AttrVect argument outAv is an allocated data structure. The user must deallocate it using the routine AttrVect_clean() when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine SpatialAverageRAttrVSP_(inAv, outAv, Weights, comm)

 ! USES:

     use m_stdio
     use m_die
     use m_mpif90
     use m_realkinds, only : SP, FP

     use m_AttrVect, only : AttrVect
     use m_AttrVect, only : AttrVect_init => init
     use m_AttrVect, only : AttrVect_zero => zero
     use m_AttrVect, only : AttrVect_clean => clean
     use m_AttrVect, only : AttrVect_nRAttr => nRAttr
     use m_AttrVect, only : AttrVect_indexRA => indexRA

     use m_List, only : List
     use m_List, only : List_nullify => nullify

     implicit none
```

INPUT PARAMETERS:

```
     type(AttrVect),     intent(IN)  :: inAv
     real(SP), dimension(:), pointer :: Weights
     integer, optional,  intent(IN)  :: comm
```

OUTPUT PARAMETERS:

```
     type(AttrVect),     intent(OUT) :: outAv
```

REVISION HISTORY:

10Jun02 - J.W. Larson <larson@mcs.anl.gov> - initial version

12.2.3 MaskedSpatialIntegralRAttrVSP_ - Masked spatial integral.

This routine computes masked spatial integrals of the REAL attributes of the input AttrVect argument inAv, returning the masked integrals in the output AttrVect argument outAv. The masked integral is computed using weights stored in the input REAL array argument SpatialWeights. Integer masking (if desired) is provided in the optional input INTEGER array iMask, and real masking (if desired) is provided in the optional input REAL array rMask. If SpatialIntegralRAttrV_() is invoked with the optional LOGICAL input argument SumWeights set as .TRUE., then the weights are also summed and stored in outAv (and can be referenced with the attribute name defined by the optional input CHARACTER argument WeightSumTag. If SpatialIntegralRAttrV_() is invoked with the optional INTEGER argument comm (a Fortran MPI communicator handle), the summation operations for the integral are completed on the local process, then reduced across the communicator, with all processes receiving the result. Otherwise, the integral is assumed to be local (or equivalent to a global address space).
N.B.: The local lengths of the AttrVect argument inAv and the input array Weights must be equal. That is, there must be a one-to-one correspondence between the field point values stored in inAv and the point weights stored in SpatialWeights.
N.B.: If SpatialIntegralRAttrV_() is invoked with the optional LOGICAL input argument SumWeights set as .TRUE.. In this case, the none of REAL attribute tags in inAv may be named the same as the string contained in WeightSumTag, which is an attribute name reserved for the sum of the weights in the output AttrVect outAv.
N.B.: The output AttrVect argument outAv is an allocated data structure. The user must deallocate it using the routine AttrVect_clean() when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine MaskedSpatialIntegralRAttrVSP_(inAv, outAv, SpatialWeights, iMask, &
                                          rMask, UseFastMethod, SumWeights, &
                                          WeightSumTag, comm)

 ! USES:

     use m_stdio
     use m_die
     use m_mpif90
     use m_realkinds, only : SP, FP

     use m_AttrVect, only : AttrVect
     use m_AttrVect, only : AttrVect_lsize => lsize

     use m_AttrVectReduce, only : AttrVect_GlobalWeightedSumRAttr => &
                                  GlobalWeightedSumRAttr
     use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                  LocalWeightedSumRAttr
     implicit none
```

INPUT PARAMETERS:

```
     type(AttrVect),                  intent(IN) :: inAv
     real(SP),dimension(:),           pointer    :: SpatialWeights
     integer, dimension(:), optional, pointer    :: iMask
     real(SP),dimension(:), optional, pointer    :: rMask
     logical,                         intent(IN) :: UseFastMethod
     logical,               optional, intent(IN) :: SumWeights
     character(len=*),      optional, intent(IN) :: WeightSumTag
     integer,               optional, intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),                  intent(OUT) :: outAv
```

REVISION HISTORY:

```
10Jun02 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

12.2.4 MaskedSpatialAverageRAttrVSP_ - Masked spatial average.

[NEEDS **LOTS** of work...] This routine computes spatial integrals of the `REAL` attributes of the `REAL` attributes of the input `AttrVect` argument `inAv`. `SpatialIntegralRAttrV_()` takes the input `AttrVect` argument `inAv` and computes the spatial integral using weights stored in the input `REAL` array argument `Weights`. The integral of each `REAL` attribute is returned in the output `AttrVect` argument `outAv`. If `SpatialIntegralRAttrV_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`, then the weights are also summed and stored in `outAv` (and can be referenced with the attribute name `WeightTag`. If `SpatialIntegralRAttrV_()` is invoked with the optional `INTEGER` argument `comm` (a Fortran MPI communicator handle), the summation operations for the integral are completed on the local process, then reduced across the communicator, with all processes receiving the result.
N.B.: The local lengths of the `AttrVect` argument `inAv` and the input array `Weights` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv` and the point weights stored in `Weights`.
N.B.: If `SpatialIntegralRAttrV_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`. In this case, the none of `REAL` attribute tags in `inAv` may be named the same as the string contained in `WeightTag`, which is an attribute name reserved for the sum of the weights in the output `AttrVect outAv`.
N.B.: The output `AttrVect` argument `outAv` is an allocated data structure. The user must deallocate it using the routine `AttrVect_clean()` when it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine MaskedSpatialAverageRAttrVSP_(inAv, outAv, SpatialWeights, iMask, &
                                         rMask, UseFastMethod, comm)

 ! USES:

     use m_stdio
     use m_die
     use m_mpif90
     use m_realkinds, only : SP, FP

     use m_AttrVect, only : AttrVect
     use m_AttrVect, only : AttrVect_init => init
     use m_AttrVect, only : AttrVect_zero => zero
     use m_AttrVect, only : AttrVect_clean => clean
     use m_AttrVect, only : AttrVect_lsize => lsize
     use m_AttrVect, only : AttrVect_nRAttr => nRAttr
     use m_AttrVect, only : AttrVect_indexRA => indexRA

     use m_List, only : List
     use m_List, only : List_nullify => nullify

     implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),                  intent(IN) :: inAv
```

```
real(SP),  dimension(:),          pointer    :: SpatialWeights
integer, dimension(:), optional, pointer    :: iMask
real(SP),dimension(:), optional, pointer    :: rMask
logical,                          intent(IN) :: UseFastMethod
integer,               optional, intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),                   intent(OUT) :: outAv
```

REVISION HISTORY:

11Jun02 - J.W. Larson <larson@mcs.anl.gov> - initial version

12.2.5 PairedSpatialIntegralRAttrVSP_ - Do two spatial integrals at once.

This routine computes spatial integrals of the `REAL` attributes of the `REAL` attributes of the input `AttrVect` arguments `inAv1` and `inAv2`, returning the integrals in the output `AttrVect` arguments `outAv1` and `outAv2`, respectively . The integrals of `inAv1` and `inAv2` are computed using spatial weights stored in the input `REAL` array arguments `Weights1` and `Weights2`, respectively. If `SpatialIntegralRAttrV_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`, then the weights are also summed and stored in `outAv1` and `outAv2`, and can be referenced with the attribute tags defined by the arguments `WeightName1` and `WeightName2`, respectively. This paired integral is implicitly a distributed operation (the whole motivation for pairing the integrals is to reduce communication latency costs), and the Fortran MPI communicator handle is defined by the input `INTEGER` argument `comm`. The summation is an AllReduce operation, with all processes receiving the global sum.
N.B.: The local lengths of the `AttrVect` argument `inAv1` and the input `REAL` array `Weights1` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv1` and the point weights stored in `Weights`. The same relationship must apply between `inAv2` and `Weights2`.
N.B.: If `SpatialIntegralRAttrV_()` is invoked with the optional `LOGICAL` input argument `SumWeights` set as `.TRUE.`, then the value of `WeightName1` must not conflict with any of the `REAL` attribute tags in `inAv1` and the value of `WeightName2` must not conflict with any of the `REAL` attribute tags in `inAv2`.
N.B.: The output `AttrVect` arguments `outAv1` and `outAv2` are allocated data structures. The user must deallocate them using the routine `AttrVect_clean()` when they are no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine PairedSpatialIntegralRAttrVSP_(inAv1, outAv1, Weights1, WeightName1, &
                                          inAv2, outAv2, Weights2, WeightName2, &
                                          SumWeights, comm)
 ! USES:

    use m_stdio
    use m_die
    use m_mpif90
    use m_realkinds, only : SP, FP

    use m_AttrVect, only : AttrVect
    use m_AttrVect, only : AttrVect_lsize => lsize
    use m_AttrVect, only : AttrVect_nRAttr => nRAttr

    use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                    LocalWeightedSumRAttr
```

```
implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),     intent(IN)  :: inAv1
real(SP),dimension(:),pointer   :: Weights1
character(len=*),   intent(IN)  :: WeightName1
type(AttrVect),     intent(IN)  :: inAv2
real(SP),dimension(:),pointer   :: Weights2
character(len=*),   intent(IN)  :: WeightName2
logical, optional,  intent(IN)  :: SumWeights
integer,            intent(IN)  :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),     intent(OUT) :: outAv1
type(AttrVect),     intent(OUT) :: outAv2
```

REVISION HISTORY:

10Jun02 - J.W. Larson <larson@mcs.anl.gov> - Initial version.

12.2.6 PairedSpatialAverageRAttrVSP_ - Do two spatial averages at once.

This routine computes spatial averages of the `REAL` attributes of the `REAL` attributes of the input `AttrVect` arguments `inAv1` and `inAv2`, returning the integrals in the output `AttrVect` arguments `outAv1` and `outAv2`, respectively . The averages of `inAv1` and `inAv2` are computed using spatial weights stored in the input `REAL` array arguments `Weights1` and `Weights2`, respectively. This paired average is implicitly a distributed operation (the whole motivation for pairing the integrals is to reduce communication latency costs), and the Fortran MPI communicator handle is defined by the input `INTEGER` argument `comm`. The summation is an AllReduce operation, with all processes receiving the global sum.
N.B.: The local lengths of the `AttrVect` argument `inAv1` and the array `Weights` must be equal. That is, there must be a one-to-one correspondence between the field point values stored in `inAv1` and the spatial weights stored in `Weights`
N.B.: The output `AttrVect` arguments `outAv1` and `outAv2` are allocated data structures. The user must deallocate them using the routine `AttrVect_clean()` when they are no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine PairedSpatialAverageRAttrVSP_(inAv1, outAv1, Weights1, inAv2, &
                                         outAv2, Weights2, comm)
 ! USES:

    use m_stdio
    use m_die
    use m_mpif90
    use m_realkinds, only : SP, FP

    use m_AttrVect, only : AttrVect
    use m_AttrVect, only : AttrVect_init => init
    use m_AttrVect, only : AttrVect_zero => zero
    use m_AttrVect, only : AttrVect_clean => clean
    use m_AttrVect, only : AttrVect_lsize => lsize
```

```
use m_AttrVect, only : AttrVect_nRAttr => nRAttr
use m_AttrVect, only : AttrVect_indexRA => indexRA

use m_AttrVectReduce, only : AttrVect_LocalWeightedSumRAttr => &
                                    LocalWeightedSumRAttr

use m_List, only : List
use m_List, only : List_nullify => nullify

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),     intent(IN) :: inAv1
real(SP),dimension(:),pointer  :: Weights1
type(AttrVect),     intent(IN) :: inAv2
real(SP),dimension(:),pointer  :: Weights2
integer,            intent(IN) :: comm
```

OUTPUT PARAMETERS:

```
type(AttrVect),    intent(OUT) :: outAv1
type(AttrVect),    intent(OUT) :: outAv2
```

REVISION HISTORY:

```
09May02 - J.W. Larson <larson@mcs.anl.gov> - Initial version.
```

13 Merging of Flux and State Data from Multiple Sources

13.1 Module m_Merge - Merge flux and state data from multiple sources. (Source File: m_Merge.F90)

This module supports *merging* of state and flux data from multiple components with overlapping spatial domains for use by another component. For example, let the vectors **a** and **b** be data from Components A and B that have been interpolated onto the physical grid of another component C. We wish to combine the data from A and B to get a vector **c**, which represents the merged data on the grid of component C. This merge process is an element-by-element masked weighted average:

$$c_i = \frac{\prod_{j=1}^{J} M_i^j \prod_{k=1}^{K} F_i^k a_i + \prod_{p=1}^{P} N_i^p \prod_{q=1}^{Q} G_i^q b_i}{\prod_{j=1}^{J} M_i^j \prod_{k=1}^{K} F_i^k + \prod_{p=1}^{P} N_i^p \prod_{q=1}^{Q} G_i^q},$$

Where M_i^j and N_i^p are *integer masks* (which have value either 0 or 1), and F_i^k and G_i^q are *real masks* (which are in the closed interval $[0, 1]$).
Currently, we assume that the integer and real masks are stored in the same `GeneralGrid` datatype. We also assume–and this is of critical importance to the user–that the attributes to be merged are the same for all the inputs and output. If the user violates this assumption, incorrect merges will occur for any attributes that are present in only some (that is not all) of the inputs.
This module supports explicitly the merging data from two, three, and four components. There is also a routine named `MergeInData` that allows the user to construct other merging schemes.

INTERFACE:

```
module m_Merge
```

USES:

```
No other modules used in the declaration section of this module.

implicit none

private   ! except
```

PUBLIC TYPES:

```
None.
```

PUBLIC MEMBER FUNCTIONS:

```
public :: MergeTwo      ! Merge Output from two components
                        ! for use by a third.
public :: MergeThree    ! Merge Output from three components
                        ! for use by a fourth.
public :: MergeFour     ! Merge Output from four components
                        ! for use by a fifth.
public :: MergeInData   ! Merge in data from a single component.

interface MergeTwo ; module procedure &
     MergeTwoGGSP_, &
     MergeTwoGGDP_
end interface
interface MergeThree ; module procedure &
     MergeThreeGGSP_, &
MergeThreeGGDP_
  end interface
```

```
interface MergeFour ; module procedure &
     MergeFourGGSP_, &
     MergeFourGGDP_
end interface
interface MergeInData ; module procedure &
     MergeInDataGGSP_, &
     MergeInDataGGDP_
end interface
```

PUBLIC DATA MEMBERS:

None.

REVISION HISTORY:

19Jun02 - J.W. Larson <larson@mcs.anl.gov> - Initial version.

13.1.1 MergeTwoGGSP_ - Merge Data from Two Sources

This routine merges `REAL` attribute data from two input `AttrVect` arguments `inAv1` and `inAv2` to a third `AttrVect` `outAv`. The attributes to be merged are determined entirely by the real attributes of `outAv`. If `outAv` shares one or more attributes with either of the inputs `inAv1` or `inAv2`, a merge is performed on the individual *intersections* of attributes between the pairs (`outAv`, `inAv1`) and (`outAv`, `inAv1`). Currently, it is assumed that these pairwise intersections are all equal. This assumption is of critical importance to the user. If the user violates this assumption, incorrect merges of attributes that are present in some (but not all) of the inputs will result.
The merge operatrion is a masked weighted element-by-element sum, as outlined in the following example. Let the vectors **a** and **b** be data from Components A and B that have been interpolated onto the physical grid of another component C. We wish to combine the data from A and B to get a vector **c**, which represents the merged data on the grid of component C. The merge relation to obtain the ith element of **c** is

$$c_i = \frac{1}{W_i}\left\{\prod_{j=1}^{J}\kappa_i^j\prod_{k=1}^{K}\alpha_i^k a_i + \prod_{l=1}^{L}\lambda_i^l\prod_{m=1}^{M}\beta_i^m b_i\right\},$$

where

$$W_i = \prod_{j=1}^{J}\kappa_i^j\prod_{k=1}^{K}\alpha_i^k + \prod_{l=1}^{L}\lambda_i^l\prod_{m=1}^{M}\beta_i^m.$$

The quantities κ_i^j and λ_i^l are *integer masks* (which have value either 0 or 1), and α_i^k and β_i^m are *real masks* (which are in the closed interval $[0, 1]$).
The integer and real masks are stored as attributes to the same input `GeneralGrid` argument `GGrid`. The mask attribute names are stored as substrings to the colon-separated strings contained in the input `CHARACTER` arguments `iMaskTags1`, `iMaskTags2`, `rMaskTags1`, and `rMaskTags2`. The `LOGICAL` input argument `CheckMasks` governs how the masks are applied. If `CheckMasks` = .`TRUE`., the entries are checked to ensure they meet the definitions of real and integer masks. If `CheckMasks` = .`TRUE`. then the masks are multiplied together on an element-by-element basis with no validation of their entries (this option results in slightly higher performance).
This routine returns the sume of the masked weights as a diagnostic. This quantity is returned in the output `REAL` array `WeightSum`.
The correspondence between the quantities in the above merge relation and the arguments to this routine are summarized in the table.

Quantity	Stored in Argument	Referenced by Argument
a_i	inAv1	
b_i	inAv2	
c_i	outAv	
$\kappa_i^j, j = 1, \ldots, J$	GGrid	iMaskTags1 (J items)
$\alpha_i^k, k = 1, \ldots, K$	GGrid	rMaskTags1 (K items)
$\lambda_i^l, l = 1, \ldots, L$	GGrid	iMaskTags2 (L items)
$\beta_i^m, m = 1, \ldots, M$	GGrid	rMaskTags2 (M items)
W_i	WeightSum	

INTERFACE:

```
subroutine MergeTwoGGSP_(inAv1, iMaskTags1, rMaskTags1, &
                         inAv2, iMaskTags2, rMaskTags2, &
                         GGrid, CheckMasks, outAv, WeightSum)
```

USES:

```
use m_stdio
use m_die

use m_realkinds, only : SP, FP

use m_List, only : List
use m_List, only : List_allocated => allocated

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nRAttr => nRAttr

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),                intent(IN)    :: inAv1
character(len=*),   optional,  intent(IN)    :: iMaskTags1
character(len=*),   optional,  intent(IN)    :: rMaskTags1
type(AttrVect),                intent(IN)    :: inAv2
character(len=*),   optional,  intent(IN)    :: iMaskTags2
character(len=*),   optional,  intent(IN)    :: rMaskTags2
type(GeneralGrid),             intent(IN)    :: GGrid
logical,                       intent(IN)    :: CheckMasks
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),                intent(INOUT) :: outAv
real(SP),       dimension(:), pointer        :: WeightSum
```

REVISION HISTORY:

```
19Jun02 - Jay Larson <larson@mcs.anl.gov> - Interface spec.
 3Jul02 - Jay Larson <larson@mcs.anl.gov> - Implementation.
10Jul02 - J. Larson <larson@mcs.anl.gov> - Improved argument
          checking.
```

13.1.2 MergeThreeGGSP_ - Merge Data from Three Sources

This routine merges `REAL` attribute data from three input `AttrVect` arguments `inAv1` , `inAv2`, and `inAv3` to a fourth `AttrVect outAv`. The attributes to be merged are determined entirely by the real attributes of `outAv`. If `outAv` shares one or more attributes with any of the inputs `inAv1`, `inAv2`, or `inAv3`, a merge is performed on the individual *intersections* of attributes between the pairs (`outAv`, `inAv1`), (`outAv`, `inAv2`), and (`outAv`, `inAv3`). Currently, it is assumed that these pairwise intersections are all equal. This assumption is of critical importance to the user. If the user violates this assumption, incorrect merges of any attributes present only in some (but not all) inputs will result.

The merge operatrion is a masked weighted element-by-element sum, as outlined in the following example. Let the vectors **a**,**b**, and **c** be data from Components A, B, and C that have been interpolated onto the physical grid of another component D. We wish to combine the data from A, B and C to get a vector **d**, which represents the merged data on the grid of component D. The merge relation to obtain the ith element of **d** is

$$d_i = \frac{1}{W_i}\left\{\prod_{j=1}^{J}\kappa_i^j\prod_{k=1}^{K}\alpha_i^k a_i + \prod_{l=1}^{L}\lambda_i^l\prod_{m=1}^{M}\beta_i^m b_i + \prod_{p=1}^{P}\mu_i^p\prod_{q=1}^{Q}\gamma_i^q c_i\right\},$$

where

$$W_i = \prod_{j=1}^{J}\kappa_i^j\prod_{k=1}^{K}\alpha_i^k + \prod_{l=1}^{L}\lambda_i^l\prod_{m=1}^{M}\beta_i^m + \prod_{p=1}^{P}\mu_i^p\prod_{q=1}^{Q}\gamma_i^q.$$

The quantities κ_i^j, λ_i^p, and μ_i^p are *integer masks* (which have value either 0 or 1), and α_i^k, β_i^m, and γ_i^q are *real masks* (which are in the closed interval $[0, 1]$).

The integer and real masks are stored as attributes to the same input `GeneralGrid` argument `GGrid`. The mask attribute names are stored as substrings to the colon-separated strings contained in the input `CHARACTER` arguments `iMaskTags1`, `iMaskTags2`, `iMaskTags3`, `rMaskTags1`, `rMaskTags2`, and `rMaskTags3`. The `LOGICAL` input argument `CheckMasks` governs how the masks are applied. If `CheckMasks` = .`TRUE`., the entries are checked to ensure they meet the definitions of real and integer masks. If `CheckMasks` = .`FALSE`. then the masks are multiplied together on an element-by-element basis with no validation of their entries (this option results in slightly higher performance).

This routine returns the sum of the masked weights as a diagnostic. This quantity is returned in the output `REAL` array `WeightSum`.

The correspondence between the quantities in the above merge relation and the arguments to this routine are summarized in the table.

Quantity	Stored in Argument	Referenced by Argument
a_i	inAv1	
b_i	inAv2	
c_i	inAv3	
d_i	outAv	
$\kappa_i^j, j = 1, \ldots, J$	GGrid	iMaskTags1 (J items)
$\alpha_i^k, k = 1, \ldots, K$	GGrid	rMaskTags1 (K items)
$\lambda_i^l, l = 1, \ldots, L$	GGrid	iMaskTags2 (L items)
$\beta_i^m, m = 1, \ldots, M$	GGrid	rMaskTags2 (M items)
$\mu_i^p, p = 1, \ldots, P$	GGrid	iMaskTags3 (L items)
$\gamma_i^q, q = 1, \ldots, Q$	GGrid	rMaskTags3 (M items)
W_i	WeightSum	

INTERFACE:

```
subroutine MergeThreeGGSP_(inAv1, iMaskTags1, rMaskTags1, &
                           inAv2, iMaskTags2, rMaskTags2, &
                           inAv3, iMaskTags3, rMaskTags3, &
                           GGrid, CheckMasks, outAv, WeightSum)
```

USES:

```
use m_stdio
use m_die

use m_realkinds, only : SP, FP

use m_List, only : List
use m_List, only : List_allocated => allocated

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nRAttr => nRAttr

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),               intent(IN)    :: inAv1
character(len=*),   optional, intent(IN)    :: iMaskTags1
character(len=*),   optional, intent(IN)    :: rMaskTags1
type(AttrVect),               intent(IN)    :: inAv2
character(len=*),   optional, intent(IN)    :: iMaskTags2
character(len=*),   optional, intent(IN)    :: rMaskTags2
type(AttrVect),               intent(IN)    :: inAv3
character(len=*),   optional, intent(IN)    :: iMaskTags3
character(len=*),   optional, intent(IN)    :: rMaskTags3
```

```
type(GeneralGrid),            intent(IN)    :: GGrid
logical,                      intent(IN)    :: CheckMasks
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),               intent(INOUT) :: outAv
real(SP),       dimension(:), pointer       :: WeightSum
```

REVISION HISTORY:

```
19Jun02 - Jay Larson <larson@mcs.anl.gov> - Interface spec.
 3Jul02 - Jay Larson <larson@mcs.anl.gov> - Implementation.
10Jul02 - J. Larson <larson@mcs.anl.gov> - Improved argument
          checking.
```

13.1.3 MergeFourGGSP_ - Merge Data from Four Sources

This routine merges `REAL` attribute data from four input `AttrVect` arguments `inAv1` , `inAv2`, `inAv3`, and `inAv4` to a fifth `AttrVect outAv`. The attributes to be merged are determined entirely by the real attributes of `outAv`. If `outAv` shares one or more attributes with any of the inputs `inAv1`, `inAv2`, `inAv3`, or `inAv4`, a merge is performed on the individual *intersections* of attributes between the pairs (`outAv`, `inAv1`), (`outAv`, `inAv2`), (`outAv`, `inAv3`), and (`outAv`, `inAv3`). Currently, it is assumed that these pairwise intersections are all equal. This assumption is of critical importance to the user. If the user violates this assumption, incorrect merges of any attributes present only in some (but not all) the inputs will result.
The merge operatrion is a masked weighted element-by-element sum, as outlined in the following example. Let the vectors **a**,**b**, **c** and **d** be data from Components A, B, C, and D that have been interpolated onto the physical grid of another component E. We wish to combine the data from A, B, C, and D to get a vector **e**, which represents the merged data on the grid of component E. The merge relation to obtain the ith element of **e** is

$$e_i = \frac{1}{W_i}\left\{\prod_{j=1}^{J}\kappa_i^j\prod_{k=1}^{K}\alpha_i^k a_i + \prod_{l=1}^{L}\lambda_i^l\prod_{m=1}^{M}\beta_i^m b_i + \prod_{p=1}^{P}\mu_i^p\prod_{q=1}^{Q}\gamma_i^q c_i + \prod_{r=1}^{R}\nu_i^r\prod_{s=1}^{S}\delta_i^s d_i\right\},$$

where

$$W_i = \prod_{j=1}^{J}\kappa_i^j\prod_{k=1}^{K}\alpha_i^k + \prod_{l=1}^{L}\lambda_i^l\prod_{m=1}^{M}\beta_i^m + \prod_{p=1}^{P}\mu_i^p\prod_{q=1}^{Q}\gamma_i^q + \prod_{r=1}^{R}\nu_i^r\prod_{s=1}^{S}\delta_i^s.$$

The quantities κ_i^j, λ_i^p, μ_i^p, and ν_i^r are *integer masks* (which have value either 0 or 1), and α_i^k, β_i^m, γ_i^q, and δ_i^s are *real masks* (which are in the closed interval $[0, 1]$).
The integer and real masks are stored as attributes to the same input `GeneralGrid` argument `GGrid`. The mask attribute names are stored as substrings to the colon-separated strings contained in the input `CHARACTER` arguments `iMaskTags1`, `iMaskTags2`, `iMaskTags3`, `iMaskTags4`, `rMaskTags1`, and `rMaskTags2`, `rMaskTags3`, and `rMaskTags4`, . The `LOGICAL` input argument `CheckMasks` governs how the masks are applied. If `CheckMasks` = `.TRUE.`, the entries are checked to ensure they meet the definitions of real and integer masks. If `CheckMasks` = `.FALSE.` then the masks are multiplied together on an element-by-element basis with no validation of their entries (this option results in slightly higher performance).
This routine returns the sume of the masked weights as a diagnostic. This quantity is returned in the output `REAL` array `WeightSum`.
The correspondence between the quantities in the above merge relation and the arguments to this routine are summarized in the table.

Quantity	Stored in Argument	Referenced by Argument
a_i	inAv1	
b_i	inAv2	
c_i	inAv3	
d_i	inAv4	
e_i	outAv	
$\kappa_i^j, j = 1, \ldots, J$	GGrid	iMaskTags1 (J items)
$\alpha_i^k, k = 1, \ldots, K$	GGrid	rMaskTags1 (K items)
$\lambda_i^l, l = 1, \ldots, L$	GGrid	iMaskTags2 (L items)
$\beta_i^m, m = 1, \ldots, M$	GGrid	rMaskTags2 (M items)
$\mu_i^p, p = 1, \ldots, P$	GGrid	iMaskTags3 (L items)
$\gamma_i^q, q = 1, \ldots, Q$	GGrid	rMaskTags3 (M items)
$\nu_i^r, r = 1, \ldots, R$	GGrid	iMaskTags4 (L items)
$\delta_i^s, s = 1, \ldots, S$	GGrid	rMaskTags4 (M items)
W_i	WeightSum	

INTERFACE:

```
subroutine MergeFourGGSP_(inAv1, iMaskTags1, rMaskTags1, &
                          inAv2, iMaskTags2, rMaskTags2, &
                          inAv3, iMaskTags3, rMaskTags3, &
                          inAv4, iMaskTags4, rMaskTags4, &
                          GGrid, CheckMasks, outAv, WeightSum)
```

USES:

```
use m_stdio
use m_die

use m_realkinds, only : SP, FP

use m_List, only : List
use m_List, only : List_allocated => allocated

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nRAttr => nRAttr

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),               intent(IN)    :: inAv1
character(len=*),   optional, intent(IN)    :: iMaskTags1
```

```
character(len=*),   optional, intent(IN)    :: rMaskTags1
type(AttrVect),               intent(IN)    :: inAv2
character(len=*),   optional, intent(IN)    :: iMaskTags2
character(len=*),   optional, intent(IN)    :: rMaskTags2
type(AttrVect),               intent(IN)    :: inAv3
character(len=*),   optional, intent(IN)    :: iMaskTags3
character(len=*),   optional, intent(IN)    :: rMaskTags3
type(AttrVect),               intent(IN)    :: inAv4
character(len=*),   optional, intent(IN)    :: iMaskTags4
character(len=*),   optional, intent(IN)    :: rMaskTags4
type(GeneralGrid),            intent(IN)    :: GGrid
logical,                      intent(IN)    :: CheckMasks
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),               intent(INOUT) :: outAv
real(SP),       dimension(:), pointer       :: WeightSum
```

REVISION HISTORY:

```
19Jun02 - Jay Larson <larson@mcs.anl.gov> - Interface spec.
 3Jul02 - Jay Larson <larson@mcs.anl.gov> - Implementation.
10Jul02 - J. Larson <larson@mcs.anl.gov> - Improved argument
          checking.
```

13.1.4 MergeInDataGGSP_ - Add Data into a Merge

This routine takes input field data from the input `AttrVect` argument `inAv`, and merges the real attributes it shares with the input/output `AttrVect` argument `outAv`. The merge is a masked merge of the form

$$c_i = c_i + \prod_{j=1}^{J} M_i^j \prod_{k=1}^{K} F_i^k a_i,$$

where c_i represents one element of one of the real attributes of `outAv`, and a_i represents one element of one of the real attributes of `inAv`. The M_i^j are *integer masks* which have value either 0 or 1, and are integer attributes of the input `GeneralGrid` argument `GGrid`. The F_i^k are *real masks* whose values are in the closed interval $[0, 1]$, and are real attributes of the input `GeneralGrid` argument `GGrid`. The input `CHARACTER` argument `iMaskTags` is a string of colon- delimited strings that name the integer attributes in `GGrid` that are used as the masks M_i^j. The input `CHARACTER` argument `rMaskTags` is a string of colon-delimited strings that name the real attributes in `GGrid` that are used as the masks F_i^k. The output `REAL` array `WeightSum` is used to store a running sum of the product of the masks. The `LOGICAL` input argument `CheckMasks` governs how the masks are applied. If `CheckMasks` = `.TRUE.`, the entries are checked to ensure they meet the definitions of real and integer masks. If `CheckMasks` = `.FALSE.` then the masks are multiplied together on an element-by-element basis with no validation of their entries (this option results in slightly higher performance).
`N.B.:` The lengths of the `AttrVect` arguments `inAv` and `outAv` must be equal, and this length must also equal the lengths of `GGrid` and `WeightSum`.
`N.B.:` This algorithm assumes the `AttrVect` argument `outAv` has been created, and its real attributes have been initialized.
`N.B.:` This algorithm assumes that the array `WeightSum` has been created and initialized.

INTERFACE:

```
subroutine MergeInDataGGSP_(inAv, iMaskTags, rMaskTags, GGrid, &
                            CheckMasks, outAv, WeightSum)
```

USES:

```
use m_stdio
use m_die

use m_realkinds, only : SP, FP

use m_String, only : String
use m_String, only : String_clean => clean
use m_String, only : String_ToChar => toChar

use m_List, only : List
use m_List, only : List_init => init
use m_List, only : List_clean => clean
use m_List, only : List_nitem => nitem
use m_List, only : List_get => get
use m_List, only : List_identical => identical
use m_List, only : List_allocated => allocated

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nRAttr => nRAttr
use m_AttrVect, only : SharedAttrIndexList

use m_GeneralGrid, only : GeneralGrid
use m_GeneralGrid, only : GeneralGrid_lsize => lsize
use m_GeneralGrid, only : GeneralGrid_exportIAttr => exportIAttr
use m_GeneralGrid, only : GeneralGrid_exportRAttr => exportRAttr

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),               intent(IN)    :: inAv
character(len=*),   optional, intent(IN)    :: iMaskTags
character(len=*),   optional, intent(IN)    :: rMaskTags
type(GeneralGrid),            intent(IN)    :: GGrid
logical,                      intent(IN)    :: CheckMasks
```

INPUT/OUTPUT PARAMETERS:

```
type(AttrVect),               intent(INOUT) :: outAv
real(SP),       dimension(:), pointer       :: WeightSum
```

REVISION HISTORY:

```
  19Jun02 - Jay Larson <larson@mcs.anl.gov> - initial verson.
  10Jul02 - J. Larson <larson@mcs.anl.gov> - Improved argument
            checking.
```

14 Time Averaging

14.1 Module m_Accumulator - Time Averaging/Accumlation Buffer (Source File: m_Accumulator.F90)

An *accumulator* is a data class used for computing running sums and/or time averages of `AttrVect` class data. The period of time over which data are accumulated/averaged is the *accumulation cycle*, which is defined by the total number of accumulation steps (the component `Accumulator%num_steps`). When the accumulation routine `accumulate_` is invoked, the number of accumulation cycle steps (the component `Accumulator%steps_done`)is incremented, and compared with the number of steps in the accumulation cycle to determine if the accumulation cycle has been completed. The accumulation buffers of the `Accumulator` are stored in an `AttrVect` (namely the component `Accumulator%data`), which allows the user to define the number of variables and their names at run-time. Finally, one can define for each field being accumulated the specific accumulation *action*. Currently, there are two options: Time Averaging and Time Summation. The user chooses the specific action by setting an integer action flag for each attribute being accumulated. The supported options are defined by the public data member constants `MCT_SUM` and `MCT_AVG`.
This module also supports a simple usage of accumulator where all the actions are SUM (`inits_` and `initavs_`) and the user must call `average_` to calculate the average from the current value of `Accumulator%steps_done`. `Accumulator%num_steps` is ignored in this case.

INTERFACE:

```
  module m_Accumulator
```

USES:

```
       use m_List, only : List
       use m_AttrVect, only : AttrVect
       use m_realkinds,only : SP,DP,FP

       implicit none

       private ! except
```

PUBLIC TYPES:

```
       public :: Accumulator ! The class data structure

     Type Accumulator
 #ifdef SEQUENCE
       sequence
 #endif
       integer :: num_steps      ! total number of accumulation steps
       integer :: steps_done     ! number of accumulation steps performed
       integer, pointer, dimension(:) :: iAction ! index of integer actions
       integer, pointer, dimension(:) :: rAction ! index of real actions
       type(AttrVect) :: data    ! accumulated sum field storage
     End Type Accumulator
```

PUBLIC MEMBER FUNCTIONS:

```
       public :: init            ! creation method
       public :: initp           ! partial creation method (MCT USE ONLY)
       public :: clean ! destruction method
       public :: initialized     ! check if initialized
       public :: lsize ! local length of the data arrays
       public :: NumSteps        ! number of steps in a cycle
       public :: StepsDone       ! number of steps completed in the
```

```
                              ! current cycle
    public :: nIAttr ! number of integer fields
    public :: nRAttr ! number of real fields
    public :: indexIA ! index the integer fields
    public :: indexRA ! index the real fields
    public :: getIList ! Return tag from INTEGER
                              ! attribute list
    public :: getRList ! Return tag from REAL attribute
                              ! list
    public :: exportIAttr  ! Return INTEGER attribute as a vector
    public :: exportRAttr  ! Return REAL attribute as a vector
    public :: importIAttr  ! Insert INTEGER vector as attribute
    public :: importRAttr  ! Insert REAL vector as attribute
    public :: zero         ! Clear an accumulator
    public :: SharedAttrIndexList ! Returns the number of shared
   ! attributes, and lists of the
   ! respective locations of these
   ! shared attributes
    public :: accumulate   ! Add AttrVect data into an Accumulator
    public :: average      ! Calculate an average in an Accumulator

Definition of interfaces for the methods for the Accumulator:

  interface init   ; module procedure &
     init_,&
     inits_,&
     initv_,  &
     initavs_
  end interface
  interface initp  ; module procedure initp_ ; end interface
  interface clean  ; module procedure clean_  ; end interface
  interface initialized; module procedure initialized_ ; end interface
  interface lsize  ; module procedure lsize_  ; end interface
  interface NumSteps  ; module procedure NumSteps_  ; end interface
  interface StepsDone  ; module procedure StepsDone_  ; end interface
  interface nIAttr ; module procedure nIAttr_ ; end interface
  interface nRAttr ; module procedure nRAttr_ ; end interface
  interface indexIA; module procedure indexIA_; end interface
  interface indexRA; module procedure indexRA_; end interface
  interface getIList; module procedure getIList_; end interface
  interface getRList; module procedure getRList_; end interface
  interface exportIAttr ; module procedure exportIAttr_ ; end interface
  interface exportRAttr ; module procedure &
       exportRAttrSP_, &
       exportRAttrDP_
  end interface
  interface importIAttr ; module procedure importIAttr_ ; end interface
  interface importRAttr ; module procedure &
       importRAttrSP_, &
       importRAttrDP_
  end interface
  interface zero ; module procedure zero_ ; end interface
  interface SharedAttrIndexList ; module procedure   &
     aCaCSharedAttrIndexList_,  &
     aVaCSharedAttrIndexList_
  end interface
  interface accumulate ; module procedure accumulate_ ; end interface
  interface average ; module procedure average_ ; end interface
```

PUBLIC DATA MEMBERS:

```
  public :: MCT_SUM
  public :: MCT_AVG

integer, parameter :: MCT_SUM = 1
integer, parameter :: MCT_AVG = 2
```

REVISION HISTORY:

```
7Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
7Feb01 - Jay Larson <larson@mcs.anl.gov> - Public interfaces
         to getIList() and getRList().
9Aug01 - E.T. Ong <eong@mcs.anl.gov> - added initialized and
         initp_ routines. Added 'action' in Accumulator type.
6May02 - Jay Larson <larson@mcs.anl.gov> - added import/export
          routines.
26Aug02 - E.T. Ong <eong@mcs.anl.gov> - thourough code revision;
          no added routines
10Jan08 - R. Jacob <jacob@mcs.anl.gov> - add simple accumulator
          use support and check documentation.
```

14.1.1 init_ - Initialize an Accumulator and its Registers

This routine allocates space for the output `Accumulator` argument `aC`, and at a minimum sets the number of time steps in an accumulation cycle (defined by the input `INTEGER` argument `num_steps`), and the *length* of the `Accumulator` register buffer (defined by the input `INTEGER` argument `lsize`). If one wishes to accumulate integer fields, the list of these fields is defined by the input `CHARACTER` argument `iList`, which is specified as a colon-delimited set of substrings (further information regarding this is available in the routine `init_()` of the module `m_AttrVect`). If no value of `iList` is supplied, no integer attribute accumulation buffers will be allocated. The accumulation action on each of the integer attributes can be defined by supplying the input `INTEGER` array argument `iAction(:)` (whose length must correspond to the number of items in `iList`). The values of the elements of `iAction(:)` must be one of the values among the public data members defined in the declaration section of this module. If the integer attributes are to be accumulated (i.e. one supplies `iList`), but `iAction(:)` is not specified, the default action for all integer accumulation operations will be summation. The input arguments `rList` and `rAction(:)` define the names of the real variables to be accumulated and the accumulation action for each. The arguments `rList` and `rAction(:)` are related to each other the same way as `iList` and `iAction(:)`. Finally, the user can manually set the number of completed steps in an accumulation cycle (e.g. for restart purposes) by supplying a value for the optional input `INTEGER` argument `steps_done`.

INTERFACE:

```
subroutine init_(aC, iList, iAction, rList, rAction, lsize, &
                 num_steps,steps_done)
```

USES:

```
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero

use m_List, only: List
use m_List, only: List_nullify => nullify
use m_List, only: List_init => init
use m_List, only: List_nitem => nitem
use m_List, only: List_clean => clean
```

```
use m_stdio
use m_die

implicit none
```

INPUT PARAMETERS:

```
character(len=*),      optional, intent(in)  :: iList
integer, dimension(:), optional, intent(in)  :: iAction
character(len=*),      optional, intent(in)  :: rList
integer, dimension(:), optional, intent(in)  :: rAction
integer,                         intent(in)  :: lsize
integer,                         intent(in)  :: num_steps
integer,               optional, intent(in)  :: steps_done
```

OUTPUT PARAMETERS:

```
type(Accumulator),               intent(out) :: aC
```

REVISION HISTORY:

```
11Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
27JUL01 - E.T. Ong <eong@mcs.anl.gov> - added iAction, rAction,
          niAction, and nrAction to accumulator type. Also defined
          MCT_SUM and MCT_AVG for accumulator module.
```

14.1.2 inits_ - Initialize a simple Accumulator and its Registers

This routine allocates space for the output simple `Accumulator` argument `aC`, and sets the *length* of the `Accumulator` register buffer (defined by the input `INTEGER` argument `lsize`). If one wishes to accumulate integer fields, the list of these fields is defined by the input `CHARACTER` argument `iList`, which is specified as a colon-delimited set of substrings (further information regarding this is available in the routine `init_()` of the module `m_AttrVect`). If no value of `iList` is supplied, no integer attribute accumulation buffers will be allocated. The input argument `rList` define the names of the real variables to be accumulated. Finally, the user can manually set the number of completed steps in an accumulation cycle (e.g. for restart purposes) by supplying a value for the optional input `INTEGER` argument `steps_done`. Its default value is zero.
In a simple accumulator, the action is always SUM.

INTERFACE:

```
subroutine inits_(aC, iList, rList, lsize,steps_done)
```

USES:

```
use m_List, only : List_init => init
use m_List, only : List_clean => clean
use m_List, only : List_nitem => nitem
use m_AttrVect, only : AttrVect_init => init
use m_AttrVect, only : AttrVect_zero => zero
use m_die

implicit none
```

INPUT PARAMETERS:

```
character(len=*),      optional, intent(in)  :: iList
character(len=*),      optional, intent(in)  :: rList
integer,                         intent(in)  :: lsize
integer,               optional, intent(in)  :: steps_done
```

OUTPUT PARAMETERS:

```
type(Accumulator),               intent(out) :: aC
```

REVISION HISTORY:

```
10Jan08 - R. Jacob <jacob@mcs.anlgov> - initial version based on init_
```

14.1.3 initp_ - Initialize an Accumulator but not its Registers

This routine is an internal service routine for use by the other initialization routines in this module. It sets up some—but not all—of the components of the output `Accumulator` argument `aC`. This routine can set up the following components of `aC`:

1. `aC%iAction`, the array of accumlation actions for the integer attributes of `aC` (if the input `INTEGER` array argument `iAction(:)` is supplied);
2. `aC%rAction`, the array of accumlation actions for the real attributes of `aC` (if the input `INTEGER` array argument `rAction(:)` is supplied);
3. `aC%num_steps`, the number of steps in an accumulation cycle (if the input `INTEGER` argument `num_steps` is supplied); and
4. `aC%steps_done`, the number of steps completed so far in an accumulation cycle (if the input `INTEGER` argument `steps_done` is supplied).

INTERFACE:

```
subroutine initp_(aC, iAction, rAction, num_steps, steps_done)
```

USES:

```
use m_die

implicit none
```

INPUT PARAMETERS:

```
integer, dimension(:), optional, intent(in)  :: iAction
integer, dimension(:), optional, intent(in)  :: rAction
integer,                         intent(in)  :: num_steps
integer,               optional, intent(in)  :: steps_done
```

OUTPUT PARAMETERS:

```
type(Accumulator),               intent(out) :: aC
```

REVISION HISTORY:

```
11Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
27JUL01 - E.T. Ong <eong@mcs.anl.gov> - added iAction, rAction,
          niAction, and nrAction to accumulator type. Also defined
          MCT_SUM and MCT_AVG for accumulator module.
```

14.1.4 initv_ - Initialize One Accumulator using Another

This routine takes the integer and real attribute information (including accumulation action settings for each attribute) from a previously initialized `Accumulator` (the input argument `bC`), and uses it to create another `Accumulator` (the output argument `aC`). In the absence of the `INTEGER` input arguments `lsize`, `num_steps`, and `steps_done`, `aC` will inherit from `bC` its length, the number of steps in its accumulation cycle, and the number of steps completed in its present accumulation cycle, respectively.

INTERFACE:

```
subroutine initv_(aC, bC, lsize, num_steps, steps_done)
```

USES:

```
use m_List,   only : List
use m_List,   only : ListExportToChar => exportToChar
use m_List,   only : List_copy        => copy
use m_List,   only : List_allocated   => allocated
use m_List,   only : List_clean       => clean
use m_die

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),           intent(in)  :: bC
integer,           optional, intent(in)  :: lsize
integer,           optional, intent(in)  :: num_steps
integer,           optional, intent(in)  :: steps_done
```

OUTPUT PARAMETERS:

```
type(Accumulator),           intent(out) :: aC
```

REVISION HISTORY:

```
11Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
17May01 - R. Jacob <jacob@mcs.anl.gov> - change string_get to
          list_get
27JUL01 - E.T. Ong <eong@mcs.anl.gov> - added iaction,raction
          compatibility
 2Aug02 - J. Larson <larson@mcs.anl.gov> made argument num_steps
          optional
```

14.1.5 initavs_ - Initialize a simple Accumulator from an AttributeVector

This routine takes the integer and real attribute information (including from a previously initialized `AttributeVector` (the input argument `aV`), and uses it to create a simple (sum only) `Accumulator` (the output argument `aC`). In the absence of the `INTEGER` input argument `lsize`, `aC` will inherit from `Av` its length. In the absence of the optional INTEGER argument, `steps_done` will be set to zero.

INTERFACE:

```
subroutine initavs_(aC, aV, acsize, steps_done)
```

USES:

```
use m_AttrVect, only: AttrVect_lsize => lsize
use m_AttrVect, only: AttrVect_nIAttr => nIAttr
use m_AttrVect, only: AttrVect_nRAttr => nRAttr
use m_AttrVect, only: AttrVect_exIL2c => exportIListToChar
use m_AttrVect, only: AttrVect_exRL2c => exportRListToChar
use m_die

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),           intent(in)     :: aV
integer,           optional, intent(in)  :: acsize
integer,           optional, intent(in)  :: steps_done
```

OUTPUT PARAMETERS:

```
type(Accumulator),           intent(out) :: aC
```

REVISION HISTORY:

```
10Jan08 - R. Jacob <jacob@mcs.anl.gov> - initial version based on initv_
```

14.1.6 clean_ - Destroy an Accumulator

This routine deallocates all allocated memory structures associated with the input/output `Accumulator` argument `aC`. The success (failure) of this operation is signified by the zero (non-zero) value of the optional `INTEGER` output argument `stat`. If `clean_()` is invoked with `stat` present, it is the user's obligation to check this return code and act accordingly. If `stat` is not supplied and any of the deallocation operations fail, this routine will terminate execution with an error statement.

INTERFACE:

```
subroutine clean_(aC, stat)
```

USES:

```
use m_mall
use m_stdio
use m_die
use m_AttrVect, only : AttrVect_clean => clean

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(Accumulator), intent(inout) :: aC
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
11Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
27JUL01 - E.T. Ong <eong@mcs.anl.gov> - deallocate pointers iAction
          and rAction.
 1Mar02 - E.T. Ong <eong@mcs.anl.gov> removed the die to prevent
          crashes and added stat argument.
```

14.1.7 initialized_ - Check if an Accumulator is Initialized

This logical function returns a value of `.TRUE.` if the input `Accumulator` argument `aC` is initialized correctly. The term "correctly initialized" means there is internal consistency between the number of integer and real attributes in `aC`, and their respective data structures for accumulation registers, and accumulation action flags. The optional `LOGICAL` input argument `die_flag` if present, can result in messages written to `stderr`:

- if `die_flag` is true and `aC` is correctly initialized, and
- if `die_flag` is false and `aC` is incorrectly initialized.

Otherwise, inconsistencies in how `aC` is set up will result in termination with an error message. The optional `CHARACTER` input argument `source_name` allows the user to, in the event of error, generate traceback information (e.g., the name of the routine that invoked this one).

INTERFACE:

```
logical function initialized_(aC, die_flag, source_name)
```

USES:

```
use m_stdio
use m_die
use m_List, only : List
use m_List, only : List_allocated => allocated

use m_AttrVect, only : AttrVect
use m_AttrVect, only : Attr_nIAttr => nIAttr
use m_AttrVect, only : Attr_nRAttr => nRAttr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),          intent(in) :: aC
logical,          optional, intent(in) :: die_flag
character(len=*), optional, intent(in) :: source_name
```

REVISION HISTORY:

```
7AUG01 - E.T. Ong <eong@mcs.anl.gov> - initital prototype
```

14.1.8 lsize_ - Length of an Accumulator

This `INTEGER` query function returns the number of data points for which the input `Accumulator` argument `aC` is performing accumulation. This value corresponds to the length of the `AttrVect` component `aC%data` that stores the accumulation registers.

INTERFACE:

```
integer function lsize_(aC)
```

USES:

```
use m_AttrVect, only : AttrVect_lsize => lsize

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator), intent(in) :: aC
```

REVISION HISTORY:

```
12Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.9 NumSteps_ - Number of Accumulation Cycle Time Steps

This `INTEGER` query function returns the number of time steps in an accumulation cycle for the input `Accumulator` argument `aC`.

INTERFACE:

```
integer function NumSteps_(aC)
```

USES:

```
use m_die,   only : die
use m_stdio, only : stderr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator), intent(in) :: aC
```

REVISION HISTORY:

```
7Aug02 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.10 StepsDone_ - Number of Completed Steps in the Current Cycle

This `INTEGER` query function returns the of time steps that have been completed in the current accumulation cycle for the input `Accumulator` argument `aC`.

INTERFACE:

```
integer function StepsDone_(aC)
```

USES:

```
use m_die,   only : die
use m_stdio, only : stderr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator), intent(in) :: aC
```

REVISION HISTORY:

```
7Aug02 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.11 nIAttr_ - Return the Number of INTEGER Attributes

This INTEGER query function returns the number of integer attributes that are stored in the input Accumulator argument aC. This value is equal to the number of integer attributes in the AttrVect component aC%data that stores the accumulation registers.

INTERFACE:

```
integer function nIAttr_(aC)
```

USES:

```
use m_AttrVect, only : AttrVect_nIAttr => nIAttr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),intent(in) :: aC
```

REVISION HISTORY:

```
12Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.12 nRAttr_ - number of REAL fields stored in the Accumulator.

This INTEGER query function returns the number of real attributes that are stored in the input Accumulator argument aC. This value is equal to the number of real attributes in the AttrVect component aC%data that stores the accumulation registers.

INTERFACE:

```
integer function nRAttr_(aC)
```

USES:

```
use m_AttrVect, only : AttrVect_nRAttr => nRAttr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),intent(in) :: aC
```

REVISION HISTORY:

```
12Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.13 getIList_ - Retrieve a Numbered INTEGER Attribute Name

This routine returns as a String (see the mpeu module m_String for information) the name of the ith item in the integer registers of the Accumulator argument aC.

INTERFACE:

```
subroutine getIList_(item, ith, aC)
```

USES:

```
use m_AttrVect, only : AttrVect_getIList => getIList
use m_String,   only : String

implicit none
```

INPUT PARAMETERS:

```
integer,           intent(in)  :: ith
type(Accumulator), intent(in)  :: aC
```

OUTPUT PARAMETERS:

```
type(String),      intent(out) :: item
```

REVISION HISTORY:

```
12Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.14 getRList_ - Retrieve a Numbered REAL Attribute Name

This routine returns as a `String` (see the mpeu module `m_String` for information) the name of the `ith` item in the real registers of the `Accumulator` argument `aC`.

INTERFACE:

```
subroutine getRList_(item, ith, aC)
```

USES:

```
use m_AttrVect, only : AttrVect_getRList => getRList
use m_String,   only : String

implicit none
```

INPUT PARAMETERS:

```
integer,          intent(in)  :: ith
type(Accumulator),intent(in)  :: aC
```

OUTPUT PARAMETERS:

```
type(String),     intent(out) :: item
```

REVISION HISTORY:

```
12Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.15 indexIA_ - Index an INTEGER Attribute

This `INTEGER` query function returns the index in the integer accumulation register buffer of the `Accumulator` argument `aC` the attribute named by the `CHARACTER` argument `item`. That is, all the accumulator running tallies for the attribute named `item` reside in

```
aC%data%iAttr(indexIA_(aC,item),:).
```

The user may request traceback information (e.g., the name of the routine from which this one is called) by providing values for either of the optional `CHARACTER` arguments `perrWith` or `dieWith` In the event `indexIA_()` can not find `item` in `aC`, the routine behaves as follows:

1. if neither `perrWith` nor `dieWith` are present, `indexIA_()` returns a value of zero;
2. if `perrWith` is present, but `dieWith` is not, an error message is written to `stderr` incorporating user-supplied traceback information stored in the argument `perrWith`;
3. if `dieWith` is present, execution terminates with an error message written to `stderr` that incorporates user-supplied traceback information stored in the argument `dieWith`.

INTERFACE:

```
integer function indexIA_(aC, item, perrWith, dieWith)
```

USES:

```
use m_AttrVect, only : AttrVect_indexIA => indexIA
use m_die,  only : die
use m_stdio,only : stderr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),          intent(in) :: aC
character(len=*),           intent(in) :: item
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

REVISION HISTORY:

```
14Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.16 indexRA_ - index the Accumulator real attribute list.

This `INTEGER` query function returns the index in the real accumulation register buffer of the `Accumulator` argument `aC` the attribute named by the `CHARACTER` argument `item`. That is, all the accumulator running tallies for the attribute named `item` reside in

```
aC%data%rAttr(indexRA_(aC,item),:).
```

The user may request traceback information (e.g., the name of the routine from which this one is called) by providing values for either of the optional `CHARACTER` arguments `perrWith` or `dieWith` In the event `indexRA_()` can not find `item` in `aC`, the routine behaves as follows:

1. if neither `perrWith` nor `dieWith` are present, `indexRA_()` returns a value of zero;
2. if `perrWith` is present, but `dieWith` is not, an error message is written to `stderr` incorporating user-supplied traceback information stored in the argument `perrWith`;
3. if `dieWith` is present, execution terminates with an error message written to `stderr` that incorporates user-supplied traceback information stored in the argument `dieWith`.

INTERFACE:

```
integer function indexRA_(aC, item, perrWith, dieWith)
```

USES:

```
use m_AttrVect, only : AttrVect_indexRA => indexRA
use m_die,  only : die
use m_stdio,only : stderr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),          intent(in) :: aC
character(len=*),           intent(in) :: item
character(len=*), optional, intent(in) :: perrWith
character(len=*), optional, intent(in) :: dieWith
```

REVISION HISTORY:

```
14Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.17 exportIAttr_ - Export INTEGER Attribute to a Vector

This routine extracts from the input `Accumulator` argument `aC` the integer attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`, and returns it in the `INTEGER` output array `outVect`, and its length in the output `INTEGER` argument `lsize`.
N.B.: This routine will fail if the `AttrTag` is not in the `Accumulator` `List` component `aC%data%iList`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `outVect` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `outVect`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) at the time this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `outVect`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportIAttr_(aC, AttrTag, outVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_exportIAttr => exportIAttr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),      intent(in)  :: aC
character(len=*),       intent(in)  :: AttrTag
```

OUTPUT PARAMETERS:

```
integer,  dimension(:), pointer     :: outVect
integer,  optional,     intent(out) :: lsize
```

REVISION HISTORY:

6May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.

14.1.18 exportRAttrSP_ - Export REAL Attribute to a Vector

This routine extracts from the input `Accumulator` argument `aC` the real attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`, and returns it in the `REAL` output array `outVect`, and its length in the output `INTEGER` argument `lsize`.
N.B.: This routine will fail if the `AttrTag` is not in the `Accumulator List` component `aC%data%iList`.
N.B.: The flexibility of this routine regarding the pointer association status of the output argument `outVect` means the user must invoke this routine with care. If the user wishes this routine to fill a pre-allocated array, then obviously this array must be allocated prior to calling this routine. If the user wishes that the routine *create* the output argument array `outVect`, then the user must ensure this pointer is not allocated (i.e. the user must nullify this pointer) at the time this routine is invoked.
N.B.: If the user has relied on this routine to allocate memory associated with the pointer `outVect`, then the user is responsible for deallocating this array once it is no longer needed. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine exportRAttrSP_(aC, AttrTag, outVect, lsize)
```

USES:

```
use m_die
use m_stdio

use m_AttrVect,      only : AttrVect_exportRAttr => exportRAttr

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),      intent(in)  :: aC
character(len=*),       intent(in)  :: AttrTag
```

OUTPUT PARAMETERS:

```
real(SP), dimension(:), pointer     :: outVect
integer,  optional,     intent(out) :: lsize
```

REVISION HISTORY:

6May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.

14.1.19 importIAttr_ - Import INTEGER Attribute from a Vector

This routine imports data provided in the input `INTEGER` vector `inVect` into the `Accumulator` argument `aC`, storing it as the integer attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`. The input `INTEGER` argument `lsize` is used to ensure there is sufficient space in the `Accumulator` to store the data.
N.B.: This routine will fail if the `AttrTag` is not in the `Accumulator List` component `aC%data%rList`.

INTERFACE:

```
subroutine importIAttr_(aC, AttrTag, inVect, lsize)
```

USES:

```
use m_die
use m_stdio ,        only : stderr

use m_AttrVect,      only : AttrVect_importIAttr => importIAttr

implicit none
```

INPUT PARAMETERS:

```
character(len=*),       intent(in)    :: AttrTag
integer, dimension(:),  pointer       :: inVect
integer,                intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(Accumulator),      intent(inout) :: aC
```

REVISION HISTORY:

```
6May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

14.1.20 importRAttrSP_ - Import REAL Attribute from a Vector

This routine imports data provided in the input `REAL` vector `inVect` into the `Accumulator` argument `aC`, storing it as the real attribute corresponding to the tag defined in the input `CHARACTER` argument `AttrTag`. The input `INTEGER` argument `lsize` is used to ensure there is sufficient space in the `Accumulator` to store the data.
N.B.: This routine will fail if the `AttrTag` is not in the `Accumulator` `List` component `aC%data%rList`.

INTERFACE:

```
subroutine importRAttrSP_(aC, AttrTag, inVect, lsize)
```

USES:

```
use m_die
use m_stdio ,        only : stderr

use m_AttrVect,      only : AttrVect_importRAttr => importRAttr

implicit none
```

INPUT PARAMETERS:

```
character(len=*),       intent(in)    :: AttrTag
real(SP), dimension(:), pointer       :: inVect
integer,                intent(in)    :: lsize
```

INPUT/OUTPUT PARAMETERS:

```
type(Accumulator),      intent(inout) :: aC
```

REVISION HISTORY:

```
6May02 - J.W. Larson <larson@mcs.anl.gov> - initial prototype.
```

14.1.21 zero_ - Zero an Accumulator

This subroutine clears the the `Accumulator` argument `aC`. This is accomplished by setting the number of completed steps in the accumulation cycle to zero, and zeroing out all of the accumlation registers.

INTERFACE:

```
subroutine zero_(aC)
```

USES:

```
use m_AttrVect, only : AttrVect_zero => zero

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(Accumulator), intent(inout) :: aC
```

REVISION HISTORY:

```
7Aug02 - Jay Larson <larson@mcs.anl.gov> - initial prototype
```

14.1.22 aCaCSharedAttrIndexList_ - Cross-index Two Accumulators

`aCaCSharedAttrIndexList_()` takes a pair of user-supplied `Accumulator` variables `aC1` and `aC2`, and for choice of either `REAL` or `INTEGER` attributes (as specified literally in the input `CHARACTER` argument `attrib`) returns the number of shared attributes `NumShared`, and arrays of indices `Indices1` and `Indices2` to their storage locations in `aC1` and `aC2`, respectively.
N.B.: This routine returns two allocated arrays—`Indices1(:)` and `Indices2(:)`—which must be deallocated once the user no longer needs them. Failure to do this will create a memory leak.

INTERFACE:

```
subroutine aCaCSharedAttrIndexList_(aC1, aC2, attrib, NumShared, &
                                    Indices1, Indices2)
```

USES:

```
use m_stdio
use m_die,         only : MP_perr_die, die, warn

use m_List,     only : GetSharedListIndices

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),    intent(in)  :: aC1
type(Accumulator),    intent(in)  :: aC2
character*7,          intent(in)  :: attrib
```

OUTPUT PARAMETERS:

```
integer,              intent(out) :: NumShared
integer,dimension(:), pointer     :: Indices1
integer,dimension(:), pointer     :: Indices2
```

REVISION HISTORY:

```
7Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

14.1.23 aVaCSharedAttrIndexList_ - Cross-index with an AttrVect

aVaCSharedAttrIndexList_() a user-supplied AttrVect variable aV and an Accumulator variable aC, and for choice of either REAL or INTEGER attributes (as ! specified literally in the input CHARACTER argument attrib) returns the number of shared attributes NumShared, and arrays of indices Indices1 and Indices2 to their storage locations in aV and aC, respectively.
N.B.: This routine returns two allocated arrays—Indices1(:) and Indices2(:)—which must be deallocated once the user no longer needs them. Failure to do this will create a memory leak.

INTERFACE:

```
subroutine aVaCSharedAttrIndexList_(aV, aC, attrib, NumShared, &
                                    Indices1, Indices2)
```

USES:

```
use m_stdio
use m_die,          only : MP_perr_die, die, warn

use m_AttrVect,     only : AttrVect

use m_List,      only : GetSharedListIndices


implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),        intent(in)  :: aV
type(Accumulator),     intent(in)  :: aC
character(len=*),      intent(in)  :: attrib
```

OUTPUT PARAMETERS:

```
integer,               intent(out) :: NumShared
integer,dimension(:), pointer     :: Indices1
integer,dimension(:), pointer     :: Indices2
```

REVISION HISTORY:

```
7Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

14.1.24 accumulate_–Acumulate from an AttrVect to an Accumulator.

This routine performs time *accumlation* of data present in an MCT field data AttrVect variable aV and combines it with the running tallies stored in the MCT Accumulator variable aC. This routine automatically identifies which fields are held in common by aV and aC and uses the accumulation action information stored in aC to decide how each field in aV is to be combined into its corresponding running tally in aC. The accumulation operations currently supported are:

- MCT_SUM: Add the current values in the Av to the current values in Ac.
- MCT_AVG: Same as MCT_SUM except when steps_done is equal to num_steps then perform one more sum and replaced with average.

This routine also automatically increments the counter in aC signifying the number of steps completed in the accumulation cycle.
NOTE: The user must reset (zero) the Accumulator after the average has been formed or the next call to accumulate will add to the average.

INTERFACE:

```
subroutine accumulate_(aV, aC)
```

USES:

```
use m_stdio, only : stdout,stderr
use m_die,   only : die

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nIAttr => nIAttr
use m_AttrVect, only : AttrVect_nRAttr => nRAttr
use m_AttrVect, only : AttrVect_indexRA => indexRA
use m_AttrVect, only : AttrVect_indexIA => indexIA

implicit none
```

INPUT PARAMETERS:

```
type(AttrVect),     intent(in)    :: aV      ! Input AttrVect
```

INPUT/OUTPUT PARAMETERS:

```
type(Accumulator),  intent(inout) :: aC      ! Output Accumulator
```

REVISION HISTORY:

```
18Sep00 - J.W. Larson <larson@mcs.anl.gov> -- initial version.
 7Feb01 - J.W. Larson <larson@mcs.anl.gov> -- General version.
10Jun01 - E.T. Ong -- fixed divide-by-zero problem in integer
          attribute accumulation.
27Jul01 - E.T. Ong <eong@mcs.anl.gov> -- removed action argument.
          Make compatible with new Accumulator type.
```

14.1.25 average_ – Force an average to be taken on an Accumulator

This routine will compute the average of the current values in an `Accumulator` using the current value of `steps_done` in the `Accumulator`

INTERFACE:

```
subroutine average_(aC)
```

USES:

```
use m_stdio, only : stdout,stderr
use m_die,   only : die

use m_AttrVect, only : AttrVect
use m_AttrVect, only : AttrVect_lsize => lsize
use m_AttrVect, only : AttrVect_nIAttr => nIAttr
use m_AttrVect, only : AttrVect_nRAttr => nRAttr
use m_AttrVect, only : AttrVect_indexRA => indexRA
use m_AttrVect, only : AttrVect_indexIA => indexIA

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(Accumulator),  intent(inout) :: aC      ! Output Accumulator
```

REVISION HISTORY:

```
11Jan08 - R.Jacob <jacob@mcs.anl.gov> -- initial version based on accumulate_
```

14.2 Module m_AccumulatorComms - MPI Communication Methods for the Accumulator (Source File: m_AccumulatorComms.F90)

This module contains communications methods for the `Accumulator` datatype (see `m_Accumulator` for details). MCT's communications are implemented in terms of the Message Passing Interface (MPI) standard, and we have as best as possible, made the interfaces to these routines appear as similar as possible to the corresponding MPI routines. For the `Accumulator`, we currently support only the following collective operations: broadcast, gather, and scatter. The gather and scatter operations rely on domain decomposition descriptors that are defined elsewhere in MCT: the `GlobalMap`, which is a one-dimensional decomposition (see the MCT module `m_GlobalMap` for more details); and the `GlobalSegMap`, which is a segmented decomposition capable of supporting multidimensional domain decompositions (see the MCT module `m_GlobalSegMap` for more details).

INTERFACE:

```
module m_AccumulatorComms
```

USES:

```
No external modules are used in the declaration section of this module.

    implicit none

    private ! except
```

PUBLIC MEMBER FUNCTIONS:

```
List of communications Methods for the Accumulator class

    public :: gather ! gather all local vectors to the root
    public :: scatter ! scatter from the root to all PEs
    public :: bcast ! bcast from root to all PEs

Definition of interfaces for the communication methods for
the Accumulator:

  interface gather ; module procedure &
            GM_gather_, &
            GSM_gather_
  end interface
  interface scatter ; module procedure &
            GM_scatter_, &
            GSM_scatter_
  end interface
  interface bcast  ; module procedure bcast_  ; end interface
```

REVISION HISTORY:

```
31Oct00 - Jay Larson <larson@mcs.anl.gov> - initial prototype--
          These routines were separated from the module m_Accumulator
15Jan01 - Jay Larson <larson@mcs.anl.gov> - Specification of
          APIs for the routines GSM_gather_() and GSM_scatter_().
10May01 - Jay Larson <larson@mcs.anl.gov> - Changes in the
          comms routine to match the MPI model for collective
          communications, and general clean-up of prologues.
 9Aug01 - E.T. Ong <eong@mcs.anl.gov> - Added private routine
          bcastp_. Used new Accumulator routines initp_ and
          initialized_ to simplify the routines.
26Aug02 - E.T. Ong <eong@mcs.anl.gov> - thourough code revision;
           no added routines
```

14.2.1 GM_gather_ - Gather Accumulator Distributed by a GlobalMap

`GM_gather()` takes a distributed (across the communicator associated with the handle `comm`) input `Accumulator` argument `iC` and gathers its data to the `Accumulator oC` on the `root`. The decomposition of `iC` is described by the input `GlobalMap` argument `Gmap`. The success (failure) of this operation is signified by the zero (nonzero) value of the optional output argument `stat`.

INTERFACE:

```
subroutine GM_gather_(iC, oC, GMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalMap, only : GlobalMap
use m_AttrVect, only : AttrVect_clean => clean
use m_Accumulator, only : Accumulator
use m_Accumulator, only : Accumulator_initialized => initialized
use m_Accumulator, only : Accumulator_initv => init
use m_AttrVectComms, only : AttrVect_gather => gather

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator), intent(in)  :: iC
type(GlobalMap) ,  intent(in)  :: GMap
integer,           intent(in)  :: root
integer,           intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(Accumulator), intent(out) :: oC
integer, optional,intent(out)  :: stat
```

REVISION HISTORY:

```
13Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
31Oct00 - Jay Larson <larson@mcs.anl.gov> - relocated to the
          module m_AccumulatorComms
15Jan01 - Jay Larson <larson@mcs.anl.gov> - renamed GM_gather_
10May01 - Jay Larson <larson@mcs.anl.gov> - revamped comms
          model to match MPI comms model, and cleaned up prologue
 9Aug01 - E.T. Ong <eong@mcs.anl.gov> - 2nd prototype. Used the
          intiialized_ and accumulator init routines.
```

14.2.2 GSM_gather_ - Gather Accumulator Distributed by a GlobalSegMap

This routine takes the distrubuted (on the communcator associated with the handle `comm`) input `Accumulator` argument `iC` gathers it to the the `Accumulator` argument `oC` (valid only on the `root`). The decompositon of `iC` is contained in the input `GlobalSegMap` argument `GSMap`. The success (failure) of this operation is signified by the zero (nonzero) returned value of the `INTEGER` flag `stat`.

INTERFACE:

```
subroutine GSM_gather_(iC, oC, GSMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalSegMap, only : GlobalSegMap
use m_AttrVect, only : AttrVect_clean => clean
use m_Accumulator, only : Accumulator
use m_Accumulator, only : Accumulator_initv => init
use m_Accumulator,   only : Accumulator_initialized => initialized
use m_AttrVectComms, only : AttrVect_gather => gather

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),  intent(in) :: iC
type(GlobalSegMap), intent(in) :: GSMap
integer,            intent(in) :: root
integer,            intent(in) :: comm
```

OUTPUT PARAMETERS:

```
type(Accumulator), intent(out) :: oC
integer, optional, intent(out) :: stat
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
10May01 - Jay Larson <larson@mcs.anl.gov> - Initial code and
               cleaned up prologue.
     09Aug01 - E.T. Ong <eong@mcs.anl.gov> - 2nd prototype. Used the
               intiialized_ and accumulator init routines.
```

14.2.3 GM_scatter_ - Scatter an Accumulator using a GlobalMap

This routine takes the input `Accumulator` argument `iC` (valid only on the `root`), and scatters it to the distributed `Accumulator` argument `oC` on the processes associated with the communicator handle `comm`. The decompositon used to scatter the data is contained in the input `GlobalMap` argument `GMap`. The success (failure) of this operation is signified by the zero (nonzero) returned value of the `INTEGER` flag `stat`.

INTERFACE:

```
subroutine GM_scatter_(iC, oC, GMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalMap,   only : GlobalMap
use m_Accumulator, only : Accumulator
use m_Accumulator, only : Accumulator_initv => init
```

```
use m_Accumulator, only : Accumulator_initialized => initialized
use m_AttrVect, only : AttrVect_clean => clean
use m_AttrVectComms, only : AttrVect_scatter => scatter

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator), intent(in)  :: iC
type(GlobalMap),   intent(in)  :: GMap
integer,           intent(in)  :: root
integer,           intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(Accumulator), intent(out) :: oC
integer, optional, intent(out) :: stat
```

REVISION HISTORY:

```
14Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
31Oct00 - Jay Larson <larson@mcs.anl.gov> - moved from the module
              m_Accumulator to m_AccumulatorComms
15Jan01 - Jay Larson <larson@mcs.anl.gov> - renamed GM_scatter_.
     10May01 - Jay Larson <larson@mcs.anl.gov> - revamped code to fit
               MPI-like comms model, and cleaned up prologue.
     09Aug01 - E.T. Ong <eong@mcs.anl.gov> - 2nd prototype. Used the
               initialized_, Accumulator init_, and bcastp_ routines.
```

14.2.4 GSM_scatter_ - Scatter an Accumulator using a GlobalSegMap

This routine takes the input `Accumulator` argument `iC` (valid only on the `root`), and scatters it to the distributed `Accumulator` argument `oC` on the processes associated with the communicator handle `comm`. The decompositon used to scatter the data is contained in the input `GlobalSegMap` argument `GSMap`. The success (failure) of this operation is signified by the zero (nonzero) returned value of the `INTEGER` flag `stat`.

INTERFACE:

```
subroutine GSM_scatter_(iC, oC, GSMap, root, comm, stat)
```

USES:

```
use m_stdio
use m_die
use m_mpif90

use m_GlobalSegMap, only : GlobalSegMap
use m_Accumulator, only : Accumulator
use m_Accumulator, only : Accumulator_initv => init
use m_Accumulator, only : Accumulator_initialized => initialized
use m_AttrVect, only : AttrVect_clean => clean
use m_AttrVectComms, only : AttrVect_scatter => scatter

implicit none
```

INPUT PARAMETERS:

```
type(Accumulator),  intent(in)  :: iC
type(GlobalSegMap), intent(in)  :: GSMap
integer,            intent(in)  :: root
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(Accumulator),  intent(out) :: oC
integer, optional,  intent(out) :: stat
```

REVISION HISTORY:

```
15Jan01 - Jay Larson <larson@mcs.anl.gov> - API specification.
10May01 - Jay Larson <larson@mcs.anl.gov> - Initial code/prologue
     09Aug01 - E.T. Ong <eong@mcs.anl.gov> 2nd prototype. Used the
               initialized and accumulator init routines.
```

14.2.5 bcast_ - Broadcast an Accumulator

This routine takes the input `Accumulator` argument `aC` (on input valid only on the `root`), and broadcasts it to all the processes associated with the communicator handle `comm`. The success (failure) of this operation is signified by the zero (nonzero) returned value of the `INTEGER` flag `stat`.

INTERFACE:

```
subroutine bcast_(aC, root, comm, stat)
```

USES:

```
use m_die
use m_mpif90
use m_AttrVectComms, only : AttrVect_bcast => bcast

use m_Accumulator, only : Accumulator
use m_Accumulator, only : Accumulator_initialized => initialized

implicit none
```

INPUT PARAMETERS:

```
integer,intent(in) :: root
integer,intent(in) :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(Accumulator), intent(inout) :: aC ! (IN) on root, (OUT) elsewhere
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
14Sep00 - Jay Larson <larson@mcs.anl.gov> - initial prototype
31Oct00 - Jay Larson <larson@mcs.anl.gov> - moved from the module
               m_Accumulator to m_AccumulatorComms
     09May01 - Jay Larson <larson@mcs.anl.gov> - cleaned up prologue
     09Aug01 - E.T. Ong <eong@mcs.anl.gov> - 2nd prototype. Made use of
               bcastp_ routine. Also more argument checks.
```

14.2.6 bcastp_ - Broadcast an Accumulator (but Not its Registers)

This routine broadcasts all components of the accumulator aC except for aCto be used by accumulator scatter and gather routines.

INTERFACE:

```
subroutine bcastp_(aC, root, comm, stat)
```

USES:

```
use m_die
use m_mpif90
use m_AttrVectComms, only : AttrVect_bcast => bcast
use m_Accumulator, only : Accumulator
use m_Accumulator, only : Accumulator_initp => initp
use m_Accumulator, only : Accumulator_nIAttr => nIAttr
use m_Accumulator, only : Accumulator_nRAttr => nRAttr

implicit none
```

INPUT PARAMETERS:

```
integer,intent(in) :: root
integer,intent(in) :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(Accumulator), intent(inout) :: aC ! (IN) on root, (OUT) elsewhere
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
09Aug01 - E.T. Ong <eong@mcs.anl.gov> - initial prototype
```

15 Global To Local Index Translation

15.1 Module m_GlobalToLocal - Global to Local Index Translation (Source File: m_GlobalToLocal.F90)

This module contains routines for translating global array indices into their local counterparts (that is, the indices into the local data structure holding a given process' chunk of a distributed array). The MCT domain decomposition descriptors `GlobalMap` and `GlobalSegMap` are both supported. Indices can be translated one-at-a-time using the `GlobalToLocalIndex` routine or many at once using the `GlobalToLocalIndices` routine.
This module also provides facilities for setting the local row and column indices for a `SparseMatrix` through the `GlobalToLocalMatrix` routines.

INTERFACE:

```
module m_GlobalToLocal
```

USES:

```
No external modules are used in the declaration section of this module.

    implicit none

    private   ! except
```

PUBLIC MEMBER FUNCTIONS:

```
    public :: GlobalToLocalIndex   ! Translate Global to Local index
                                   ! (i.e. recover local index for a
                                   ! point from its global index).

    public :: GlobalToLocalIndices ! Translate Global to Local indices
                                   ! (i.e. recover local starts/lengths
                                   ! of distributed data segments).

    public :: GlobalToLocalMatrix  ! Re-indexing of row or column
                                   ! indices for a SparseMatrix

  interface GlobalToLocalIndices ; module procedure   &
     GlobalSegMapToIndices_,  &   ! local arrays of starts/lengths
     GlobalSegMapToNavigator_, &  ! return local indices as Navigator
     GlobalSegMapToIndexArr_
  end interface

  interface GlobalToLocalIndex ; module procedure &
     GlobalSegMapToIndex_, &
     GlobalMapToIndex_
  end interface

  interface GlobalToLocalMatrix ; module procedure &
     GlobalSegMapToLocalMatrix_
  end interface
```

SEE ALSO:

```
The MCT modules {\tt m\_GlobalMap} and {m\_GlobalSegMap} for more
information regarding MCT's domain decomposition descriptors.
The MCT module {\tt m\_SparseMatrix} for more information regarding
the {\tt SparseMatrix} datatype.
```

REVISION HISTORY:

```
2Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial prototype
```

15.1.1 GlobalSegMapToIndices_ - Return _local_ indices in arrays.

`GlobalSegMapToIndices_()` takes a user-supplied `GlobalSegMap` data type `GSMap`, which desribes a decomposition on the input MPI communicator corresponding to the Fortran `INTEGER` handle `comm` to translate the global directory of segment locations into local indices for referencing the on-pe storage of the mapped distributed data.
N.B.: This routine returns two allocated arrays—`start(:)` and `length(:)`—which must be deallocated once the user no longer needs them. Failure to do this will create a memory leak.

INTERFACE:

```
subroutine GlobalSegMapToIndices_(GSMap, comm, start, length)
```

USES:

```
use m_mpif90
use m_die,          only : MP_perr_die, die, warn
use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg
use m_GlobalSegMap, only : GlobalSegMap_nlseg => nlseg

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),   intent(in) :: GSMap ! Output GlobalSegMap
integer,              intent(in) :: comm  ! communicator handle
```

OUTPUT PARAMETERS:

```
integer,dimension(:), pointer :: start  ! local segment start indices
integer,dimension(:), pointer :: length ! local segment sizes
```

REVISION HISTORY:

```
2Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

15.1.2 GlobalSegMapToIndex_ - Global to Local Index Translation

This `INTEGER` query function takes a user-supplied `GlobalSegMap` data type `GSMap`, which desribes a decomposition on the input MPI communicator corresponding to the Fortran `INTEGER` handle `comm`, and the input global index value `i_g`, and returns a positive local index value if the datum `i_g`. If the datum `i_g` is not stored on the local process ID, a value of `-1` is returned.

INTERFACE:

```
integer function GlobalSegMapToIndex_(GSMap, i_g, comm)
```

USES:

```
use m_mpif90
use m_die,          only : MP_perr_die, die, warn
use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg
use m_GlobalSegMap, only : GlobalSegMap_nlseg => nlseg

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)  :: GSMap ! Output GlobalSegMap
integer,            intent(in)  :: i_g   ! global index
integer,            intent(in)  :: comm  ! communicator handle
```

REVISION HISTORY:

```
2Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

15.1.3 GlobalSegMapToIndexArr_ - Global to Local Index Array Translation

Given a `GlobalSegMap` data type `GSMap` and MPI communicator corresponding to the Fortran `INTEGER` handle `comm`, convert an array of global index values `i_global()` to an array of local index values `i_local()`. If the datum `i_global(j)` is not stored on the local process ID, then `i_local(j)` will be set to `-1`/

INTERFACE:

```
subroutine GlobalSegMapToIndexArr_(GSMap, i_global, i_local, nindex, comm)
```

USES:

```
use m_stdio
use m_mpif90
use m_die,          only : MP_perr_die, die, warn
use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg
use m_GlobalSegMap, only : GlobalSegMap_nlseg => nlseg

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)  :: GSMap ! Output GlobalSegMap
integer,            intent(in)  :: i_global(:)   ! global index
integer,            intent(out) :: i_local(:)    ! local index
integer,            intent(in)  :: nindex          ! size of i_global()
integer,            intent(in)  :: comm  ! communicator handle
```

REVISION HISTORY:

```
12-apr-2006   R. Loy <rloy@mcs.anl.gov> - initial version
```

15.1.4 GlobalMapToIndex_ - Global to Local Index Translation

This INTEGER query function takes as its input a user-supplied GlobalMap data type GMap, which desribes a decomposition on the input MPI communicator corresponding to the Fortran INTEGER handle comm, and the input global index value i_g, and returns a positive local index value if the datum i_g. If the datum i_g is not stored on the local process ID, a value of -1 is returned.

INTERFACE:

```
integer function GlobalMapToIndex_(GMap, i_g, comm)
```

USES:

```
use m_mpif90
use m_die,          only : MP_perr_die, die, warn
use m_GlobalMap,    only : GlobalMap

implicit none
```

INPUT PARAMETERS:

```
type(GlobalMap), intent(in)  :: GMap  ! Input GlobalMap
integer,         intent(in)  :: i_g   ! global index
integer,         intent(in)  :: comm  ! communicator handle
```

REVISION HISTORY:

2Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version

15.1.5 GlobalSegMapToNavigator_ - Return Navigator to Local Segments

This routine takes as its input takes a user-supplied GlobalSegMap data type GSMap, which desribes a decomposition on the input MPI communicator corresponding to the Fortran INTEGER handle comm, and returns the local segment start index and length information for referencing the on-pe storage of the mapped distributed data. These data are returned in the form of the output Navigator argument Nav.
N.B.: This routine returns a Navigator variable Nav, which must be deallocated once the user no longer needs it. Failure to do this will create a memory leak.

INTERFACE:

```
subroutine GlobalSegMapToNavigator_(GSMap, comm, oNav)
```

USES:

```
use m_mpif90
use m_die,          only : MP_perr_die, die, warn
use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg
use m_GlobalSegMap, only : GlobalSegMap_nlseg => nlseg
use m_Navigator, only    : Navigator
use m_Navigator, only    : Navigator_init => init

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)  :: GSMap ! Input GlobalSegMap
integer,            intent(in)  :: comm  ! communicator handle
```

OUTPUT PARAMETERS:

```
type(Navigator),    intent(out) :: oNav   ! Output Navigator
```

REVISION HISTORY:

```
2Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```

15.1.6 GlobalSegMapToLocalMatrix_ - Set Local SparseMatrix Indices

This routine takes as its input a user-supplied `GlobalSegMap` domain decomposition `GSMap`, which describes the decomposition of either the rows or columns of the input/output `SparseMatrix` argument `sMat` on the communicator associated with the `INTEGER` handle `comm`, and to translate the global row or column indices of `sMat` into their local counterparts. The choice of either row or column is governed by the value of the input `CHARACTER` argument `RCFlag`. One sets this variable to either `'ROW'` or `'row'` to specify row re-indexing (which are stored in `sMat` and retrieved by indexing the attribute `lrow`), and `'COLUMN'` or `'column'` to specify column re-indexing (which are stored in `sMat` and retrieved by indexing the `SparseMatrix` attribute `lcol`).

INTERFACE:

```
subroutine GlobalSegMapToLocalMatrix_(sMat, GSMap, RCFlag, comm)
```

USES:

```
use m_stdio
use m_die,          only : die

use m_SparseMatrix, only : SparseMatrix
use m_SparseMatrix, only : SparseMatrix_indexIA => indexIA
use m_SparseMatrix, only : SparseMatrix_lsize => lsize

use m_GlobalSegMap, only : GlobalSegMap


implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap), intent(in)    :: GSMap  ! Input GlobalSegMap
character(len=*),   intent(in)    :: RCFlag ! 'row' or 'column'
integer,            intent(in)    :: comm   ! communicator handle
```

INPUT/OUTPUT PARAMETERS:

```
type(SparseMatrix), intent(inout) :: sMat
```

SEE ALSO:

```
The MCT module m_SparseMatrix for more information about the
SparseMatrix type and its storage of global and local row-and
column indices.
```

REVISION HISTORY:

```
    3May01 - J.W. Larson <larson@mcs.anl.gov> - initial version, which
             is _extremely_ slow, but safe.  This must be re-examined
             later.
```

16 Convert From Global Map To Global Segment Map

16.1 Module m_ConvertMaps - Conversion Between MCT Domain Decomposition Descriptors (Source File: m_ConvertMaps.F90)

This module contains routines to convert between the `GlobalMap` and `GlobalSegMap` types. Since the `GlobalMap` is a 1-D decomposition with one contiguous segment per process, it is always possible to create a `GlobalSegMap` containing the same decomposition information. In the unusual case that a `GlobalSegMap` contains *at most* one segment per process, and no two segments overlap, it is possible to create a `GlobalMap` describing the same decomposition.

INTERFACE:

```
module m_ConvertMaps
```

USES:

```
      use m_GlobalMap,    only : GlobalMap
      use m_GlobalSegMap, only : GlobalSegMap

      implicit none

      private   ! except
```

PUBLIC MEMBER FUNCTIONS:

```
      public :: GlobalMapToGlobalSegMap
      public :: GlobalSegMapToGlobalMap


    interface GlobalMapToGlobalSegMap ; module procedure &
        GlobalMapToGlobalSegMap_
    end interface
    interface GlobalSegMapToGlobalMap ; module procedure &
        GlobalSegMapToGlobalMap_
    end interface
```

REVISION HISTORY:

```
   12Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial module
```

16.1.1 GlobalMapToGlobalSegMap_ - Convert GlobalMap to GlobalSegMap

This routine takes an input `GlobalMap` argument `GMap`, and converts its decomposition information into the output `GlobalSegMap` argument `GSMap`. Since the `GlobalMap` is a very special case of the more general `GlobalSegMap` decomposition, this conversion is always possible.
The motivation of this routine is the fact that the majority of the APIs for MCT services require the user to supply a `GlobalSegMap` as a domain decomposition descriptor argument. This routine is the means by which the user can enjoy the convenience and simplicity of the `GlobalMap` datatype (where it is appropriate), but still access all of the MCT's functionality.
N.B.: This routine creates an allocated structure `GSMap`. The user is responsible for deleting this structure using the `clean()` method for the `GlobalSegMap` when `GSMap` is no longer needed. Failure to do so will create a memory leak.

INTERFACE:

```
  subroutine GlobalMapToGlobalSegMap_(GMap, GSMap)
```

USES:

```
use m_stdio, only : stderr
use m_die,   only : MP_perr_die, die, warn

use m_GlobalMap,    only : GlobalMap

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_init => init

use m_MCTWorld, only : ThisMCTWorld
use m_MCTWorld, only : MCTWorld_ComponentNumProcs => ComponentNumProcs

implicit none
```

INPUT PARAMETERS:

```
type(GlobalMap),    intent(in)  :: GMap
```

OUTPUT PARAMETERS:

```
type(GlobalSegMap), intent(out) :: GSMap
```

REVISION HISTORY:

```
12Feb01 - J.W. Larson <larson@mcs.anl.gov> - Prototype code.
24Feb01 - J.W. Larson <larson@mcs.anl.gov> - Finished code.
```

16.1.2 GlobalSegMapToGlobalMap_ - Convert GlobalSegMap to GlobalMap

This routine takes an input `GlobalSegMap` argument `GSMap`, and examines it to determine whether or not it may be expressed in `GlobalMap` form. A `GlobalSegMap` can be converted to a `GlobalMap` if and only if:

1. Each process on the communicator covered by the `GlobalSegMap` contains *at most one* segment;
2. The `GlobalSegMap` is *not* haloed (that is, none of the segments overlap); and
3. The start indices of the segments are in the same order as their respective process ID numbers.

If these conditions are satisfied, `GlobalSegMapToGlobalMap_()` creates an output `GlobalMap` argument `GMap` describing the same decomposition as `GSMap`. If these conditions are not satisfied, map conversion can not occur, and `GlobalSegMapToGlobalMap_()` has one of two outcomes:

1. If the optional output `INTEGER` argument `status` is provided, `GlobalSegMapToGlobalMap_()` returns without creating `GMap`, and returns a non-zero value for `status`.
2. If the optional output `INTEGER` argument `status` is not provided, execution will terminate with an error message.

The optional output `INTEGER` argument `status`, if provided will be returned from `GlobalSegMapToGlobalMap_()` with a value explained by the table below:
N.B.: This routine creates an allocated structure `GMap`. The user is responsible for deleting this structure using the `clean()` method for the `GlobalMap` when `GMap` is no longer needed. Failure to do so will create a memory leak.

INTERFACE:

```
subroutine GlobalSegMapToGlobalMap_(GSMap, GMap, status)
```

Value of status	Significance
0	Map Conversion Successful
1	Unsuccessful–more than one segment per process, or a negative numer of segments (ERROR)
2	Unsuccessful–GSMap haloed
3	Unsuccessful–GSMap segments out-of-order with respect to resident process ID ranks

USES:

```
use m_stdio, only : stderr
use m_die,   only : MP_perr_die, die

use m_SortingTools , only : IndexSet
use m_SortingTools , only : IndexSort
use m_SortingTools , only : Permute

use m_MCTWorld, only : MCTWorld
use m_MCTWorld, only : ThisMCTWorld
use m_MCTWorld, only : ComponentNumProcs

use m_GlobalSegMap, only : GlobalSegMap
use m_GlobalSegMap, only : GlobalSegMap_comp_id => comp_id
use m_GlobalSegMap, only : GlobalSegMap_gsize => gsize
use m_GlobalSegMap, only : GlobalSegMap_haloed => haloed
use m_GlobalSegMap, only : GlobalSegMap_ngseg => ngseg
use m_GlobalSegMap, only : GlobalSegMap_nlseg => nlseg
use m_GlobalSegMap, only : GlobalSegMap_active_pes => active_pes

use m_GlobalMap,    only : GlobalMap

implicit none
```

INPUT PARAMETERS:

```
type(GlobalSegMap),           intent(in)  :: GSMap
```

OUTPUT PARAMETERS:

```
type(GlobalMap),              intent(out) :: GMap
integer,            optional, intent(out) :: status
```

REVISION HISTORY:

```
12Feb01 - J.W. Larson <larson@mcs.anl.gov> - API / first prototype.
21Sep02 - J.W. Larson <larson@mcs.anl.gov> - Near-complete Implementation,
          still, do not call!
```

Part III
Documentation of MPEU Datatypes Used to Define MCT Datatypes

17 The String Datatype

17.1 Module m_String - The String Datatype (Source File: m_String.F90)

The `String` datatype is an encapsulated pointer to a one-dimensional array of single characters. This allows one to define variable-length strings, and arrays of variable-length strings.

INTERFACE:

```
  module m_String
```

USES:

```
   No external modules are used in the declaration section of this module.

       implicit none

       private ! except
```

PUBLIC TYPES:

```
       public :: String ! The class data structure

     Type String
 #ifdef SEQUENCE
       sequence
 #endif
       character(len=1),dimension(:),pointer :: c
     End Type String
```

PUBLIC MEMBER FUNCTIONS:

```
       public :: toChar
       public :: char ! convert to a CHARACTER(*)

       public :: String_init
       public :: init ! set a CHARACTER(*) type to a String

       public :: String_clean
       public :: clean ! Deallocate memory occupied by  a String

       public :: String_len
       public :: len ! length of a String

       public :: String_bcast
       public :: bcast ! Broadcast a String

       public :: String_mci      ! Track memory used to store a String
       public :: String_mco

       public :: ptr_chars       ! Assign a pointer to a String's
```

```
                                ! character buffer

 interface char;  module procedure &
str2ch0_,&
ch12ch0_
 end interface

 interface toChar;  module procedure &
str2ch0_,&
ch12ch0_
 end interface

 interface String_init;  module procedure &
initc_,&
initc1_,&
inits_
 end interface

 interface init;  module procedure &
initc_,&
initc1_,&
inits_
 end interface

 interface String_clean; module procedure clean_; end interface
 interface clean; module procedure clean_; end interface
 interface String_len; module procedure len_; end interface
 interface len; module procedure len_; end interface
 interface String_bcast; module procedure bcast_; end interface
 interface bcast; module procedure bcast_; end interface

 interface String_mci; module procedure &
   mci0_,&
   mci1_,&
   mci2_,&
   mci3_
 end interface

 interface String_mco; module procedure &
   mco0_,&
   mco1_,&
   mco2_,&
   mco3_
 end interface

 interface ptr_chars; module procedure &
   ptr_chars_
 end interface
```

REVISION HISTORY:

```
  22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

17.1.1 str2ch0_ - Convert a String to a CHARACTER

This function returns the contents of the character buffer of the input `String` argument `str` as a `CHARCTER` suitable for printing.

INTERFACE:

```
function str2ch0_(str)
```

USES:

```
No external modules are used by this function.

   implicit none
```

INPUT PARAMETERS:

```
   type(String),              intent(in) :: str
```

OUTPUT PARAMETERS:

```
   character(len=size(str%c,1))            :: str2ch0_
```

REVISION HISTORY:

```
23Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

17.1.2 ch12ch0_ - Convert a CHARACTER(:) to a CHARACTER(*)

This function takes an input one-dimensional array of single characters and returns a single character string.

INTERFACE:

```
function ch12ch0_(ch1)
```

USES:

```
No external modules are used by this function.

    implicit none
```

INPUT PARAMETERS:

```
   character(len=1), dimension(:), intent(in) :: ch1
```

OUTPUT PARAMETERS:

```
   character(len=size(ch1,1))                  :: ch12ch0_
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

17.1.3 initc_ - Create a String using a CHARACTER

This routine takes an input scalar `CHARACTER` argument `chr`, and uses it to create the output `String` argument `str`.

INTERFACE:

```
subroutine initc_(str, chr)
```

USES:

```
use m_die, only : die,perr
use m_mall,only : mall_mci,mall_ison

implicit none
```

INPUT PARAMETERS:

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

OUTPUT PARAMETERS:

```
type(String),     intent(out) :: str
```

REVISION HISTORY:

```
23Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

17.1.4 initc1_ - Create a String using a CHARACTER array

This routine takes an input `CHARACTER(:)` argument `chr`, and uses it to create the output `String` argument `str`.

INTERFACE:

```
subroutine initc1_(str, chr)
```

USES:

```
use m_die, only : die,perr
use m_mall,only : mall_mci,mall_ison

implicit none
```

INPUT PARAMETERS:

```
character,     dimension(:), intent(in)  :: chr
```

OUTPUT PARAMETERS:

```
type(String),                intent(out) :: str
```

REVISION HISTORY:

```
2Aug02 - J. Larson <larson@mcs.anl.gov> - initial prototype
```

17.1.5 inits_ - Initialization of a String from another String

This routine takes an input `String` argument `iStr` and creates an output `String` argument `oStr`. In other words, it copies `iStr` to `oStr`.

INTERFACE:

```
subroutine inits_(oStr, iStr)
```

USES:

```
use m_die, only : die
use m_mall,only : mall_mci,mall_ison

implicit none
```

INPUT PARAMETERS:

```
type(String),  intent(in)  :: iStr
```

OUTPUT PARAMETERS:

```
type(String),  intent(out) :: oStr
```

REVISION HISTORY:

```
 07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
- initial prototype/prolog/code
```

17.1.6 clean_ - Deallocate Memory Occupied by a String

This routine deallocates memory associated with the input/output `String` argument `str`. This amounts to deallocating `str%c`.

INTERFACE:

```
subroutine clean_(str)
```

USES:

```
use m_die, only : die,perr
use m_mall,only : mall_mco,mall_ison

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(String), intent(inout) :: str
```

REVISION HISTORY:

```
23Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

17.1.7 bcast_ - MPI Broadcast of a rank-0 String

This routine performs an MPI broadcast of the input/output `String` argument `Str` on a communicator associated with the Fortran integer handle `comm`. The broadcast originates from the process with rank given by `root` on `comm`. The `String` argument `Str` is on entry valid only on the `root` process, and is valid on exit on all processes on the communicator `comm`. The success (failure) is signified by a zero (non-zero) value of the optional `INTEGER` output argument `stat`.

INTERFACE:

```
subroutine bcast_(Str, root, comm, stat)
```

USES:

```
use m_mpif90
use m_die, only : perr,die
use m_mall,only : mall_mci,mall_ison

implicit none
```

INPUT PARAMETERS:

```
integer,           intent(in)    :: root
integer,           intent(in)    :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(String),      intent(inout) :: Str ! (IN) on the root,
                                        ! (OUT) elsewhere
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
27Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

17.1.8 mci0_ - checking in a String scalar

INTERFACE:

```
subroutine mci0_(marg,thread)
```

USES:

```
use m_mall, only : mall_ci

implicit none
```

INPUT PARAMETERS:

```
type(String),     intent(in) :: marg
character(len=*), intent(in) :: thread
```

REVISION HISTORY:

```
07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
- initial prototype/prolog/code
```

17.1.9 mco0_ - checking out a String scalar

INTERFACE:

```
subroutine mco0_(marg,thread)
```

USES:

```
      use m_mall, only : mall_co

      implicit none

      type(String),    intent(in) :: marg
      character(len=*),intent(in) :: thread
```

REVISION HISTORY:

```
   07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
  - initial prototype/prolog/code
```

17.1.10 mci1_ - checking in a String scalar

INTERFACE:

```
subroutine mci1_(marg,thread)
```

USES:

```
      use m_mall, only : mall_ci

      implicit none
```

INPUT PARAMETERS:

```
      type(String),     dimension(:), intent(in) :: marg
      character(len=*),               intent(in) :: thread
```

REVISION HISTORY:

```
   07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
  - initial prototype/prolog/code
```

17.1.11 mco1_ - checking out a String scalar

INTERFACE:

```
subroutine mco1_(marg,thread)
```

USES:

```
      use m_mall, only : mall_co

      implicit none
```

INPUT PARAMETERS:

```
      type(String),     dimension(:), intent(in) :: marg
      character(len=*),               intent(in) :: thread
```

REVISION HISTORY:

```
    07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
   - initial prototype/prolog/code
```

17.1.12 mci2_ - checking in a String scalar

INTERFACE:

```
  subroutine mci2_(marg, thread)
```

USES:

```
      use m_mall, only : mall_ci

      implicit none
```

INPUT PARAMETERS:

```
      type(String),     dimension(:,:), intent(in) :: marg
      character(len=*),                 intent(in) :: thread
```

REVISION HISTORY:

```
    07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
   - initial prototype/prolog/code
```

17.1.13 mco2_ - checking out a String scalar

INTERFACE:

```
  subroutine mco2_(marg,thread)
```

USES:

```
      use m_mall, only : mall_co

      implicit none
```

INPUT PARAMETERS:

```
      type(String),     dimension(:,:), intent(in) :: marg
      character(len=*),                 intent(in) :: thread
```

REVISION HISTORY:

```
    07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
   - initial prototype/prolog/code
```

17.1.14 mci3_ - checking in a String scalar

INTERFACE:

```
subroutine mci3_(marg,thread)
```

USES:

```
use m_mall, only : mall_ci

implicit none
```

INPUT PARAMETERS:

```
type(String),     dimension(:,:,:), intent(in) :: marg
character(len=*),                   intent(in) :: thread
```

REVISION HISTORY:

```
 07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
- initial prototype/prolog/code
```

17.1.15 mco3_ - checking out a String scalar

INTERFACE:

```
subroutine mco3_(marg,thread)
```

USES:

```
use m_mall, only : mall_co

implicit none
```

INPUT PARAMETERS:

```
type(String),     dimension(:,:,:), intent(in) :: marg
character(len=*),                   intent(in) :: thread
```

REVISION HISTORY:

```
 07Feb00 - Jing Guo <guo@dao.gsfc.nasa.gov>
- initial prototype/prolog/code
```

17.1.16 len_ = len of a String

INTERFACE:

```
integer function len_(str)
```

USES:

```
No external modules are used by this function.

    implicit none
```

INPUT PARAMETERS:

```
    type(String),intent(in) :: str
```

REVISION HISTORY:

```
 10Apr00 - Jing Guo <guo@dao.gsfc.nasa.gov>
- initial prototype/prolog/code
```

17.1.17 ptr_chars_ - direct

This pointer-valued function provides a direct interface to the character buffer in the input `String` argument `str`. That is, `ptr_chars_ => str%c`.

INTERFACE:

```
function ptr_chars_(str)
```

USES:

```
No external modules are used by this function.

    implicit none
```

INPUT PARAMETERS:

```
    type(String),                   intent(in) :: str
```

OUTPUT PARAMETERS:

```
    character(len=1), dimension(:), pointer    :: ptr_chars_
```

REVISION HISTORY:

```
 10Apr00 - Jing Guo <guo@dao.gsfc.nasa.gov>
- initial prototype/prolog/code
```

18 The List Datatype

18.1 Module m_List - A List Manager (Source File: m_List.F90)

A *List* is a character buffer comprising substrings called *items* separated by colons, combined with indexing information describing (1) the starting point in the character buffer of each substring, and (2) the length of each substring. The only constraints on the valid list items are (1) the value of an item does not contain the ":" delimitter, and (2) leading and trailing blanks are stripped from any character string presented to define a list item (although any imbeded blanks are retained).
Example: Suppose we wish to define a List containing the items `'latitude'`, `'longitude'`, and `'pressure'`. The character buffer of the List containing these items will be the 27-character string

```
'latitude:longitude:pressure'
```

and the indexing information is summarized in the table below.

Item	Starting Point in Buffer	Length
latitude	1	8
longitude	9	9
pressure	20	8

One final note: All operations for the `List` datatype are **case sensitive**.

INTERFACE:

```
module m_List
```

USES:

```
No other Fortran modules are used.

    implicit none

    private ! except
```

PUBLIC TYPES:

```
    public :: List ! The class data structure

    Type List
#ifdef SEQUENCE
     sequence
#endif
  character(len=1),dimension(:),pointer :: bf
  integer,       dimension(:,:),pointer :: lc
    End Type List
```

PUBLIC MEMBER FUNCTIONS:

```
    public :: init
    public :: clean
    public :: nullify
    public :: index
    public :: get_indices
    public :: test_indices
    public :: nitem
```

```
    public :: get
    public :: identical
    public :: assignment(=)
    public :: allocated
    public :: copy
    public :: exportToChar
    public :: exportToString
    public :: CharBufferSize
    public :: append
    public :: concatenate
    public :: bcast
    public :: send
    public :: recv
    public :: GetSharedListIndices

interface init ; module procedure &
    init_,&
    initStr_,&
    initstr1_
end interface
interface clean; module procedure clean_; end interface
interface nullify; module procedure nullify_; end interface
interface index; module procedure &
    index_,     &
    indexStr_
end interface
interface get_indices; module procedure get_indices_; end interface
interface test_indices; module procedure test_indices_; end interface
interface nitem; module procedure nitem_; end interface
interface get  ; module procedure &
    get_,&
    getall_,&
    getrange_
end interface
interface identical; module procedure identical_; end interface
interface assignment(=)
  module procedure copy_
end interface
interface allocated ; module procedure &
     allocated_
end interface
interface copy ; module procedure copy_ ;  end interface
interface exportToChar ; module procedure &
     exportToChar_
end interface
interface exportToString ; module procedure &
     exportToString_
end interface
interface CharBufferSize ; module procedure &
    CharBufferSize_
end interface
interface append ; module procedure append_ ; end interface
interface concatenate ; module procedure concatenate_ ; end interface
interface bcast; module procedure bcast_; end interface
interface send; module procedure send_; end interface
interface recv; module procedure recv_; end interface
interface GetSharedListIndices; module procedure &
    GetSharedListIndices_
end interface
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
16May01 - J. Larson <larson@mcs.anl.gov> - Several changes / fixes:
          public interface for copy_(), corrected version of copy_(),
          corrected version of bcast_().
15Oct01 - J. Larson <larson@mcs.anl.gov> - Added the LOGICAL
          function identical_().
14Dec01 - J. Larson <larson@mcs.anl.gov> - Added the LOGICAL
          function allocated_().
13Feb02 - J. Larson <larson@mcs.anl.gov> - Added the List query
          functions exportToChar() and CharBufferLength().
13Jun02-  R.L. Jacob <jacob@mcs.anl.gov> - Move GetSharedListIndices
          from mct to this module.
```

18.1.1 init_ - Initialize a List from a CHARACTER String

A list is a string in the form of "`Larry:Moe:Curly`", or "`lat:lon:lev`", combined with substring location and length information. Through the initialization call, the items delimited by "`:`" are stored as an array of sub- strings of a long string, accessible through an array of substring indices. The only constraints now on the valid list entries are, (1) the value of an entry does not contain "`:`", and (2) The leading and the trailing blanks are insignificant, although any imbeded blanks are. For example,

```
call init_(aList, 'batman  :SUPERMAN:Green Lantern:  Aquaman')
```

will result in `aList` having four items: 'batman', 'SUPERMAN', 'Green Lantern', and 'Aquaman'. That is

```
aList%bf =  'batman:SUPERMAN:Green Lantern:Aquaman'
```

INTERFACE:

```
subroutine init_(aList,Values)
```

USES:

```
use m_die,only : die
use m_mall,only : mall_mci,mall_ison

implicit none
```

INPUT PARAMETERS:

```
character(len=*),intent(in) :: Values ! ":" delimited names
```

OUTPUT PARAMETERS:

```
type(List),intent(out)   :: aList  ! an indexed string values
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

18.1.2 initStr_ - Initialize a List Using the String Type

This routine initializes a `List` datatype given an input `String` datatype (see `m_String` for more information regarding the `String` type). The contents of the input `String` argument `pstr` must adhere to the restrictions stated for character input stated in the prologue of the routine `init_()` in this module.

INTERFACE:

```
subroutine initStr_(aList, pstr)
```

USES:

```
use m_String, only : String,toChar

implicit none
```

INPUT PARAMETERS:

```
type(String),intent(in)   :: pstr
```

OUTPUT PARAMETERS:

```
type(List),intent(out)   :: aList  ! an indexed string values
```

REVISION HISTORY:

```
23Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

18.1.3 initStr1_ - Initialize a List Using an Array of Strings

This routine initializes a `List` datatype given as input array of `String` datatypes (see `m_String` for more information regarding the `String` type). The contents of each `String` element of the input array `strs` must adhere to the restrictions stated for character input stated in the prologue of the routine `init_()` in this module. Specifically, no element in `strs` may contain the colon `:` delimiter, and any leading or trailing blanks will be stripped (though embedded blank spaces will be retained). For example, consider an invocation of `initStr1_()` where the array `strs(:)` contains four entries: `strs(1)='John'`, `strs(2)=' Paul'`, `strs(3)='George '`, and `strs(4)=' Ringo'`. The resulting `List` output `aList` will have

```
aList%bf =  'John:Paul:George:Ringo'
```

INTERFACE:

```
subroutine initStr1_(aList, strs)
```

USES:

```
use m_String, only : String,toChar
use m_String, only : len
use m_String, only : ptr_chars
use m_die,only : die

implicit none
```

INPUT PARAMETERS:

```
type(String),dimension(:),intent(in)   :: strs
```

OUTPUT PARAMETERS:

```
type(List),intent(out)   :: aList  ! an indexed string values
```

REVISION HISTORY:

```
23Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

18.1.4 clean_ - Deallocate Memory Used by a List

This routine deallocates the allocated memory components of the input/output `List` argument `aList`. Specifically, it deallocates `aList%bf` and `aList%lc`. If the optional output `INTEGER` arguemnt `stat` is supplied, no warning will be printed if the Fortran intrinsic `deallocate()` returns with an error condition.

INTERFACE:

```
subroutine clean_(aList, stat)
```

USES:

```
use m_die,  only : warn
use m_mall, only : mall_mco,mall_ison

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(List),        intent(inout) :: aList
```

OUTPUT PARAMETERS:

```
integer, optional, intent(out)   :: stat
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
 1Mar02 - E.T. Ong <eong@mcs.anl.gov> - added stat argument and
          removed die to prevent crashes.
```

18.1.5 nullify_ - Nullify Pointers in a List

In Fortran 90, pointers may have three states: (1) `ASSOCIATED`, that is the pointer is pointing at a target, (2) `UNASSOCIATED`, and (3) `UNINITIALIZED`. On some platforms, the Fortran intrinsic function `associated()` will view uninitialized pointers as `UNASSOCIATED` by default. This is not always the case. It is good programming practice to nullify pointers if they are not to be used. This routine nullifies the pointers present in the `List` datatype.

INTERFACE:

```
subroutine nullify_(aList)
```

USES:

```
use m_die,only : die

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(List),intent(inout) :: aList
```

REVISION HISTORY:

```
18Jun01 - J.W. Larson - <larson@mcs.anl.gov> - initial version
```

18.1.6 nitem_ - Return the Number of Items in a List

This function enumerates the number of items in the input `List` argument `aList`. For example, suppose

```
aList%bf = 'John:Paul:George:Ringo'
```

Then,

$$\texttt{nitem\_}(\texttt{aList}) = 4.$$

INTERFACE:

```
integer function nitem_(aList)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(List),intent(in) :: aList
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
10Oct01 - J.W. Larson <larson@mcs.anl.gov> - modified routine to
          check pointers aList%bf and aList%lc using  the f90
          intrinsic ASSOCIATED before proceeding with the item
          count.  If these pointers are UNASSOCIATED, an item
          count of zero is returned.
```

18.1.7 index_ - Return Rank in a List of a Given Item (CHARACTER)

This function returns the rank of an item (defined by the `CHARACTER` argument `item`) in the input `List` argument `aList`. If `item` is not present in `aList`, then zero is returned. For example, suppose

```
aList%bf = 'Bob:Carol:Ted:Alice'
```

Then, index_(aList,'Ted') = 3, index_(aList,'Carol') = 2, and index_(aList,'TheDude') = 0.

INTERFACE:

```
integer function index_(aList, item)
```

USES:

```
use m_String, only : toChar

implicit none
```

INPUT PARAMETERS:

```
type(List),      intent(in) :: aList ! a List of names
character(len=*),intent(in) :: item ! a given item name
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

18.1.8 indexStr_ - Return Rank in a List of a Given Item (String)

This function performs the same operation as the function `index_()`, but the item to be indexed is instead presented in the form of a `String` datatype (see the module `m_String` for more information about the `String` type). This routine searches through the input `List` argument `aList` for an item that matches the item defined by `itemStr`, and if a match is found, the rank of the item in the list is returned (see also the prologue for the routine `index_()` in this module). If no match is found, a value of zero is returned.

INTERFACE:

```
integer function indexStr_(aList, itemStr)
```

USES:

```
use m_String,only : String,toChar

implicit none
```

INPUT PARAMETERS:

```
type(List),      intent(in) :: aList ! a List of names
type(String),    intent(in) :: itemStr
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
25Oct02 - R. Jacob <jacob@mcs.anl.gov> - just call index_ above
```

18.1.9 allocated_ - Check Pointers in a List for Association Status

This function checks the input `List` argument `inList` to determine whether or not it has been allocated. It does this by invoking the Fortran90 intrinsic function `associated()` on the pointers `inList%bf` and `inList%lc`. If both of these pointers are associated, the return value is `.TRUE.`.
N.B.: In Fortran90, pointers have three different states: `ASSOCIATED`, `UNASSOCIATED`, and `UNDEFINED`. If a pointer is `UNDEFINED`, this function may return either `.TRUE.` or `.FALSE.` values, depending on the Fortran90 compiler. To avoid such problems, we advise that users invoke the `List` method `nullify()` to nullify any `List` pointers for `List` variables that are not initialized.

INTERFACE:

```
logical function allocated_(inList)
```

USES:

```
use m_die,only : die

implicit none
```

INPUT PARAMETERS:

```
type(List), intent(in) :: inList
```

REVISION HISTORY:

```
14Dec01 - J. Larson <larson@mcs.anl.gov> - inital version
```

18.1.10 copy_ - Copy a List

This routine copies the contents of the input `List` argument `xL` into the output `List` argument `yL`.

INTERFACE:

```
subroutine copy_(yL,xL) ! yL=xL
```

USES:

```
use m_die,only : die
use m_stdio
use m_String ,only : String
use m_String ,only : String_clean
use m_mall,only : mall_mci,mall_ison

implicit none
```

INPUT PARAMETERS:

```
type(List),intent(in)  :: xL
```

OUTPUT PARAMETERS:

```
type(List),intent(out) :: yL
```

REVISION HISTORY:

```
22Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
16May01 - J. Larson <larson@mcs.anl.gov> - simpler, working
          version that exploits the String datatype (see m_String)
 1Aug02 - Larson/Ong - Added logic for correct copying of blank
          Lists.
```

18.1.11 exportToChar_ - Export List to a CHARACTER

This function returns the character buffer portion of the input `List` argument `inList`—that is, the contents of `inList%bf`—as a `CHARACTER` (suitable for printing). An example of the use of this function is:

```
write(stdout,'(1a)') exportToChar(inList)
```

which writes the contents of `inList%bf` to the Fortran device `stdout`.

INTERFACE:

```
function exportToChar_(inList)
```

USES:

```
    use m_die,    only : die
    use m_stdio,  only : stderr
    use m_String, only : String
    use m_String, only : String_ToChar => toChar
    use m_String, only : String_clean

    implicit none

! INPUT PARAMETERS:

    type(List),        intent(in)  :: inList

! OUTPUT PARAMETERS:

    character(len=size(inList%bf,1)) :: exportToChar_
```

REVISION HISTORY:

```
13Feb02 - J. Larson <larson@mcs.anl.gov> - initial version.
06Jun03 - R. Jacob <jacob@mcs.anl.gov> - return blank if List is not allocated
```

18.1.12 exportToString_ - Export List to a String

This function returns the character buffer portion of the input `List` argument `inList`—that is, the contents of `inList%bf`—as a `String` (see the mpeu module m_String for more information regarding the `String` type). This function was created to circumvent problems with implementing inheritance of the function `exportToChar_()` to other datatypes build on top of the `List` type.

INTERFACE:

```
function exportToString_(inList)
```

USES:

```
      use m_die,    only : die
      use m_stdio,  only : stderr

      use m_String, only : String
      use m_String, only : String_init => init

      implicit none

  ! INPUT PARAMETERS:

      type(List),       intent(in) :: inList

  ! OUTPUT PARAMETERS:

      type(String)                 :: exportToString_
```

REVISION HISTORY:

```
  14Aug02 - J. Larson <larson@mcs.anl.gov> - initial version.
```

18.1.13 CharBufferSize_ - Return size of a List's Character Buffer

This function returns the length of the character buffer portion of the input `List` argument `inList` (that is, the number of characters stored in `inList%bf`) as an `INTEGER`. Suppose for the sake of argument that `inList` was created using the following call to `init_()`:

```
   call init_(inList, 'Groucho:Harpo:Chico:Zeppo')
```

Then, using the above example value of `inList`, we can use `CharBufferSize_()` as follows:

```
  integer :: BufferLength
  BufferLength = CharBufferSize(inList)
```

and the resulting value of `BufferLength` will be 25.

INTERFACE:

```
 integer function CharBufferSize_(inList)
```

USES:

```
      use m_die,    only : die
      use m_stdio,  only : stderr

      implicit none

  ! INPUT PARAMETERS:

      type(List),         intent(in) :: inList
```

REVISION HISTORY:

```
  13Feb02 - J. Larson <larson@mcs.anl.gov> - initial version.
```

18.1.14 get_ - Retrieve a Numbered Item from a List as a String

This routine retrieves a numbered item (defined by the input `INTEGER` argument `ith`) from the input `List` argument `aList`, and returns it in the output `String` argument `itemStr` (see the module `m_String` for more information about the `String` type). If the argument `ith` is nonpositive, or greater than the number of items in `aList`, a String containing one blank space is returned.

INTERFACE:

```
subroutine get_(itemStr, ith, aList)
```

USES:

```
use m_String, only : String, init, toChar

implicit none
```

INPUT PARAMETERS:

```
integer,     intent(in)  :: ith
type(List),  intent(in)  :: aList
```

OUTPUT PARAMETERS:

```
type(String),intent(out) :: itemStr
```

REVISION HISTORY:

```
23Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
14May07 - Larson, Jacob - add space to else case string so function
                          matches documentation.
```

18.1.15 getall_ - Return all Items from a List as one String

This routine returns all the items from the input `List` argument `aList` in the output `String` argument `itemStr` (see the module `m_String` for more information about the `String` type). The contents of the character buffer in `itemStr` will be the all of the items in `aList`, separated by the colon delimiter.

INTERFACE:

```
subroutine getall_(itemStr, aList)
```

USES:

```
use m_String, only : String, init, toChar

implicit none
```

INPUT PARAMETERS:

```
type(List),   intent(in)  :: aList
```

OUTPUT PARAMETERS:

```
type(String), intent(out) :: itemStr
```

REVISION HISTORY:

```
23Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
```

18.1.16 getrange_ - Return a Range of Items from a List as one String

This routine returns all the items ranked `i1` through `i2` from the input `List` argument `aList` in the output `String` argument `itemStr` (see the module `m_String` for more information about the `String` type). The contents of the character buffer in `itemStr` will be items in `i1` through `i2` `aList`, separated by the colon delimiter.

INTERFACE:

```
subroutine getrange_(itemStr, i1, i2, aList)
```

USES:

```
use m_die,    only : die
use m_stdio,  only : stderr
use m_String, only : String,init,toChar

implicit none
```

INPUT PARAMETERS:

```
integer,     intent(in)  :: i1
integer,     intent(in)  :: i2
type(List),  intent(in)  :: aList
```

OUTPUT PARAMETERS:

```
type(String),intent(out) :: itemStr
```

REVISION HISTORY:

```
23Apr98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
26Jul02 - J. Larson - Added argument checks.
```

18.1.17 identical_ - Compare Two Lists for Equality

This function compares the string buffer and indexing information in the two input `List` arguments `yL` and `xL`. If the string buffers and index buffers of `yL` and `xL` match, this function returns a value of `.TRUE.`. Otherwise, it returns a value of `.FALSE.`

INTERFACE:

```
logical function identical_(yL, xL)
```

USES:

```
use m_die,only : die
use m_String ,only : String
use m_String ,only : String_clean

implicit none
```

INPUT PARAMETERS:

```
type(List), intent(in) :: yL
type(List), intent(in) :: xL
```

REVISION HISTORY:

```
14Oct01 - J. Larson <larson@mcs.anl.gov> - original version
```

18.1.18 get_indices_ - Index Multiple Items in a List

This routine takes as input a `List` argument `aList`, and a `CHARACTER` string Values, which is a colon- delimited string of items, and returns an `INTEGER` array `indices(:)`, which contain the rank of each item in `aList`. For example, suppose `aList` was created from the character string

```
'happy:sleepy:sneezey:grumpy:dopey::bashful:doc'
```

and get_indices_() is invoked as follows:

```
call get_indices_(indices, aList, 'sleepy:grumpy:bashful:doc')
```

The array `indices(:)` will be returned with 4 entries: `indices`(1) = 2, `indices`(2) = 4, `indices`(3) = 6, and `indices`(4) = 7.
N.B.: This routine operates on the assumption that each of the substrings in the colon-delimited string `Values` is an item in `aList`. If this assumption is invalid, this routine terminates execution with an error message.
N.B.: The pointer `indices` must be `UNASSOCIATED` on entry to this routine, and will be `ASSOCIATED` upon return. After this pointer is no longer needed, it should be deallocated. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine get_indices_(indices, aList, Values)
```

USES:

```
use m_stdio
use m_die
use m_String, only : String
use m_String, only : String_clean => clean
use m_String, only : String_toChar => toChar

implicit none
```

INPUT PARAMETERS:

```
type(List),            intent(in) :: aList  ! an indexed string values
character(len=*),      intent(in) :: Values ! ":" delimited names
```

OUTPUT PARAMETERS:

```
integer, dimension(:), pointer    :: indices
```

REVISION HISTORY:

```
31May98 - Jing Guo <guo@thunder> - initial prototype/prolog/code
12Feb03 - J. Larson <larson@mcs.anl.gov> Working refactored version
```

18.1.19 test_indices_ - Test/Index Multiple Items in a List

This routine takes as input a `List` argument `aList`, and a `CHARACTER` string Values, which is a colon- delimited string of items, and returns an `INTEGER` array `indices(:)`, which contain the rank of each item in `aList`. For example, suppose `aList` was created from the character string

```
'happy:sleepy:sneezey:grumpy:dopey::bashful:doc'
```

and `test_indices_()` is invoked as follows:

```
call test_indices_(indices, aList, 'sleepy:grumpy:bashful:doc')
```

The array `indices(:)` will be returned with 4 entries: `indices`$(1) = 2$, `indices`$(2) = 4$, `indices`$(3) = 6$, and `indices`$(4) = 7$.
Now suppose `test_indices_()` is invoked as follows:

```
call test_indices_(indices, aList, 'sleepy:grumpy:bashful:Snow White')
```

The array `indices(:)` will be returned with 4 entries: `indices`$(1) = 2$, `indices`$(2) = 4$, `indices`$(3) = 6$, and `indices`$(4) = 0$.
N.B.: This routine operates on the assumption that one or more of the substrings in the colon-delimited string `Values` is may not be an item in `aList`. If an item in `Values` is *not* in `aList`, its corresponding entry in `indices(:)` is set to zero.
N.B.: The pointer `indices` must be `UNASSOCIATED` on entry to this routine, and will be `ASSOCIATED` upon return. After this pointer is no longer needed, it should be deallocated. Failure to do so will result in a memory leak.

INTERFACE:

```
subroutine test_indices_(indices, aList, Values)
```

USES:

```
use m_stdio
use m_die
use m_String, only : String
use m_String, only : String_clean => clean
use m_String, only : String_toChar => toChar

implicit none
```

INPUT PARAMETERS:

```
type(List),            intent(in) :: aList  ! an indexed string values
character(len=*),      intent(in) :: Values ! ":" delimited names
```

OUTPUT PARAMETERS:

```
integer, dimension(:), pointer    :: indices
```

REVISION HISTORY:

12Feb03 - J. Larson <larson@mcs.anl.gov> Working refactored version

---

18.1.20 append_ - Append One List Onto the End of Another

This routine takes two `List` arguments `iList1` and `iList2`, and appends `List2` onto the end of `List1`.
N.B.: There is no check for shared items in the arguments `List1` and `List2`. It is the user's responsibility to ensure `List1` and `List2` share no items. If this routine is invoked in such a manner that `List1` and `List2` share common items, the resultant value of `List1` will produce ambiguous results for some of the `List` query functions.
N.B.: The outcome of this routine is order dependent. That is, the entries of `iList2` will follow the *input* entries in `iList1`.

INTERFACE:

```
subroutine append_(iList1, iList2)
```

USES:

```
use m_stdio
use m_die, only : die

use m_mpif90

use m_String, only:  String
use m_String, only:  String_toChar => toChar
use m_String, only:  String_len
use m_String, only:  String_clean => clean

implicit none
```

INPUT PARAMETERS:

```
type(List),         intent(in)    :: iList2
```

INPUT/OUTPUT PARAMETERS:

```
type(List),         intent(inout) :: iList1
```

REVISION HISTORY:

6Aug02 - J. Larson - Initial version

18.1.21 concatenate_ - Concatenates two Lists to form a Third List.

This routine takes two input `List` arguments `iList1` and `iList2`, and concatenates them, producing an output `List` argument `oList`.
N.B.: The nature of this routine is such that one must **never** supply as the actual value of `oList` the same value supplied for either `iList1` or `iList2`.
N.B.: The outcome of this routine is order dependent. That is, the entries of `iList2` will follow `iList1`.

INTERFACE:

```
subroutine concatenate_(iList1, iList2, oList)
```

USES:

```
use m_stdio
use m_die, only : die

use m_mpif90

use m_String, only:  String
use m_String, only:  String_init => init
use m_String, only:  String_clean => clean

implicit none
```

INPUT PARAMETERS:

```
type(List),         intent(in)  :: iList1
type(List),         intent(in)  :: iList2
```

OUTPUT PARAMETERS:

```
type(List),         intent(out) :: oList
```

BUGS:

```
CHARACTER variables as intermediate storage.  The lengths of these
scratch variables is hard-wired to 10000, which should be large enough
for most applications.  This undesirable feature should be corrected
ASAP.
```

REVISION HISTORY:

```
 8May01 - J.W. Larson - initial version.
17May01 - J.W. Larson - Re-worked and tested successfully.
17Jul02 - E. Ong - fixed the bug mentioned above
```

18.1.22 bcast_ - MPI Broadcast for the List Type

This routine takes an input `List` argument `iList` (on input, valid on the root only), and broadcasts it.
N.B.: The outcome of this routine, `ioList` on non-root processes, represents allocated memory. When this `List` is no longer needed, it must be deallocated by invoking the routine `List_clean()`. Failure to do so will cause a memory leak.

INTERFACE:

```
subroutine bcast_(ioList, root, comm, status)
```

USES:

```
use m_stdio,  only : stderr
use m_die, only : MP_perr_die, die

use m_String, only:  String
use m_String, only:  String_bcast => bcast
use m_String, only:  String_clean => clean

use m_mpif90

implicit none
```

INPUT PARAMETERS:

```
integer,            intent(in)     :: root
integer,            intent(in)     :: comm
```

INPUT/OUTPUT PARAMETERS:

```
type(List),         intent(inout)  :: ioList
```

OUTPUT PARAMETERS:

```
integer, optional,  intent(out)    :: status
```

REVISION HISTORY:

```
 7May01 - J.W. Larson - initial version.
14May01 - R.L. Jacob - fix error checking
16May01 - J.W. Larson - new, simpler String-based algorigthm
          (see m_String for details), which works properly on
          the SGI platform.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize status
          (if present).
```

18.1.23 send_ - MPI Point-to-Point Send for the List Type

This routine takes an input `List` argument `inList` and sends it to processor `dest` on the communicator associated with the fortran 90 `INTEGER` handle `comm`. The message is tagged by the input `INTEGER` argument `TagBase`. The success (failure) of this operation is reported in the zero (nonzero) optional output argument `status`.
N.B.: One must avoid assigning elsewhere the MPI tag values `TagBase` and `TagBase+1`. This is because `send_()` performs the send of the `List` as a pair of operations. The first send is the number of characters in `inList%bf`, and is given MPI tag value `TagBase`. The second send is the `CHARACTER` data present in `inList%bf`, and is given MPI tag value `TagBase+1`.

INTERFACE:

```
subroutine send_(inList, dest, TagBase, comm, status)
```

USES:

```
use m_stdio
use m_die, only : MP_perr_die

use m_mpif90

use m_String, only:  String
use m_String, only:  String_toChar => toChar
use m_String, only:  String_len
use m_String, only:  String_clean => clean

implicit none
```

INPUT PARAMETERS:

```
type(List),         intent(in)  :: inList
integer,            intent(in)  :: dest
integer,            intent(in)  :: TagBase
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
integer, optional,  intent(out) :: status
```

REVISION HISTORY:

```
 6Jun01 - J.W. Larson - initial version.
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize status
          (if present).
```

18.1.24 recv_ - MPI Point-to-Point Receive for the List Type

This routine receives the output `List` argument `outList` from processor `source` on the communicator associated with the fortran 90 `INTEGER` handle `comm`. The message is tagged by the input `INTEGER` argument `TagBase`. The success (failure) of this operation is reported in the zero (nonzero) optional output argument `status`.
N.B.: One must avoid assigning elsewhere the MPI tag values `TagBase` and `TagBase+1`. This is because `recv_()` performs the receive of the `List` as a pair of operations. The first receive is the number of characters in `outList%bf`, and is given MPI tag value `TagBase`. The second receive is the `CHARACTER` data present in `outList%bf`, and is given MPI tag value `TagBase+1`.

INTERFACE:

```
subroutine recv_(outList, source, TagBase, comm, status)
```

USES:

```
use m_stdio, only : stderr
use m_die,   only : MP_perr_die

use m_mpif90

use m_String, only : String

implicit none
```

INPUT PARAMETERS:

```
integer,            intent(in)  :: source
integer,            intent(in)  :: TagBase
integer,            intent(in)  :: comm
```

OUTPUT PARAMETERS:

```
type(List),         intent(out) :: outList
integer, optional,  intent(out) :: status
```

REVISION HISTORY:

```
 6Jun01 - J.W. Larson - initial version.
11Jun01 - R. Jacob - small bug fix; status in MPI_RECV
13Jun01 - J.W. Larson <larson@mcs.anl.gov> - Initialize status
          (if present).
```

18.1.25 GetSharedListIndices_ - Index Shared Items for Two Lists

`GetSharedListIndices_()` compares two user- supplied `List` arguments `List1` and `Lis2` to determine: the number of shared items `NumShared`, and arrays of the locations `Indices1` and `Indices2` in `List1` and `List2`, respectively.
N.B.: This routine returns two allocated arrays: `Indices1(:)` and `Indices2(:)`. Both of these arrays must be deallocated once they are no longer needed. Failure to do this will create a memory leak.

INTERFACE:

```
subroutine GetSharedListIndices_(List1, List2, NumShared, Indices1, &
                                 Indices2)
```

USES:

```
use m_die,  only : MP_perr_die, die, warn

use m_String, only : String
use m_String, only : String_clean => clean

implicit none
```

INPUT PARAMETERS:

```
type(List),    intent(in)  :: List1
type(List),    intent(in)  :: List2
```

OUTPUT PARAMETERS:

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

integer,dimension(:), pointer  :: Indices1
integer,dimension(:), pointer  :: Indices2
```

REVISION HISTORY:

```
7Feb01 - J.W. Larson <larson@mcs.anl.gov> - initial version
```