Chombo&INS with Particles Reference Manual

1.2

Generated by Doxygen 1.3.2

Wed Jun 2 13:34:23 2004

Contents

Chapter 1

Chombo: Block-Structured Adaptive Mesh Refinement Library

`Chombo Web Page`

	What is Chombo?

The Chombo package provides a set of tools for implementing finite difference methods for the solution of partial differential equations on block-structured adaptively refined rectangular grids. Both elliptic and time-dependent modules are included. Support for parallel platforms and standardized self-describing file formats are included.

Chombo provides a distributed infrastructure for parallel calculations over block-structured, adaptively refined grids. Chombo's design is uniquely flexible and accessible. Any collaborator will be able to develop parallel applications to solve the partial differential equations in which she is interested with far shorter development times than would be possible without the infrastructure. Very careful design and documentation allows said collaborator to enter the software at many levels. She will be able to use Chombo to investigate deep technical issues of adaptive mesh refinement algorithms or to simply adapt the example applications to solve different scientific problems.

Chapter 2

Chombo&INS Namespace Index

2.1 Chombo&INS Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Chombo&INS Hierarchical Index

3.1 Chombo&INS Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Chombo&INS Compound Index

4.1 Chombo&INS Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

Chombo&INS File Index

5.1 Chombo&INS File List

Here is a list of all files with brief descriptions:

Chapter 6

Chombo&INS Namespace Documentation

6.1 std Namespace Reference

Chapter 7

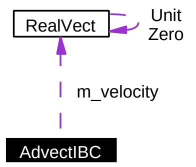
Chombo&INS Class Documentation

7.1 AdvectIBC Class Reference

IBC for simple advection.

`#include <AdvectIBC.H>`

Collaboration diagram for AdvectIBC:



Public Member Functions

- **AdvectIBC** ()

    Null constructor.

- **AdvectIBC** (const **Real** &a_gamma, const **Real** &a_ambientDensity, const **Real** &a_deltaDensity, const int &a_pressure, const **RealVect** &a_center, const **Real** &a_size, const **RealVect** &a_velocity)

    Constructor which defines parameters used by Fortran routines.

- ~**AdvectIBC** ()

    Destructor.

- PhysIBC ∗ **new_physIBC** ()

    Factory method - this object is its own factory.

- void **setDefaultValues** ()

set default boundary values

- void **fluxBC** (**FArrayBox** &a_F, const **FArrayBox** &a_W, const **FArrayBox** &a_-Wextrap, const int &a_dir, const **Side::LoHiSide** &a_side, const **Real** &a_time)

  Set boundary fluxes.

- void **setBdrySlopes** (**FArrayBox** &a_dW, const **FArrayBox** &a_W, const int &a_dir, const **Real** &a_time)

  Set boundary slopes.

- void **initialize** (**LevelData**< **FArrayBox** > &a_U)

  Set up initial conditions.

- void **advectionVel** (const **RealVect** &a_advVel)

  set velocity

- const **RealVect** & **advectionVel** () const
- void **probType** (const int a_probtype)

  set problem type

- void **setBoundaryValue** (**Real** a_bcVal, int a_dir, **Side::LoHiSide** a_hiLo)

  set boundary value (default is 0)

- **Real getBoundaryValue** (int a_dir, **Side::LoHiSide** a_hiLo) const

  access boundary value

- void **setSlopeValue** (**Real** a_slopeVal, int a_dir, **Side::LoHiSide** a_hiLo)

  set slope value (default is 0)

- **Real getSlopeValue** (int a_dir, **Side::LoHiSide** a_hiLo) const

  access slope value

- int **probType** () const

  accessor

Protected Attributes

- **RealVect m_velocity**

  advection velocity field

- **Real m_bcVal** [**SpaceDim**][2]
- **Real m_slopeVal** [**SpaceDim**][2]
- bool **m_isBCvalSet**
- bool **m_isSlopeValSet**
- int **m_probtype**

  initial condition type

7.1.1 Detailed Description

IBC for simple advection.

Parameters:

7.1.2 Constructor & Destructor Documentation

7.1.2.1 AdvectIBC::AdvectIBC ()

Null constructor.

7.1.2.2 AdvectIBC::AdvectIBC (const Real & *a_gamma*, const Real & *a_ambientDensity*, const Real & *a_deltaDensity*, const int & *a_pressure*, const RealVect & *a_center*, const Real & *a_size*, const RealVect & *a_velocity*)

Constructor which defines parameters used by Fortran routines.

7.1.2.3 AdvectIBC::~AdvectIBC () `[inline]`

Destructor.

7.1.3 Member Function Documentation

7.1.3.1 const RealVect& AdvectIBC::advectionVel () const

7.1.3.2 void AdvectIBC::advectionVel (const RealVect & *a_advVel*)

set velocity

7.1.3.3 void AdvectIBC::fluxBC (FArrayBox & *a_F*, const FArrayBox & *a_W*, const FArrayBox & *a_Wextrap*, const int & *a_dir*, const Side::LoHiSide & *a_side*, const Real & *a_time*)

Set boundary fluxes.

7.1.3.4 Real AdvectIBC::getBoundaryValue (int *a_dir*, Side::LoHiSide *a_hiLo*) const

access boundary value

7.1.3.5 Real AdvectIBC::getSlopeValue (int *a_dir*, Side::LoHiSide *a_hiLo*) const

access slope value

7.1.3.6 void AdvectIBC::initialize (LevelData< FArrayBox > & *a_U*)

Set up initial conditions.

7.1.3.7 PhysIBC* AdvectIBC::new_physIBC ()

Factory method - this object is its own factory.

Return a pointer to a new PhysIBC object with m_isDefined = false (i.e., its define() must be called before it is used) and m_isFortranCommonSet set to value of m_isFortranCommonset in the current (factory) object.

7.1.3.8 int AdvectIBC::probType () const

accessor

7.1.3.9 void AdvectIBC::probType (const int *a_probtype*)

set problem type

7.1.3.10 void AdvectIBC::setBdrySlopes (FArrayBox & *a_dW*, const FArrayBox & *a_W*, const int & *a_dir*, const Real & *a_time*)

Set boundary slopes.

The boundary slopes in a_dW are already set to one sided difference approximations. If this function doesn't change them they will be used for the slopes at the boundaries.

7.1.3.11 void AdvectIBC::setBoundaryValue (Real *a_bcVal*, int *a_dir*, Side::LoHiSide *a_hiLo*)

set boundary value (default is 0)

7.1.3.12 void AdvectIBC::setDefaultValues ()

set default boundary values

7.1.3.13 void AdvectIBC::setSlopeValue (Real *a_slopeVal*, int *a_dir*, Side::LoHiSide *a_hiLo*)

set slope value (default is 0)

7.1.4 Member Data Documentation

7.1.4.1 Real AdvectIBC::m_bcVal[SpaceDim][2] `[protected]`

7.1.4.2 bool AdvectIBC::m_isBCvalSet `[protected]`

7.1.4.3 bool AdvectIBC::m_isSlopeValSet `[protected]`

7.1.4.4 int AdvectIBC::m_probtype `[protected]`

initial condition type

7.1.4.5 Real AdvectIBC::m_slopeVal[SpaceDim][2] [protected]

7.1.4.6 RealVect AdvectIBC::m_velocity [protected]

advection velocity field

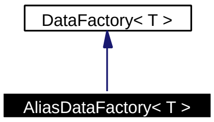
The documentation for this class was generated from the following file:

- **AdvectIBC.H**

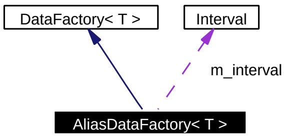
7.2 AliasDataFactory< T > Class Template Reference

`#include <BoxLayoutData.H>`

Inheritance diagram for AliasDataFactory< T >:



Collaboration diagram for AliasDataFactory< T >:



Public Member Functions

- virtual ~**AliasDataFactory** ()
- **AliasDataFactory** (**BoxLayoutData**< T > *a_original, const **Interval** &interval)
- void **define** (**BoxLayoutData**< T > *a_original, const **Interval** &interval)
- virtual T * **create** (const **Box** &box, int ncomps, const **DataIndex** &a_datInd) const

Protected Attributes

- **BoxLayoutData< T > * m_origPointer**
- **Interval m_interval**

template<class T> class AliasDataFactory< T >

7.2.1 Constructor & Destructor Documentation

7.2.1.1 template<class T> virtual AliasDataFactory< T >::~AliasDataFactory () `[inline, virtual]`

7.2.1.2 template<class T> AliasDataFactory< T >::AliasDataFactory (BoxLayoutData< T > * *a_original*, const Interval & *interval*)

7.2.2 Member Function Documentation

7.2.2.1 template<class T> T * AliasDataFactory< T >::create (const Box & *box*, int *ncomps*, const DataIndex & *a_datInd*) const `[virtual]`

creates a new 'T' object and returns a pointer to it. Responsiblitly for calling operator 'delete' on this pointer is passed to the user.

Implements **DataFactory< T >** (p. 206).

7.2.2.2 template<class T> void AliasDataFactory< T >::define (BoxLayoutData< T > * *a_original*, const Interval & *interval*)

7.2.3 Member Data Documentation

7.2.3.1 template<class T> Interval AliasDataFactory< T >::m_interval [protected]

7.2.3.2 template<class T> BoxLayoutData<T>* AliasDataFactory< T >::m_origPointer [protected]

The documentation for this class was generated from the following file:

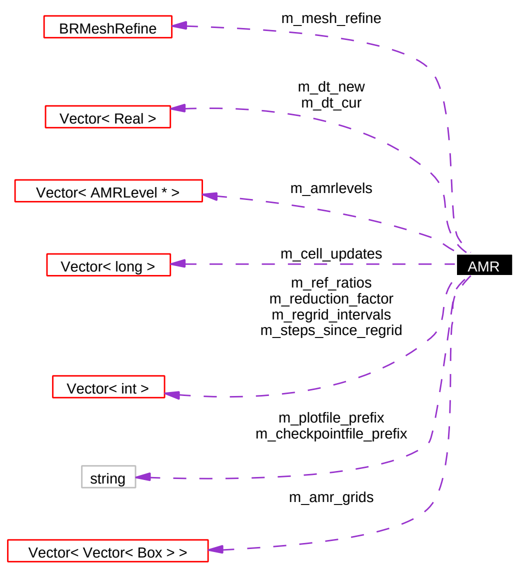
- **BoxLayoutData.H**

7.3 AMR Class Reference

Framework for Berger-Oliger timestepping for AMR.

`#include <AMR.H>`

Collaboration diagram for AMR:



Public Member Functions

- **AMR** ()
- ~**AMR** ()
- void **define** (int a_max_level, const **Vector**< int > &a_ref_ratios, const **Box** &a_prob_domain, const **AMRLevelFactory** *const a_amrLevelFact)
- void **define** (int a_max_level, const **Vector**< int > &a_ref_ratios, const **ProblemDomain** &a_prob_domain, const **AMRLevelFactory** *const a_amrLevelFact)
- bool **isDefined** () const
- bool **isSetUp** () const
- void **setupForNewAMRRun** ()
- void **setupForFixedHierarchyRun** (const **Vector**< **Vector**< **Box** > > &a_amr_grids, int a_proper_nest=1)
- void **conclude** () const
- void **run** (**Real** a_max_time, int a_max_step)
- void **plotPrefix** (const std::string &a_plotfile_prefix)
- void **checkpointPrefix** (const std::string &a_checkpointfile_prefix)
- void **plotInterval** (int a_plot_interval)
- void **checkpointInterval** (int a_checkpoint_interval)
- void **maxGridSize** (int a_max_grid_size)
- void **maxBaseGridSize** (int a_max_base_grid_size)
- void **dtToleranceFactor** (**Real** a_dt_tolerance_factor)
- void **fillRatio** (**Real** a_fillRat)
- void **blockFactor** (int a_blockFactor)

- void **gridBufferSize** (int a_grid_buffer_size)
- int **maxGridSize** () const
- int **maxBaseGridSize** () const
- void **verbosity** (int a_verbosity)
- void **regridIntervals** (const **Vector**< int > &a_regridIntervals)
- int **verbosity** () const
- void **maxDtGrow** (**Real** a_dtGrowFactor)
- void **timeEps** (**Real** a_timeEps)
- **Real maxDtGrow** () const
- **Real timeEps** () const
- void **fixedDt** (**Real** a_dt)
- **Real fixedDt** () const
- **Vector**< **AMRLevel** ∗ > **getAMRLevels** ()
- **Real getCurrentTime** () const

Protected Member Functions

- int **timeStep** (int a_level, int a_stepsLeft, bool a_coarseTimeBoundary)
- void **clearMemory** ()
- void **regrid** (int a_base_level)
- void **initialGrid** ()
- void **writePlotFile** () const
- void **writeCheckpointFile** () const
- void **assignDt** ()
- bool **needToRegrid** (int a_level, int a_numStepsLeft) const
- void **makeBaseLevelMesh** (**Vector**< **Box** > &a_grids) const
- void **setDefaultValues** ()

Protected Attributes

- int **m_blockFactor**
- **Real m_fillRatio**
- int **m_max_level**
- int **m_finest_level_old**
- int **m_finest_level**
- int **m_checkpoint_interval**
- int **m_plot_interval**
- int **m_max_grid_size**
- int **m_max_base_grid_size**
- **Real m_dt_tolerance_factor**
- **Real m_fixedDt**
- bool **m_isDefined**
- bool **m_isSetUp**
- **Vector**< **AMRLevel** ∗ > **m_amrlevels**
- **Vector**< int > **m_ref_ratios**
- **Vector**< int > **m_reduction_factor**
- **Vector**< int > **m_regrid_intervals**
- **Real m_dt_base**
- **Vector**< **Real** > **m_dt_new**

- **Vector< Real > m_dt_cur**
- **Real m_maxDtGrow**
- **Real m_time_eps**
- **BRMeshRefine m_mesh_refine**
- bool **m_use_meshrefine**
- **Vector< Vector< Box > > m_amr_grids**
- int **m_cur_step**
- int **m_restart_step**
- int **m_lastcheck_step**
- **Real m_cur_time**
- **Vector**< int > **m_steps_since_regrid**
- **Vector**< long > **m_cell_updates**
- std::string **m_plotfile_prefix**
- std::string **m_checkpointfile_prefix**
- int **m_verbosity**

7.3.1 Detailed Description

Framework for Berger-Oliger timestepping for AMR.

This class is a framework for Berger-Oliger timestepping for adaptive mesh refinement of time-dependent problems. It is applicable to both hyperbolic and parabolic problems. It represents a hierarchy of levels of refinement as a collection of **AMRLevel** objects.

The usage pattern is this:

1. Call define to define the stuff that does not change throughout the run (maxlevel, refinement ratios, domain, and operator).
2. Modify any parameters you like (blocking factor and so forth using access functions.
3. Call any one of the three setup functions so AMR can set up all its internal data structures.
4. Call run to run the calculation.
5. Call conclude to output (to stdout) how many cells were updated and so forth.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 AMR::AMR ()

Default constructor. Use must subsequently call a **define()** function and a setUp function.

7.3.2.2 AMR::~AMR ()

Destructor.

7.3.3 Member Function Documentation

7.3.3.1 void AMR::assignDt () `[protected]`

7.3.3.2 void AMR::blockFactor (int *a_blockFactor*)

Set the blocking factor for meshrefinement. Should be called after define and before setup.

7.3.3.3 void AMR::checkpointInterval (int *a_checkpoint_interval*)

Sets the interval to write checkpoint files, in terms of the base level time step.

7.3.3.4 void AMR::checkpointPrefix (const std::string & *a_checkpointfile_prefix*)

Sets the checkpoint file prefix. Should be called after define and before setup.

7.3.3.5 void AMR::clearMemory () `[protected]`

7.3.3.6 void AMR::conclude () const

You should call this last. It writes the last checkpoint file and tells you how many cells you updated and all that.

7.3.3.7 void AMR::define (int *a_max_level*, const Vector< int > & *a_ref_ratios*, const ProblemDomain & *a_prob_domain*, const AMRLevelFactory *const *a_amrLevelFact*)

Defines this object. User must call a setup function before running. Arguments:

- a_prob_domain (not modified): problem domain on the base level.
- a_ref_ratios (not modified): refinement ratios. There must be at least a_max_level+1 elements, or an error will result. Element zero is the base level.
- a_max_level (not modified): the maximum level allowed, where the base level is zero. There will be a total of a_max_level+1 levels, since level zero and level a_max_level will both exist.
- a_amrlevel_ptr (not modified): pointer to a physics class factory object. The object it points to is used to construct the collection of **AMRLevel** objects in this AMR as objects of the physics class type. It can be destructed after this **define()** returns.

7.3.3.8 void AMR::define (int *a_max_level*, const Vector< int > & *a_ref_ratios*, const Box & *a_prob_domain*, const AMRLevelFactory *const *a_amrLevelFact*)

Defines this object. User must call a setup function before running. Arguments:

- a_prob_domain (not modified): problem domain on the base level.
- a_ref_ratios (not modified): refinement ratios. There must be at least a_max_level+1 elements, or an error will result. Element zero is the base level.
- a_max_level (not modified): the maximum level allowed, where the base level is zero. There will be a total of a_max_level+1 levels, since level zero and level a_max_level will both exist.
- a_amrlevel_ptr (not modified): pointer to a physics class factory object. The object it points to is used to construct the collection of **AMRLevel** objects in this AMR as objects of the physics class type. It can be destructed after this **define()** returns.

7.3.3.9 void AMR::dtToleranceFactor (Real *a_dt_tolerance_factor*)

Set the factor by which the current dt must exceed the new (max) dt for time subcycling to occur (i.e., reduction of the current dt by powers of 2).

7.3.3.10 void AMR::fillRatio (Real *a_fillRat*)

Set the fill ratio for meshrefinement. Should be called after define and before setup.

7.3.3.11 Real AMR::fixedDt () const

Returns the fixed timestep.

7.3.3.12 void AMR::fixedDt (Real *a_dt*)

Set a fixed timestep.

7.3.3.13 Vector<AMRLevel*> AMR::getAMRLevels ()

Returns a vector of all the AMR levels

7.3.3.14 Real AMR::getCurrentTime () const

Returns the current time

7.3.3.15 void AMR::gridBufferSize (int *a_grid_buffer_size*)

Set the buffering for the meshrefinement. Should be called after define and before setup.

7.3.3.16 void AMR::initialGrid () `[protected]`

7.3.3.17 bool AMR::isDefined () const

Has a define function been called? Lots of these functions will assert fail if not.

7.3.3.18 bool AMR::isSetUp () const

Has a setUp function been called? If not, you can't call run

7.3.3.19 void AMR::makeBaseLevelMesh (Vector< Box > & *a_grids*) const `[protected]`

7.3.3.20 int AMR::maxBaseGridSize () const

Returns the maximum grid size for level 0 grids.

7.3.3.21 void AMR::maxBaseGridSize (int *a_max_base_grid_size*)

Set the maximum grid size for level 0 grids. Defaults to maxGridSize, so should be called after maxGridSize and before setup

7.3.3.22 Real AMR::maxDtGrow () const

Returns current maximum factor by which a timestep can grow.

7.3.3.23 void AMR::maxDtGrow (Real *a_dtGrowFactor*)

Set maximum factor by which a timestep can grow.

7.3.3.24 int AMR::maxGridSize () const

Returns the maximum grid size.

7.3.3.25 void AMR::maxGridSize (int *a_max_grid_size*)

Set the maximum grid size. Should be called after define and before setup.

7.3.3.26 bool AMR::needToRegrid (int *a_level*, int *a_numStepsLeft*) const `[protected]`

7.3.3.27 void AMR::plotInterval (int *a_plot_interval*)

Sets the interval to write plot files, in terms of the base level time step.

7.3.3.28 void AMR::plotPrefix (const std::string & *a_plotfile_prefix*)

Sets the plot file prefix. Should be called after define and before setup.

7.3.3.29 void AMR::regrid (int *a_base_level*) `[protected]`

7.3.3.30 void AMR::regridIntervals (const Vector< int > & *a_regridIntervals*)

Sets the regridding intervals. This should be OK to call any time after define and before run.

7.3.3.31 void AMR::run (Real *a_max_time*, int *a_max_step*)

Runs the calculation. User must have previously called both the define function and a setup function.

7.3.3.32 void AMR::setDefaultValues () [protected]

7.3.3.33 void AMR::setupForFixedHierarchyRun (const Vector< Vector< Box > > & *a_amr_grids*, int *a_proper_nest* = 1)

This sets the hierarchy and sets regrid_intervals to -1 (turns off regridding). If you want to keep regridding on, reset regridIntervals after this call;

7.3.3.34 void AMR::setupForNewAMRRun ()

Sets up this object for cold start. User must have previously called **define()**. Need to call this or setupForRestart or setupforFixedHierarchyRun before you run.

7.3.3.35 Real AMR::timeEps () const

Returns amount by which two times may differ and still be considered equal (used to determine whether we've reached stop time yet)

7.3.3.36 void AMR::timeEps (Real *a_timeEps*)

Set amount by which two times may differ and still be considered equal (used to determine whether we've reached stop time yet) In practice, this is multiplied by the current base-level timestep, so we consider ourselves finished if (stopTime - time) < timeEps∗dt0

7.3.3.37 int AMR::timeStep (int *a_level*, int *a_stepsLeft*, bool *a_coarseTimeBoundary*) [protected]

7.3.3.38 int AMR::verbosity () const

Returns current verbosity level.

-inf to 0: print nothing.

1: prints a message every coarse time step.

2: also prints a message when regridding and writing files.

3: also prints function trace, and some more inf during file I/O.

4 to inf: also prints list of boxes and processor maps during regrid.

7.3.3.39 void AMR::verbosity (int *a_verbosity*)

Sets verbosity level to a_verbosity.

-inf to 0: print nothing.

1: prints a message every coarse time step.

2: also prints a message when regridding and writing files.

3: also prints function trace, and some more inf during file I/O.

4 to inf: also prints list of boxes and processor maps during regrid.

This should be OK to call any time after define and before run.

7.3.3.40 void AMR::writeCheckpointFile () const [protected]

7.3.3.41 void AMR::writePlotFile () const [protected]

7.3.4 Member Data Documentation

7.3.4.1 Vector<Vector<Box> > AMR::m_amr_grids [protected]

7.3.4.2 Vector<AMRLevel*> AMR::m_amrlevels [protected]

7.3.4.3 int AMR::m_blockFactor [protected]

7.3.4.4 Vector<long> AMR::m_cell_updates [protected]

7.3.4.5 int AMR::m_checkpoint_interval [protected]

7.3.4.6 std::string AMR::m_checkpointfile_prefix [protected]

7.3.4.7 int AMR::m_cur_step [protected]

7.3.4.8 Real AMR::m_cur_time [protected]

7.3.4.9 Real AMR::m_dt_base [protected]

7.3.4.10 Vector<Real> AMR::m_dt_cur [protected]

7.3.4.11 Vector<Real> AMR::m_dt_new [protected]

7.3.4.12 Real AMR::m_dt_tolerance_factor [protected]

7.3.4.13 Real AMR::m_fillRatio [protected]

7.3.4.14 int AMR::m_finest_level [protected]

7.3.4.15 int AMR::m_finest_level_old [protected]

7.3.4.16 Real AMR::m_fixedDt [protected]

7.3.4.17 bool AMR::m_isDefined [protected]

7.3.4.18 bool AMR::m_isSetUp [protected]

7.3.4.19 int AMR::m_lastcheck_step [protected]

7.3.4.20 int AMR::m_max_base_grid_size [protected]

7.3.4.21 int AMR::m_max_grid_size [protected]

7.3.4.22 int AMR::m_max_level [protected]

7.3.4.23 Real AMR::m_maxDtGrow [protected]

7.3.4.24 BRMeshRefine AMR::m_mesh_refine [protected]

7.3.4.25 int AMR::m_plot_interval [protected]

7.3.4.26 std::string AMR::m_plotfile_prefix [protected]

7.3.4.27 Vector<int> AMR::m_reduction_factor [protected]

7.3.4.28 Vector<int> AMR::m_ref_ratios [protected]

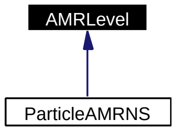
- **AMR.H**

7.4 AMRLevel Class Reference

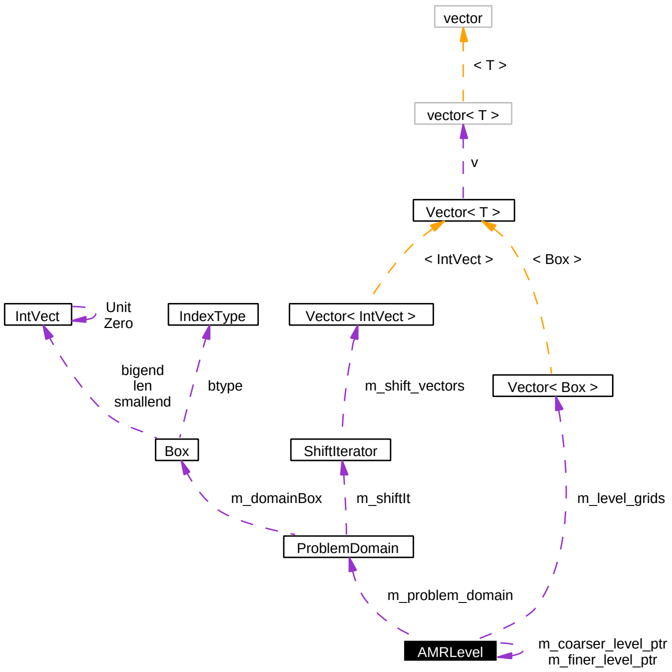
Abstract base class for time-dependent data at a level of refinement.

`#include <AMRLevel.H>`

Inheritance diagram for AMRLevel:



Collaboration diagram for AMRLevel:



Public Member Functions

- **AMRLevel** ()
- virtual ∼**AMRLevel** ()
- virtual void **define** (**AMRLevel** ∗a_coarser_level_ptr, const **Box** &a_problem_domain, int a_level, int a_ref_ratio)
- virtual void **define** (**AMRLevel** ∗a_coarser_level_ptr, const **ProblemDomain** &a_problem_domain, int a_level, int a_ref_ratio)
- virtual void **finerLevelPtr** (**AMRLevel** ∗a_finer_level_ptr)
- virtual void **dt** (**Real** a_dt)
- virtual void **time** (**Real** a_time)
- virtual **Real dt** () const

- virtual **Real time** () const
- virtual void **initialDtMultiplier** (**Real** a_initial_dt_multiplier)
- virtual **Real initialDtMultiplier** () const
- virtual const **ProblemDomain** & **problemDomain** () const
- virtual **Vector**< **Box** > **boxes** () const
- bool **isDefined** () const
- virtual int **refRatio** () const
- virtual **Real advance** ()=0
- virtual void **postTimeStep** ()=0
- virtual void **tagCells** (**IntVectSet** &a_tags)=0
- virtual void **tagCellsInit** (**IntVectSet** &a_tags)=0
- virtual void **regrid** (const **Vector**< **Box** > &a_new_grids)=0
- virtual void **postRegrid** (int a_base_level)
- virtual void **initialGrid** (const **Vector**< **Box** > &a_new_grids)=0
- virtual void **initialData** ()=0
- virtual void **postInitialize** ()=0
- virtual **Real computeDt** ()=0
- virtual **Real computeInitialDt** ()=0

Static Public Member Functions

- void **verbosity** (int a_verbosity)
- int **verbosity** ()

Protected Attributes

- **ProblemDomain m_problem_domain**
- **Vector**< **Box** > **m_level_grids**
- int **m_level**
- int **m_ref_ratio**
- **Real m_initial_dt_multiplier**
- **Real m_dt**
- **Real m_time**
- **AMRLevel** ∗ **m_coarser_level_ptr**
- **AMRLevel** ∗ **m_finer_level_ptr**
- bool **m_isDefined**

Static Protected Attributes

- int **s_verbosity**

7.4.1 Detailed Description

Abstract base class for time-dependent data at a level of refinement.

This is an abstract base class for data at the same level of refinement within a hierarchy of levels. The concrete class derived from AMRLevel is called a physics class. The domain of a level is a disjoint union of rectangles in a logically rectangular index space. Data is defined within this domain. There is also a problem domain, which may be larger, within which data can, in theory, be interpolated from some coarser level.

AMRLevel is intimately entangled with the **AMR** class. The **AMR** contains a collection of AMRLevel objects which represent a hierarchy of levels of refinement. The class **AMR** is responsible for calling the correct member functions of AMRLevel. The user is responsible for implementing the physics class, and might never call any AMRLevel functions described here.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 AMRLevel::AMRLevel ()

Default constructor.

7.4.2.2 virtual AMRLevel::∼AMRLevel () `[virtual]`

Destructor.

7.4.3 Member Function Documentation

7.4.3.1 virtual Real AMRLevel::advance () `[pure virtual]`

Advances this level by one time step.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 452).

7.4.3.2 virtual Vector<Box> AMRLevel::boxes () const `[virtual]`

Returns the domain of this level.

7.4.3.3 virtual Real AMRLevel::computeDt () `[pure virtual]`

Returns maximum stable time step for this level.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 453).

7.4.3.4 virtual Real AMRLevel::computeInitialDt () `[pure virtual]`

Returns maximum stable time step for this level with initial data.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 453).

7.4.3.5 virtual void AMRLevel::define (AMRLevel ∗ *a_coarser_level_ptr*, const ProblemDomain & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) `[virtual]`

Defines this AMRLevel.

- a_coarser_level_ptr (not modified): pointer to next coarser level object.

- a_problem_domain (not modified): problem domain of this level.
- a_level (not modified): index of this level. The base level is zero.
- a_ref_ratio (not modified): the refinement ratio between this level and the next finer level.

Reimplemented in **ParticleAMRNS** (p. 454).

7.4.3.6 virtual void AMRLevel::define (AMRLevel * *a_coarser_level_ptr*, const Box & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) `[virtual]`

Defines this AMRLevel.

- a_coarser_level_ptr (not modified): pointer to next coarser level object.
- a_problem_domain (not modified): problem domain of this level.
- a_level (not modified): index of this level. The base level is zero.
- a_ref_ratio (not modified): the refinement ratio between this level and the next finer level.

Reimplemented in **ParticleAMRNS** (p. 454).

7.4.3.7 virtual Real AMRLevel::dt () const `[virtual]`

Returns the current value of the time step.

7.4.3.8 virtual void AMRLevel::dt (Real *a_dt*) `[virtual]`

Sets the time step to a_dt.

7.4.3.9 virtual void AMRLevel::finerLevelPtr (AMRLevel * *a_finer_level_ptr*) `[virtual]`

Sets the pointer-to-finer-level member to a_finer_level_ptr.

7.4.3.10 virtual void AMRLevel::initialData () `[pure virtual]`

Initializes data.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 456).

7.4.3.11 virtual Real AMRLevel::initialDtMultiplier () const `[virtual]`

Returns the initial dt multiplier.

7.4.3.12 virtual void AMRLevel::initialDtMultiplier (Real *a_initial_dt_multiplier*) `[virtual]`

Sets the initial dt multiplier to a_initial_dt_multiplier.

7.4.3.13 virtual void AMRLevel::initialGrid (const Vector< Box > & *a_new_grids*) [pure virtual]

Initializes this level to have the specified domain a_new_grids.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 456).

7.4.3.14 bool AMRLevel::isDefined () const

Returns true if any **AMRLevel::define** function has been called, false otherwise.

7.4.3.15 virtual void AMRLevel::postInitialize () [pure virtual]

Things to do after initialization.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 458).

7.4.3.16 virtual void AMRLevel::postRegrid (int *a_base_level*) [virtual]

Performs any post-regridding operations which are necessary.

This is not a pure virtual function to preserve compatibility with earlier versions of AMRLevel. The **AMRLevel::postRegrid()** instantiation is a no-op.

Reimplemented in **ParticleAMRNS** (p. 458).

7.4.3.17 virtual void AMRLevel::postTimeStep () [pure virtual]

Things to do after advancing this level by one time step.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 458).

7.4.3.18 virtual const ProblemDomain& AMRLevel::problemDomain () const [virtual]

Returns the problem domain of this level.

7.4.3.19 virtual int AMRLevel::refRatio () const [virtual]

Returns the refinement ratio between this level and the next finer level.

7.4.3.20 virtual void AMRLevel::regrid (const Vector< Box > & *a_new_grids*) [pure virtual]

Redefines this level to have the specified domain a_new_grids.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 459).

7.4.3.21 virtual void AMRLevel::tagCells (IntVectSet & *a_tags*) [pure virtual]

Creates tagged cells for dynamic mesh refinement.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 459).

7.4.3.22 virtual void AMRLevel::tagCellsInit (IntVectSet & *a_tags*) [pure virtual]

Creates tagged cells for mesh refinement at initialization.

This is a pure virtual function and MUST be defined in the derived class.

Implemented in **ParticleAMRNS** (p. 460).

7.4.3.23 virtual Real AMRLevel::time () const [virtual]

Returns the current value of the time on this level.

7.4.3.24 virtual void AMRLevel::time (Real *a_time*) [virtual]

Sets the time to a_time.

7.4.3.25 int AMRLevel::verbosity () [static]

Returns current verbosity level. Minimum verbosity is 0, for which nothing is printed.

7.4.3.26 void AMRLevel::verbosity (int *a_verbosity*) [static]

Sets verbosity level to a_verbosity. Minimum verbosity is 0, for which nothing is printed.

7.4.4 Member Data Documentation

7.4.4.1 AMRLevel* AMRLevel::m_coarser_level_ptr [protected]

7.4.4.2 Real AMRLevel::m_dt [protected]

7.4.4.3 AMRLevel* AMRLevel::m_finer_level_ptr [protected]

7.4.4.4 Real AMRLevel::m_initial_dt_multiplier [protected]

7.4.4.5 bool AMRLevel::m_isDefined [protected]

7.4.4.6 int AMRLevel::m_level [protected]

7.4.4.7 Vector<Box> AMRLevel::m_level_grids [protected]

7.4.4.8 ProblemDomain AMRLevel::m_problem_domain [protected]

7.4.4.9 int AMRLevel::m_ref_ratio [protected]

7.4.4.10 Real AMRLevel::m_time [protected]

7.4.4.11 int AMRLevel::s_verbosity [static, protected]

The documentation for this class was generated from the following file:

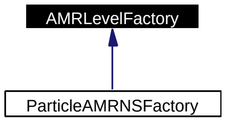
- **AMRLevel.H**

7.5 AMRLevelFactory Class Reference

Factory class to produce an **AMRLevel**.

`#include <AMRLevelFactory.H>`

Inheritance diagram for AMRLevelFactory:



Public Member Functions

- virtual **AMRLevel** * **new_amrlevel** () const=0

7.5.1 Detailed Description

Factory class to produce an **AMRLevel**.

This is needed for **LevelData** construction.

7.5.2 Member Function Documentation

7.5.2.1 virtual AMRLevel* AMRLevelFactory::new_amrlevel () const `[pure virtual]`

Factory function. Creates a new **AMRLevel**. In your instantiation you should include defining all the stuff that is not part of the **AMRLevel** API.

Implemented in **ParticleAMRNSFactory** (p. 470).

The documentation for this class was generated from the following file:

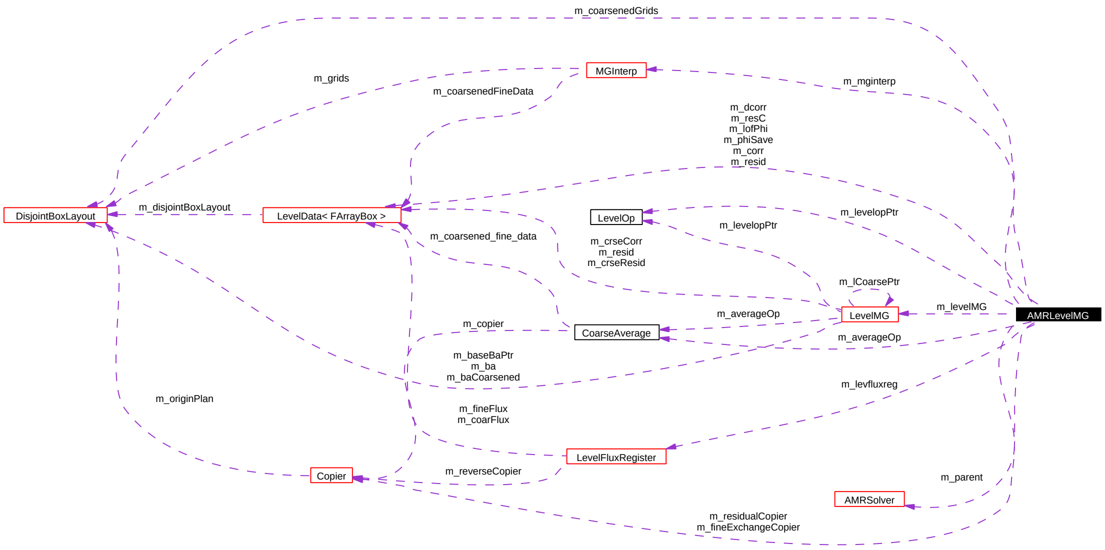
- **AMRLevelFactory.H**

7.6 AMRLevelMG Class Reference

Class which manages data and operations on a level for **AMRSolver**.

#include <AMRLevelMG.H>

Collaboration diagram for AMRLevelMG:



Public Member Functions

- **AMRLevelMG** ()

  Null constructor.

- ~**AMRLevelMG** ()

  Destructor.

- **AMRLevelMG** (const **AMRSolver** ∗const a_parent, int a_level, const **LevelOp** ∗const a_opin, int a_ncomp=1)

  Define AMRLevelMG from the inputs.

- void **define** (const **AMRSolver** ∗const a_parent, int a_level, const **LevelOp** ∗const a_opin, int a_ncomp=1)

  Define AMRLevelMG from the inputs.

- void **setNumSmoothUp** (int a_numSmoothUp)

  Set number of multigrid smoothings on way up V-cycle; Default == 4.

- void **setNumSmoothDown** (int a_numSmoothDown)

  Set number of multigrid smoothings on way down V-cycle; Default == 4.

- void **applyAMROperator** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, **LevelData**< **FArrayBox** > &a_Lofphi)

Complete, 3-level operator.

- void **applyAMROperatorHphys** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, **LevelData**< **FArrayBox** > &a_LofPhi)

  Complete, 3-level operator with homogeneous physicalboundary conditions.

- void **computeAMRResidual** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)

  Compute complete, 3-level residual and put it into local data m_resid.

- void **computeAMRResidualHphys** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)

  Compute complete, 3-level residual with homogeneous physical BCs.

- void **computeAMRResidual** (**LevelData**< **FArrayBox** > &a_resid, **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)

  Compute complete, 3-level residual and return it in a_resid.

- void **upSweep** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)

  Sweep up multigrid v-cycle.

- void **downSweep** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)

  Sweep down v-cycle.

- **Vector**< **Real** > **computeResidualNorm** (int normType) const

  Compute and return norm of internal data m_resid.

- **LevelOp** ∗ **levelOpPtr** () const

  Return pointer to the **LevelOp** *used to define this AMRLevelMG.*

- bool **isDefined** () const

  Returns true if this object has been defined.

- **Vector**< **Real** > **computeNorm** (const **LevelData**< **FArrayBox** > &a_mfinput, int a_normType) const

  Returns normType norm of mfab.

- void **setConvergenceMetric** (**Real** a_metric, int a_comp)

  sets convergences metrics (if appropriate) for this problem.

Protected Member Functions

- void **smooth** (**LevelData**< **FArrayBox** > &a_soln, const **LevelData**< **FArrayBox** > &a_rhs)
- void **clearMemory** ()
- void **setDefaultValues** ()

- void **reflux** (**Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel, **LevelData**< **FArrayBox** > &a_Lofphi)
- void **initFRCoarse** (**Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel)
- void **incrementFRFine** (**Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel)

Protected Attributes

- const **AMRSolver** * **m_parent**
- **LevelMG m_levelMG**
- **MGInterp m_mginterp**
- **LevelFluxRegister m_levfluxreg**
- bool **m_isDefined**
- **DisjointBoxLayout m_coarsenedGrids**
- **LevelData**< **FArrayBox** > **m_resid**
- **LevelData**< **FArrayBox** > **m_corr**
- **LevelData**< **FArrayBox** > **m_dcorr**
- **LevelData**< **FArrayBox** > **m_lofPhi**
- **LevelData**< **FArrayBox** > **m_phiSave**
- **LevelData**< **FArrayBox** > **m_resC**
- **Copier m_residualCopier**
- **Copier m_fineExchangeCopier**
- int **m_level**
- **LevelOp** * **m_levelopPtr**
- **CoarseAverage m_averageOp**
- bool **m_arrayViewVerbose**

Friends

- class **AMRSolver**

7.6.1 Detailed Description

Class which manages data and operations on a level for **AMRSolver**.

AMRLevelMG manages the operations on a level for **AMRSolver**. **AMRSolver** contains a vector of AMRLevelMG's (one for each level). This class should be considered internal to **AMRSolver** and should not be considered part of the Chombo API.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 AMRLevelMG::AMRLevelMG ()

Null constructor.

7.6.2.2 AMRLevelMG::~AMRLevelMG ()

Destructor.

7.6.2.3 AMRLevelMG::AMRLevelMG (const AMRSolver *const *a_parent*, int *a_level*, const LevelOp *const *a_opin*, int *a_ncomp* = 1)

Define AMRLevelMG from the inputs.

7.6.3 Member Function Documentation

7.6.3.1 void AMRLevelMG::applyAMROperator (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, LevelData< FArrayBox > & *a_Lofphi*)

Complete, 3-level operator.

Applies multilevel operator with inhomogeneous coarse-fine boundary conditions (and inhomogeneous domain boundary conditions. Refluxing is used to compute the operator at coarse-fine boundaries.

7.6.3.2 void AMRLevelMG::applyAMROperatorHphys (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, LevelData< FArrayBox > & *a_LofPhi*)

Complete, 3-level operator with homogeneous physicalboundary conditions.

Applies multilevel operator with inhomogeneous physical boundary conditions. Refluxing is used to compute the operator at coarse-fine boundaries.

7.6.3.3 void AMRLevelMG::clearMemory () `[protected]`

Delete internal memory and set pointers to null

7.6.3.4 void AMRLevelMG::computeAMRResidual (LevelData< FArrayBox > & *a_resid*, Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*)

Compute complete, 3-level residual and return it in a_resid.

7.6.3.5 void AMRLevelMG::computeAMRResidual (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*)

Compute complete, 3-level residual and put it into local data m_resid.

7.6.3.6 void AMRLevelMG::computeAMRResidualHphys (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*)

Compute complete, 3-level residual with homogeneous physical BCs.

Compute complete, 3-level residual with homogeneous physical BCs and inhomogeneous C/F BCs and put it into local data m_resid

7.6.3.7 Vector<Real> AMRLevelMG::computeNorm (const LevelData< FArrayBox > & *a_mfinput*, int *a_normType*) const

Returns normType norm of mfab.

7.6.3.8 Vector<Real> AMRLevelMG::computeResidualNorm (int *normType*) const

Compute and return norm of internal data m_resid.

7.6.3.9 void AMRLevelMG::define (const AMRSolver *const *a_parent*, int *a_level*, const LevelOp *const *a_opin*, int *a_ncomp* = 1)

Define AMRLevelMG from the inputs.

The AMRLevelMG will only work for the input **AMRSolver**. A **LevelMG** is created.

7.6.3.10 void AMRLevelMG::downSweep (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*)

Sweep down v-cycle.

7.6.3.11 void AMRLevelMG::incrementFRFine (Vector< LevelData< FArrayBox > * > & *a_phiLevel*) `[protected]`

Increment fluxregisters with fine flux

7.6.3.12 void AMRLevelMG::initFRCoarse (Vector< LevelData< FArrayBox > * > & *a_phiLevel*) `[protected]`

Fill fluxregisters with coarse flux

7.6.3.13 bool AMRLevelMG::isDefined () const

Returns true if this object has been defined.

7.6.3.14 LevelOp* AMRLevelMG::levelOpPtr () const

Return pointer to the **LevelOp** used to define this AMRLevelMG.

7.6.3.15 void AMRLevelMG::reflux (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, LevelData< FArrayBox > & *a_Lofphi*) `[protected]`

Reflux enforces flux-matching from finer levels. Steps involved:

- a) Initialize flux register with coarse flux
- b) Increment flux register with sum of fine fluxes
- b) Modify solution with flux diffrence

7.6.3.16 void AMRLevelMG::setConvergenceMetric (Real *a_metric*, int *a_comp*)

sets convergences metrics (if appropriate) for this problem.

If we're using this class in the context of a solve, we may wish to set a metric against which to definte convergence (for example, the norm of the RHS). As currently implemented, this is simply a pass-through to the **LevelOp** classes from this class.

7.6.3.17 void AMRLevelMG::setDefaultValues () [protected]

Set default values. does not deal with memory

7.6.3.18 void AMRLevelMG::setNumSmoothDown (int *a_numSmoothDown*)

Set number of multigrid smoothings on way down V-cycle; Default == 4.

7.6.3.19 void AMRLevelMG::setNumSmoothUp (int *a_numSmoothUp*)

Set number of multigrid smoothings on way up V-cycle; Default == 4.

7.6.3.20 void AMRLevelMG::smooth (LevelData< FArrayBox > & *a_soln*, const LevelData< FArrayBox > & *a_rhs*) [protected]

Use short v-cycle or levelmg's smoother to reduce high wave numbers of residual

7.6.3.21 void AMRLevelMG::upSweep (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*)

Sweep up multigrid v-cycle.

7.6.4 Friends And Related Function Documentation

7.6.4.1 friend class AMRSolver [friend]

7.6.5 Member Data Documentation

7.6.5.1 bool AMRLevelMG::m_arrayViewVerbose [protected]

Debugging option to use ArrayView to examine what happens

7.6.5.2 CoarseAverage AMRLevelMG::m_averageOp [protected]

Used for averaging residual to coarser level

7.6.5.3 DisjointBoxLayout AMRLevelMG::m_coarsenedGrids [protected]

Coarsened version of the grids at this level

7.6.5.4 LevelData<FArrayBox> AMRLevelMG::m_corr [protected]

Correction

7.6.5.5 LevelData<FArrayBox> AMRLevelMG::m_dcorr [protected]

The infamous correction to the correction

7.6.5.6 Copier AMRLevelMG::m_fineExchangeCopier [protected]

Copier that's used for exchange on the fine grid data before fine flux register data is incremented.

7.6.5.7 bool AMRLevelMG::m_isDefined [protected]

Has this AMRLevelMG been defined yet?

7.6.5.8 int AMRLevelMG::m_level [protected]

Which **AMR** Level we are on

7.6.5.9 LevelMG AMRLevelMG::m_levelMG [protected]

LevelMG level solver object to relax on this level

7.6.5.10 LevelOp* AMRLevelMG::m_levelopPtr [protected]

The **LevelOp** which encapsulates which operator we're using in this solve.

7.6.5.11 LevelFluxRegister AMRLevelMG::m_levfluxreg [protected]

Flux register with next coarser level

7.6.5.12 LevelData<FArrayBox> AMRLevelMG::m_lofPhi [protected]

Storage for L(phi)

7.6.5.13 MGInterp AMRLevelMG::m_mginterp [protected]

For interpolation

7.6.5.14 const AMRSolver* AMRLevelMG::m_parent [protected]

The parent **AMRSolver** which manages the overall multilevel solve

7.6.5.15 LevelData<FArrayBox> AMRLevelMG::m_phiSave [protected]

Extra copy of solution

7.6.5.16 LevelData<FArrayBox> AMRLevelMG::m_resC [protected]

Storage for coarsened residual

7.6.5.17 LevelData<FArrayBox> AMRLevelMG::m_resid [protected]

Residual

7.6.5.18 Copier AMRLevelMG::m_residualCopier [protected]

Copier that operates for moving coarsened residual to coarse level grid space.

The documentation for this class was generated from the following file:

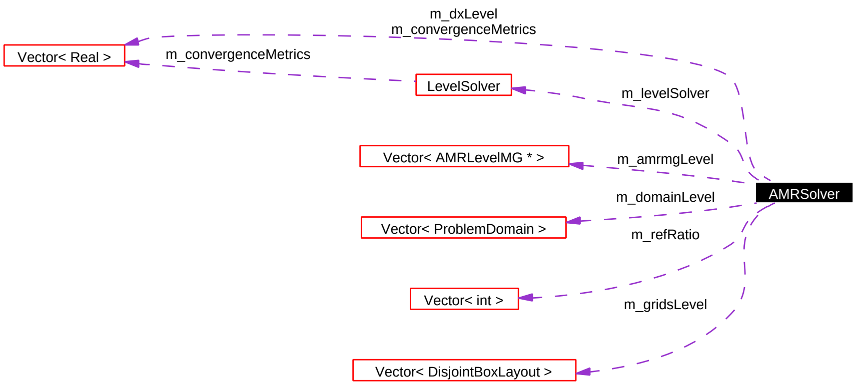
- **AMRLevelMG.H**

7.7 AMRSolver Class Reference

Class which manages grid hierarchy and composite elliptic solution.

`#include <AMRSolver.H>`

Collaboration diagram for AMRSolver:



Public Member Functions

- **AMRSolver** ()

  Creates a AMRSolver whose internal state is undefined.

- **AMRSolver** (const **Vector**< **DisjointBoxLayout** > &a_gridsLevel, const **Vector**< **Box** > &a_domainLevel, const **Vector**< **Real** > &a_dxLevel, const **Vector**< int > &a_refRatio, int a_numLevels, int a_lBase, const **LevelOp** ∗const a_opin, int a_ncomp=1, bool a_limitCoarsening=false)

  Creates a fully-defined AMRSolver.

- **AMRSolver** (const **Vector**< **DisjointBoxLayout** > &a_gridsLevel, const **Vector**< **ProblemDomain** > &a_domainLevel, const **Vector**< **Real** > &a_dxLevel, const **Vector**< int > &a_refRatio, int a_numLevels, int a_lBase, const **LevelOp** ∗const a_opin, int a_ncomp=1, bool a_limitCoarsening=false)

  Creates a fully-defined AMRSolver.

- void **define** (const **Vector**< **DisjointBoxLayout** > &a_gridsLevel, const **Vector**< **Box** > &a_domainLevel, const **Vector**< **Real** > &a_dxLevel, const **Vector**< int > &a_refRatio, int a_numLevels, int a_lBase, const **LevelOp** ∗const a_opin, int a_ncomp=1, bool a_limitCoarsening=false)

  Defines AMRSolver object's internal state.

- void **define** (const **Vector**< **DisjointBoxLayout** > &a_gridsLevel, const **Vector**< **ProblemDomain** > &a_domainLevel, const **Vector**< **Real** > &a_dxLevel, const **Vector**< int > &a_refRatio, int a_numLevels, int a_lBase, const **LevelOp** ∗const a_opin, int a_ncomp=1, bool a_limitCoarsening=false)

  Defines AMRSolver object's internal state.

- **~AMRSolver** ()

Destructor.

- bool **isDefined** () const

  Returns true if full define function has been called.

- void **setNumSmoothUp** (int a_numSmoothUp)

  Set number of multigrid smoothings on way up v-cycle. Default is 4.

- void **setNumSmoothDown** (int a_numSmoothDown)

  Set number of multigrid smoothings on way down v-cycle; Default is 4.

- void **setTolerance** (**Real** a_tolerance)

  Set tolerance of iterative solution. Default is 1.0e-10.

- void **setOperatorTolerance** (**Real** a_operatorTolerance)

  Set "operator tolerance" of iterative solution.

- void **setMaxIter** (int a_maxIter)

  Set max number of iterations. Default is 42.

- void **setMinIter** (int a_minIter)

  Set min number of iterations.

- void **setNumVCyclesBottom** (int a_numVCycleBottom)

  Set the number of V-cycles performed at the bottom of the **AMR** *v-cycle.*

- void **setConvergenceMetric** (**Real** a_metric, int a_comp)

  set the convergence metrics (if appropriate) for this problem

- void **solveAMR** (**Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > * > &a_rhsLevel, bool a_initializePhiToZero=true)

  Solves on hierarchy to tolerance m_tolerance.

- void **AMRVCycleMG** (**Vector**< **LevelData**< **FArrayBox** > * > &a_corrLevel, const **Vector**< **LevelData**< **FArrayBox** > * > &a_residLevel)

  Does one relaxation V-cycle using a MG solver.

- **Vector**< **Real** > **computeResidualNorm** (**Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > * > &a_rhsLevel, int a_normType)

  Calculate norm of multilevel residual on levels lBase to lmax.

- **Vector**< **Real** > **computeResidualNorm** (**Vector**< **LevelData**< **FArrayBox** > * > &a_residLevel, **Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > * > &a_rhsLevel, int a_normType)

  Calculate norm of multilevel residual on levels lBase to lmax.

- void **computeAMRResidual** (**Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > * > &a_rhsLevel, **LevelData**< **FArrayBox** > &a_res, int a_ilev)

Calculate multilevel residual on level ilev.

- void **applyAMROperator** (**Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel, **LevelData**< **FArrayBox** > &a_LofPhi, int a_ilev)

  Calculate multilevel L(phi). includes refluxing and all that.

- void **applyAMROperatorHphys** (**Vector**< **LevelData**< **FArrayBox** > * > &a_phiLevel, **LevelData**< **FArrayBox** > &a_LofPhi, int a_ilev)

  Calculate multilevel L(phi) with homogeneous physical BCs.

- void **setVerbose** (bool a_verbose)

  set whether the solver does i/o while solving. default is true

Protected Member Functions

- void **setDefaultValues** ()
- void **clear** ()
- **Vector**< **Real** > **computeNorm** (const **Vector**< **LevelData**< **FArrayBox** > * > &a_phiVect, int a_normType)

Protected Attributes

- **Real m_errorTolerance**
- **Real m_tolerance**
- **Vector**< **Real** > **m_convergenceMetrics**
- **Real m_operatorTolerance**
- int **m_numLevels**
- int **m_finestLevel**
- **Vector**< int > **m_refRatio**
- **Vector**< **Real** > **m_dxLevel**
- **Vector**< **DisjointBoxLayout** > **m_gridsLevel**
- **Vector**< **ProblemDomain** > **m_domainLevel**
- int **m_maxIter**
- int **m_minIter**
- int **m_numSmoothUp**
- int **m_numVCyclesBottom**
- int **m_numSmoothDown**
- **Vector**< **AMRLevelMG** * > **m_amrmgLevel**
- bool **m_isDefined**
- **LevelSolver m_levelSolver**
- int **m_lBase**
- int **m_ncomp**
- bool **m_verbose**

Friends

- class **AMRLevelMG**

7.7.1 Detailed Description

Class which manages grid hierarchy and composite elliptic solution.

AMRSolver manages the AMR/multigrid solution to the elliptic equation on a multiple level grid that satisfies certain proper nesting conditions. It can be used either as a solver, or to apply the **AMR** / multigrid V-cycle as a preconditioner for some other iterative method, such as a Krylov method.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 AMRSolver::AMRSolver ()

Creates a AMRSolver whose internal state is undefined.

Need to call define(...) function to use any of the functionality of the class

7.7.2.2 AMRSolver::AMRSolver (const Vector< DisjointBoxLayout > & *a_gridsLevel*, const Vector< Box > & *a_domainLevel*, const Vector< Real > & *a_dxLevel*, const Vector< int > & *a_refRatio*, int *a_numLevels*, int *a_lBase*, const LevelOp *const *a_opin*, int *a_ncomp* = 1, bool *a_limitCoarsening* = false)

Creates a fully-defined AMRSolver.

Calls define function with identical arguments

7.7.2.3 AMRSolver::AMRSolver (const Vector< DisjointBoxLayout > & *a_gridsLevel*, const Vector< ProblemDomain > & *a_domainLevel*, const Vector< Real > & *a_dxLevel*, const Vector< int > & *a_refRatio*, int *a_numLevels*, int *a_lBase*, const LevelOp *const *a_opin*, int *a_ncomp* = 1, bool *a_limitCoarsening* = false)

Creates a fully-defined AMRSolver.

Calls define function with identical arguments

7.7.2.4 AMRSolver::~AMRSolver ()

Destructor.

7.7.3 Member Function Documentation

7.7.3.1 void AMRSolver::AMRVCycleMG (Vector< LevelData< FArrayBox > * > & *a_corrLevel*, const Vector< LevelData< FArrayBox > * > & *a_residLevel*)

Does one relaxation V-cycle using a MG solver.

Problem is assumed to already be in residual-correction form.

Inputs:

- **corrLevel** - pointers to current guess at the correction values for levels lMin = max(lBase-1,0) ... lMax. **Vector** index corresponds to level number.

- **residLevel** - pointers to **AMR** residual for levels lMin ... lMax. **Vector** index corresponds to level number.

Outputs:

- **corrLevel - LevelData<FArrayBox>**s pointed to for levels lMin,...,lMax are updated in place.

7.7.3.2 void AMRSolver::applyAMROperator (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, LevelData< FArrayBox > & *a_LofPhi*, int *a_ilev*)

Calculate multilevel L(phi). includes refluxing and all that.

Includes refluxing and all that. This is the three-level operator.

7.7.3.3 void AMRSolver::applyAMROperatorHphys (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, LevelData< FArrayBox > & *a_LofPhi*, int *a_ilev*)

Calculate multilevel L(phi) with homogeneous physical BCs.

Calculate multilevel L(phi) with homogeneous physical BCs (but with inhomogeneous C/F BCs). includes refluxing and all that This is the three-level operator.

7.7.3.4 void AMRSolver::clear () `[protected]`

Clear internal storage

7.7.3.5 void AMRSolver::computeAMRResidual (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*, LevelData< FArrayBox > & *a_res*, int *a_ilev*)

Calculate multilevel residual on level ilev.

7.7.3.6 Vector<Real> AMRSolver::computeNorm (const Vector< LevelData< FArrayBox > * > & *a_phiVect*, int *a_normType*) `[protected]`

compute multilevel norm on levels lBase to lMax for all components in a_phiVect. Multilevel norm does not include covered regions.

7.7.3.7 Vector<Real> AMRSolver::computeResidualNorm (Vector< LevelData< FArrayBox > * > & *a_residLevel*, Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*, int *a_normType*)

Calculate norm of multilevel residual on levels lBase to lmax.

Does not include data covered by finer levels. Also returns residual in argument a_residLevel.

7.7.3.8 Vector<Real> AMRSolver::computeResidualNorm (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*, int *a_normType*)

Calculate norm of multilevel residual on levels lBase to lmax.

Does not include data covered by finer levels.

7.7.3.9 void AMRSolver::define (const Vector< DisjointBoxLayout > & *a_gridsLevel*, const Vector< ProblemDomain > & *a_domainLevel*, const Vector< Real > & *a_dxLevel*, const Vector< int > & *a_refRatio*, int *a_numLevels*, int *a_lBase*, const LevelOp *const *a_opin*, int *a_ncomp* = 1, bool *a_limitCoarsening* = false)

Defines AMRSolver object's internal state.

Inputs:

- **gridsLevel** – The grids at all levels. Each element in the vector is a level's worth of grids. gridsLevel[0] are grids for level 0 and so forth. **Vector** index corresponds to level number.
- **domainLevel** – The domains at all levels. Each element in the vector is a level's domain. domainLevel[0] is the domain for level 0 and so forth. **Vector** index corresponds to level number.
- **dxLevel** – The grid spacing at all levels. Each element in the vector is a level's $$. dxLevel[0] – is $$ for level 0 and so forth. **Vector** index corresponds to level number.
- **refRatio** – The refinement ratio between all levels. refRatio[0] is the refinement ratio between level 0 and level 1; **Vector** index corresponds to level number.
- **numlevels** – The number of **AMR** levels in the calculation. The length of the **Vector** arguments has to be at least numlevels.
- **lBase** – coarsest level on which solution is to be computed. This needs to be set at the time of definition, in order to build the bottom **LevelSolver**.
- **a_opin** – The levelop to use in the solution.
- **a_limitCoarsening**: if true, only do multigrid coarsening down to next coarser **AMR** level (only coarsen by a_nRefCrse). If false, coarsen as far as possible. Only relevant when lBase > 0.

7.7.3.10 void AMRSolver::define (const Vector< DisjointBoxLayout > & *a_gridsLevel*, const Vector< Box > & *a_domainLevel*, const Vector< Real > & *a_dxLevel*, const Vector< int > & *a_refRatio*, int *a_numLevels*, int *a_lBase*, const LevelOp *const *a_opin*, int *a_ncomp* = 1, bool *a_limitCoarsening* = false)

Defines AMRSolver object's internal state.

Inputs:

- **gridsLevel** – The grids at all levels. Each element in the vector is a level's worth of grids. gridsLevel[0] are grids for level 0 and so forth. **Vector** index corresponds to level number.

- **domainLevel** – The domains at all levels. Each element in the vector is a level's domain. domainLevel[0] is the domain for level 0 and so forth. **Vector** index corresponds to level number.
- **dxLevel** – The grid spacing at all levels. Each element in the vector is a level's $$. dxLevel[0] – is $$ for level 0 and so forth. **Vector** index corresponds to level number.
- **refRatio** – The refinement ratio between all levels. refRatio[0] is the refinement ratio between level 0 and level 1; **Vector** index corresponds to level number.
- **numlevels** – The number of **AMR** levels in the calculation. The length of the **Vector** arguments has to be at least numlevels.
- **lBase** – coarsest level on which solution is to be computed. This needs to be set at the time of definition, in order to build the bottom **LevelSolver**.
- **a_opin** – The levelop to use in the solution.
- **a_limitCoarsening**: if true, only do multigrid coarsening down to next coarser **AMR** level (only coarsen by a_nRefCrse). If false, coarsen as far as possible. Only relevant when lBase > 0.

7.7.3.11 bool AMRSolver::isDefined () const

Returns true if full define function has been called.

7.7.3.12 void AMRSolver::setConvergenceMetric (Real *a_metric*, int *a_comp*)

set the convergence metrics (if appropriate) for this problem

sets metric against which we define convergence (for example, the norm of the RHS). Also passes a_metric through to the **LevelMG** (and eventually to a bottom solver.)

7.7.3.13 void AMRSolver::setDefaultValues () `[protected]`

Set data to default values. not for external use

7.7.3.14 void AMRSolver::setMaxIter (int *a_maxIter*)

Set max number of iterations. Default is 42.

7.7.3.15 void AMRSolver::setMinIter (int *a_minIter*)

Set min number of iterations.

Only relevant when residual is not decreasing fast enough to satisfy the "operator tolerance". Default is 4.

7.7.3.16 void AMRSolver::setNumSmoothDown (int *a_numSmoothDown*)

Set number of multigrid smoothings on way down v-cycle; Default is 4.

7.7.3.17 void AMRSolver::setNumSmoothUp (int *a_numSmoothUp*)

Set number of multigrid smoothings on way up v-cycle. Default is 4.

7.7.3.18 void AMRSolver::setNumVCyclesBottom (int *a_numVCycleBottom*)

Set the number of V-cycles performed at the bottom of the **AMR** v-cycle.

Set the number of v-cycles performed from the base level to the maximum coarsening of the base level during one overall v-cycle on the **AMR** hierarchy. Default is 1.

7.7.3.19 void AMRSolver::setOperatorTolerance (Real *a_operatorTolerance*)

Set "operator tolerance" of iterative solution.

Iteration will stop if (new_residual/old_residual) > 1-operatorTolerance (and at least minIter iterations have been performed). Default is 1.0e-5.

7.7.3.20 void AMRSolver::setTolerance (Real *a_tolerance*)

Set tolerance of iterative solution. Default is 1.0e-10.

7.7.3.21 void AMRSolver::setVerbose (bool *a_verbose*)

set whether the solver does i/o while solving. default is true

7.7.3.22 void AMRSolver::solveAMR (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*, bool *a_initializePhiToZero* = true)

Solves on hierarchy to tolerance m_tolerance.

Solves the elliptic equation over the hierarchy of levels lBase ... lMax where $lMax = numlevels-1$. If lBase > 0, then the data at level lBase - 1 is used to interpolate boundary conditions at boundary cells that are not adjacent to the domain boundary.

Inputs:

- **phiLevel** - pointers to current guess at the solution values for levels (lMin = max(lBase-1,0)) ... lMax. **Vector** index corresponds to level number.

- **rhsLevel** - pointers to right-hand side for levels lmin ... lMax. **Vector** index corresponds to level number.

Outputs:

- phiLevel - **LevelData<FArrayBox>**s pointed to for levels lMin,...,lMax are updated in place.

7.7.4 Friends And Related Function Documentation

7.7.4.1 friend class AMRLevelMG [friend]

7.7.5 Member Data Documentation

7.7.5.1 Vector<AMRLevelMG*> AMRSolver::m_amrmgLevel [protected]

Array of AMRLevelMGs containing data at each refinement level

7.7.5.2 Vector<Real> AMRSolver::m_convergenceMetrics [protected]

metrics used to judge convergence

7.7.5.3 Vector<ProblemDomain> AMRSolver::m_domainLevel [protected]

Domains.

7.7.5.4 Vector<Real> AMRSolver::m_dxLevel [protected]

Grid spacing.

7.7.5.5 Real AMRSolver::m_errorTolerance [protected]

Error tolerance for error estimation

7.7.5.6 int AMRSolver::m_finestLevel [protected]

Maximum number of working level is m_numLevels - 1

7.7.5.7 Vector<DisjointBoxLayout> AMRSolver::m_gridsLevel [protected]

Grids.

7.7.5.8 bool AMRSolver::m_isDefined [protected]

Has this solver been defined?

7.7.5.9 int AMRSolver::m_lBase [protected]

7.7.5.10 LevelSolver AMRSolver::m_levelSolver [protected]

7.7.5.11 int AMRSolver::m_maxIter [protected]

Maximum number of solver iterations

7.7.5.12 int AMRSolver::m_minIter [protected]

Minimum number of solver iterations before "hung converence" criteria is valid

7.7.5.13 int AMRSolver::m_ncomp [protected]

7.7.5.14 int AMRSolver::m_numLevels [protected]

Number of levels allowed

7.7.5.15 int AMRSolver::m_numSmoothDown [protected]

Number of smoother iterations on down cycle

7.7.5.16 int AMRSolver::m_numSmoothUp [protected]

Number of smoother iterations on up cycle

7.7.5.17 int AMRSolver::m_numVCyclesBottom [protected]

Number of VCycles for max iter of levelsolver

7.7.5.18 Real AMRSolver::m_operatorTolerance [protected]

Tolerance on maxOperator

7.7.5.19 Vector<int> AMRSolver::m_refRatio [protected]

Refinement ratio.

7.7.5.20 Real AMRSolver::m_tolerance [protected]

Solver tolerance

7.7.5.21 bool AMRSolver::m_verbose [protected]

default is true

The documentation for this class was generated from the following file:

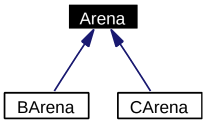
- **AMRSolver.H**

7.8 Arena Class Reference

A Virtual Base Class for Dynamic Memory Managemen.

`#include <Arena.H>`

Inheritance diagram for Arena:



Public Types

- typedef void(∗ **FP**)()

Public Member Functions

- **Arena** ()

    base class constructor

- virtual ∼**Arena** ()

    base class destructor.

- virtual void ∗ **alloc** (size_t sz)=0
- virtual void **free** (void ∗pt)=0

Static Public Member Functions

- size_t **align** (size_t sz)

7.8.1 Detailed Description

A Virtual Base Class for Dynamic Memory Managemen.

A Virtual Base Class for Dynamic Memory Management

This is a virtual base class for objects that manage their own dynamic memory allocation. Since it is a virtual base class, you have to derive something from it to use it.

7.8.2 Member Typedef Documentation

7.8.2.1 typedef void(∗ Arena::FP)()

7.8.3 Constructor & Destructor Documentation

7.8.3.1 Arena::Arena ()

base class constructor

base class constructor. registers Arena object with memory tracking system

7.8.3.2 virtual Arena::∼Arena () `[virtual]`

base class destructor.

base class destructor. unregisters Arena object with memory tracking system

7.8.4 Member Function Documentation

7.8.4.1 size_t Arena::align (size_t *sz*) `[inline, static]`

: Given a minimum required arena size of sz bytes, this returns the next largest arena size that will hold an integral number of objects of the largest of the types void∗, long, double and function pointer.

7.8.4.2 virtual void∗ Arena::alloc (size_t *sz*) `[pure virtual]`

: Allocate a dynamic memory arena of size sz. A pointer to this memory should be returned.

Implemented in **BArena** (p. 67), and **CArena** (p. 167).

7.8.4.3 virtual void Arena::free (void ∗ *pt*) `[pure virtual]`

: A pure virtual function for deleting the arena pointed to by pt.

Implemented in **BArena** (p. 67), and **CArena** (p. 168).

The documentation for this class was generated from the following file:

- **Arena.H**

7.9 Arena::Word Union Reference

`#include <Arena.H>`

Public Attributes

- void * **p**
- double **d**
- long **l**
- **FP f**

7.9.1 Member Data Documentation

7.9.1.1 double Arena::Word::d

7.9.1.2 FP Arena::Word::f

7.9.1.3 long Arena::Word::l

7.9.1.4 void* Arena::Word::p

The documentation for this union was generated from the following file:

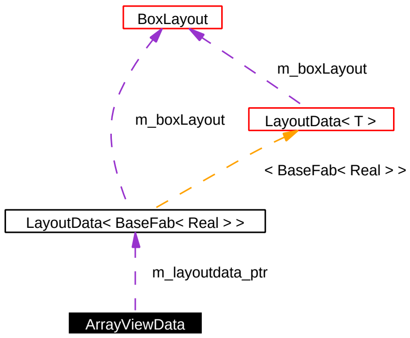
- **Arena.H**

7.10 ArrayViewData Class Reference

Special **LayoutData** type for use with ArrayView.

`#include <ArrayViewData.H>`

Collaboration diagram for ArrayViewData:



Public Member Functions

- **ArrayViewData** ()
- ~**ArrayViewData** ()
- **ArrayViewData** (**LayoutData**< **BaseFab**< **Real** > > *a_layoutdata_ptr)
- **ArrayViewData** (**LayoutData**< **FArrayBox** > *a_layoutdata_ptr)
- **BaseFab**< **Real** > & **operator**[ ] (int a_elem)

    Access a single FAB in the **LayoutData** *using its internal index.*

- void **set** (int a_elem, **BaseFab**< **Real** > *a_fab_ptr)

    Substitute a different FAB into the **LayoutData**.

7.10.1 Detailed Description

Special **LayoutData** type for use with ArrayView.

ArrayViewData is a wrapper around **LayoutData** for **BaseFab<Real>** or **FArrayBox** for use in ArrayView. It provides random access to the individual FABs and the ability to directly set the data in them. It is intended to work only in ArrayView. Any other attept to use this class is unsupported.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 ArrayViewData::ArrayViewData () `[inline]`

7.10.2.2 ArrayViewData::~ArrayViewData () `[inline]`

7.10.2.3 ArrayViewData::ArrayViewData (LayoutData< BaseFab< Real > > * *a_layoutdata_ptr*) `[inline]`

7.10.2.4 ArrayViewData::ArrayViewData (LayoutData< FArrayBox > * *a_layoutdata_ptr*) `[inline]`

7.10.3 Member Function Documentation

7.10.3.1 BaseFab<Real>& ArrayViewData::operator[] (int *a_elem*) `[inline]`

Access a single FAB in the **LayoutData** using its internal index.

7.10.3.2 void ArrayViewData::set (int *a_elem*, BaseFab< Real > * *a_fab_ptr*) `[inline]`

Substitute a different FAB into the **LayoutData**.

The caller is responsible for managing the memory used by the FAB that is being replaced.

The documentation for this class was generated from the following file:

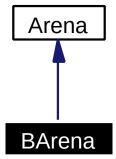
- **ArrayViewData.H**

7.11 BArena Class Reference

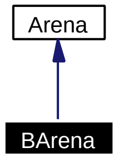
A Concrete Class for Dynamic Memory Management.

`#include <Arena.H>`

Inheritance diagram for BArena:



Collaboration diagram for BArena:



Public Member Functions

- **BArena** (const char *name="unnamed")
- virtual void * **alloc** (size_t sz)
- virtual void **free** (void *pt)

  Deletes the arena pointed to by pt.

7.11.1 Detailed Description

A Concrete Class for Dynamic Memory Management.

This is the simplest dynamic memory management class derived from **Arena**.

Makes calls to ::operator new() and ::operator delete().

7.11.2 Constructor & Destructor Documentation

7.11.2.1 BArena::BArena (const char * *name* = "unnamed")

optional

Parameters:

name used by memory tracker to distinguish between different memory Arenas

7.11.3 Member Function Documentation

7.11.3.1 virtual void* BArena::alloc (size_t *sz*) [virtual]

: Allocates a dynamic memory arena of size sz. Returns a pointer to this memory.

Implements **Arena** (p. 62).

7.11.3.2 virtual void BArena::free (void * *pt*) [virtual]

Deletes the arena pointed to by pt.

Implements **Arena** (p. 62).

The documentation for this class was generated from the following file:

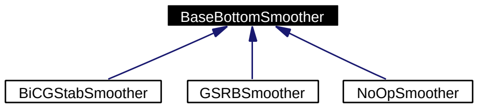
- **Arena.H**

7.12 BaseBottomSmoother Class Reference

Pure base class to encapsulate bottom smooth for LevelOp-derived objects.

`#include <BaseBottomSmoother.H>`

Inheritance diagram for BaseBottomSmoother:



Public Member Functions

- **BaseBottomSmoother** ()

  Null constructor.

- virtual ~**BaseBottomSmoother** ()

  Destructor.

- virtual **BaseBottomSmoother** ∗ **new_bottomSmoother** () const=0

  This gets around the "no virtual constructor" rule.

- virtual void **doBottomSmooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs, **LevelOp** ∗a_levelopPtr)=0

  Do smoothing.

- virtual void **setConvergenceMetric** (**Real** a_metric, int a_comp=0)=0

  sets metric to judge convergence (normally norm(rhs) of outer problem

- virtual void **setSolverTol** (**Real** a_tol)=0

  set solver tolerance

7.12.1 Detailed Description

Pure base class to encapsulate bottom smooth for LevelOp-derived objects.

The BaseBottomSmoother class is a virtual base class for implementing different bottomSmoother algorithms for LevelOp-derived classes. It takes a **LevelOp** pointer as an argument during its definition, in order to access the problem-dependent (equation-dependent) functionality of the specific **LevelOp** derivative class.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 BaseBottomSmoother::BaseBottomSmoother () `[inline]`

Null constructor.

7.12.2.2 virtual BaseBottomSmoother::~BaseBottomSmoother () [inline, virtual]

Destructor.

7.12.3 Member Function Documentation

7.12.3.1 virtual void BaseBottomSmoother::doBottomSmooth (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*, LevelOp * *a_levelopPtr*) [pure virtual]

Do smoothing.

Implemented in **BiCGStabSmoother** (p. 92), **GSRBSmoother** (p. 285), and **NoOpSmoother** (p. 428).

7.12.3.2 virtual BaseBottomSmoother* BaseBottomSmoother::new_bottomSmoother () const [pure virtual]

This gets around the "no virtual constructor" rule.

Implemented in **BiCGStabSmoother** (p. 93), **GSRBSmoother** (p. 285), and **NoOpSmoother** (p. 428).

7.12.3.3 virtual void BaseBottomSmoother::setConvergenceMetric (Real *a_metric*, int *a_comp* = 0) [pure virtual]

sets metric to judge convergence (normally norm(rhs) of outer problem

To make the iterative solver convergence more consistent, the solver will consider itself "solved" if the residual is reduced to epsilon*metric, where epsilon is the solver tolerance (set by setSolverTol). A good starting point for the metric is the norm of the RHS of the original problem (i.e. the rhs sent in to **LevelSolver** or **AMRSolver**). Default value is 1.0.

Implemented in **BiCGStabSmoother** (p. 93), **GSRBSmoother** (p. 285), and **NoOpSmoother** (p. 428).

7.12.3.4 virtual void BaseBottomSmoother::setSolverTol (Real *a_tol*) [pure virtual]

set solver tolerance

Implemented in **BiCGStabSmoother** (p. 93), **GSRBSmoother** (p. 286), and **NoOpSmoother** (p. 428).

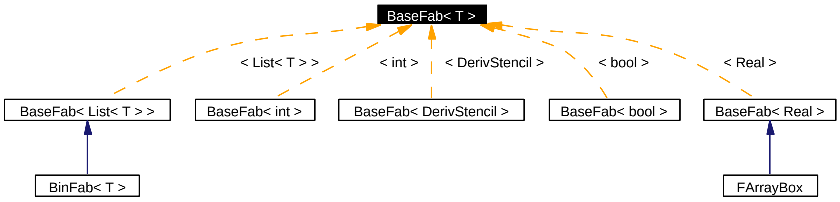
The documentation for this class was generated from the following file:

- **BaseBottomSmoother.H**

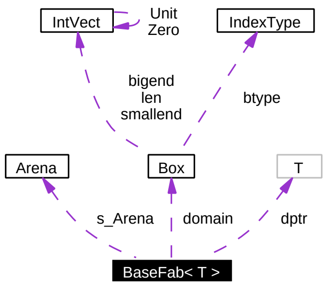
7.13 BaseFab< T > Class Template Reference

`#include <BaseFab.H>`

Inheritance diagram for BaseFab< T >:



Collaboration diagram for BaseFab< T >:



Public Member Functions

- **BaseFab** ()

    { *constructors, destructor and defines*}

- **BaseFab** (const **Box** &bx, int n, T ∗alias=NULL)
- **BaseFab** (const **Interval** &a_comps, **BaseFab**< T > &a_original)
- virtual ∼**BaseFab** ()
- void **resize** (const **Box** &b, int N=1, T ∗alias=NULL)
- virtual void **define** (const **Box** &box, int comps, T ∗alias=NULL)
- virtual void **define** (const **Interval** &a_comps, **BaseFab**< T > &a_original)
- void **clear** ()
- int **nComp** () const

    { *accessors*}

- const **Box** & **box** () const
- const int ∗ **size** () const
- const **IntVect** & **smallEnd** () const
- const **IntVect** & **bigEnd** () const
- **Interval interval** () const
- T & **operator()** (const **IntVect** &p, int N)
- T & **operator()** (const **IntVect** &p)
- const T & **operator()** (const **IntVect** &p, int N) const

- const T & **operator()** (const **IntVect** &p) const
- void **getVal** (T *data, const **IntVect** &pos, int N, int numcomp) const
- void **getVal** (T *data, const **IntVect** &pos) const
- const int * **loVect** () const

  { *Fortran interface functions*}

- const int * **hiVect** () const
- const int * **nCompPtr** () const
- T * **dataPtr** (int N=0)
- const T * **dataPtr** (int N=0) const
- bool **contains** (const **BaseFab**< T > &fab) const

  { *comparison functions*}

- bool **contains** (const **Box** &bx) const
- void **setVal** (T x, const **Box** &bx, int nstart, int ncomp)

  { *data modification functions*}

- void **setVal** (T x, const **Box** &bx, int N)
- void **setVal** (T x, int N)
- void **setVal** (T x)
- **BaseFab**< T > & **copy** (const **BaseFab**< T > &src, const **Box** &srcbox, int srccomp, const **Box** &destbox, int destcomp, int numcomp)
- **BaseFab**< T > & **copy** (const **BaseFab**< T > &src, int srccomp, int destcomp, int numcomp=1)
- **BaseFab**< T > & **copy** (const **BaseFab**< T > &src, const **Box** &destbox)
- **BaseFab**< T > & **copy** (const **BaseFab**< T > &src)
- void **copy** (const **Box** &RegionFrom, const **Interval** &Cdest, const **Box** &RegionTo, const **BaseFab**< T > &src, const **Interval** &Csrc)
- **BaseFab**< T > & **shift** (const **IntVect** &v)

  { *domain modification functions*}

- **BaseFab**< T > & **shift** (int idir, int n_cell)
- **BaseFab**< T > & **shiftHalf** (int dir, int num_halfs)
- **BaseFab**< T > & **shiftHalf** (const **IntVect** &num_halfs)
- virtual int **size** (const **Box** &b, const **Interval** &comps) const

  { *linearization functions*}

- virtual void **linearOut** (void *buf, const **Box** &R, const **Interval** &comps) const
- virtual void **linearIn** (void *buf, const **Box** &R, const **Interval** &comps)

Static Public Member Functions

- int **preAllocatable** ()

Protected Member Functions

- void **define** ()
- void **undefine** ()
- virtual void **performCopy** (const **BaseFab**< T > &src, const **Box** &srcbox, int srccomp, const **Box** &destbox, int destcomp, int numcomp)
- void **performSetVal** (T x, const **Box** &bx, int nstart, int numcomp)

Static Protected Member Functions

- std::string **name** ()

Protected Attributes

- **Box domain**
- int **nvar**
- long **numpts**
- long **truesize**
- T * **dptr**
- bool **aliased**

Static Protected Attributes

- **Arena** * **s_Arena** = NULL

7.13.1 Detailed Description

template<class T> class BaseFab< T >

A Fortran Array-like Container

BaseFab emulates the Fortran array concept. Useful operations can be performed upon BaseFab's in C++, and they provide a convenient interface to Fortran when it is necessary to retreat into that language.

'BaseFab' is a template class. Through use of the template, a 'BaseFab' may be based upon any class. So far at least, most applications have been based upon simple types like 'integer's, 'real's, or 'doubleprecision's. Most applications do not use BaseFab's directly, but utilize specialized classes derived from BaseFab.

It will be easier to use a 'BaseFab' if you understand the following concepts. 'BaseFab' objects depend on the dimensionality of space (indirectly through the DOMAIN 'Box' member). It is typical to define the macro 'CH_SPACEDIM' to be 1, 2, or 3 to indicate the dimension of space. See the discussion of class 'Box' for more information. A 'BaseFab' contains a 'Box' DOMAIN, which indicates the integer indexing space over which the array is defined. A 'BaseFab' also has NVAR components. By components, we mean that for each point in the rectangular indexing space, there are NVAR values associated with that point. A Fortran array corresponding to a 'BaseFab' would have (CH_SPACEDIM+1) dimensions.

By design, the array layout in a 'BaseFab' mirrors that of a Fortran array. The first index (x direction for example) varies most rapidly, the next index (y direction), if any, varies next fastest. The component index varies last, after all the spatial indices.

It is sometimes convenient to be able to treat a sub-array within an existing 'BaseFab' as a 'BaseFab' in its own right. This is often referred to as 'aliasing' the 'BaseFab'. Note that when aliasing is used, the BaseFab's domain will not, in general, be the same as the parent BaseFab's domain, nor will the number of components. BaseFab is a dimension dependent class, so CH_SPACEDIM must be defined as either 1, 2, or 3 when compiling.

This is NOT a polymorphic class.

It does NOT provide a copy constructor or assignment operator.

T MUST have a default constructor and an assignment operator.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 template<class T> BaseFab< T >::BaseFab () [inline]

{ constructors, destructor and defines}

: Constructs an invalid ‘BaseFab’. The domain is invalid, the number of components is zero, and no actual array memory is allocated. An invalid ‘BaseFab’ must be **resize()**d (see ‘BaseFab::resize’) before use.

7.13.2.2 template<class T> BaseFab< T >::BaseFab (const Box & *bx*, int *n*, T * *alias* = NULL) [inline]

Constructs a BaseFab with desired domain and number of components.

7.13.2.3 template<class T> BaseFab< T >::BaseFab (const Interval & *a_comps*, BaseFab< T > & *a_original*) [inline]

Constructs an 'aliased' BaseFab of the requested interval of the argument BaseFab. This BaseFab does not allocate any memory, but sets its data pointer into the memory pointed to by the argument BaseFab. It is the users responsiblity to ensure this aliased BaseFab is not used after the original BaseFab has deleted its data ptr (resize, define(..) called, or destruction, etc.).

This aliased BaseFab will also generate side effects (modifying the values of data in one will modify the other's data).

This aliased BaseFab will have a_comps.size() components, starting at zero.

7.13.2.4 template<class T> BaseFab< T >::~BaseFab () [inline, virtual]

The destructor deletes the array memory. Unless this was an aliased BaseFab.

7.13.3 Member Function Documentation

7.13.3.1 template<class T> const IntVect & BaseFab< T >::bigEnd () const [inline]

: Returns the upper corner of the domain. See class ‘Box’ for analogue.

7.13.3.2 template<class T> const Box & BaseFab< T >::box () const [inline]

Returns the domain (box) where the array is defined.

7.13.3.3 template<class T> void BaseFab< T >::clear () [inline]

: The function returns the ‘BaseFab’ to the invalid state. (See comments for constructors above.) The memory is freed.

Reimplemented in **BinFab< T >** (p. 98).

7.13.3.4 template<class T> bool BaseFab< T >::contains (const Box & *bx*) const [inline]

: Returns true if bx is totally contained within the domain of this 'BaseFab'.

7.13.3.5 template<class T> bool BaseFab< T >::contains (const BaseFab< T > & *fab*) const [inline]

{ comparison functions}

: Returns true if the domain of fab is totally contained within the domain of this 'BaseFab'.

7.13.3.6 template<class T> void BaseFab< T >::copy (const Box & *RegionFrom*, const Interval & *Cdest*, const Box & *RegionTo*, const BaseFab< T > & *src*, const Interval & *Csrc*) [inline]

Copy from a subsection of one box into another. Assumes the boxes are both in the same index space, and that box R is completely contained in both the source and destination boxes.

7.13.3.7 template<class T> BaseFab< T > & BaseFab< T >::copy (const BaseFab< T > & *src*) [inline]

: Modifies this BaseFab by coping the contents of the argument BaseFab into it. A copy within the intersecting region of the domains of the two BaseFabs is performed. All components are copied.

7.13.3.8 template<class T> BaseFab< T > & BaseFab< T >::copy (const BaseFab< T > & *src*, const Box & *destbox*) [inline]

: Modifies this BaseFab by coping the contents of the argument BaseFab into it. A copy within the intersecting region of the domains of the two BaseFabs and the specified **Box** destbox is performed. All components are copied.

7.13.3.9 template<class T> BaseFab< T > & BaseFab< T >::copy (const BaseFab< T > & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1) [inline]

: Modifies this BaseFab by copying the contents of the argument BaseFab into it. A copy within the intersecting region of the domains of the two BaseFabs is performed. The user specifies how many components are copied, starting at component srccomp in src and stored starting at component destcomp.

7.13.3.10 template<class T> BaseFab< T > & BaseFab< T >::copy (const BaseFab< T > & *src*, const Box & *srcbox*, int *srccomp*, const Box & *destbox*, int *destcomp*, int *numcomp*) [inline]

: Modifies this BaseFab by copying the contents of the argument BaseFab into it. This, the most general form of copy, specifies the contents of any sub-box srcbox in 'BaseFab' src may be copied into a (possibly different) destbox in the destination 'BaseFab'. Note that although the srcbox and the destbox may be disjoint, they must be the same size and shape. If the sizes differ, the copy is undefined and a runtime error results. This copy function is the only one of the copy functions to allow a copy between differing boxes. The user also specifies how many components are copied, starting at component srccomp in src and stored starting at component destcomp. The results are UNDEFINED if the src and dest BaseFabs are the same and the srcbox and destbox overlap.

7.13.3.11 template<class T> const T * BaseFab< T >::dataPtr (int *N* = 0) const [inline]

: Returns a constant pointer to an object of type T that is the value of the Nth component associated with the cell at the low end of the domain. This is commonly used to get a pointer to data in the array which is then handed off to a Fortran subroutine. It should not be used in any other context!!! Remember that data is stored in Fortran array order, with the component index coming last. In other words, 'dataPtr' returns a pointer to all the Nth components.

7.13.3.12 template<class T> T * BaseFab< T >::dataPtr (int *N* = 0) [inline]

: Returns a pointer to an object of type T that is the value of the Nth component associated with the cell at the low end of the domain. This is commonly used to get a pointer to data in the array which is then handed off to a Fortran subroutine. It should not be used in any other context!!! Remember that data is stored in Fortran array order, with the component index coming last. In other words, 'dataPtr' returns a pointer to all the Nth components.

7.13.3.13 template<class T> void BaseFab< T >::define () [inline, protected]

7.13.3.14 template<class T> void BaseFab< T >::define (const Interval & *a_comps*, BaseFab< T > & *a_original*) [inline, virtual]

alias define. no memory allocated. this BaseFab sets its data ptr directly in the a_original BaseFab data.

7.13.3.15 template<class T> virtual void BaseFab< T >::define (const Box & *box*, int *comps*, T * *alias* = NULL) [inline, virtual]

: Make BaseFab with desired domain and number of components. existing data is lost. Data is in uninialized state.

Reimplemented in **FArrayBox** (p. 266).

7.13.3.16 template<class T> void BaseFab< T >::getVal (T * *data*, const IntVect & *pos*) const [inline]

: This function puts all component values, starting at component 0, from position pos in the domain into array data, that must be allocated by the user.

7.13.3.17 template<class T> void BaseFab< T >::getVal (T * *data*, const IntVect & *pos*, int *N*, int *numcomp*) const [inline]

: This function puts numcomp component values, starting at component N, from position pos in the domain into array data, that must be allocated by the user.

7.13.3.18 template<class T> const int * BaseFab< T >::hiVect () const [inline]

: Returns the upper corner of the domain. Instead of returning them in the form of IntVects, as in smallEnd and bigEnd, it returns the values as a pointer to an array of constant integers. This is useful when interfacing to Fortran subroutines. It should not be used in any other context!!!

7.13.3.19 template<class T> Interval BaseFab< T >::interval () const [inline]

Returns an **Interval** for the entire range on components.

7.13.3.20 template<class T> void BaseFab< T >::linearIn (void * *buf*, const Box & *R*, const Interval & *comps*) [inline, virtual]

Reimplemented in **BinFab< T >** (p. 98).

7.13.3.21 template<class T> void BaseFab< T >::linearOut (void * *buf*, const Box & *R*, const Interval & *comps*) const [inline, virtual]

Write a linear representation of the internal data. Assumes that sufficient memory for the buffer has already been allocated by the caller.

Reimplemented in **BinFab< T >** (p. 98).

7.13.3.22 template<class T> const int * BaseFab< T >::loVect () const [inline]

{ Fortran interface functions}

: Returns the lower corner of the domain. Instead of returning them in the form of IntVects, as in smallEnd and bigEnd, it returns the values as a pointer to an array of constant integers. This is useful when interfacing to Fortran subroutines. It should not be used in any other context!!!

7.13.3.23 template<class T> std::string BaseFab< T >::name () [inline, static, protected]

7.13.3.24 template<class T> int BaseFab< T >::nComp () const [inline]

{ accessors}

Returns the number of components.

7.13.3.25 template<class T> const int * BaseFab< T >::nCompPtr () const [inline]

: Returns a pointer to an integer that contains the number of components in the BaseFab. This is useful when interfacing to Fortran subroutines. It should not be used in any other context!!!

7.13.3.26 template<class T> const T & BaseFab< T >::operator() (const IntVect & *p*) const [inline]

7.13.3.27 template<class T> const T & BaseFab< T >::operator() (const IntVect & *p*, int *N*) const [inline]

: Returns a conatant reference to the Nth component value defined at position p in the domain. This operator may be inefficient if the C++ compiler is unable to optimize the C++ code.

7.13.3.28 template<class T> T & BaseFab< T >::operator() (const IntVect & *p*) [inline]

7.13.3.29 template<class T> T & BaseFab< T >::operator() (const IntVect & *p*, int *N*) [inline]

: Returns a modifiable lvalue reference to the Nth component value defined at position p in the domain. This operator may be inefficient if the C++ compiler is unable to optimize the C++ code.

7.13.3.30 template<class T> void BaseFab< T >::performCopy (const BaseFab< T > & *src*, const Box & *srcbox*, int *srccomp*, const Box & *destbox*, int *destcomp*, int *numcomp*) [inline, protected, virtual]

Reimplemented in **FArrayBox** (p. 272).

7.13.3.31 template<class T> void BaseFab< T >::performSetVal (T *x*, const Box & *bx*, int *nstart*, int *numcomp*) [inline, protected]

7.13.3.32 template<class T> int BaseFab< T >::preAllocatable () [inline, static]

Reimplemented in **BinFab< T >** (p. 99).

7.13.3.33 template<class T> void BaseFab< T >::resize (const Box & *b*, int *N* = 1, T * *alias* = NULL)

This function resizes a 'BaseFab' so it covers the 'Box' B with N components. The default action is that under **resize()**ing, the memory allocated for the 'BaseFab' only grows and never shrinks. This function is particularly useful when a 'BaseFab' is used as a temporary space which must be a different size whenever it is used. Resize()ing a temp will often be faster than re-allocating a 'BaseFab' because memory allocation can often be avoided.

7.13.3.34 template<class T> void BaseFab< T >::setVal (T *x*) [inline]

Modifies this BaseFab so that all values of all components are set to the given value x.

7.13.3.35 template<class T> void BaseFab< T >::setVal (T *x*, int *N*) [inline]

Modifies this BaseFab so that all values of component N are set to the given value x.

7.13.3.36 template<class T> void BaseFab< T >::setVal (T *x*, const Box & *bx*, int *N*) [inline]

Modifies this BaseFab so that all values of component N in the specified **Box** bx are set to the given value x.

7.13.3.37 template<class T> void BaseFab< T >::setVal (T *x*, const Box & *bx*, int *nstart*, int *ncomp*) [inline]

{ data modification functions}

: The setVal functions set subregions in the 'BaseFab' to a constant value. This most general form specifies the sub-box, the starting component number, and the number of components to be set.

7.13.3.38 template<class T> BaseFab< T > & BaseFab< T >::shift (int *idir*, int *n_cell*) [inline]

Modifies the domain of this BaseFab by shifting it n_cells indexing positions in coordinate direction idir. Directions are zero-based. It is an error if not 0 <= idir < SpaceDim.There is no effect upon the array memory.

7.13.3.39 template<class T> BaseFab< T > & BaseFab< T >::shift (const IntVect & *v*) [inline]

{ domain modification functions}

Modifies the domain of this BaseFab by shifting. Equivalent to fab.shift(0,iv[0]).shift(1,iv[1]) .... There is no effect upon the array memory.

7.13.3.40 template<class T> BaseFab< T > & BaseFab< T >::shiftHalf (const IntVect & *num_halfs*) [inline]

Modifies the domain of this BaseFab by shifting by half indices. Equivalent to fab.shiftHalf(0,iv[0]).shiftHalf(1,iv[1]) ... There is no effect upon the array memory.

7.13.3.41 template<class T> BaseFab< T > & BaseFab< T >::shiftHalf (int *dir*, int *num_halfs*) [inline]

Modifies the domain of this BaseFab by shifting by "half" indices, thereby converting the **Box** from type CELL to NODE or vice-versa. fab.shiftHalf(0,1) shifts the domain to the right by 1/2 cells. fab.shiftHalf(1,-3) shifts the domain in the -j direction by 3/2 cells. NOTE: If num is EVEN the shift is num/2 full zones and hence will not change the type. This is: fab.shifthalf(4) == fab.shift(2). Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. There is no effect upon the array memory.

7.13.3.42 template<class T> int BaseFab< T >::size (const Box & *b*, const Interval & *comps*) const [inline, virtual]

{ linearization functions}

Returns the size, in number of bytes, of a flat linear representation of the data in this object in the area defined by the input **Box** R and the component **Interval** comps. The size does not include the size of R and comps.

Reimplemented in **BinFab< T >** (p. 99).

7.13.3.43 template<class T> const int * BaseFab< T >::size () const [inline]

: Returns a pointer to an array of SpaceDim integers giving the length of the domain in each direction.

7.13.3.44 template<class T> const IntVect & BaseFab< T >::smallEnd () const [inline]

: Returns the lower corner of the domain. See class 'Box' for analogue.

7.13.3.45 template<class T> void BaseFab< T >::undefine () [inline, protected]

7.13.4 Member Data Documentation

7.13.4.1 template<class T> bool BaseFab< T >::aliased [protected]

7.13.4.2 template<class T> Box BaseFab< T >::domain [protected]

7.13.4.3 template<class T> T* BaseFab< T >::dptr [protected]

7.13.4.4 template<class T> long BaseFab< T >::numpts [protected]

7.13.4.5 template<class T> int BaseFab< T >::nvar [protected]

7.13.4.6 template<class T> Arena * BaseFab< T >::s_Arena = NULL [static, protected]

7.13.4.7 template<class T> long BaseFab< T >::truesize [protected]

The documentation for this class was generated from the following files:

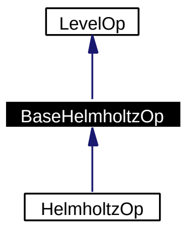
- **BaseFab.H**
- **BaseFabImplem.H**

7.14 BaseHelmholtzOp Class Reference

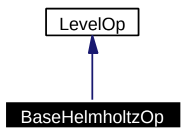
Pure-virtual base Helmholtz operator class (alpha*I + div*beta*grad).

`#include <BaseHelmholtzOp.H>`

Inheritance diagram for BaseHelmholtzOp:



Collaboration diagram for BaseHelmholtzOp:



Public Member Functions

- virtual void **scaleAlpha** (**Real** a_scale)=0

  *Scale the helmholtz coeficient alpha in (alpha*I + beta*laplacian)phi.*

- virtual void **scaleBeta** (**Real** a_scale)=0

  *Scale the helmholtz coefficient beta in (alpha*I + beta*laplacian)phi.*

- virtual void **setDomainGhostBC** (const **DomainGhostBC** &a_dombc)=0

  set (or reset) the physical boundary conditions

- **BaseHelmholtzOp** ()

  Null Constructor.

- virtual ~**BaseHelmholtzOp** ()

  Destructor.

7.14.1 Detailed Description

Pure-virtual base Helmholtz operator class (alpha*I + div*beta*grad).

Virtual base class to solve Helmholtz-type problems. The raison d'Etre for this class is to allow for scaling of Alpha and Beta for any Helmholtz-type operator

7.14.2 Constructor & Destructor Documentation

7.14.2.1 BaseHelmholtzOp::BaseHelmholtzOp () `[inline]`

Null Constructor.

7.14.2.2 virtual BaseHelmholtzOp::~BaseHelmholtzOp () `[inline, virtual]`

Destructor.

7.14.3 Member Function Documentation

7.14.3.1 virtual void BaseHelmholtzOp::scaleAlpha (Real *a_scale*) `[pure virtual]`

Scale the helmholtz coeficient alpha in (alpha*I + beta*laplacian)phi.

(alpha := a_scale*alpha)

Implemented in **HelmholtzOp** (p. 292).

7.14.3.2 virtual void BaseHelmholtzOp::scaleBeta (Real *a_scale*) `[pure virtual]`

Scale the helmholtz coefficient beta in (alpha*I + beta*laplacian)phi.

(beta := a_scale*beta)

Implemented in **HelmholtzOp** (p. 293).

7.14.3.3 virtual void BaseHelmholtzOp::setDomainGhostBC (const DomainGhostBC & *a_dombc*) `[pure virtual]`

set (or reset) the physical boundary conditions

Implemented in **HelmholtzOp** (p. 293).

The documentation for this class was generated from the following file:

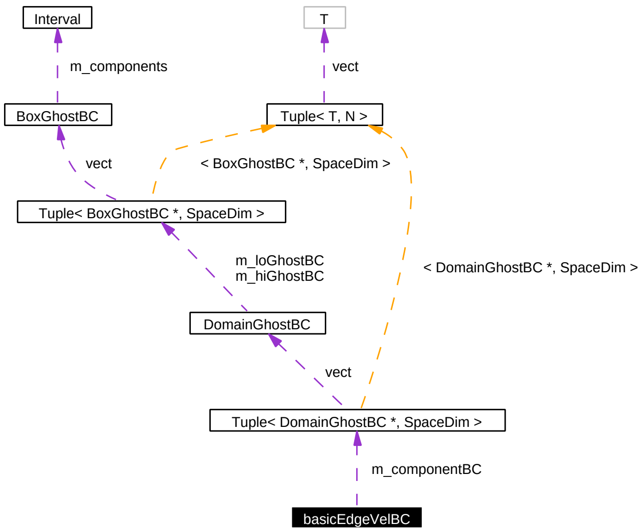
- **BaseHelmholtzOp.H**

7.15 basicEdgeVelBC Class Reference

this is a basic physical BC class designed to handle velocities

`#include <basicVelBC.H>`

Collaboration diagram for basicEdgeVelBC:



Public Member Functions

- **basicEdgeVelBC** ()
- virtual ~**basicEdgeVelBC** ()
- virtual void **setBCType** (const **DomainGhostBC** &a_ghostBC, int a_component)

  one **DomainGhostBC** *is set for each velocity component*

- **basicEdgeVelBC** (const int a_component)
- virtual void **applyBCs** (**FluxBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx) const
- virtual void **applyBCs** (**FluxBox** &a_state, const **Box** &a_domain, const **Real** a_dx) const

  old interface using **Box** *instead of* **ProblemDomain**

- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx) const
- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx) const

  old interface using **Box** *instead of* **ProblemDomain**

- virtual void **applyBCs** (**LevelData**< **FluxBox** > &a_stateMF, const **ProblemDomain** &a_domain, const **Real** a_dx) const
- virtual void **applyBCs** (**LevelData**< **FluxBox** > &a_stateMF, const **Box** &a_domain, const **Real** a_dx) const

  old interface using **Box** *instead of* **ProblemDomain**

- virtual void **applyBCs** (**FluxBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx, int a_comp) const

  single-component version

- virtual void **applyBCs** (**FluxBox** &a_state, const **Box** &a_domain, const **Real** a_dx, int a_comp) const

  single-component version using **Box** *instead of* **ProblemDomain**

- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx, int a_comp) const

  single-component version

- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx, int a_comp) const

  single-component version using **Box** *instead of* **ProblemDomain**

- const **DomainGhostBC** & **DomGhostBC** (const int dir) const

  returns **DomainGhostBC** *for direction Dir*

- void **isSingleComponent** (bool a_isSingleComp)

  will this be applied to single-component FluxBoxes?

- bool **isSingleComponent** () const

  will this be applied to single-component FluxBoxes?

Protected Attributes

- **Tuple< DomainGhostBC *, SpaceDim > m_componentBC**
- bool **m_isSingleComponent**

  if this is will be applied to single component FluxBoxes

7.15.1 Detailed Description

this is a basic physical BC class designed to handle velocities

this class is handles velocities, which are by their nature multicomponent

7.15.2 Constructor & Destructor Documentation

7.15.2.1 basicEdgeVelBC::basicEdgeVelBC ()

7.15.2.2 virtual basicEdgeVelBC::~basicEdgeVelBC () `[virtual]`

7.15.2.3 basicEdgeVelBC::basicEdgeVelBC (const int *a_component*)

7.15.3 Member Function Documentation

7.15.3.1 virtual void basicEdgeVelBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*, int *a_comp*) const `[virtual]`

single-component version using **Box** instead of **ProblemDomain**

7.15.3.2 virtual void basicEdgeVelBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*, int *a_comp*) const `[virtual]`

single-component version

7.15.3.3 virtual void basicEdgeVelBC::applyBCs (FluxBox & *a_state*, const Box & *a_domain*, const Real *a_dx*, int *a_comp*) const `[virtual]`

single-component version using **Box** instead of **ProblemDomain**

7.15.3.4 virtual void basicEdgeVelBC::applyBCs (FluxBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*, int *a_comp*) const `[virtual]`

single-component version

7.15.3.5 virtual void basicEdgeVelBC::applyBCs (LevelData< FluxBox > & *a_stateMF*, const Box & *a_domain*, const Real *a_dx*) const `[virtual]`

old interface using **Box** instead of **ProblemDomain**

7.15.3.6 virtual void basicEdgeVelBC::applyBCs (LevelData< FluxBox > & *a_stateMF*, const ProblemDomain & *a_domain*, const Real *a_dx*) const `[virtual]`

7.15.3.7 virtual void basicEdgeVelBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*) const `[virtual]`

old interface using **Box** instead of **ProblemDomain**

7.15.3.8 virtual void basicEdgeVelBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*) const `[virtual]`

7.15.3.9 virtual void basicEdgeVelBC::applyBCs (FluxBox & *a_state*, const Box & *a_domain*, const Real *a_dx*) const `[virtual]`

old interface using **Box** instead of **ProblemDomain**

7.15.3.10 virtual void basicEdgeVelBC::applyBCs (FluxBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*) const `[virtual]`

7.15.3.11 const DomainGhostBC& basicEdgeVelBC::DomGhostBC (const int *dir*) const

returns **DomainGhostBC** for direction Dir

7.15.3.12 bool basicEdgeVelBC::isSingleComponent () const

will this be applied to single-component FluxBoxes?

7.15.3.13 void basicEdgeVelBC::isSingleComponent (bool *a_isSingleComp*)

will this be applied to single-component FluxBoxes?

Single component Fluxboxes, like advection velocities, will have each direction be a single BC (x-dir fab is x-velocity, y-dir fab is y-velocity, etc). multicomponent edge-centered velocities are arranged so that velocity components align with components in Fabs, so that the x-edge-centered fab will have SpaceDim components: x-velocity, y-velocity, (and z-velocity in 3d). Each of these will have a different BC.

7.15.3.14 virtual void basicEdgeVelBC::setBCType (const DomainGhostBC & *a_ghostBC*, int *a_component*) `[virtual]`

one **DomainGhostBC** is set for each velocity component

7.15.4 Member Data Documentation

7.15.4.1 Tuple<DomainGhostBC*, SpaceDim> basicEdgeVelBC::m_componentBC `[protected]`

7.15.4.2 bool basicEdgeVelBC::m_isSingleComponent `[protected]`

if this is will be applied to single component FluxBoxes

The documentation for this class was generated from the following file:

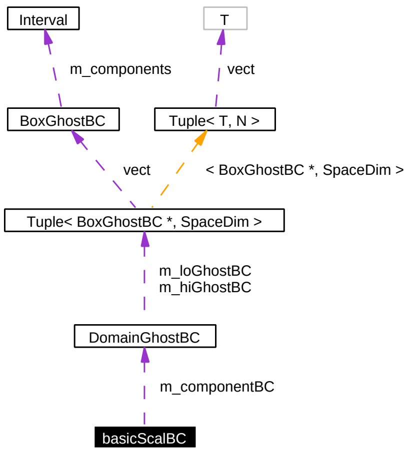
- **basicVelBC.H**

7.16 basicScalBC Class Reference

this is a basic physical BC class designed to handle scalars

`#include <basicScalBC.H>`

Collaboration diagram for basicScalBC:



Public Member Functions

- **basicScalBC** ()
- virtual ~**basicScalBC** ()
- virtual void **setBCType** (const **DomainGhostBC** &a_ghostBC)

    DomainGhostBC *is set.*

- virtual void **setAllBCTypes** ()=0

    or the derived class can do it for you

- **basicScalBC** ()
- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx) const
- virtual void **applyBCs** (**LevelData**< **FArrayBox** > &a_stateMF, const **ProblemDomain** &a_domain, const **Real** a_dx) const
- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx) const

    deprecated interface

- virtual void **applyBCs** (**LevelData**< **FArrayBox** > &a_stateMF, const **Box** &a_domain, const **Real** a_dx) const

    deprecated interface

Protected Attributes

- **DomainGhostBC** ∗ **m_componentBC**

7.16.1 Detailed Description

this is a basic physical BC class designed to handle scalars

7.16.2 Constructor & Destructor Documentation

7.16.2.1 basicScalBC::basicScalBC ()

7.16.2.2 virtual basicScalBC::~basicScalBC () [virtual]

7.16.2.3 basicScalBC::basicScalBC ()

7.16.3 Member Function Documentation

7.16.3.1 virtual void basicScalBC::applyBCs (LevelData< FArrayBox > & *a_stateMF*, const Box & *a_domain*, const Real *a_dx*) const [virtual]

deprecated interface

7.16.3.2 virtual void basicScalBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*) const [virtual]

deprecated interface

7.16.3.3 virtual void basicScalBC::applyBCs (LevelData< FArrayBox > & *a_stateMF*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

7.16.3.4 virtual void basicScalBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

7.16.3.5 virtual void basicScalBC::setAllBCTypes () [pure virtual]

or the derived class can do it for you

7.16.3.6 virtual void basicScalBC::setBCType (const DomainGhostBC & *a_ghostBC*) [virtual]

DomainGhostBC is set.

7.16.4 Member Data Documentation

7.16.4.1 DomainGhostBC* basicScalBC::m_componentBC [protected]

The documentation for this class was generated from the following file:

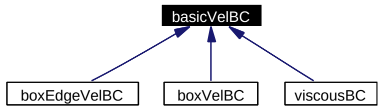
- **basicScalBC.H**

7.17 basicVelBC Class Reference

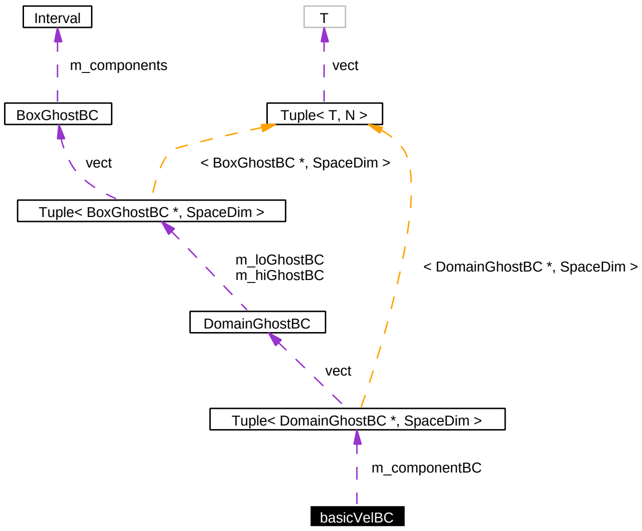
this is a basic physical BC class designed to handle velocities

`#include <basicVelBC.H>`

Inheritance diagram for basicVelBC:



Collaboration diagram for basicVelBC:



Public Member Functions

- **basicVelBC** ()
- virtual ~**basicVelBC** ()
- virtual void **setBCType** (const **DomainGhostBC** &a_ghostBC, int a_component)

  one **DomainGhostBC** *is set for each velocity component*

- **basicVelBC** (const int a_component)
- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx) const
- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx) const

  old interface using **Box** *instead of* **ProblemDomain**

- virtual void **applyBCs** (**LevelData**< **FArrayBox** > &a_stateMF, const **ProblemDomain** &a_domain, const **Real** a_dx) const
- virtual void **applyBCs** (**LevelData**< **FArrayBox** > &a_stateMF, const **Box** &a_domain, const **Real** a_dx) const

old interface using **Box** *instead of* **ProblemDomain**

- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx, int a_comp) const

  single-component version

- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx, int a_comp) const

  single-component version using **Box** *instead of* **ProblemDomain**

- const **DomainGhostBC** & **DomGhostBC** (const int dir) const

  returns **DomainGhostBC** *for direction Dir*

Protected Attributes

- **Tuple< DomainGhostBC ∗, SpaceDim > m_componentBC**

7.17.1 Detailed Description

this is a basic physical BC class designed to handle velocities

this class is handles velocities, which are by their nature multicomponent

7.17.2 Constructor & Destructor Documentation

7.17.2.1 basicVelBC::basicVelBC ()

7.17.2.2 virtual basicVelBC::~basicVelBC () `[virtual]`

7.17.2.3 basicVelBC::basicVelBC (const int *a_component*)

7.17.3 Member Function Documentation

7.17.3.1 virtual void basicVelBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*, int *a_comp*) const `[virtual]`

single-component version using **Box** instead of **ProblemDomain**

Reimplemented in **boxEdgeVelBC** (p. 135), and **viscousBC** (p. 631).

7.17.3.2 virtual void basicVelBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*, int *a_comp*) const `[virtual]`

single-component version

Reimplemented in **boxEdgeVelBC** (p. 136), and **viscousBC** (p. 632).

7.17.3.3 virtual void basicVelBC::applyBCs (LevelData< FArrayBox > & *a_stateMF*, const Box & *a_domain*, const Real *a_dx*) const [virtual]

old interface using **Box** instead of **ProblemDomain**

Reimplemented in **viscousBC** (p. 631).

7.17.3.4 virtual void basicVelBC::applyBCs (LevelData< FArrayBox > & *a_stateMF*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

Reimplemented in **viscousBC** (p. 632).

7.17.3.5 virtual void basicVelBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*) const [virtual]

old interface using **Box** instead of **ProblemDomain**

Reimplemented in **boxEdgeVelBC** (p. 136), and **viscousBC** (p. 631).

7.17.3.6 virtual void basicVelBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

Reimplemented in **boxEdgeVelBC** (p. 136), and **viscousBC** (p. 632).

7.17.3.7 const DomainGhostBC& basicVelBC::DomGhostBC (const int *dir*) const

returns **DomainGhostBC** for direction Dir

7.17.3.8 virtual void basicVelBC::setBCType (const DomainGhostBC & *a_ghostBC*, int *a_component*) [virtual]

one **DomainGhostBC** is set for each velocity component

7.17.4 Member Data Documentation

7.17.4.1 Tuple<DomainGhostBC*, SpaceDim> basicVelBC::m_componentBC [protected]

The documentation for this class was generated from the following file:

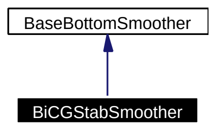
- **basicVelBC.H**

7.18 BiCGStabSmoother Class Reference

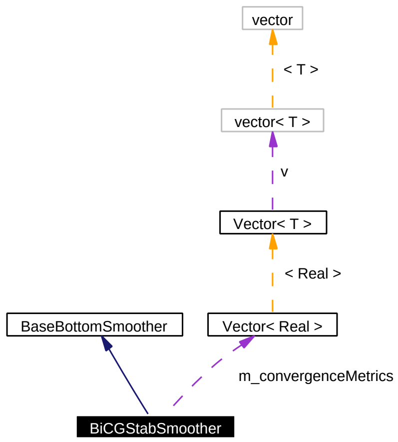
Implements BiCGStab as a bottom smoother for LevelOp-derived classes.

`#include <BiCGStabSmoother.H>`

Inheritance diagram for BiCGStabSmoother:



Collaboration diagram for BiCGStabSmoother:



Public Member Functions

- **BiCGStabSmoother** ()

  Null constructor.

- virtual ~**BiCGStabSmoother** ()

  Destructor.

- virtual **BaseBottomSmoother** ∗ **new_bottomSmoother** () const

  "Virtual constructor" workaround.

- void **setMaxIter** (int a_maxIter)

  Set maximum number of BiCGStab iterations.

- void **setSolverTol** (**Real** a_solverTol)

  Set solver tolerance.

- void **setEnableRestart** (bool a_enableRestart)

  Set whether to enable restart capabillity.

- void **setVerbose** (bool a_verbose)

  Set verbosity level.

- virtual void **doBottomSmooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs, **LevelOp** *a_levelopPtr)

  Do smoothing.

- virtual void **setConvergenceMetric** (**Real** a_metric, int a_comp)

  set metric for judging convergence (if not set, defaults to 1.0)

Protected Attributes

- int **m_maxIter**
- **Vector**< **Real** > **m_convergenceMetrics**
- **Real m_solverTol**
- **Real m_small**
- **Real m_convergeSmall**
- bool **m_enableRestart**
- bool **m_verbose**

7.18.1 Detailed Description

Implements BiCGStab as a bottom smoother for LevelOp-derived classes.

The BiCGStabSmoother class implements the BiCGStab algorithm for LevelOp-derived classes

7.18.2 Constructor & Destructor Documentation

7.18.2.1 BiCGStabSmoother::BiCGStabSmoother ()

Null constructor.

7.18.2.2 virtual BiCGStabSmoother::~BiCGStabSmoother () `[virtual]`

Destructor.

7.18.3 Member Function Documentation

7.18.3.1 virtual void BiCGStabSmoother::doBottomSmooth (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*, LevelOp * *a_levelopPtr*) `[virtual]`

Do smoothing.

Performs BiCGStab smoothing of L(a_phi) = a_rhs; L is the operator defined in a_levelopPtr, and the preconditioner is a_levelopPtr->smooth

Implements **BaseBottomSmoother** (p. 69).

7.18.3.2 virtual BaseBottomSmoother* BiCGStabSmoother::new_bottomSmoother () const [virtual]

"Virtual constructor" workaround.

This gets around the "no virtual constructor" rule

Implements **BaseBottomSmoother** (p. 69).

7.18.3.3 virtual void BiCGStabSmoother::setConvergenceMetric (Real *a_metric*, int *a_comp*) [virtual]

set metric for judging convergence (if not set, defaults to 1.0)

Implements **BaseBottomSmoother** (p. 69).

7.18.3.4 void BiCGStabSmoother::setEnableRestart (bool *a_enableRestart*)

Set whether to enable restart capabillity.

If this is true, if BiCGStab reaches a hang point, the solver will recompute the residual and attempt to restart the computation

7.18.3.5 void BiCGStabSmoother::setMaxIter (int *a_maxIter*)

Set maximum number of BiCGStab iterations.

7.18.3.6 void BiCGStabSmoother::setSolverTol (Real *a_solverTol*) [virtual]

Set solver tolerance.

Set solver tolerance

Implements **BaseBottomSmoother** (p. 69).

7.18.3.7 void BiCGStabSmoother::setVerbose (bool *a_verbose*)

Set verbosity level.

Set whether i/o is done when solving (default is false)

7.18.4 Member Data Documentation

7.18.4.1 Vector<Real> BiCGStabSmoother::m_convergenceMetrics [protected]

Solver convergence metrics

7.18.4.2 Real BiCGStabSmoother::m_convergeSmall [protected]

Small parameter to check for size of correction (to prevent hanging)

7.18.4.3 bool BiCGStabSmoother::m_enableRestart [protected]

If true, enable restarting capability

7.18.4.4 int BiCGStabSmoother::m_maxIter [protected]

Maximum number of BiCGStab iterations

7.18.4.5 Real BiCGStabSmoother::m_small [protected]

Small parameter to check for division by zero

7.18.4.6 Real BiCGStabSmoother::m_solverTol [protected]

Solver tolerance (we declare the problem solved if the residual is smaller than solverTol*convergenceMetric

7.18.4.7 bool BiCGStabSmoother::m_verbose [protected]

Default is false

The documentation for this class was generated from the following file:

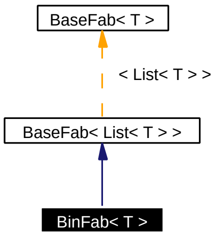
- **BiCGStabSmoother.H**

7.19 BinFab< T > Class Template Reference

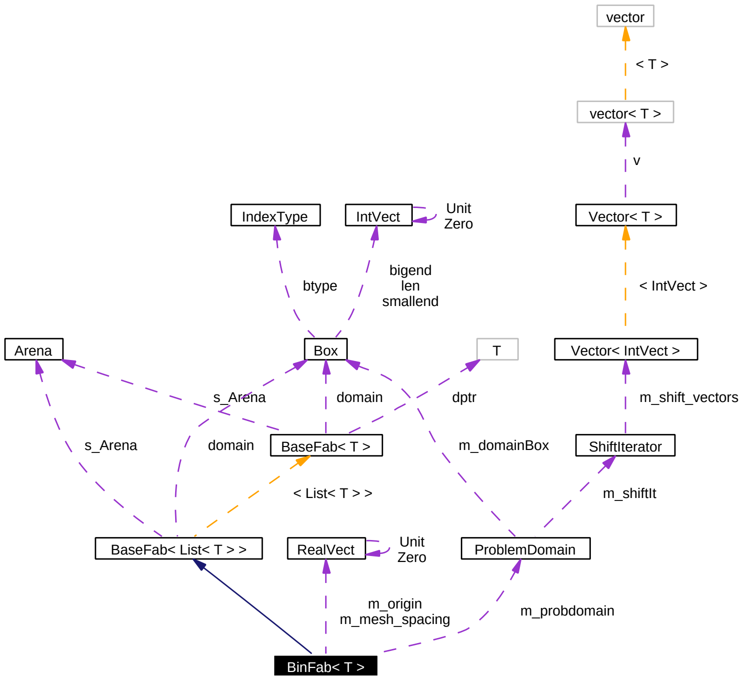
Base class for particle data on a **Box**.

`#include <BinFab.H>`

Inheritance diagram for BinFab< T >:



Collaboration diagram for BinFab< T >:



Public Member Functions

- **BinFab** ()

  Null constructor, copy constructor and operator= can be compiler defined.

- **BinFab** (const **Box** &a_domain, const **RealVect** &a_mesh_spacing, const **RealVect** &a_origin, const **ProblemDomain** &a_probdomain)

  Constructs an empty `BinFab` *on* a_domain.

- **BinFab** (const **BinFab** &a_binfab)

  Copy constructor.

- virtual ~**BinFab** ()

  Destructor.

- virtual void **define** (const **Box** &a_domain, const **RealVect** &a_mesh_spacing, const **RealVect** &a_origin, const **ProblemDomain** &a_probdomain)

  Same as the constructor.

- virtual void **reBin** ()

  Sort all the items into the appropriate bins.

- virtual void **addItem** (const T &a_item)

  Copy one item into the appropriate bin.

- virtual void **addItems** (const **List**< T > &a_list)

  Copy a list of items into the appropriate bins.

- virtual void **addItemsDestructive** (**List**< T > &a_list)

  Move items off the argument list and into the appropriate bins.

- virtual void **addItemsDestructive** (**List**< T > &a_list,const **Box** &a_valid, bool a_in=true)

  Move items into the appropriate bins and remove from the arg list only if the dest is in the **Box***.*

- virtual void **clear** ()

  Delete all the items in this `BinFab` *and reset it to the null* **Box***.*

- virtual int **numItems** (const **Box** &a_box=**Box**(), const **Interval** &a_comps=**Interval**()) const

  Count the number of items in this `BinFab`.

- virtual int **size** (const **Box** &a_box, const **Interval** &a_comps) const

  The number of bytes used by **linearIn()/linearOut***().*

- virtual void **linearOut** (void *a_buf, const **Box** &a_box, const **Interval** &a_comps) const

  Write a serialized (packed) representation into a_buf*.*

- virtual void **linearIn** (void *a_buf, const **Box** &a_box, const **Interval** &a_comps)

  Extract a serialized (packed) representation from a_buf*.*

Static Public Member Functions

- int **preAllocatable** ()

Protected Member Functions

- **IntVect locateBin** (const T &a_item) const
  compute the cell index containing the physical position of the item

Protected Attributes

- **RealVect m_origin**
  location in physical space of the 0 index
- **RealVect m_mesh_spacing**
  cell size in physical space
- **ProblemDomain m_probdomain**
  contains the **Box** *on which this BinFab is defined*

7.19.1 Detailed Description

template<class T> class BinFab< T >

Base class for particle data on a **Box**.

This is a class for holding and sorting particle items. Class must have a **RealVect** ::position() const function which is used to place the item in an appropriate bin. See class **BinItem** for a base class for particles. All BinFabs are assumed to have 1 component.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 template<class T> BinFab< T >::BinFab ()

Null constructor, copy constructor and operator= can be compiler defined.

7.19.2.2 template<class T> BinFab< T >::BinFab (const Box & *a_domain*, const RealVect & *a_mesh_spacing*, const RealVect & *a_origin*, const ProblemDomain & *a_probdomain*)

Constructs an empty `BinFab` on *a_domain*.

7.19.2.3 template<class T> BinFab< T >::BinFab (const BinFab< T > & *a_binfab*)

Copy constructor.

7.19.2.4 template<class T> BinFab< T >::~BinFab () `[virtual]`

Destructor.

7.19.3 Member Function Documentation

7.19.3.1 template<class T> void BinFab< T >::addItem (const T & *a_item*) `[virtual]`

Copy one item into the appropriate bin.

7.19.3.2 template<class T> void BinFab< T >::addItems (const List< T > & *a_list*) `[virtual]`

Copy a list of items into the appropriate bins.

7.19.3.3 template<class T> void BinFab< T >::addItemsDestructive (List< T > & *a_List*, const Box & *a_valid*, bool *a_in* = true) `[virtual]`

Move items into the appropriate bins and remove from the arg list only if the dest is in the **Box**.

If the destination cell is in the specified box (on not in if a_in is false) then remove the particle from the argument **List**, else copy it.

7.19.3.4 template<class T> void BinFab< T >::addItemsDestructive (List< T > & *a_list*) `[virtual]`

Move items off the argument list and into the appropriate bins.

7.19.3.5 template<class T> void BinFab< T >::clear () `[virtual]`

Delete all the items in this `BinFab` and reset it to the null **Box**.

Reimplemented from **BaseFab< List< T > >** (p. 73).

7.19.3.6 template<class T> void BinFab< T >::define (const Box & *a_domain*, const RealVect & *a_mesh_spacing*, const RealVect & *a_origin*, const ProblemDomain & *a_probdomain*) `[virtual]`

Same as the constructor.

7.19.3.7 template<class T> void BinFab< T >::linearIn (void * *a_buf*, const Box & *a_box*, const Interval & *a_comps*) `[virtual]`

Extract a serialized (packed) representation from *a_buf*.

Given a linear representation of the class data previously made using **linearIn()** in *a_buf*, set the data for this class on the specified **Box**.

Reimplemented from **BaseFab< List< T > >** (p. 76).

7.19.3.8 template<class T> void BinFab< T >::linearOut (void * *a_buf*, const Box & *a_box*, const Interval & *a_comps*) const `[virtual]`

Write a serialized (packed) representation into *a_buf*.

Write a linear representation of the items in the specified **Box**. Assumes that sufficient memory for the buffer has already been allocated by the caller.

Reimplemented from **BaseFab< List< T > >** (p. 76).

7.19.3.9 template<class T> IntVect BinFab< T >::locateBin (const T & *a_item*) const [inline, protected]

compute the cell index containing the physical position of the item

7.19.3.10 template<class T> int BinFab< T >::numItems (const Box & *a_box* = Box(), const Interval & *a_comps* = Interval()) const [virtual]

Count the number of items in this `BinFab`.

The default is to do the whole Fab for all components. This is slow, so use it sparingly.

7.19.3.11 template<class T> int BinFab< T >::preAllocatable () [inline, static]

Reimplemented from **BaseFab< List< T > >** (p. 77).

7.19.3.12 template<class T> void BinFab< T >::reBin () [virtual]

Sort all the items into the appropriate bins.

7.19.3.13 template<class T> int BinFab< T >::size (const Box & *a_box*, const Interval & *a_comps*) const [virtual]

The number of bytes used by **linearIn()/linearOut**().

Returns the size, in number of bytes, of a flat linear representation of the items in the specified **Box**.

Reimplemented from **BaseFab< List< T > >** (p. 78).

7.19.4 Member Data Documentation

7.19.4.1 template<class T> RealVect BinFab< T >::m_mesh_spacing [protected]

cell size in physical space

7.19.4.2 template<class T> RealVect BinFab< T >::m_origin [protected]

location in physical space of the 0 index

7.19.4.3 template<class T> ProblemDomain BinFab< T >::m_probdomain [protected]

contains the **Box** on which this BinFab is defined

The documentation for this class was generated from the following files:

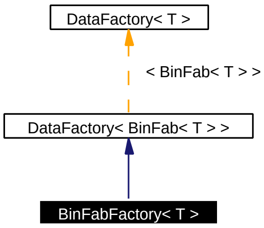
- **BinFab.H**
- **BinFabImplem.H**

7.20 BinFabFactory< T > Class Template Reference

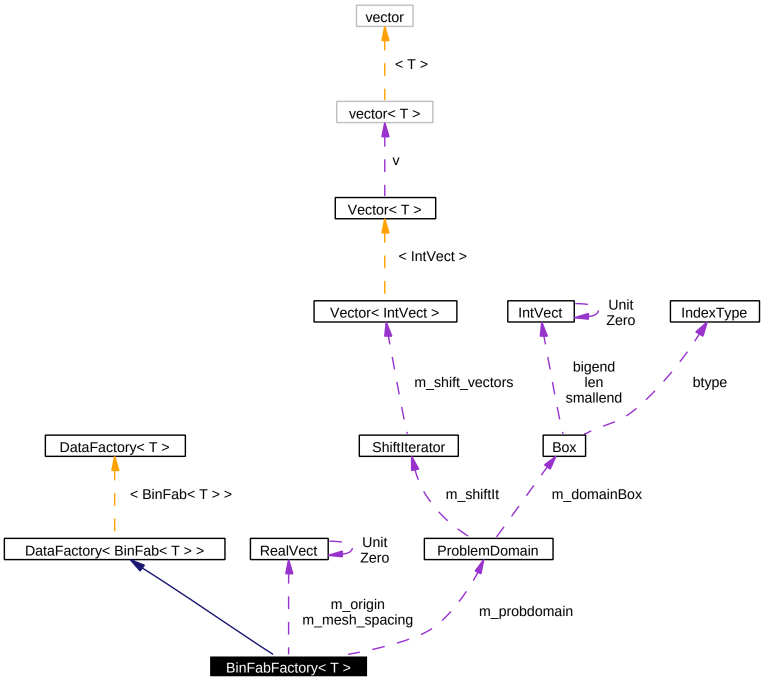
Factory class to produce one or more instances of **BinFab** on the same level.

`#include <BinFabFactory.H>`

Inheritance diagram for BinFabFactory< T >:



Collaboration diagram for BinFabFactory< T >:



Public Member Functions

- virtual **BinFab**< T > ∗ **create** (const **Box** &a_box, int a_ncomps, const **DataIndex** &a_dit) const

  Factory function.

- **BinFabFactory** (const **RealVect** &a_mesh_spacing, const **RealVect** &a_origin, const **ProblemDomain** &a_probdomain)

  Use these parameters for all manufactured **BinFab** *instances.*

- virtual ~**BinFabFactory** ()

  Destructor.

7.20.1 Detailed Description

template<class T> class BinFabFactory< T >

Factory class to produce one or more instances of **BinFab** on the same level.

The mesh and **ProblemDomain** will be the same for all instances created by the factory. This is needed for **LevelData** construction.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 template<class T> BinFabFactory< T >::BinFabFactory (const RealVect & *a_mesh_spacing*, const RealVect & *a_origin*, const ProblemDomain & *a_probdomain*)

Use these parameters for all manufactured **BinFab** instances.

7.20.2.2 template<class T> BinFabFactory< T >::~BinFabFactory () `[virtual]`

Destructor.

7.20.3 Member Function Documentation

7.20.3.1 template<class T> BinFab< T > * BinFabFactory< T >::create (const Box & *a_box*, int *a_ncomps*, const DataIndex & *a_dit*) const `[virtual]`

Factory function.

Creates a new **BinFab** object and returns a pointer to it. Responsiblitly for calling operator 'delete' on this pointer is passed to the user.

Assumes *a_ncomps* is 1 and ignores *a_dit*.

Implements **DataFactory< BinFab< T > >** (p. 206).

The documentation for this class was generated from the following files:

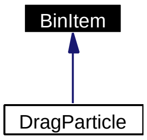
- **BinFabFactory.H**
- **BinFabFactoryI.H**

7.21 BinItem Class Reference

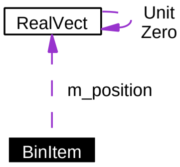
Virtual base class for particle data.

`#include <BinItem.H>`

Inheritance diagram for BinItem:



Collaboration diagram for BinItem:



Public Member Functions

- **BinItem** ()

  Null constructor, copy constructor and operator= can be compiler defined.

- **BinItem** (const **RealVect** &a_position)

  Constructs a `BinItem` *at* a_position.

- virtual ~**BinItem** ()

  Destructor.

- virtual void **define** (const **RealVect** &a_position)

  Same as the constructor.

- void **setPosition** (const **RealVect** &a_position)

  Change the position.

- void **setPosition** (const **Real** a_position, const int a_dimension)

  Change one component of the position.

- **RealVect** & **position** ()

  Return a reference to the position.

- const **RealVect** & **position** () const

  Return a constant reference to the position.

- **Real position** (const int a_dir) const
  Gets one component of the position.
- virtual int **size** () const
  Number of bytes used by **linearIn()/linearOut**().
- virtual void **linearOut** (void *a_buf) const
  Write a serialized (packed) representation into a_buf.
- virtual void **linearIn** (void *a_buf)
  Extract a serialized (packed) representation from a_buf.
- int **preAllocatable** ()
- virtual void **print** () const
  Write a human-readable version to stdout (for debugging).

Protected Attributes

- **RealVect m_position**
  this is the position in space of this item

Friends

- std::ostream & **operator**<< (std::ostream &ostr, const **BinItem** &a_item)

7.21.1 Detailed Description

Virtual base class for particle data.

BinItem is a base class to provide the necessary interface with **BinFab** for particle-in-bin/cell problems. Since the only attribute of a particle needed by the **BinFab** is position, it is the only attribute present in this class. All other attributes (charge, velocity, etc) must appear in a derived class.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 BinItem::BinItem ()

Null constructor, copy constructor and operator= can be compiler defined.

7.21.2.2 BinItem::BinItem (const RealVect & *a_position*)

Constructs a `BinItem` at *a_position*.

7.21.2.3 virtual BinItem::~BinItem () `[virtual]`

Destructor.

7.21.3 Member Function Documentation

7.21.3.1 virtual void BinItem::define (const RealVect & *a_position*) [virtual]

Same as the constructor.

Reimplemented in **DragParticle** (p. 243).

7.21.3.2 virtual void BinItem::linearIn (void * *a_buf*) [virtual]

Extract a serialized (packed) representation from *a_buf*.

Given a linear representation of the class data previously made using **linearIn()** in *a_buf*, set the data for this class.

7.21.3.3 virtual void BinItem::linearOut (void * *a_buf*) const [virtual]

Write a serialized (packed) representation into *a_buf*.

Write a linear representation of the internal data. Assumes that sufficient memory for the buffer has already been allocated by the caller.

Reimplemented in **DragParticle** (p. 243).

7.21.3.4 Real BinItem::position (const int *a_dir*) const

Gets one component of the position.

7.21.3.5 const RealVect& BinItem::position () const

Return a constant reference to the position.

7.21.3.6 RealVect& BinItem::position ()

Return a reference to the position.

7.21.3.7 int BinItem::preAllocatable () [inline]

Reimplemented in **DragParticle** (p. 244).

7.21.3.8 virtual void BinItem::print () const [virtual]

Write a human-readable version to stdout (for debugging).

7.21.3.9 void BinItem::setPosition (const Real *a_position*, const int *a_dimension*)

Change one component of the position.

7.21.3.10 void BinItem::setPosition (const RealVect & *a_position*)

Change the position.

7.21.3.11 virtual int BinItem::size () const [virtual]

Number of bytes used by **linearIn()/linearOut**().

Returns the size, in number of bytes, of a flat linear representation of the data in this object.

Reimplemented in **DragParticle** (p. 245).

7.21.4 Friends And Related Function Documentation

7.21.4.1 std::ostream& operator<< (std::ostream & *ostr*, const BinItem & *a_item*) [friend]

7.21.5 Member Data Documentation

7.21.5.1 RealVect BinItem::m_position [protected]

this is the position in space of this item

The documentation for this class was generated from the following file:

- **BinItem.H**

7.22 BitSet Class Reference

#include <BitSet.H>

Public Member Functions

- **BitSet** ()
- **BitSet** (int bits, bool init)
- void **define** (int bits, bool init)
- **BitSet** (const **BitSet** &rhs)
- **BitSet** & **operator=** (const **BitSet** &rhs)
- ~**BitSet** ()
- bool **operator**[ ] (int i) const
- void **setTrue** (int i)
- void **setFalse** (int i)
- bool **isEmpty** () const
- bool **isFull** () const
- int **size** () const
- int **linearSize** () const
- void **linearIn** (const void *const inBuf)
- void **linearOut** (void *const a_outBuf) const

Static Public Member Functions

- int **initialize** ()

Static Public Attributes

- long int **bytes**
- long int **peak**

Friends

- class **BitSetIterator**

7.22.1 Constructor & Destructor Documentation

7.22.1.1 BitSet::BitSet ()

7.22.1.2 BitSet::BitSet (int *bits*, bool *init*)

7.22.1.3 BitSet::BitSet (const BitSet & *rhs*)

7.22.1.4 BitSet::~BitSet ()

7.22.2 Member Function Documentation

7.22.2.1 void BitSet::define (int *bits*, bool *init*)

7.22.2.2 int BitSet::initialize () `[static]`

7.22.2.3 bool BitSet::isEmpty () const

returns 'true' if the entire bitset is zero

7.22.2.4 bool BitSet::isFull () const

returns 'true' if entire bitset is 1

7.22.2.5 void BitSet::linearIn (const void *const *inBuf*)

7.22.2.6 void BitSet::linearOut (void *const *a_outBuf*) const

7.22.2.7 int BitSet::linearSize () const

7.22.2.8 BitSet& BitSet::operator= (const BitSet & *rhs*)

7.22.2.9 bool BitSet::operator[] (int *i*) const `[inline]`

7.22.2.10 void BitSet::setFalse (int *i*)

7.22.2.11 void BitSet::setTrue (int *i*)

7.22.2.12 int BitSet::size () const `[inline]`

7.22.3 Friends And Related Function Documentation

7.22.3.1 friend class BitSetIterator `[friend]`

7.22.4 Member Data Documentation

7.22.4.1 long int BitSet::bytes `[static]`

7.22.4.2 long int BitSet::peak `[static]`

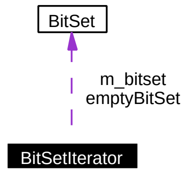
The documentation for this class was generated from the following file:

- **BitSet.H**

7.23 BitSetIterator Class Reference

`#include <BitSet.H>`

Collaboration diagram for BitSetIterator:



Public Member Functions

- **BitSetIterator** ()
- **BitSetIterator** (const **BitSet** &bitset)
- bool **operator()** () const
- bool **ok** () const
- void **operator++** ()
- void **begin** ()
- void **end** ()

7.23.1 Constructor & Destructor Documentation

7.23.1.1 BitSetIterator::BitSetIterator () `[inline]`

7.23.1.2 BitSetIterator::BitSetIterator (const BitSet & *bitset*) `[inline]`

7.23.2 Member Function Documentation

7.23.2.1 void BitSetIterator::begin ()

7.23.2.2 void BitSetIterator::end ()

7.23.2.3 bool BitSetIterator::ok () const `[inline]`

7.23.2.4 bool BitSetIterator::operator() () const `[inline]`

7.23.2.5 void BitSetIterator::operator++ () `[inline]`

The documentation for this class was generated from the following file:

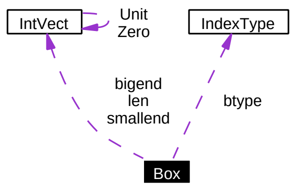
- **BitSet.H**

7.24 Box Class Reference

A Rectangular Domain on an Integer Lattice.

`#include <Box.H>`

Collaboration diagram for Box:



Public Member Functions

- **Box** ()
- ~**Box** ()
- **Box** (const **IntVect** &small, const **IntVect** &big)
- void **define** (const **IntVect** &small, const **IntVect** &big)
- **Box** (const **IntVect** &small, const int ∗vec_len)
- **Box** (const **IntVect** &small, const **IntVect** &big, const **IntVect** &typ)
- **Box** (const **IntVect** &small, const **IntVect** &big, const **IndexType** &t)
- **Box** (const **Box** &b)
- void **define** (const **Box** &b)
- **Box copy** () const
- const **IntVect** & **smallEnd** () const

  { *Accessors*}

- **IntVect sideEnd** (**Side::LoHiSide** a_side) const
- int **smallEnd** (int dir) const
- const **IntVect** & **bigEnd** () const
- int **bigEnd** (int dir) const
- const int ∗ **loVect** () const
- const int ∗ **hiVect** () const
- const int ∗ **getVect** () const
- long **index** (const **IntVect** &v) const
- **IndexType ixType** () const

  centering

- **IntVect type** () const
- **IndexType::CellIndex type** (int dir) const
- const **IntVect** & **size** () const

  size functions

- int **size** (int dir) const
- bool **numPtsOK** () const
- long **numPts** () const
- bool **volumeOK** () const

- long **volume** () const
- int **longside** (int &dir) const
- int **longside** () const
- int **shortside** (int &dir) const
- int **shortside** () const
- bool **isEmpty** () const

    { *Comparison Functions*}

- bool **contains** (const **IntVect** &p) const
- bool **contains** (const **Box** &b) const
- bool **intersects** (const **Box** &b) const
- bool **intersectsNotEmpty** (const **Box** &b) const
- bool **sameSize** (const **Box** &b) const
- bool **sameType** (const **Box** &b) const
- bool **operator==** (const **Box** &b) const
- bool **eq** (const **Box** &b) const
- bool **operator!=** (const **Box** &b) const
- bool **neq** (const **Box** &b) const
- bool **cellCentered** () const
- bool **operator<** (const **Box** &rhs) const
- bool **lt** (const **Box** &rhs) const
- **Box** & **operator=** (const **Box** &b)

    { *Modification Functions*}

- **Box** & **setSmall** (const **IntVect** &sm)
- **Box** & **setSmall** (int dir, int sm_index)
- **Box** & **setBig** (const **IntVect** &bg)
- **Box** & **setBig** (int dir, int bg_index)
- **Box** & **setRange** (int dir, int sm_index, int n_cells=1)
- **Box** & **convert** (**IndexType** typ)

    centering type conversion functions

- **Box** & **convert** (const **IntVect** &typ)
- **Box** & **convert** (int dir, **IndexType::CellIndex** typ)
- **Box** & **surroundingNodes** ()
- **Box** & **surroundingNodes** (int dir)
- **Box** & **surroundingNodes_int** (int dir)
- **Box** & **enclosedCells** ()
- **Box** & **enclosedCells** (int dir)
- **Box** & **enclosedCells_int** (int dir)
- **Box** & **shift** (int dir, int nzones)

    shift functions

- **Box** & **shift** (const **IntVect** &iv)
- **Box** & **shift_intvect** (const **IntVect** &iv)
- **Box** & **shiftHalf** (int dir, int num_halfs)
- **Box** & **shiftHalf** (const **IntVect** &iv)
- **Box** & **shiftHalf_intvect** (const **IntVect** &iv)
- **Box** & **operator+=** (const **IntVect** &v)
- **Box operator+** (const **IntVect** &v) const

- **Box** & **operator-=** (const **IntVect** &v)
- **Box** **operator-** (const **IntVect** &v) const
- **Box** **operator &** (const **Box** &) const

  intersection functions

- **Box** & **operator &=** (const **Box** &)
- **Box** & **minBox** (const **Box** &b)
- **Box** & **grow** (int i)

  grow functions

- **Box** & **grow** (const **IntVect** &v)
- **Box** & **grow** (int idir, int n_cell)
- **Box** & **growLo** (int idir, int n_cell=1)
- **Box** & **growDir** (int a_idir, const **Side::LoHiSide** &a_sd, int a_cell)
- **Box** & **growHi** (int idir, int n_cell=1)
- **Box** & **refine** (int refinement_ratio)

  refinement

- **Box** & **refine** (const **IntVect** &refinement_ratio)
- **Box** & **coarsen** (int refinement_ratio)

  coarsening

- **Box** & **coarsen** (const **IntVect** &refinement_ratio)
- void **next** (**IntVect** &) const
- void **next** (**IntVect** &p, const int ∗shv) const
- **Box** **chop** (int dir, int chop_pnt)

  misc

- void **p** () const
- void **dumpOn** (std::ostream &strm) const
- void **computeBoxLen** ()

  { *Box Constants*}

- void **computeBoxLenNotEmpty** ()

Protected Member Functions

- bool **numPtsOK** (long &N) const
- bool **volumeOK** (long &N) const

Protected Attributes

- **IntVect smallend**
- **IntVect bigend**
- **IntVect len**
- **IndexType btype**

Friends

- class **HDF5Handle**
- **Box surroundingNodes** (const **Box** &b, int dir)
- **Box surroundingNodes** (const **Box** &b)
- **Box enclosedCells** (const **Box** &b, int dir)
- **Box enclosedCells** (const **Box** &b)
- **Box bdryBox** (const **Box** &b, int dir, **Side::LoHiSide** a_sd, int **len**)

  neighbor box functions

- **Box bdryLo** (const **Box** &b, int dir, int **len**)
- **Box bdryHi** (const **Box** &b, int dir, int **len**)
- **Box adjCellLo** (const **Box** &b, int dir, int **len**)
- **Box adjCellHi** (const **Box** &b, int dir, int **len**)
- **Box adjCellBox** (const **Box** &b, int dir, **Side::LoHiSide** a_side, int **len**)
- **Box minBox** (const **Box** &b1, const **Box** &b2)
- **Box grow** (const **Box** &b, int i)
- **Box grow** (const **Box** &b, const **IntVect** &v)
- **Box refine** (const **Box** &b, int refinement_ratio)
- **Box refine** (const **Box** &b, const **IntVect** &refinement_ratio)
- **Box coarsen** (const **Box** &b, int refinement_ratio)
- **Box coarsen** (const **Box** &b, const **IntVect** &refinement_ratio)
- std::ostream & **operator**<< (std::ostream &os, const **Box** &bx)

  { *I/O Functions*}

- std::istream & **operator**>> (std::istream &os, **Box** &bx)

7.24.1 Detailed Description

A Rectangular Domain on an Integer Lattice.

A Box is an abstraction for defining discrete rectangular regions of SpaceDim-dimensioned indexing space. Boxes have an **IndexType**, which defines **IndexType::CELL** or **IndexType::NODE** based points for each direction and a low and high **IntVect** which defines the lower and upper corners of the Box. Boxes can exist in positive and negative indexing space.

Box is a dimension dependent class, so SpaceDim must be defined as either 1, 2, or 3 when compiling.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 Box::Box ()

The default constructor. The constructed Box is empty.

7.24.2.2 Box::~Box () `[inline]`

The destructor.

7.24.2.3 Box::Box (const IntVect & *small*, const IntVect & *big*)

Construct cell-centered type Box. It is an error if small is greater than big.

7.24.2.4 Box::Box (const IntVect & *small*, const int * *vec_len*)

Construct Box with specified lengths. It is an error if the lengths are negative.

7.24.2.5 Box::Box (const IntVect & *small*, const IntVect & *big*, const IntVect & *typ*)

Construct Box with given type. small and big are expected to be consistent with given type. It is an error if small is greater than big. It is an error if typ is not convertible to an **IndexType**.

7.24.2.6 Box::Box (const IntVect & *small*, const IntVect & *big*, const IndexType & *t*)

Construct Box with given type. small and big are expected to be consistent with given type. It is an error if small is greater than big.

7.24.2.7 Box::Box (const Box & *b*) `[inline]`

The copy constructor.

7.24.3 Member Function Documentation

7.24.3.1 int Box::bigEnd (int *dir*) const `[inline]`

Returns the coordinate of the high end of this Box in the given direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.2 const IntVect & Box::bigEnd () const `[inline]`

Returns the upper corner of this Box.

7.24.3.3 bool Box::cellCentered () const `[inline]`

Returns true if this Box is cell-centered in all indexing directions.

7.24.3.4 Box Box::chop (int *dir*, int *chop_pnt*)

misc

Chops the Box at the chop in the dir direction returns one Box, modifies the object Box. The union of the two is the original Box. The modified Box is the low end, the returned Box is the high end. If type(dir) = CELL, the Boxes are disjoint with the chop included in the high end (new Box). It is an ERROR if chop is the low end of the orig Box. If type(dir) = NODE, the chop is included in both Boxes but is the only point in common. It is also an error if the chop is an end node of the Box. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.5 Box& Box::coarsen (const IntVect & *refinement_ratio*)

Modifies this Box by coarsening by given (positive) refinement ratio. The Empty Box is not modified by this function. \

NOTE: if type(dir) = CELL centered: lo <- lo/ratio and hi <- hi/ratio.\

NOTE: if type(dir) = NODE centered: lo <- lo/ratio and hi <- hi/ratio + ((hi%ratio)==0 ? 0 : 1).\

That is, refinement of coarsened Box must contain the original Box.

7.24.3.6 Box& Box::coarsen (int *refinement_ratio*)

coarsening

Modifies this Box by coarsening it by given (positive) refinement ratio. The Empty Box is not modified by this function. \

NOTE: if type(dir) = CELL centered: lo <- lo/ratio and hi <- hi/ratio.\

NOTE: if type(dir) = NODE centered: lo <- lo/ratio and hi <- hi/ratio + ((hi%ratio)==0 ? 0 : 1).\

That is, refinement of coarsened Box must contain the original Box.

7.24.3.7 void Box::computeBoxLen () `[inline]`

{ Box Constants}

This is a canonical empty Box of cell-centered type.

7.24.3.8 void Box::computeBoxLenNotEmpty () `[inline]`

7.24.3.9 bool Box::contains (const Box & *b*) const `[inline]`

Returns true if argument is contained within this Box. It is an error if the Boxes have different types. An empty Box does not contain any **IntVect**.

7.24.3.10 bool Box::contains (const IntVect & *p*) const `[inline]`

Returns true if argument is contained within this Box. An empty Box does not contain and is not contained by any Box, including itself.

7.24.3.11 Box& Box::convert (int *dir*, IndexType::CellIndex *typ*)

Modifies this Box by converting from the current type into the argument type. This may change the Box coordinates:\

type CELL -> NODE : increase coordinate by one on high end.\

type NODE -> CELL : reduce coordinate by one on high end.\

Other type mappings make no change. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.12 Box & Box::convert (const IntVect & *typ*) [inline]

Modifies this Box by converting from the current type into the argument type. This may change the Box coordinates:\ type CELL -> NODE : increase coordinate by one on high end.\ type NODE -> CELL : reduce coordinate by one on high end.\ other type mappings make no change. It is an error if typ is not convertible to an **IndexType**.

7.24.3.13 Box& Box::convert (IndexType *typ*)

centering type conversion functions

Modifies this Box by converting from the current type into the argument type. This may change the Box coordinates:\

type CELL -> NODE : increase coordinate by one on high end.\

type NODE -> CELL : reduce coordinate by one on high end.\

other type mappings make no change.

7.24.3.14 Box Box::copy () const [inline]

7.24.3.15 void Box::define (const Box & *b*)

7.24.3.16 void Box::define (const IntVect & *small*, const IntVect & *big*)

Construct cell-centered type Box. It is an error if small is greater than big.

7.24.3.17 void Box::dumpOn (std::ostream & *strm*) const

Gives more detail than printOn. Useful for exiting due to an error.

7.24.3.18 Box & Box::enclosedCells (int *dir*) [inline]

Modifies this Box by converting it to CELL type in given direction. This decreases the component of the upper corner in the given direction by one. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The Empty Box is not modified by this function except for changing of type.

7.24.3.19 Box& Box::enclosedCells ()

Modifies this Box by converting it to CELL type in all directions. This decreases all components of the upper corner by one. The Empty Box is not modified by this function except for changing of type.

7.24.3.20 Box& Box::enclosedCells_int (int *dir*)

7.24.3.21 bool Box::eq (const Box & *b*) const

7.24.3.22 const int * Box::getVect () const [inline]

Returns a constant pointer to the array of coordinates in the Box. Useful for calls to Fortran, but otherwise too dangerous for use.

7.24.3.23 Box & Box::grow (int *idir*, int *n_cell*) [inline]

Modifies this Box by growing it on the low and high end by n cells in direction idir. The Empty Box is not modified by this function. NOTE: n negative shrinks this Box by that number of cells. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.24 Box & Box::grow (const IntVect & *v*) [inline]

Modifies this Box by growing it in each direction by specified amount. The Empty Box is not modified by this function. NOTE: components of iv may be negative, which would shrink this Box. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box.

7.24.3.25 Box & Box::grow (int *i*) [inline]

grow functions

Modifies this Box by growing it in all directions by given amount. The Empty Box is not modified by this function. NOTE: n negative shrinks the Box by that number of cells. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box.

7.24.3.26 Box & Box::growDir (int *a_idir*, const Side::LoHiSide & *a_sd*, int *a_cell*) [inline]

growLo if a_sd == **Side::Lo** \ growHi if a_sd= = **Side::Hi**

7.24.3.27 Box & Box::growHi (int *idir*, int *n_cell* = 1) [inline]

Modifies this Box by growing it on the high end by n cells in direction idir. The Empty Box is not modified by this function. NOTE: n negative shrinks the Box by that number of cells. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.28 Box & Box::growLo (int *idir*, int *n_cell* = 1) [inline]

Modifies this Box by growing it on the low end by n cells in direction idir. The Empty Box is not modified by this function. NOTE: n negative shrinks this Box by that number of cells. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.29 const int * Box::hiVect () const [inline]

Returns a constant pointer the array of high end coordinates. Useful for calls to Fortran. It should not be used in any other circumstances.

7.24.3.30 long Box::index (const IntVect & *v*) const [inline]

Returns offset of point from smallend; i.e. index(smallend) -> 0, bigend would return **volume()**-1. Is used in accessing **FArrayBox**.

7.24.3.31 bool Box::intersects (const Box & *b*) const

Returns true if this Box and the argument have non-null intersections. It is an error if the Boxes have different types. An empty Box does not intersect any Box, including itself.

7.24.3.32 bool Box::intersectsNotEmpty (const Box & *b*) const

Returns true if this Box and the argument have non-null intersections. It is an error if the Boxes have different types. This routine does not perform the check to see if *this or b are empty boxes. It is the callers responsibility to ensure that this never happens. If you are unsure, the use the .intersects(..) routine.

7.24.3.33 bool Box::isEmpty () const [inline]

{ Comparison Functions}

Returns true if this Box is empty.

7.24.3.34 IndexType Box::ixType () const [inline]

centering

Return the indexing type of this Box.

7.24.3.35 int Box::longside () const

Returns length of the longest side of this Box. Ignores type.

7.24.3.36 int Box::longside (int & *dir*) const

Returns length of the longest side of this Box. The argument dir is modified to give direction with longest side: 0...SpaceDim-1. Ignores type.

7.24.3.37 const int * Box::loVect () const [inline]

Returns a constant pointer to the array of low end coordinates. Useful for calls to Fortran. It should not be used in any other circumstances.

7.24.3.38 bool Box::lt (const Box & *rhs*) const

7.24.3.39 Box& Box::minBox (const Box & *b*)

Modifies this Box to that of the minimum Box containing both this Box and the argument Box. Both Boxes must have identical type.

7.24.3.40 bool Box::neq (const Box & *b*) const

7.24.3.41 void Box::next (IntVect & *p*, const int ∗ *shv*) const

7.24.3.42 void Box::next (IntVect &) const

7.24.3.43 long Box::numPts () const

Return the volume, in indexing space, of region enclosed by this Box. This is identical to **volume()**. Abort()s if the number cannot be represented in a long.\

7.24.3.44 bool Box::numPtsOK (long & *N*) const `[protected]`

7.24.3.45 bool Box::numPtsOK () const `[inline]`

Is the number calculated by **numPts()** representable in a long?

7.24.3.46 Box Box::operator & (const Box &) const `[inline]`

intersection functions

Returns the Box that is intersection of this Box and the argument Box. Intersection is commutative. The Boxes MUST be of same type. The intersection of the Empty Box and any Box is the Empty Box.

7.24.3.47 Box& Box::operator &= (const Box &)

Modifies this Box by intersection with the argument Box. The Boxes MUST be of the same type. The intersection of the Empty Box and any Box is the Empty Box.

7.24.3.48 bool Box::operator!= (const Box & *b*) const `[inline]`

Returns true if this Box and the argument differ, including type.

7.24.3.49 Box Box::operator+ (const IntVect & *v*) const `[inline]`

Returns a Box that is this Box shifted by the given **IntVect**. The Empty Box is not modified by this function.

7.24.3.50 Box & Box::operator+= (const IntVect & *v*) `[inline]`

Modifies this Box by shifting it by given **IntVect**. The Empty Box is not modified by this function.

7.24.3.51 Box Box::operator- (const IntVect & *v*) const [inline]

Returns a Box that is this Box shifted by the given **IntVect**. The Empty Box is not modified by this function.

7.24.3.52 Box & Box::operator-= (const IntVect & *v*) [inline]

Modifies this Box by shifting it by given **IntVect**. The Empty Box is not modified by this function.

7.24.3.53 bool Box::operator< (const Box & *rhs*) const [inline]

Returns true if this Box is lexigraphically less than rhs box. All comparison is based on lower box corner. In the name of coding efficiency, we do not handle the case where two boxes have the same lower left corner. In DEBUG mode, this is checked for with an assert.

7.24.3.54 Box & Box::operator= (const Box & *b*) [inline]

{ Modification Functions}

The assignment operator.

7.24.3.55 bool Box::operator== (const Box & *b*) const [inline]

Returns true if this Box and the argument are identical, including type.

7.24.3.56 void Box::p () const

print to **pout()**

7.24.3.57 Box& Box::refine (const IntVect & *refinement_ratio*)

Modifies this Box by refining it by given (positive) refinement ratio. The Empty Box is not modified by this function. \

NOTE: if type(dir) = CELL centered: lo <- lo∗ratio and hi <- (hi+1)∗ratio - 1.\

NOTE: if type(dir) = NODE centered: lo <- lo∗ratio and hi <- hi∗ratio.

7.24.3.58 Box& Box::refine (int *refinement_ratio*)

refinement

Modifies this Box by refining it by given (positive) refinement ratio. The Empty Box is not modified by this function. \

NOTE: if type(dir) = CELL centered: lo <- lo∗ratio and hi <- (hi+1)∗ratio - 1. \

NOTE: if type(dir) = NODE centered: lo <- lo∗ratio and hi <- hi∗ratio.

7.24.3.59 bool Box::sameSize (const Box & *b*) const [inline]

Returns true if this Box and the argument are the same size, ie translates of each other. It is an error if they have different types.

7.24.3.60 bool Box::sameType (const Box & *b*) const [inline]

Returns true if this Box and the argument have same type.

7.24.3.61 Box & Box::setBig (int *dir*, int *bg_index*) [inline]

Redefines the upper end of this Box in the specified coordinate direction. It is an error if the specified value is less than the exisiting lower end of this Box. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.62 Box & Box::setBig (const IntVect & *bg*) [inline]

Redefines the upper corner of this Box. It is an error if the specified corner is less than the exisiting lower corner of this Box.

7.24.3.63 Box & Box::setRange (int *dir*, int *sm_index*, int *n_cells* = 1) [inline]

Set the entire range in a given direction, starting at sm with length n. It is an error if n <= 0.

7.24.3.64 Box & Box::setSmall (int *dir*, int *sm_index*) [inline]

Redefines the lower end of this Box in the specified coordinate direction. It is an error if the specified value is greater than the exisiting upper end of this Box. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.65 Box & Box::setSmall (const IntVect & *sm*) [inline]

Redefine the lower corner of this Box. It is an error if the specified corner is greater than the exisiting upper corner of this Box.

7.24.3.66 Box & Box::shift (const IntVect & *iv*) [inline]

Modifies this Box by shifting. Equivalent to b.shift(0,iv[0]).shift(1,iv[1]) .... The Empty Box is not modified by this function.

7.24.3.67 Box & Box::shift (int *dir*, int *nzones*) [inline]

shift functions

Modifies this Box by shifting it nzones indexing positions in coordinate direction dir. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The Empty Box is not modified by this function.

7.24.3.68 Box& Box::shift_intvect (const IntVect & *iv*)

7.24.3.69 Box& Box::shiftHalf (const IntVect & *iv*)

Modifies this Box by shifting by half indices. Equivalent to b.shiftHalf(0,iv[0]).shiftHalf(1,iv[1]) ... The Empty Box is not modified by this function except for changing of type.

7.24.3.70 Box& Box::shiftHalf (int *dir*, int *num_halfs*)

This member modifies this Box by shifting by "half" indices, thereby converting the Box from type CELL to NODE or vice-versa. b.shiftHalf(0,1) shifts b to the right by 1/2 cells. b.shiftHalf(1,-3) shifts b in the -j direction by 3/2 cells. NOTE: If num is EVEN the shift is num/2 full zones and hence will not change the type. This is: b.shifthalf(4) == b.shift(2). Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The Empty Box is not modified by this function except for changing of type.

7.24.3.71 Box& Box::shiftHalf_intvect (const IntVect & *iv*)

7.24.3.72 int Box::shortside () const

Returns length of the shortest side of this Box. Ignores type.

7.24.3.73 int Box::shortside (int & *dir*) const

Returns length of shortest side of this Box. The argument dir is modified to give direction with shortest side: 0...SpaceDim-1. Ignores type.

7.24.3.74 IntVect Box::sideEnd (Side::LoHiSide *a_side*) const `[inline]`

return smallend if side is low, bigend if side is high.

7.24.3.75 int Box::size (int *dir*) const `[inline]`

Return the size of this Box in the specified coordinate direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.76 const IntVect & Box::size () const `[inline]`

size functions

Return an **IntVect** containing the size of this Box in each coordinate direction.

7.24.3.77 int Box::smallEnd (int *dir*) const `[inline]`

Returns the coordinate of the low end of this Box in the given direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.78 const IntVect & Box::smallEnd () const [inline]

{ Accessors}

Returns the lower corner of this Box.

7.24.3.79 Box & Box::surroundingNodes (int *dir*) [inline]

Modifies this Box by converting it to NODE type in given direction. This increases the component of the upper corner in the given direction by one. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The Empty Box is not modified by this function except for changing of type.

7.24.3.80 Box& Box::surroundingNodes ()

Modifies this Box by converting it to NODE type in all directions. This increases all components of the upper corner by one. The Empty Box is not modified by this function except for changing of type.

7.24.3.81 Box& Box::surroundingNodes_int (int *dir*)

7.24.3.82 IndexType::CellIndex Box::type (int *dir*) const [inline]

Return the indexing type of this Box in the specified direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

7.24.3.83 IntVect Box::type () const [inline]

Return the indexing type of this Box.

7.24.3.84 long Box::volume () const

Return the volume, in indexing space, of region enclosed by this Box. This is identical to **numPts()**. Abort()s if the number cannot be represented in a long.\

7.24.3.85 bool Box::volumeOK (long & *N*) const [protected]

7.24.3.86 bool Box::volumeOK () const [inline]

Is the number calculated by **volume()** representable in a long?

7.24.4 Friends And Related Function Documentation

7.24.4.1 Box adjCellBox (const Box & *b*, int *dir*, Side::LoHiSide *a_side*, int *len*) [friend]

7.24.4.2 Box adjCellHi (const Box & *b*, int *dir*, int *len*) [friend]

Returns the cell centered Box of the given length adjacent to the argument Box on the high end along the given coordinate direction. The return Box is identical to the argument Box in the other directions. The return Box and the argument Box have an empty intersection. \

NOTE: len >= 1. \

NOTE: Box retval = adjCellHi(b,dir,len) is equivalent to the following set of operations: \

Box retval(b); \

retval.convert(dir,Box::CELL); \

retval.setrange(dir,retval.bigEnd(dir)+1,len);\

Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty Box is an Empty Box of the appropriate type.

7.24.4.3 Box adjCellLo (const Box & *b*, int *dir*, int *len*) [friend]

Returns the cell centered Box of the given length adjacent to the argument Box on the low end along the given coordinate direction. The return Box is identical to the argument Box in the other directions. The return Box and the argument Box have an empty intersection. \

NOTE: len >= 1. \

NOTE: Box retval = adjCellLo(b,dir,len) is equivalent to the following set of operations: \

Box retval(b); \

retval.convert(dir,Box::CELL); \

retval.setrange(dir,retval.smallEnd(dir)-len,len); \

Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty Box is an Empty Box of the appropriate type.

7.24.4.4 Box bdryBox (const Box & *b*, int *dir*, Side::LoHiSide *a_sd*, int *len*) [friend]

neighbor box functions

7.24.4.5 Box bdryHi (const Box & *b*, int *dir*, int *len*) [friend]

Returns the edge-centered Box (in direction dir) defining the high side of the argument Box. The return Box will have the given length in the given direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty Box is an Empty Box of the appropriate type.

7.24.4.6 Box bdryLo (const Box & *b*, int *dir*, int *len*) [friend]

Returns the edge-centered Box (in direction dir) defining the low side of the argument Box. The output Box will have the given length in the given direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty Box is an Empty Box of the appropriate type.

7.24.4.7 Box coarsen (const Box & *b*, const IntVect & *refinement_ratio*) [friend]

Returns a Box that is the argument Box coarsened by given (positive) refinement ratio. The Empty Box is not modified by this function. \

NOTE: if type(dir) = CELL centered: lo <- lo/ratio and hi <- hi/ratio.\

NOTE: if type(dir) = NODE centered: lo <- lo/ratio and hi <- hi/ratio + ((hi%ratio)==0 ? 0 : 1).\

That is, refinement of coarsened Box must contain the original Box.

7.24.4.8 Box coarsen (const Box & *b*, int *refinement_ratio*) [friend]

Returns a Box that is the argument Box coarsened by given (positive) refinement ratio. The Empty Box is not modified by this function. \

NOTE: if type(dir) = CELL centered: lo <- lo/ratio and hi <- hi/ratio.\

NOTE: if type(dir) = NODE centered: lo <- lo/ratio and hi <- hi/ratio + ((hi%ratio)==0 ? 0 : 1).\

That is, refinement of coarsened Box must contain the original Box.

7.24.4.9 Box enclosedCells (const Box & *b*) [friend]

Returns a Box with CELL based coordinates in all directions that is enclosed by argument Box. The upper corner of the return Box will be one less in all components than the upper corner of the argument Box. The Empty Box is not modified by this function except for changing of type.

7.24.4.10 Box enclosedCells (const Box & *b*, int *dir*) [friend]

Returns a Box with CELL based coordinates in direction dir that is enclosed by argument Box. NOTE: equivalent to b.convert(dir,CELL). NOTE: error if b.type(dir) == CELL. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The Empty Box is not modified by this function except for changing of type.

7.24.4.11 Box grow (const Box & *b*, const IntVect & *v*) [friend]

Returns a Box that is the argument Box grown in each direction by specified amount. The Empty Box is not modified by this function. NOTE: components of iv may be negative, which would return a shrunken Box. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box.

7.24.4.12 Box grow (const Box & *b*, int *i*) [friend]

Returns a Box that is the argument Box grown in all directions by given amount. The Empty Box is not modified by this function. NOTE: n negative shrinks the Box by that number of cells. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box.

7.24.4.13 friend class HDF5Handle [friend]

7.24.4.14 Box minBox (const Box & *b1*, const Box & *b2*) [friend]

Returns a Box that is the minimum Box containing both the argument Boxes. Both Boxes must have identical type.

7.24.4.15 std::ostream& operator<< (std::ostream & *os*, const Box & *bx*) [friend]

{ I/O Functions}

Write an ASCII representation to the ostream.

7.24.4.16 std::istream& operator>> (std::istream & *os*, Box & *bx*) [friend]

Read from istream.

7.24.4.17 Box refine (const Box & *b*, const IntVect & *refinement_ratio*) [friend]

Returns a Box that is the argument Box refined by given (positive) refinement ratio. The Empty Box is not modified by this function. \

NOTE: if type(dir) = CELL centered: lo <- lo*ratio and hi <- (hi+1)*ratio - 1.\

NOTE: if type(dir) = NODE centered: lo <- lo*ratio and hi <- hi*ratio.

7.24.4.18 Box refine (const Box & *b*, int *refinement_ratio*) [friend]

Returns a Box that is the argument Box refined by given (positive) refinement ratio. The Empty Box is not modified by this function. \

NOTE: if type(dir) = CELL centered: lo <- lo*ratio and hi <- (hi+1)*ratio - 1.\

NOTE: if type(dir) = NODE centered: lo <- lo*ratio and hi <- hi*ratio.

7.24.4.19 Box surroundingNodes (const Box & *b*) [friend]

Returns a Box with NODE based coordinates in all directions that encloses argument Box. The upper corner of the return Box will be one more in all components than the upper corner of the argument Box. The Empty Box is not modified by this function except for changing of type.

7.24.4.20 Box surroundingNodes (const Box & *b*, int *dir*) [friend]

Returns a Box with NODE based coordinates in direction dir that encloses the argument Box. NOTE: equivalent to b.convert(dir,NODE). NOTE: error if b.type(dir) == NODE. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The Empty Box is not modified by this function except for changing of type.

7.24.5 Member Data Documentation

7.24.5.1 IntVect Box::bigend [protected]

7.24.5.2 IndexType Box::btype [protected]

7.24.5.3 IntVect Box::len [protected]

7.24.5.4 IntVect Box::smallend [protected]

The documentation for this class was generated from the following file:

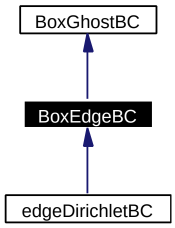
- **Box.H**

7.25 BoxEdgeBC Class Reference

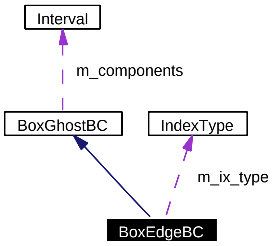
class to do edge-centered physical boundary conditions

`#include <edgeGhostBC.H>`

Inheritance diagram for BoxEdgeBC:



Collaboration diagram for BoxEdgeBC:



Public Member Functions

- virtual ~**BoxEdgeBC** ()
- **BoxEdgeBC** ()
- **BoxEdgeBC** (int a_dir, **Side::LoHiSide** a_sd)

  sets interval to be (0,0)
- **BoxEdgeBC** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps)
- **BoxEdgeBC** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps, **IndexType** &a_-ixtype)
- const **IndexType** & **ixType** () const

  returns the type of centering of this edge-centered BC
- void **ixType** (const **IndexType** &a_ixType)

Protected Member Functions

- virtual **BoxGhostBC** ∗ **new_boxghostbc** () const

  virtual constructor workaround
- void **define** (int a_dir, **Side::LoHiSide** a_sd)

sets interval to be (0,0)

- void **define** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps)
- void **define** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps, **IndexType** &a_ixtype)
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

  deprecated interface

- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

  deprecated interface

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

  deprecated interface

- virtual void **applyBCs** (const **Box** &a_bcbox, **FArrayBox** &a_state, const **FArrayBox** &a_neumfac, const **FArrayBox** &a_dircfac, const **FArrayBox** &a_inhmval, **Real** a_dx) const

7.25.1 Detailed Description

class to do edge-centered physical boundary conditions

BoxEdgeBC is designed to do physical BC's on edge centered data. It is derived from **BoxGhostBC** so that it can be used seamlessly with BoxGhostBCs.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 virtual BoxEdgeBC::~BoxEdgeBC () `[virtual]`

7.25.2.2 BoxEdgeBC::BoxEdgeBC ()

7.25.2.3 BoxEdgeBC::BoxEdgeBC (int *a_dir*, Side::LoHiSide *a_sd*)

sets interval to be (0,0)

7.25.2.4 BoxEdgeBC::BoxEdgeBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*)

7.25.2.5 BoxEdgeBC::BoxEdgeBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*, IndexType & *a_ixtype*)

7.25.3 Member Function Documentation

7.25.3.1 virtual void BoxEdgeBC::applyBCs (const Box & *a_bcbox*, FArrayBox & *a_state*, const FArrayBox & *a_neumfac*, const FArrayBox & *a_dircfac*, const FArrayBox & *a_inhmval*, Real *a_dx*) const `[protected, virtual]`

Actually applies the boundary conditions defined by the values in a_neumfac, a_dircfac, and a_inhmval to a_state in the region defined by a_bcBox (which should be the appropriate ghost cells)

Reimplemented from **BoxGhostBC** (p. 140).

7.25.3.2 virtual void BoxEdgeBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

Reimplemented from **BoxGhostBC** (p. 140).

7.25.3.3 virtual void BoxEdgeBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.25.3.4 virtual void BoxEdgeBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

Reimplemented from **BoxGhostBC** (p. 141).

7.25.3.5 virtual void BoxEdgeBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.25.3.6 void BoxEdgeBC::define (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*, IndexType & *a_ixtype*) [protected]

7.25.3.7 void BoxEdgeBC::define (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*) [protected]

Full define function

Reimplemented from **BoxGhostBC** (p. 141).

7.25.3.8 void BoxEdgeBC::define (int *a_dir*, Side::LoHiSide *a_sd*) [protected]

sets interval to be (0,0)

Reimplemented from **BoxGhostBC** (p. 141).

7.25.3.9 virtual void BoxEdgeBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const [protected, virtual]

deprecated interface

Implements **BoxGhostBC** (p. 141).

Reimplemented in **edgeDirichletBC** (p. 248).

7.25.3.10 virtual void BoxEdgeBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const [protected, virtual]

Pure-virtual Worker function to fill arrays which define the boundary condition. Essentially, if the boundary condition is represented as A∗phi + B∗dphi/dn = C, then a_dircfac is A, a_neumfac is B, and a_inhmval is C.

Implements **BoxGhostBC** (p. 141).

Reimplemented in **edgeDirichletBC** (p. 248).

7.25.3.11 void BoxEdgeBC::ixType (const IndexType & *a_ixType*)

7.25.3.12 const IndexType& BoxEdgeBC::ixType () const

returns the type of centering of this edge-centered BC

7.25.3.13 virtual BoxGhostBC∗ BoxEdgeBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround

Implements **BoxGhostBC** (p. 141).

Reimplemented in **edgeDirichletBC** (p. 249).

The documentation for this class was generated from the following file:

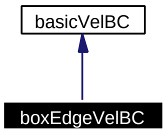
- **edgeGhostBC.H**

7.26 boxEdgeVelBC Class Reference

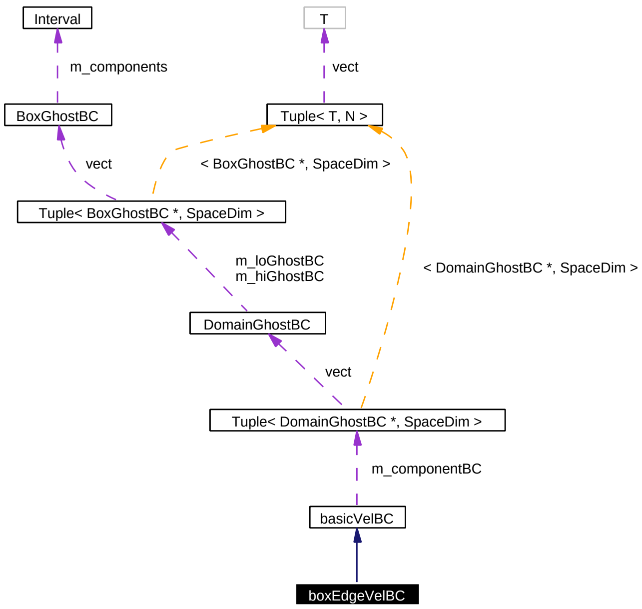
implements edge-centered velocity BC's for flow in a box!

`#include <boxVelBC.H>`

Inheritance diagram for boxEdgeVelBC:



Collaboration diagram for boxEdgeVelBC:



Public Member Functions

- **boxEdgeVelBC** (bool isViscous=false)
- virtual **~boxEdgeVelBC** ()
- virtual void **setAllBCTypes** ()
- bool **isViscous** ()

  access function

- void **isViscous** (bool a_isViscous)
- virtual void **applyBCs** (**LevelData**< **FluxBox** > &a_edgeStateMF, const **ProblemDomain** &a_domain, const **Real** a_dx) const

  also need to implement edge-centered version of applyBCs

- virtual void **applyBCs** (**LevelData**< **FluxBox** > &a_edgeStateMF, const **Box** &a_domain, const **Real** a_dx) const

  also need to implement edge-centered version of applyBCs (old interface)

- virtual void **applyBCs** (**FluxBox** &a_edgeStateMF, const **ProblemDomain** &a_domain, const **Real** a_dx) const

  also need to implement edge-centered version of applyBCs

- virtual void **applyBCs** (**FluxBox** &a_edgeStateMF, const **Box** &a_domain, const **Real** a_dx) const

  also need to implement edge-centered version of applyBCs (old interface)

- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx) const

  since the other two functions hide the applyBCs function...

- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx) const

  since the other two functions hide the applyBCs function (old interface)

- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx, int a_velComp) const

  this one assumes that each edge-centered FAB is one component

- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx, int a_velComp) const

  this one assumes each edge-centered FAB is one component (old interface)

Protected Attributes

- bool **m_isViscous**

7.26.1 Detailed Description

implements edge-centered velocity BC's for flow in a box!

7.26.2 Constructor & Destructor Documentation

7.26.2.1 boxEdgeVelBC::boxEdgeVelBC (bool *isViscous* = false)

7.26.2.2 virtual boxEdgeVelBC::~boxEdgeVelBC () `[inline, virtual]`

7.26.3 Member Function Documentation

7.26.3.1 virtual void boxEdgeVelBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*, int *a_velComp*) const `[virtual]`

this one assumes each edge-centered FAB is one component (old interface)

Reimplemented from **basicVelBC** (p. 89).

7.26.3.2 virtual void boxEdgeVelBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*, int *a_velComp*) const [virtual]

this one assumes that each edge-centered FAB is one component

Reimplemented from **basicVelBC** (p. 89).

7.26.3.3 virtual void boxEdgeVelBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*) const [virtual]

since the other two functions hide the applyBCs function (old interface)

Reimplemented from **basicVelBC** (p. 90).

7.26.3.4 virtual void boxEdgeVelBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

since the other two functions hide the applyBCs function...

Reimplemented from **basicVelBC** (p. 90).

7.26.3.5 virtual void boxEdgeVelBC::applyBCs (FluxBox & *a_edgeStateMF*, const Box & *a_domain*, const Real *a_dx*) const [virtual]

also need to implement edge-centered version of applyBCs (old interface)

7.26.3.6 virtual void boxEdgeVelBC::applyBCs (FluxBox & *a_edgeStateMF*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

also need to implement edge-centered version of applyBCs

7.26.3.7 virtual void boxEdgeVelBC::applyBCs (LevelData< FluxBox > & *a_edgeStateMF*, const Box & *a_domain*, const Real *a_dx*) const [virtual]

also need to implement edge-centered version of applyBCs (old interface)

7.26.3.8 virtual void boxEdgeVelBC::applyBCs (LevelData< FluxBox > & *a_edgeStateMF*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

also need to implement edge-centered version of applyBCs

7.26.3.9 void boxEdgeVelBC::isViscous (bool *a_isViscous*)

7.26.3.10 bool boxEdgeVelBC::isViscous ()

access function

7.26.3.11 virtual void boxEdgeVelBC::setAllBCTypes () [virtual]

7.26.4 Member Data Documentation

7.26.4.1 bool boxEdgeVelBC::m_isViscous [protected]

The documentation for this class was generated from the following file:

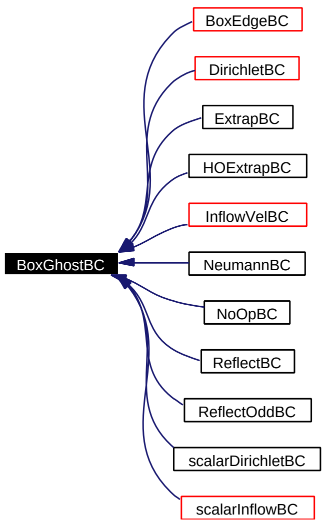
- **boxVelBC.H**

7.27 BoxGhostBC Class Reference

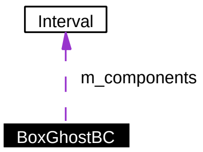
Class to encapsulate the operations of ghost-cell boundary conditions.

`#include <GhostBC.H>`

Inheritance diagram for BoxGhostBC:



Collaboration diagram for BoxGhostBC:



Public Member Functions

- **BoxGhostBC** ()

  Null constructor.

- virtual ~**BoxGhostBC** ()

  Destructor.

- **BoxGhostBC** (int a_dir, **Side::LoHiSide** a_sd)

  Sets interval to be (0,0).

- **BoxGhostBC** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps)

  Full constructor.

Protected Member Functions

- virtual **BoxGhostBC** * **new_boxghostbc** () const=0
- void **define** (int a_dir, **Side::LoHiSide** a_sd)
- void **define** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps)
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const
- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const=0
- virtual void **applyBCs** (const **Box** &a_bcbox, **FArrayBox** &a_state, const **FArrayBox** &a_neumfac, const **FArrayBox** &a_dircfac, const **FArrayBox** &a_inhmval, **Real** a_dx) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const=0

Protected Attributes

- **Side::LoHiSide m_side**
- int **m_direction**
- **Interval m_components**

Friends

- class **DomainGhostBC**

7.27.1 Detailed Description

Class to encapsulate the operations of ghost-cell boundary conditions.

BoxGhostBC is a class which encapsulates the operations of ghost-cell boundary conditions at a single face. If the solution is phi and the face normal direction is x, the boundary conditions usually used can be expressed as A*phi + B*dphi/dx = C. The derived class provides a function to to fill A B and C. These functions are only meant to be called by **DomainGhostBC**.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 BoxGhostBC::BoxGhostBC () `[inline]`

Null constructor.

7.27.2.2 virtual BoxGhostBC::~BoxGhostBC () `[inline, virtual]`

Destructor.

7.27.2.3 BoxGhostBC::BoxGhostBC (int *a_dir*, Side::LoHiSide *a_sd*)

Sets interval to be (0,0).

7.27.2.4 BoxGhostBC::BoxGhostBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*)

Full constructor.

7.27.3 Member Function Documentation

7.27.3.1 virtual void BoxGhostBC::applyBCs (const Box & *a_bcbox*, FArrayBox & *a_state*, const FArrayBox & *a_neumfac*, const FArrayBox & *a_dircfac*, const FArrayBox & *a_inhmval*, Real *a_dx*) const `[protected, virtual]`

Actually applies the boundary conditions defined by the values in a_neumfac, a_dircfac, and a_inhmval to a_state in the region defined by a_bcBox (which should be the appropriate ghost cells)

Reimplemented in **BoxEdgeBC** (p. 131).

7.27.3.2 virtual void BoxGhostBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented in **BoxEdgeBC** (p. 131), **NoOpBC** (p. 425), **ExtrapBC** (p. 254), **HOExtrapBC** (p. 298), **ReflectBC** (p. 586), and **ReflectOddBC** (p. 589).

7.27.3.3 virtual void BoxGhostBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_domain. (Deprecated function – use **ProblemDomain** instead of a **Box** to represent the domain)

Reimplemented in **BoxEdgeBC** (p. 131), **NoOpBC** (p. 425), **ExtrapBC** (p. 253), **HOExtrapBC** (p. 297), **ReflectBC** (p. 586), and **ReflectOddBC** (p. 589).

7.27.3.4 virtual void BoxGhostBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented in **BoxEdgeBC** (p. 131), **NoOpBC** (p. 426), **ExtrapBC** (p. 254), **HOExtrapBC** (p. 298), **ReflectBC** (p. 586), and **ReflectOddBC** (p. 589).

7.27.3.5 virtual void BoxGhostBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const [protected, virtual]

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_-domain. (Deprecated function – use **ProblemDomain** instead of a **Box** to represent the domain)

Reimplemented in **BoxEdgeBC** (p. 131), **NoOpBC** (p. 425), **ExtrapBC** (p. 254), **HOExtrapBC** (p. 298), **ReflectBC** (p. 587), and **ReflectOddBC** (p. 590).

7.27.3.6 void BoxGhostBC::define (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*) [protected]

Full define function

Reimplemented in **BoxEdgeBC** (p. 132), and **NoOpBC** (p. 426).

7.27.3.7 void BoxGhostBC::define (int *a_dir*, Side::LoHiSide *a_sd*) [protected]

Sets interval to be (0,0)

Reimplemented in **BoxEdgeBC** (p. 132), and **NoOpBC** (p. 426).

7.27.3.8 virtual void BoxGhostBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const [protected, pure virtual]

Pure-virtual Worker function to fill arrays which define the boundary condition. Essentially, if the boundary condition is represented as A*phi + B*dphi/dn = C, then a_dircfac is A, a_neumfac is B, and a_inhmval is C.

Implemented in **InflowVelBC** (p. 305), **BoxEdgeBC** (p. 132), **edgeDirichletBC** (p. 248), **NoOpBC** (p. 426), **ExtrapBC** (p. 254), **HOExtrapBC** (p. 298), **ReflectBC** (p. 587), **ReflectOddBC** (p. 590), and **NeumannBC** (p. 422).

7.27.3.9 virtual void BoxGhostBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const [protected, pure virtual]

Deprecated function – use ProplemDomain version instead

Implemented in **InflowVelBC** (p. 305), **BoxEdgeBC** (p. 132), **edgeDirichletBC** (p. 248), **NoOpBC** (p. 426), **ExtrapBC** (p. 254), **HOExtrapBC** (p. 298), **ReflectBC** (p. 587), **ReflectOddBC** (p. 590), **NormalInflowVelBC** (p. 430), and **NeumannBC** (p. 422).

7.27.3.10 virtual BoxGhostBC* BoxGhostBC::new_boxghostbc () const [protected, pure virtual]

Virtual constructor workaround.

Implemented in **InflowVelBC** (p. 306), **BoxEdgeBC** (p. 132), **edgeDirichletBC** (p. 249), **NoOpBC** (p. 426), **ExtrapBC** (p. 254), **HOExtrapBC** (p. 298), **ReflectBC** (p. 587), **ReflectOddBC** (p. 590), **NormalInflowVelBC** (p. 430), and **NeumannBC** (p. 423).

7.27.4 Friends And Related Function Documentation

7.27.4.1 friend class DomainGhostBC [friend]

Reimplemented in **InflowVelBC** (p. 306), **edgeDirichletBC** (p. 249), **ExtrapBC** (p. 255), **HOExtrapBC** (p. 298), **ReflectBC** (p. 587), **ReflectOddBC** (p. 590), **scalarDirichletBC** (p. 592), **scalarInflowBC** (p. 594), **InhomoDirichletBC** (p. 308), **scalarInflowChannelBC** (p. 596), **NormalInflowVelBC** (p. 430), **NeumannBC** (p. 423), and **DirichletBC** (p. 224).

7.27.5 Member Data Documentation

7.27.5.1 Interval BoxGhostBC::m_components [protected]

Which components this boundary condition should be applied to

7.27.5.2 int BoxGhostBC::m_direction [protected]

Which direction this boundary condition is for

7.27.5.3 Side::LoHiSide BoxGhostBC::m_side [protected]

Which side this boundary condition is on

The documentation for this class was generated from the following file:

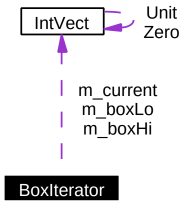
- **GhostBC.H**

7.28 BoxIterator Class Reference

iterates through the IntVects of a **Box**

`#include <BoxIterator.H>`

Collaboration diagram for BoxIterator:



Public Member Functions

- **BoxIterator** ()
- **BoxIterator** (const **Box** &bx)
- void **setBox** (const **Box** &bx)
- void **define** (const **Box** &bx)
- **BoxIterator** (const **BoxIterator** &tx)
- ~**BoxIterator** ()
- void **begin** ()
- void **reset** ()
- void **operator++** ()
- void **next** ()
- const **IntVect** & **operator()** () const
- bool **ok** ()

Protected Attributes

- **IntVect m_current**
- **IntVect m_boxLo**
- **IntVect m_boxHi**

7.28.1 Detailed Description

iterates through the IntVects of a **Box**

BoxIterator iterates through the IntVects of a box. The actual sqeuence of IntVects is implementation-specific. Typical usage:

Box b; ... BoxIterator bit (b); for (bit.begin(); bit.ok(); ++bit) { **IntVect** iv = bit(); (do operations involving iv) }

7.28.2 Constructor & Destructor Documentation

7.28.2.1 BoxIterator::BoxIterator ()

Default constructor. This constructs an invalid iterator. The user must call define before using.

7.28.2.2 BoxIterator::BoxIterator (const Box & *bx*)

Constructs a BoxIterator and associates it with a **Box**. Arguments: bx (not modified) the **Box** to iterate over.

7.28.2.3 BoxIterator::BoxIterator (const BoxIterator & *tx*)

Copy constructor. Arguments: tx (not modified) the BoxIterator to copy.

7.28.2.4 BoxIterator::~BoxIterator ()

7.28.3 Member Function Documentation

7.28.3.1 void BoxIterator::begin ()

Sets this BoxIterator to the first **IntVect** in its **Box**. The definition of the "first" **IntVect** is implementation-dependent.

7.28.3.2 void BoxIterator::define (const Box & *bx*)

Associates a **Box** with this BoxIterator. Arguments: bx (not modified) the **Box** to iterate over.

7.28.3.3 void BoxIterator::next ()

7.28.3.4 bool BoxIterator::ok ()

Returns true if this BoxIterator's location is within its **Box**.

7.28.3.5 const IntVect& BoxIterator::operator() () const

Returns the value of the InVect for the current location of this BoxIterator.

7.28.3.6 void BoxIterator::operator++ ()

Modifies this BoxIterator to set it to the next location in its **Box**. The definition of the "next location" of a **Box** is implementation-dependent.

7.28.3.7 void BoxIterator::reset ()

Sets this BoxIterator to the first **IntVect** in its **Box**. The definition of the "first" **IntVect** is implementation-dependent.

7.28.3.8 void BoxIterator::setBox (const Box & *bx*)

7.28.4 Member Data Documentation

7.28.4.1 IntVect BoxIterator::m_boxHi [protected]

7.28.4.2 IntVect BoxIterator::m_boxLo [protected]

7.28.4.3 IntVect BoxIterator::m_current [protected]

The documentation for this class was generated from the following file:

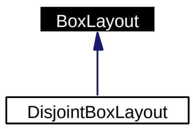
- **BoxIterator.H**

7.29 BoxLayout Class Reference

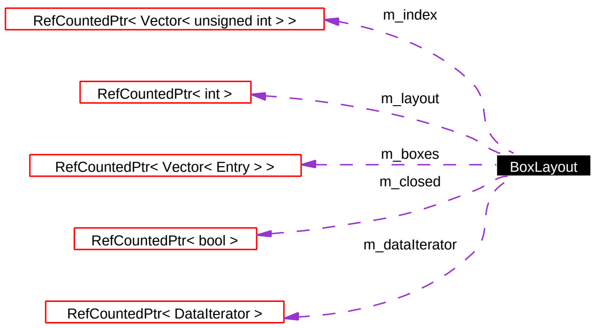
A not-necessarily-disjoint collective of boxes.

`#include <BoxLayout.H>`

Inheritance diagram for BoxLayout:



Collaboration diagram for BoxLayout:



Modification functions

- **DataIndex addBox** (const **Box** &box, int procID)
- virtual void **close** ()
- virtual void **deepCopy** (const **BoxLayout** &a_source)
- bool **coarsenable** (int refRatio) const
- void **setProcID** (const **LayoutIndex** &a_index, unsigned int a_procID)
- void **sort** ()
- void **aliasAddBox** (const **Box** &box)
- void **aliasClose** ()
- void **coarsen** (**BoxLayout** &output, const **BoxLayout** &input, int refinement)
- void **refine** (**BoxLayout** &output, const **BoxLayout** &input, int refinement)

Public Member Functions

Constructors, destructors, assignments, defines

- **BoxLayout** ()
- **BoxLayout** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs)
- virtual ~**BoxLayout** ()
- **BoxLayout** & **operator=** (const **BoxLayout** &a_rhs)
- virtual void **define** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs)

Accessors

- const **Box** & **operator**[ ] (const **LayoutIndex** &it) const
- **Box** & **operator**[ ] (const **LayoutIndex** &it)
- **Box get** (const **LayoutIndex** &it) const
- **Box get** (const **LayoutIterator** &it) const
- **Box get** (const **DataIterator** &it) const
- unsigned int **procID** (const **LayoutIndex** &a_index) const
- int **numBoxes** (const int procID) const
- unsigned int **size** () const
- unsigned int **index** (const **LayoutIndex** &index) const

Checks

- bool **operator==** (const **BoxLayout** &rhs) const
- bool **eq** (const **BoxLayout** &rhs) const
- bool **isClosed** () const
- bool **isSorted** () const
- bool **check** (const **LayoutIndex** &index) const

Iterators

- **DataIterator dataIterator** () const
  Parallel iterator.
- **LayoutIterator layoutIterator** () const
  Iterator that processes through ALL the boxes in a BoxLayout.

I/O functions

- void **print** () const
- void **p** () const

Protected Member Functions

- void **setIndexVector** ()

Protected Attributes

- **RefCountedPtr**< **Vector**< **Entry** > > **m_boxes**
- **RefCountedPtr**< **Vector**< unsigned int > > **m_index**
- **RefCountedPtr**< int > **m_layout**
- **RefCountedPtr**< bool > **m_closed**
- **RefCountedPtr**< **DataIterator** > **m_dataIterator**

Friends

- class **LayoutIterator**
- class **DataIterator**

7.29.1 Detailed Description

A not-necessarily-disjoint collective of boxes.

A BoxLayout is a collection of **Box** objects that are assigned to process numbers. Each box is associated with only one process. Processes are numbered from 0 to n-1 (for a job with n processes).

A BoxLayout can be either open or closed.

Open BoxLayout:

- Created by null construction or deepCopy.
- Boxes may be added to it.
- Non-const operations may be performed on the boxes in it.

Closed BoxLayout:

- Created by constructor with vectors of Boxes and processors given explicitly.
- Cannot be modified.
- Represented as sorted boxes.
- Many uses of BoxLayouts require a closed BoxLayout.

Ref-counting

BoxLayout is an explicitly ref-counted object.

Assignment and copy are compiler-generated. They increment the refcount on the contained data members. They perform shallow, ref-counted operations.

Refcounting is a process whereby multiple instantiations make use of a single implementation of that object and keep a tally of how many instantiations are sharing. Thus:

```
BoxLayout b1(boxes, procIDs);

          b1  ----$>$ refcount = 1
              ----$>$  m_boxes
              ----$>$  m_processors

BoxLayout b2(b1)

          b1  ----$>$ refcount = 2  $<$---- b2
              ----$>$  m_boxes      $<$----
              ----$>$  m_processors $<$----

BoxLayout b3;
b3 = b2;

          b1  ----$>$ refcount = 3  $<$---- b2
              ----$>$  m_boxes      $<$----
              ----$>$  m_processors $<$----
                      ^^^
                      |||
                       b3
```

7.29.2 Constructor & Destructor Documentation

7.29.2.1 BoxLayout::BoxLayout ()

Construct BoxLayout with no boxes.

7.29.2.2 BoxLayout::BoxLayout (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*)

Construct from a **Vector** of Boxes and a **Vector** of processor assignments. On exit, the BoxLayout will be closed.

7.29.2.3 virtual BoxLayout::~BoxLayout () `[virtual]`

Ref-counted destruction. Once the last reference to the implementation of this class is destroyed, the data members are cleaned up

7.29.3 Member Function Documentation

7.29.3.1 DataIndex BoxLayout::addBox (const Box & *box*, int *procID*)

Add a box to this BoxLayout. Takes the processor assignment. The input *box* is copied. BoxLayout must be "open" for this operation, or else a runtime error will occur.

7.29.3.2 void BoxLayout::aliasAddBox (const Box & *box*)

Not a user function, used in Chombo interface to AMRData.H interoperability tools.

7.29.3.3 void BoxLayout::aliasClose ()

Not a user function, used in Chombo interface to AMRData.H interoperability tools.

7.29.3.4 bool BoxLayout::check (const LayoutIndex & *index*) const `[inline]`

not a user function

7.29.3.5 virtual void BoxLayout::close () `[virtual]`

Mark this BoxLayout as complete and unchangeable.

Reimplemented in **DisjointBoxLayout** (p. 230).

7.29.3.6 bool BoxLayout::coarsenable (int *refRatio*) const

returns true iff:

- every **Box** in the BoxLayout can be coarsened by refRatio and return back to the original **Box** when refined by refRatio.

- refRatio must be a positive non-zero integer.

7.29.3.7 DataIterator BoxLayout::dataIterator () const

Parallel iterator.

Parallel iterator.

Returns **DataIndex** object that corresponds to boxes on the local processor only.

7.29.3.8 virtual void BoxLayout::deepCopy (const BoxLayout & *a_source*) `[virtual]`

Actual deep copy operation. New object created with copied data. This object disassociates itself with original implementation safely. This object now is considered 'open' and can be non-const modified. There is no assurance that the order in which this BoxLayout is indexed corresponds to the indexing of *a_source*.

```
BoxLayout b1(boxes, procIDs);


     b1  ----$>$ refcount = 1
         ----$>$  m_boxes
         ----$>$  m_processors


BoxLayout b2(b1)


     b1  ----$>$ refcount = 2  $<$---- b2
         ----$>$  m_boxes      $<$----
         ----$>$  m_processors $<$----


BoxLayout b3;
b3.deepCopy(b2);
     b1  ----$>$ refcount = 2  $<$---- b2  b3 ----$>$ refcount = 1
         ----$>$  m_boxes      $<$----        ----$>$  m_boxes
         ----$>$  m_processors $<$----        ----$>$  m_processors
```

Reimplemented in **DisjointBoxLayout** (p. 230).

7.29.3.9 virtual void BoxLayout::define (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*) `[virtual]`

Define this BoxLayout from a **Vector** of Boxes and a **Vector** of processor assignments. Any pre-existing layout will be lost (ref-count dropped by one). The processor assignment **Vector** must be the same length as the **Box Vector**. On exit, the BoxLayout will be closed.

Reimplemented in **DisjointBoxLayout** (p. 231).

7.29.3.10 bool BoxLayout::eq (const BoxLayout & *rhs*) const `[inline]`

Refcounted pointer check. Return `true` if these two objects share the same implementation.

7.29.3.11 Box BoxLayout::get (const DataIterator & *it*) const

Get box indexed by *it*.

7.29.3.12 Box BoxLayout::get (const LayoutIterator & *it*) const

Get box indexed by *it*.

7.29.3.13 Box BoxLayout::get (const LayoutIndex & *it*) const `[inline]`

Get box indexed by *it*.

As a consequence of the C++ compiler being free to choose which version of operator[] when the object is technically non-const, we very often get 'BoxLayout closed' errors. This is a non-overloaded get method.

7.29.3.14 unsigned int BoxLayout::index (const LayoutIndex & *index*) const `[inline]`

Not a user function. Used in I/O routine.

7.29.3.15 bool BoxLayout::isClosed () const `[inline]`

Return `true` if **close()** has been called. Closed BoxLayout is always sorted.

7.29.3.16 bool BoxLayout::isSorted () const `[inline]`

Return `true` if **sort()** has been called.

7.29.3.17 LayoutIterator BoxLayout::layoutIterator () const

Iterator that processes through ALL the boxes in a BoxLayout.

Iterator that processes through ALL the boxes in a BoxLayout.

If BoxLayout is closed, then **LayoutIterator** will return the Boxes in sorted order.

7.29.3.18 int BoxLayout::numBoxes (const int *procID*) const

Returns the number of boxes assigned to a given procID.

7.29.3.19 BoxLayout& BoxLayout::operator= (const BoxLayout & *a_rhs*)

Ref-counted assignment.

7.29.3.20 bool BoxLayout::operator== (const BoxLayout & *rhs*) const `[inline]`

Refcounted pointer check. Return `true` if these two objects share the same implementation.

7.29.3.21 Box & BoxLayout::operator[] (const LayoutIndex & *it*) [inline]

accessor operator. See also get(const DataIndex&).

7.29.3.22 const Box & BoxLayout::operator[] (const LayoutIndex & *it*) const [inline]

const accessor operator. See also get(const DataIndex&).

7.29.3.23 void BoxLayout::p () const [inline]

Invokes cout<<*this; pretty-print dump of BoxLayout.

7.29.3.24 void BoxLayout::print () const

Invokes cout<<*this; pretty-print dump of BoxLayout.

7.29.3.25 unsigned int BoxLayout::procID (const LayoutIndex & *a_index*) const [inline]

Returns the processor to which this box has been assigned. Not a user function, at least, not a new user function. It can be used safely at anytime, closed or open. A person needing this level of knowledge of the processor assignment should have non-trivial needs, like writing your own load balancer or such. Most user-level parallel coding needs can be addressed using the **DataIterator** class.

7.29.3.26 void BoxLayout::setIndexVector () [protected]

7.29.3.27 void BoxLayout::setProcID (const LayoutIndex & *a_index*, unsigned int *a_procID*) [inline]

Assign a **Box** in the BoxLayout to a processor. Requires the BoxLayout to be open.

7.29.3.28 unsigned int BoxLayout::size () const [inline]

Returns the total number of boxes in the BoxLayout.

7.29.3.29 void BoxLayout::sort ()

Sort the boxes of an open BoxLayout. No change if the BoxLayout is closed.

7.29.4 Friends And Related Function Documentation

7.29.4.1 void coarsen (BoxLayout & *output*, const BoxLayout & *input*, int *refinement*) [friend]

Coarsen a BoxLayout:

- *output* must be open
- *input* must be closed
- *refinement* must be a positive non-zero integer
- *output* and *input* do not share an implementation.

output is first deepCopy'ed from *input*, then coarsen(refinement) is called on each box of *output*.

output returns from this function closed.

LayoutIterators from *input* and *output* can perform interchangeably.

7.29.4.2 friend class DataIterator [friend]

7.29.4.3 friend class LayoutIterator [friend]

7.29.4.4 void refine (BoxLayout & *output*, const BoxLayout & *input*, int *refinement*) [friend]

Refine a BoxLayout:

- *output* must be open
- *input* must be closed
- *refinement* must be a positive non-zero integer
- *output* and *input* do not share an implementation.

output is first deepCopy'ed from *input*, then refine(refinement) is called on each box of *output*.

output returns from this function closed.

LayoutIterators from *input* and *output* can perform interchangeably.

7.29.5 Member Data Documentation

7.29.5.1 RefCountedPtr<Vector<Entry> > BoxLayout::m_boxes [protected]

7.29.5.2 RefCountedPtr<bool> BoxLayout::m_closed [protected]

7.29.5.3 RefCountedPtr<DataIterator> BoxLayout::m_dataIterator [protected]

7.29.5.4 RefCountedPtr<Vector<unsigned int> > BoxLayout::m_index [protected]

7.29.5.5 RefCountedPtr<int> BoxLayout::m_layout [protected]

The documentation for this class was generated from the following file:

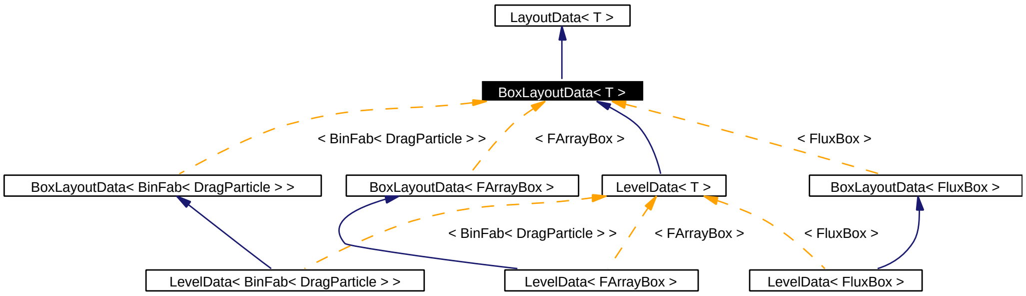
- **BoxLayout.H**

7.30 BoxLayoutData< T > Class Template Reference

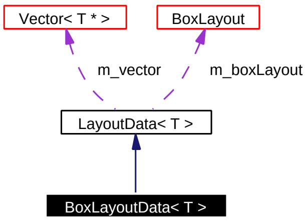
Data on a **BoxLayout**.

`#include <BoxLayoutData.H>`

Inheritance diagram for BoxLayoutData< T >:



Collaboration diagram for BoxLayoutData< T >:



Public Member Functions

- **BoxLayoutData** ()
- virtual ~**BoxLayoutData** ()
- **BoxLayoutData** (const **BoxLayout** &boxes, int comps, const **DataFactory**< T > &factory=**DefaultDataFactory**< T >())
- virtual void **define** (const **BoxLayout** &boxes, int comps, const **DataFactory**< T > &factory=**DefaultDataFactory**< T >())
- virtual void **define** (const **BoxLayoutData**< T > &da, const **DataFactory**< T > &factory=**DefaultDataFactory**< T >())
- virtual void **define** (const **BoxLayoutData**< T > &da, const **Interval** &comps, const **DataFactory**< T > &factory=**DefaultDataFactory**< T >())
- virtual void **define** (const **BoxLayout** &boxes)

  overridden and throws an error.

- int **nComp** () const
- **Interval interval** () const
- virtual void **apply** (void(∗a_Function)(const **Box** &box, int comps, T &t))
- virtual bool **isDefined** () const

Protected Member Functions

- void **setVector** (const **BoxLayoutData**< T > &da, const **Interval** &srcComps, const **Interval** &destComps)
- void **allocateGhostVector** (const **DataFactory**< T > &factory, const **IntVect** &ghost=IntVect::TheZeroVector())

Protected Attributes

- int **m_comps**
- bool **m_isdefined**

Friends

- class **LevelData**< **T** >

7.30.1 Detailed Description

template<class T> class BoxLayoutData< T >

Data on a **BoxLayout**.

A collection of Box-oriented objects. The arrangement of Boxes is given by the underlying **BoxLayout** object. BoxLayoutData attempt to prevent users from maniupulating the template class T to become out of synch with the boxes in the **BoxLayout**. Caveat emptor.

All access to the data in a BoxLayoutData is forced to be data-parallel, by use of the **DataIterator**.

class T must provide the following methods:

- `T()`
- `T(const Box& box, int comps)`
- `define(const Box& box, int comps)`
- ```
  void copy(const Box& R, const Interval& Cd, const T& source, const Interval Cs)
        Cs and Cd must have same length, but can be displaced
  ```
- ```
  static int preAllocatable()
      returns 0 if the size(...) function is strictly a function of  Box and
         Interval, and does not depend on the current state of the T object.
      return 1 if size(..) is symmetric, in that sender and receiver T object
         can size their message buffers, but a static object cannot.
      returns 2 if the object is truly dynamic.  the message size is subject to
         unique object data.
  ```
- `int size(const Box& R, const Interval& comps)`
- `void linearOut(void* buf, const Box& R, const Interval& comps)`
- `void linearIn(void* buf, const Box& R, const Interval& comps)`

In sequential codes, many of these functions will not be called, but in the general parallel case, they all will be.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 template<class T> BoxLayoutData< T >::BoxLayoutData () [inline]

7.30.2.2 template<class T> virtual BoxLayoutData< T >::~BoxLayoutData () [inline, virtual]

7.30.2.3 template<class T> BoxLayoutData< T >::BoxLayoutData (const BoxLayout & *boxes*, int *comps*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) [inline]

7.30.3 Member Function Documentation

7.30.3.1 template<class T> void BoxLayoutData< T >::allocateGhostVector (const DataFactory< T > & *factory*, const IntVect & *ghost* = IntVect::TheZeroVector()) [inline, protected]

7.30.3.2 template<class T> void BoxLayoutData< T >::apply (void(* *a_Function*)(const Box &box, int comps, T &t)) [inline, virtual]

Reimplemented in **LevelData< T >** (p. 357), **LevelData< BinFab< DragParticle > >** (p. 357), **LevelData< FluxBox >** (p. 357), and **LevelData< FArrayBox >** (p. 357).

7.30.3.3 template<class T> void BoxLayoutData< T >::define (const BoxLayout & *boxes*) [inline, virtual]

overridden and throws an error.

Reimplemented from **LayoutData< T >** (p. 346).

Reimplemented in **LevelData< T >** (p. 358), **LevelData< BinFab< DragParticle > >** (p. 358), **LevelData< FluxBox >** (p. 358), and **LevelData< FArrayBox >** (p. 358).

7.30.3.4 template<class T> void BoxLayoutData< T >::define (const BoxLayoutData< T > & *da*, const Interval & *comps*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) [inline, virtual]

Define this BoxLayoutData to be the subset of the data in da defined by the **Interval** comps.

Reimplemented in **LevelData< T >** (p. 358), **LevelData< BinFab< DragParticle > >** (p. 358), **LevelData< FluxBox >** (p. 358), and **LevelData< FArrayBox >** (p. 358).

7.30.3.5 template<class T> void BoxLayoutData< T >::define (const BoxLayoutData< T > & *da*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) [inline, virtual]

Reimplemented in **LevelData< T >** (p. 359), **LevelData< BinFab< DragParticle > >** (p. 359), **LevelData< FluxBox >** (p. 359), and **LevelData< FArrayBox >** (p. 359).

7.30.3.6 template<class T> void BoxLayoutData< T >::define (const BoxLayout & *boxes*, int *comps*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) `[inline, virtual]`

Reimplemented in **LevelData< T >** (p. 359), **LevelData< BinFab< DragParticle > >** (p. 359), **LevelData< FluxBox >** (p. 359), and **LevelData< FArrayBox >** (p. 359).

7.30.3.7 template<class T> Interval BoxLayoutData< T >::interval () const `[inline]`

7.30.3.8 template<class T> bool BoxLayoutData< T >::isDefined () const `[inline, virtual]`

7.30.3.9 template<class T> int BoxLayoutData< T >::nComp () const `[inline]`

7.30.3.10 template<class T> void BoxLayoutData< T >::setVector (const BoxLayoutData< T > & *da*, const Interval & *srcComps*, const Interval & *destComps*) `[inline, protected]`

7.30.4 Friends And Related Function Documentation

7.30.4.1 template<class T> friend class LevelData< T > `[friend]`

7.30.5 Member Data Documentation

7.30.5.1 template<class T> int BoxLayoutData< T >::m_comps `[protected]`

7.30.5.2 template<class T> bool BoxLayoutData< T >::m_isdefined `[protected]`

The documentation for this class was generated from the following file:

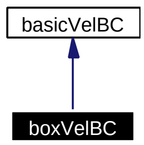
- **BoxLayoutData.H**

7.31 boxVelBC Class Reference

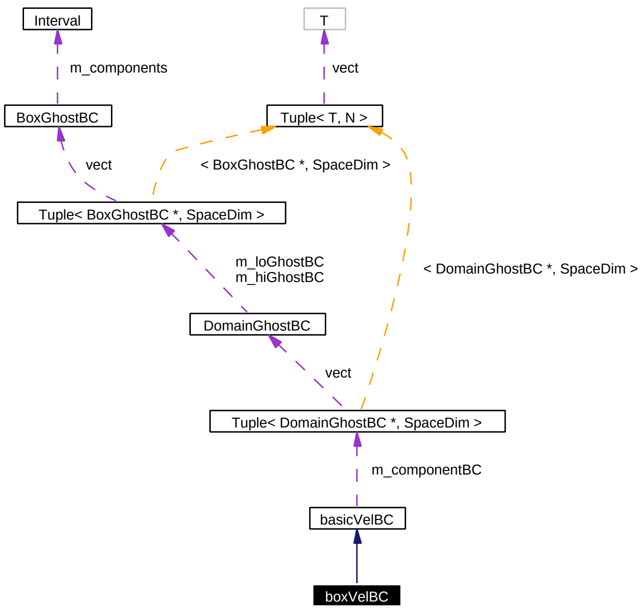
implements cell-centered velocity BC's for flow in a box!

`#include <boxVelBC.H>`

Inheritance diagram for boxVelBC:



Collaboration diagram for boxVelBC:



Public Member Functions

- **boxVelBC** (bool isViscous=false)
- virtual ~**boxVelBC** ()
- virtual void **setAllBCTypes** ()
- bool **isViscous** ()

  access function

- void **isViscous** (bool a_isViscous)

Protected Attributes

- bool **m_isViscous**

7.31.1 Detailed Description

implements cell-centered velocity BC's for flow in a box!

7.31.2 Constructor & Destructor Documentation

7.31.2.1 boxVelBC::boxVelBC (bool *isViscous* = false)

7.31.2.2 virtual boxVelBC::~boxVelBC () `[inline, virtual]`

7.31.3 Member Function Documentation

7.31.3.1 void boxVelBC::isViscous (bool *a_isViscous*)

7.31.3.2 bool boxVelBC::isViscous ()

access function

7.31.3.3 virtual void boxVelBC::setAllBCTypes () `[virtual]`

7.31.4 Member Data Documentation

7.31.4.1 bool boxVelBC::m_isViscous `[protected]`

The documentation for this class was generated from the following file:

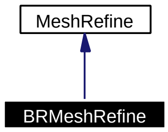
- **boxVelBC.H**

7.32 BRMeshRefine Class Reference

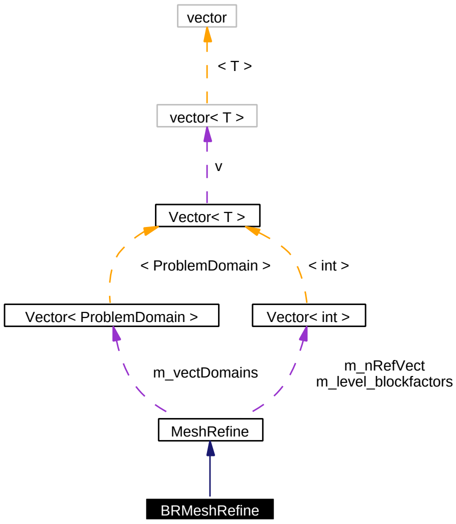
Class which manages Berger-Rigoutsos grid generation.

`#include <BRMeshRefine.H>`

Inheritance diagram for BRMeshRefine:



Collaboration diagram for BRMeshRefine:



Public Member Functions

- **BRMeshRefine** ()

  Default constructor – leaves object in an unusable state.

- **BRMeshRefine** (const **Box** &a_baseDomain, const **Vector**< int > &a_refRatios, const **Real** a_fillRatio, const int a_blockFactor, const int a_bufferSize, const int a_maxSize)

  Full constructor – leaves object in usable state.

- **BRMeshRefine** (const **ProblemDomain** &a_baseDomain, const **Vector**< int > &a_refRatios, const **Real** a_fillRatio, const int a_blockFactor, const int a_bufferSize, const int a_maxSize)

  Full constructor – leaves object in usable state.

- virtual ~**BRMeshRefine** ()

  Destructor.

- void **define** (const **Box** &a_baseDomain, const **Vector**< int > &a_refRatios, const **Real** a_fillRatio, const int a_blockFactor, const int a_bufferSize, const int a_maxSize)

  Define function – size of RefRatios will define maximum number of levels.

- void **define** (const **ProblemDomain** &a_baseDomain, const **Vector**< int > &a_refRatios, const **Real** a_fillRatio, const int a_blockFactor, const int a_bufferSize, const int a_maxSize)

  Define function – size of RefRatios will define maximum number of levels.

- void **makeBoxes** (**Vector**< **Box** > &a_mesh, const **IntVectSet** &a_tags, const **IntVectSet** &a_pnd, const **ProblemDomain** &a_domain, const int a_maxSize) const

  Constructs a set of boxes which covers a set of tagged cells.

Protected Member Functions

- void **makeBoxes** (std::list< **Box** > &a_mesh, const **IntVectSet** &a_tags, const **IntVectSet** &a_pnd, const **ProblemDomain** &a_domain, const int a_maxSize, const int a_depth) const
- void **splitBox** (std::list< **Box** > &a_boxes, const std::list< **Box** >::iterator &a_boxindex, const int a_dimension,const int a_maxboxsize) const
- **Vector**< int > **makeTrace** (const **IntVectSet** &a_Ivs,int a_dir) const
- void **makeTraces** (const **IntVectSet** &a_Ivs,**Vector**< int > *a_traces) const
- int **findSplit** (const **Vector**< int > &a_trace) const
- int **findMaxInflectionPoint** (const **Vector**< int > &a_trace,int &a_maxVal) const
- void **splitTags** (const **IntVectSet** &a_tags, const int a_split_dir,const int a_split_indx, **IntVectSet** &a_tags_lo,**IntVectSet** &a_tags_hi) const
- void **breakBoxes** (**Vector**< **Box** > &a_vboxin, const int &a_maxSize, const int &a_idir) const
- int **maxloc** (const int *a_V,const int a_Size) const
- void **makeBoxesParallel** (std::list< **Box** > &a_mesh, const **IntVectSet** &a_tags, const **IntVectSet** &a_pnd, const **ProblemDomain** &a_domain, const int a_maxSize, const int a_depth, const int a_minSize, const **Interval** &a_procInterval) const
- void **sendBoxesParallel** (const std::list< **Box** > &a_mesh, int tag) const
- void **receiveBoxesParallel** (const **Interval** &a_from, const **Interval** &a_to, std::list< **Box** > &a_mesh, int tag) const

7.32.1 Detailed Description

Class which manages Berger-Rigoutsos grid generation.

This class manages grid generation from sets of tagged cells using the Berger-Rigoutsos algorithm in the context of the **MeshRefine** class from which it is derived

There are two ways grids can be defined based on tagged cells. one takes a single **IntVectSet** of tags defined on the BaseLevel mesh and uses that set of tags for every level to be refined; the other takes a Vector<IntVectSet> of tags defined on all the mesh levels to be refined and uses those.

Long Description:

Create new meshes based on tagged cells on a range of levels of a mesh hierarchy. Each level of tagged cells is used to generate a new mesh at the next finer level. The finest level in the output mesh will be one level higher than the top of the range of levels given as input. As a special case, use the same tags (appropriately refined) for all levels.

Usage:

Call the regrid functions after computing error estimates and tagging cells. To add a new mesh level, set TopLevel to the index of the finest level in the existing mesh and define tags on the finest level. To keep the existing number of mesh levels, set TopLevel to one less than the index of the finest level and don't define any tags on the finest level. If a single **IntVectSet** of tags is passed (instead of a Vector<IntVectSet>) then the same tags (properly refined) will be used for all the new meshes up to level TopLevel+1. In any case, the meshes at levels BaseLevel and below are not modified. The output argument newmeshes will be reallocated to the necessary size before being used. When this function returns, the elements of the newmeshes vector corresponding to the unchanged levels will be filled in with copies of the levels from the old mesh vector. The variable tags is modified in an undefined way, so its contents should not be relied upon. The variable BlockFactor specifies the amount by which each box will be coarsenable. Every grid box will have an integral multiple of BlockFactor cells in each dimension and also lower index values that are integral multiples. As a side effect, the minimum box size will be BlockFactor.

Expensive validations are done only when debugging is enabled (i.e. the DEBUG make variable is "TRUE").

Usage Notes:

All the input vectors should be defined with max index >= TopLevel. They should have values for indices [BaseLevel:TopLevel]. (except for OldMeshes, which must be defined for all indices). The new mesh vector newmeshes will be redefined up to index TopLevel+1. RefRatios should be defined such that RefRatios[L] is the value to use to refine the level L mesh to produce the level L+1 mesh. The tags vector is modified in an undefined manner. The output variable newmeshes may not be completely defined if an exception occurs. The BlockFactor can be used to force a minimum box size.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 BRMeshRefine::BRMeshRefine ()

Default constructor – leaves object in an unusable state.

7.32.2.2 BRMeshRefine::BRMeshRefine (const Box & *a_baseDomain*, const Vector< int > & *a_refRatios*, const Real *a_fillRatio*, const int *a_blockFactor*, const int *a_bufferSize*, const int *a_maxSize*)

Full constructor – leaves object in usable state.

Parameters:

a_baseDomain Level 0 domain

a_refRatios Refinement ratios – refRatio[0] is btwn levels 0 and 1

a_fillRatio Measure of how efficiently tagged cells will be covered

a_blockFactor Amount by which grids are guaranteed to be coarsenable

a_bufferSize Proper nesting buffer amount

a_maxSize Maximum grid length in any direction – 0 means no limit.

7.32.2.3 BRMeshRefine::BRMeshRefine (const ProblemDomain & *a_baseDomain*, const Vector< int > & *a_refRatios*, const Real *a_fillRatio*, const int *a_blockFactor*, const int *a_bufferSize*, const int *a_maxSize*)

Full constructor – leaves object in usable state.

Parameters:

a_baseDomain Level 0 domain

a_refRatios Refinement ratios – refRatio[0] is btwn levels 0 and 1

a_fillRatio Measure of how efficiently tagged cells will be covered

a_blockFactor Amount by which grids are guaranteed to be coarsenable

a_bufferSize Proper nesting buffer amount

a_maxSize Maximum grid length in any direction – 0 means no limit.

7.32.2.4 virtual BRMeshRefine::~BRMeshRefine () `[virtual]`

Destructor.

7.32.3 Member Function Documentation

7.32.3.1 void BRMeshRefine::breakBoxes (Vector< Box > & *a_vboxin*, const int & *a_maxSize*, const int & *a_idir*) const `[protected]`

7.32.3.2 void BRMeshRefine::define (const ProblemDomain & *a_baseDomain*, const Vector< int > & *a_refRatios*, const Real *a_fillRatio*, const int *a_blockFactor*, const int *a_bufferSize*, const int *a_maxSize*)

Define function – size of RefRatios will define maximum number of levels.

Parameters:

a_baseDomain :evel 0 domain

a_refRatios Refinement ratios – refRatio[0] is btwn levels 0 and 1

a_fillRatio Measure of how efficiently tagged cells will be covered

a_blockFactor Amount by which grids are guaranteed to be coarsenable

a_bufferSize Proper nesting buffer amount

a_maxSize Maximum grid length in any direction – 0 means no limit

Reimplemented from **MeshRefine** (p. 411).

7.32.3.3 void BRMeshRefine::define (const Box & *a_baseDomain*, const Vector< int > & *a_refRatios*, const Real *a_fillRatio*, const int *a_blockFactor*, const int *a_bufferSize*, const int *a_maxSize*)

Define function – size of RefRatios will define maximum number of levels.

Parameters:

a_baseDomain Level 0 domain

a_refRatios Refinement ratios – refRatio[0] is btwn levels 0 and 1

a_fillRatio Measure of how efficiently tagged cells will be covered

a_blockFactor Amount by which grids are guaranteed to be coarsenable

a_bufferSize Proper nesting buffer amount

a_maxSize Maximum grid length in any direction – 0 means no limit

Reimplemented from **MeshRefine** (p. 412).

7.32.3.4 int BRMeshRefine::findMaxInflectionPoint (const Vector< int > & *a_trace*, int & *a_maxVal*) const [protected]

7.32.3.5 int BRMeshRefine::findSplit (const Vector< int > & *a_trace*) const [protected]

7.32.3.6 void BRMeshRefine::makeBoxes (std::list< Box > & *a_mesh*, const IntVectSet & *a_tags*, const IntVectSet & *a_pnd*, const ProblemDomain & *a_domain*, const int *a_maxSize*, const int *a_depth*) const [protected]

Function which actually implement Berger-Rigoutsos chopping.

Parameters:

a_mesh Output: refined boxes at each new level

a_tags Input: set of tagged cells to cover

a_pnd Input: proper nesting domain in which mesh boxes must live

a_domain Input: physical domain

a_maxSize Input: largest number of cells in any dimension for any box

a_depth Input: depth of this recursion in the algorithm

7.32.3.7 void BRMeshRefine::makeBoxes (Vector< Box > & *a_mesh*, const IntVectSet & *a_tags*, const IntVectSet & *a_pnd*, const ProblemDomain & *a_domain*, const int *a_maxSize*) const [virtual]

Constructs a set of boxes which covers a set of tagged cells.

Constructs a set of boxes which covers a set of tagged cells by using the Berger-Rigoutsos algorithm. Everything should be on the same level, and blocking factor is not applied. Boxes will be on the same refinement level as the tags. This would normally be a protected function, but it can be useful to call it on it's own, so it has been left public.

Parameters:

a_mesh Putput: refined boxes at each new level

a_tags Input: set of tagged cells to cover

a_pnd Input: proper nesting domain in which mesh boxes must live

a_domain Input: physical domain

a_maxSize Input: largest number of cells in any dimension for any box

Implements **MeshRefine** (p. 412).

7.32.3.8 void BRMeshRefine::makeBoxesParallel (std::list< Box > & *a_mesh*, const IntVectSet & *a_tags*, const IntVectSet & *a_pnd*, const ProblemDomain & *a_domain*, const int *a_maxSize*, const int *a_depth*, const int *a_minSize*, const Interval & *a_procInterval*) const `[protected]`

7.32.3.9 Vector<int> BRMeshRefine::makeTrace (const IntVectSet & *a_Ivs*, int *a_dir*) const `[protected]`

7.32.3.10 void BRMeshRefine::makeTraces (const IntVectSet & *a_Ivs*, Vector< int > * *a_traces*) const `[protected]`

7.32.3.11 int BRMeshRefine::maxloc (const int * *a_V*, const int *a_Size*) const `[protected]`

7.32.3.12 void BRMeshRefine::receiveBoxesParallel (const Interval & *a_from*, const Interval & *a_to*, std::list< Box > & *a_mesh*, int *tag*) const `[protected]`

7.32.3.13 void BRMeshRefine::sendBoxesParallel (const std::list< Box > & *a_mesh*, int *tag*) const `[protected]`

7.32.3.14 void BRMeshRefine::splitBox (std::list< Box > & *a_boxes*, const std::list< Box >::iterator & *a_boxindex*, const int *a_dimension*, const int *a_maxboxsize*) const `[protected]`

7.32.3.15 void BRMeshRefine::splitTags (const IntVectSet & *a_tags*, const int *a_split_dir*, const int *a_split_indx*, IntVectSet & *a_tags_lo*, IntVectSet & *a_tags_hi*) const `[protected]`

The documentation for this class was generated from the following file:

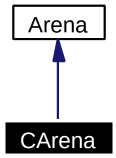
- **BRMeshRefine.H**

7.33 CArena Class Reference

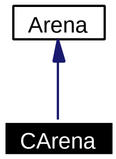
A Concrete Class for Dynamic Memory Management.

`#include <Arena.H>`

Inheritance diagram for CArena:



Collaboration diagram for CArena:



Public Types

- enum { **DefaultHunkSize** = 1024∗1024 }

  The default memory hunk size to grab from the heap.

Public Member Functions

- **CArena** (size_t hunk_size=0)
- virtual ~**CArena** ()

  The destructor.

- virtual void ∗ **alloc** (size_t nbytes)

  Allocate some memory.

- virtual void **free** (void ∗ap)
- void ∗ **calloc** (size_t nmemb, size_t size)
- void ∗ **realloc** (void ∗ptr, size_t size)

Protected Types

- typedef set< **Node** > **NL**

Protected Attributes

- std::vector< void * > **m_alloc**
- **NL m_freelist**
- **NL m_busylist**
- size_t **m_hunk**

7.33.1 Detailed Description

A Concrete Class for Dynamic Memory Management.

This is a coalescing memory manager. It allocates (possibly) large chunks of heap space and apportions it out as requested. It merges together neighboring chunks on each **free()**.

7.33.2 Member Typedef Documentation

7.33.2.1 typedef set< Node > CArena::NL `[protected]`

7.33.3 Member Enumeration Documentation

7.33.3.1 anonymous enum

The default memory hunk size to grab from the heap.

Enumeration values:
DefaultHunkSize

7.33.4 Constructor & Destructor Documentation

7.33.4.1 CArena::CArena (size_t *hunk_size* = 0)

: Construct a coalescing memory manager. 'hunk_size' is the minimum size of hunks of memory to allocate from the heap. If hunk_size == 0 we use DefaultHunkSize as specified below.

7.33.4.2 virtual CArena::~CArena () `[virtual]`

The destructor.

7.33.5 Member Function Documentation

7.33.5.1 virtual void* CArena::alloc (size_t *nbytes*) `[virtual]`

Allocate some memory.

Implements **Arena** (p. 62).

7.33.5.2 void* CArena::calloc (size_t *nmemb*, size_t *size*)

Mirror the C **calloc()** function. Returns zero'd memory of size nmemb*size. This is here so that we can implement malloc(3) and family. User's shouldn't use this function.

7.33.5.3 virtual void CArena::free (void * *ap*) [virtual]

Free up allocated memory. Merge neighboring free memory chunks into largest possible chunk.

Implements **Arena** (p. 62).

7.33.5.4 void* CArena::realloc (void * *ptr*, size_t *size*)

Mirror the C **realloc()** function. This is here so that we can implement malloc(3) and family. User's shouldn't use this function.

7.33.6 Member Data Documentation

7.33.6.1 std::vector<void*> CArena::m_alloc [protected]

7.33.6.2 NL CArena::m_busylist [protected]

7.33.6.3 NL CArena::m_freelist [protected]

7.33.6.4 size_t CArena::m_hunk [protected]

The documentation for this class was generated from the following file:

- **Arena.H**

7.34 CArena::Node Class Reference

#include <Arena.H>

Public Member Functions

- **Node** ()
- **Node** (void *block, size_t size)
- **Node** (const **Node** &rhs)
- **Node** & **operator=** (const **Node** &rhs)
- bool **operator<** (const **Node** &rhs) const
- bool **operator==** (const **Node** &rhs) const
- void * **block** () const
- void **block** (void *blk)
- size_t **size** () const
- void **size** (size_t sz)

7.34.1 Constructor & Destructor Documentation

7.34.1.1 CArena::Node::Node () `[inline]`

7.34.1.2 CArena::Node::Node (void * *block*, size_t *size*) `[inline]`

7.34.1.3 CArena::Node::Node (const Node & *rhs*) `[inline]`

7.34.2 Member Function Documentation

7.34.2.1 void CArena::Node::block (void * *blk*) `[inline]`

7.34.2.2 void* CArena::Node::block () const `[inline]`

7.34.2.3 bool CArena::Node::operator< (const Node & *rhs*) const `[inline]`

7.34.2.4 Node& CArena::Node::operator= (const Node & *rhs*) `[inline]`

7.34.2.5 bool CArena::Node::operator== (const Node & *rhs*) const `[inline]`

7.34.2.6 void CArena::Node::size (size_t *sz*) `[inline]`

7.34.2.7 size_t CArena::Node::size () const `[inline]`

The documentation for this class was generated from the following file:

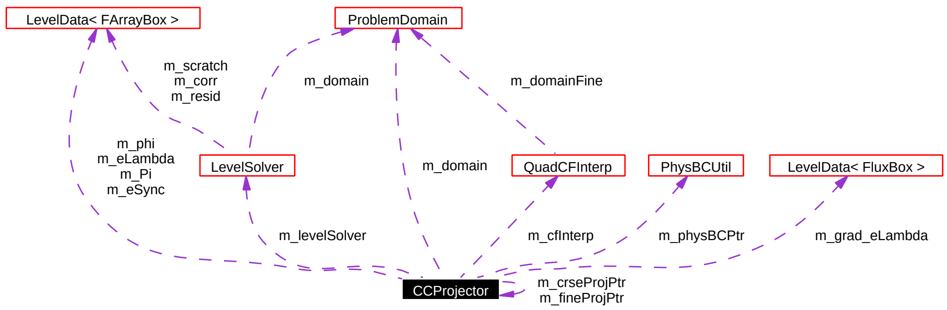
- **Arena.H**

7.35 CCProjector Class Reference

this class manages the various forms of the CC projection

`#include <CCProjector.H>`

Collaboration diagram for CCProjector:



Public Member Functions

- **CCProjector** ()
- **CCProjector** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_crseGridsPtr, const **Box** &a_domain, const **Real** a_dx, **CCProjector** ∗a_finerProj, **CCProjector** ∗a_crseProj, int a_nRefCrse, int a_level, const **PhysBCUtil** &a_physBC)

  full constructor

- **CCProjector** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_crseGridsPtr, const **ProblemDomain** &a_domain, const **Real** a_dx, **CCProjector** ∗a_finerProj, **CCProjector** ∗a_crseProj, int a_nRefCrse, int a_level, const **PhysBCUtil** &a_physBC)

  full constructor

- **~CCProjector** ()

  destructor

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_crseGridsPtr, const **Box** &a_domain, const **Real** a_dx, **CCProjector** ∗a_finerProj, **CCProjector** ∗a_crseProj, int a_nRefCrse, int a_level, const **PhysBCUtil** &a_physBC)

  define function

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_crseGridsPtr, const **ProblemDomain** &a_domain, const **Real** a_dx, **CCProjector** ∗a_finerProj, **CCProjector** ∗a_crseProj, int a_nRefCrse, int a_level, const **PhysBCUtil** &a_physBC)

  define function

- void **init** (const **CCProjector** &a_oldProj)

  initialize new projection with data from old projection

- void **variableSetUp** ()

  define static parts

- void **setCrseProj** (**CCProjector** ∗a_crseProj, int nRefCrse)
- void **setFineProj** (**CCProjector** ∗a_fineProj)
- void **limitSolverCoarsening** (bool a_limitSolverCoarsening)

  set solver parameter

- void **writeCheckpointHeader** (HDF5Handle &a_handle) const

  write checkpoint header

- void **writeCheckpointLevel** (HDF5Handle &a_handle) const

  write this class to a checkpoint file for later restart

- void **readCheckpointHeader** (HDF5Handle &a_handle)

  read the checkpoint header

- void **readCheckpointLevel** (HDF5Handle &a_handle)

  read this class from a checkpoint file

- void **levelMacProject** (**LevelData**< **FluxBox** > &uEdge, **Real** a_oldTime, **Real** a_dt)
- void **LevelProject** (**LevelData**< **FArrayBox** > &a_velocity, **LevelData**< **FArrayBox** > ∗a_crseVelPtr, const **Real** a_newTime, const **Real** a_dt)
- void **doSyncOperations** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_velocity, **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_lambda, const **Real** a_newTime, const **Real** a_dtSync)
- void **initialLevelProject** (**LevelData**< **FArrayBox** > &a_velocity, **LevelData**< **FArrayBox** > ∗a_crseVelPtr, const **Real** a_oldTime, const **Real** a_newTime)
- void **doInitialSyncOperations** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_vel, **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_lambda, const **Real** a_newTime, const **Real** a_dtSync)
- void **initialVelocityProject** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_velocity, bool a_homogeneousCFBC=true)
- void **initialVelocityProject** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_velocity, **AMRSolver** &a_solver, bool a_homogeneousCFBC=true)
- void **doPostRegridOps** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_velocity, **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_lambda, const **Real** a_dt, const **Real** a_time)
- int **getLevel** () const

  access functions

- const **ProblemDomain** & **dProblem** () const
- const **DisjointBoxLayout** & **getBoxes** () const
- int **nRefCrse** () const
- **Real dx** () const
- **Real etaLambda** () const

  returns coefficient for volume-discrepancy solve.

- **CCProjector** ∗ **fineProjPtr** () const
- **CCProjector** ∗ **crseProjPtr** () const
- **LevelData**< **FArrayBox** > & **phi** ()

  returns MAC correction

- const **LevelData**< **FArrayBox** > & **phi** () const

const version of accessor

- void **gradPhi** (**LevelData**< **FArrayBox** > &a_gradPhi, int a_dir) const

  returns edge-centered grad(phi) in direction dir

- void **gradPhi** (**LevelData**< **FluxBox** > &a_gradPhi) const

  returns all components of grad(phi) (gradPhi should be correct size)

- **LevelData**< **FArrayBox** > & **Pi** ()

  returns level-projection pressure

- void **gradPi** (**LevelData**< **FArrayBox** > &a_gradPi, int a_dir) const

  returns grad(pi) in direction dir

- void **gradPi** (**LevelData**< **FArrayBox** > &a_gradPi) const
- **LevelData**< **FArrayBox** > & **eSync** ()

  returns synchronization correction

- const **LevelData**< **FArrayBox** > & **eSync** () const

  returns synchronization correction (const version)

- void **grad_eSync** (**LevelData**< **FArrayBox** > &a_grad_eSync, int a_dir) const

  returns cell-centered grad(eSync) (composite gradient)

- void **grad_eSync** (**LevelData**< **FArrayBox** > &a_grad_eSync) const

  returns cell-centered G^{comp}(eSync)

- **LevelData**< **FArrayBox** > & **eLambda** ()

  returns volume-discrepancy correction

- const **LevelData**< **FArrayBox** > & **eLambda** () const

  returns volume-discrepancy correction (const version)

- void **grad_eLambda** (**LevelData**< **FArrayBox** > &a_grad_eLambda, int a_dir) const

  returns edge-centered grad(eLambda) in direction dir

- **LevelData**< **FluxBox** > & **grad_eLambda** ()
- const **LevelData**< **FluxBox** > & **grad_eLambda** () const
- bool **doSyncProjection** () const

  do sync projection?

- bool **doMacSync** () const

  do volume discrepancy correction?

- bool **isInitialized** () const

  has this object been completely initialized?

- bool **doQuadInterp** () const

  use quadratic interpolation (instead of extrap) for c/f bc

- **QuadCFInterp** & **quadCFInterpolator** ()
  returns predefined quadratic coarse-fine interpolation object
- bool **isFinestLevel** () const
  is this the finest level?
- void **isFinestLevel** (bool a_finest_level)
  set whether this is the finest level
- void **verbosity** (int a_verbosity)
  set verbosity
- int **verbosity** () const
  returns verbosity
- void **setPhysBC** (const **PhysBCUtil** &a_bc)
  sets physBCs
- **PhysBCUtil** * **getPhysBCPtr** () const

Protected Member Functions

- void **applyMacCorrection** (**LevelData**< **FluxBox** > &a_uEdge, **Real** CFscale)
  *returns uEdge - G(phi_mac); C/F BC is CFscale*Pi(crse)*
- void **correctCCVelocities** (**LevelData**< **FArrayBox** > &a_velocity, const **Real** scale) const
  *vel := vel - scale*G_CC(pi)*
- void **doSyncProjection** (**Vector**< **LevelData**< **FArrayBox** > * > &a_velocity, const **Real** a_newTime, const **Real** a_dtSync, **AMRSolver** &solver)
- void **initialSyncProjection** (**Vector**< **LevelData**< **FArrayBox** > * > &a_velocity, const **Real** a_newTime, const **Real** a_dtSync, **AMRSolver** &a_solver)
  perform initial sync projection
- void **computeVDCorrection** (**Vector**< **LevelData**< **FArrayBox** > * > &a_lambda, const **Real** a_newTime, const **Real** a_dtSync, **AMRSolver** &solver)
- void **applySyncCorrection** (**Vector**< **LevelData**< **FArrayBox** > * > &a_velocity, const **Real** scale, **LevelData**< **FArrayBox** > *crseCorr)
- void **computeGrad_eLambda** ()
  compute composite gradient of eLambda – does this recursively
- void **rescaleGrad_eLambda** (**Real** a_dx_lbase)
  rescales composite gradient of eLambda, recursively all finer levels
- void **defineSolver** (**AMRSolver** &a_solver, const **Vector**< **LevelData**< **FArrayBox** > * > &a_vel, bool a_freestreamSolver)
- void **postRestart** ()
  takes care of setting ghost cell values after restarting

7.35.1 Detailed Description

this class manages the various forms of the CC projection

this class performs the various cell-centered projections required by the IAMR algorithm. These include the single-level operators levelMac and levelProject, as well as the multilevel sync projection and volume discrepancy solves.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 CCProjector::CCProjector ()

7.35.2.2 CCProjector::CCProjector (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_crseGridsPtr*, const Box & *a_domain*, const Real *a_dx*, CCProjector ∗ *a_finerProj*, CCProjector ∗ *a_crseProj*, int *a_nRefCrse*, int *a_level*, const PhysBCUtil & *a_physBC*)

full constructor

7.35.2.3 CCProjector::CCProjector (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_crseGridsPtr*, const ProblemDomain & *a_domain*, const Real *a_dx*, CCProjector ∗ *a_finerProj*, CCProjector ∗ *a_crseProj*, int *a_nRefCrse*, int *a_level*, const PhysBCUtil & *a_physBC*)

full constructor

7.35.2.4 CCProjector::∼CCProjector ()

destructor

7.35.3 Member Function Documentation

7.35.3.1 void CCProjector::applyMacCorrection (LevelData< FluxBox > & *a_uEdge*, Real *CFscale*) `[protected]`

returns uEdge - G(phi_mac); C/F BC is CFscale∗Pi(crse)

7.35.3.2 void CCProjector::applySyncCorrection (Vector< LevelData< FArrayBox > ∗ > & *a_velocity*, const Real *scale*, LevelData< FArrayBox > ∗ *crseCorr*) `[protected]`

7.35.3.3 void CCProjector::computeGrad_eLambda () `[protected]`

compute composite gradient of eLambda – does this recursively

7.35.3.4 void CCProjector::computeVDCorrection (Vector< LevelData< FArrayBox > ∗ > & *a_lambda*, const Real *a_newTime*, const Real *a_dtSync*, AMRSolver & *solver*) [protected]

7.35.3.5 void CCProjector::correctCCVelocities (LevelData< FArrayBox > & *a_velocity*, const Real *scale*) const [protected]

vel := vel - scale∗G_CC(pi)

7.35.3.6 CCProjector∗ CCProjector::crseProjPtr () const

7.35.3.7 void CCProjector::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_crseGridsPtr*, const ProblemDomain & *a_domain*, const Real *a_dx*, CCProjector ∗ *a_finerProj*, CCProjector ∗ *a_crseProj*, int *a_nRefCrse*, int *a_level*, const PhysBCUtil & *a_physBC*)

define function

7.35.3.8 void CCProjector::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_crseGridsPtr*, const Box & *a_domain*, const Real *a_dx*, CCProjector ∗ *a_finerProj*, CCProjector ∗ *a_crseProj*, int *a_nRefCrse*, int *a_level*, const PhysBCUtil & *a_physBC*)

define function

7.35.3.9 void CCProjector::defineSolver (AMRSolver & *a_solver*, const Vector< LevelData< FArrayBox > ∗ > & *a_vel*, bool *a_freestreamSolver*) [protected]

7.35.3.10 void CCProjector::doInitialSyncOperations (Vector< LevelData< FArrayBox > ∗ > & *a_vel*, Vector< LevelData< FArrayBox > ∗ > & *a_lambda*, const Real *a_newTime*, const Real *a_dtSync*)

7.35.3.11 bool CCProjector::doMacSync () const

do volume discrepancy correction?

7.35.3.12 void CCProjector::doPostRegridOps (Vector< LevelData< FArrayBox > ∗ > & *a_velocity*, Vector< LevelData< FArrayBox > ∗ > & *a_lambda*, const Real *a_dt*, const Real *a_time*)

7.35.3.13 bool CCProjector::doQuadInterp () const

use quadratic interpolation (instead of extrap) for c/f bc

7.35.3.14 void CCProjector::doSyncOperations (Vector< LevelData< FArrayBox > * > & *a_velocity*, Vector< LevelData< FArrayBox > * > & *a_lambda*, const Real *a_newTime*, const Real *a_dtSync*)

7.35.3.15 void CCProjector::doSyncProjection (Vector< LevelData< FArrayBox > * > & *a_velocity*, const Real *a_newTime*, const Real *a_dtSync*, AMRSolver & *solver*) [protected]

7.35.3.16 bool CCProjector::doSyncProjection () const

do sync projection?

7.35.3.17 const ProblemDomain& CCProjector::dProblem () const

7.35.3.18 Real CCProjector::dx () const

7.35.3.19 const LevelData<FArrayBox>& CCProjector::eLambda () const

returns volume-discrepancy correction (const version)

7.35.3.20 LevelData<FArrayBox>& CCProjector::eLambda ()

returns volume-discrepancy correction

7.35.3.21 const LevelData<FArrayBox>& CCProjector::eSync () const

returns synchronization correction (const version)

7.35.3.22 LevelData<FArrayBox>& CCProjector::eSync ()

returns synchronization correction

7.35.3.23 Real CCProjector::etaLambda () const

returns coefficient for volume-discrepancy solve.

7.35.3.24 CCProjector* CCProjector::fineProjPtr () const

7.35.3.25 const DisjointBoxLayout& CCProjector::getBoxes () const

7.35.3.26 int CCProjector::getLevel () const

access functions

7.35.3.27 PhysBCUtil* CCProjector::getPhysBCPtr () const

7.35.3.28 const LevelData<FluxBox>& CCProjector::grad_eLambda () const

7.35.3.29 LevelData<FluxBox>& CCProjector::grad_eLambda ()

7.35.3.30 void CCProjector::grad_eLambda (LevelData< FArrayBox > & *a_grad_eLambda*, int *a_dir*) const

returns edge-centered grad(eLambda) in direction dir

7.35.3.31 void CCProjector::grad_eSync (LevelData< FArrayBox > & *a_grad_eSync*) const

returns cell-centered G^{comp}(eSync)

7.35.3.32 void CCProjector::grad_eSync (LevelData< FArrayBox > & *a_grad_eSync*, int *a_dir*) const

returns cell-centered grad(eSync) (composite gradient)

7.35.3.33 void CCProjector::gradPhi (LevelData< FluxBox > & *a_gradPhi*) const

returns all components of grad(phi) (gradPhi should be correct size)

7.35.3.34 void CCProjector::gradPhi (LevelData< FArrayBox > & *a_gradPhi*, int *a_dir*) const

returns edge-centered grad(phi) in direction dir

7.35.3.35 void CCProjector::gradPi (LevelData< FArrayBox > & *a_gradPi*) const

7.35.3.36 void CCProjector::gradPi (LevelData< FArrayBox > & *a_gradPi*, int *a_dir*) const

returns grad(pi) in direction dir

7.35.3.37 void CCProjector::init (const CCProjector & *a_oldProj*)

initialize new projection with data from old projection

7.35.3.38 void CCProjector::initialLevelProject (LevelData< FArrayBox > & *a_velocity*, LevelData< FArrayBox > * *a_crseVelPtr*, const Real *a_oldTime*, const Real *a_newTime*)

Initialization functions performs initial level projection – phys and copy BC's should be preset

7.35.3.39 void CCProjector::initialSyncProjection (Vector< LevelData< FArrayBox > ∗ > & *a_velocity*, const Real *a_newTime*, const Real *a_dtSync*, AMRSolver & *a_solver*) `[protected]`

perform initial sync projection

7.35.3.40 void CCProjector::initialVelocityProject (Vector< LevelData< FArrayBox > ∗ > & *a_velocity*, AMRSolver & *a_solver*, bool *a_homogeneousCFBC* = true)

7.35.3.41 void CCProjector::initialVelocityProject (Vector< LevelData< FArrayBox > ∗ > & *a_velocity*, bool *a_homogeneousCFBC* = true)

7.35.3.42 void CCProjector::isFinestLevel (bool *a_finest_level*)

set whether this is the finest level

7.35.3.43 bool CCProjector::isFinestLevel () const

is this the finest level?

7.35.3.44 bool CCProjector::isInitialized () const

has this object been completely initialized?

7.35.3.45 void CCProjector::levelMacProject (LevelData< FluxBox > & *uEdge*, Real *a_oldTime*, Real *a_dt*)

7.35.3.46 void CCProjector::LevelProject (LevelData< FArrayBox > & *a_velocity*, LevelData< FArrayBox > ∗ *a_crseVelPtr*, const Real *a_newTime*, const Real *a_dt*)

7.35.3.47 void CCProjector::limitSolverCoarsening (bool *a_limitSolverCoarsening*)

set solver parameter

7.35.3.48 int CCProjector::nRefCrse () const

7.35.3.49 const LevelData<FArrayBox>& CCProjector::phi () const

const version of accessor

7.35.3.50 LevelData<FArrayBox>& CCProjector::phi ()

returns MAC correction

7.35.3.51 LevelData<FArrayBox>& CCProjector::Pi ()

returns level-projection pressure

7.35.3.52 void CCProjector::postRestart () [protected]

takes care of setting ghost cell values after restarting

7.35.3.53 QuadCFInterp& CCProjector::quadCFInterpolator ()

returns predefined quadratic coarse-fine interpolation object

7.35.3.54 void CCProjector::readCheckpointHeader (HDF5Handle & *a_handle*)

read the checkpoint header

7.35.3.55 void CCProjector::readCheckpointLevel (HDF5Handle & *a_handle*)

read this class from a checkpoint file

7.35.3.56 void CCProjector::rescaleGrad_eLambda (Real *a_dx_lbase*) [protected]

rescales composite gradient of eLambda, recursively all finer levels

rescales grad(eLambda) by (a_dx_lbase)/m_dx for this level and recursively calls this function for finer levels. This is done for stability reasons.

7.35.3.57 void CCProjector::setCrseProj (CCProjector * *a_crseProj*, int *nRefCrse*)

7.35.3.58 void CCProjector::setFineProj (CCProjector * *a_fineProj*)

7.35.3.59 void CCProjector::setPhysBC (const PhysBCUtil & *a_bc*)

sets physBCs

7.35.3.60 void CCProjector::variableSetUp ()

define static parts

7.35.3.61 int CCProjector::verbosity () const

returns verbosity

7.35.3.62 void CCProjector::verbosity (int *a_verbosity*)

set verbosity

7.35.3.63 void CCProjector::writeCheckpointHeader (HDF5Handle & *a_handle*) const

write checkpoint header

7.35.3.64 void CCProjector::writeCheckpointLevel (HDF5Handle & *a_handle*) const

write this class to a checkpoint file for later restart

The documentation for this class was generated from the following file:

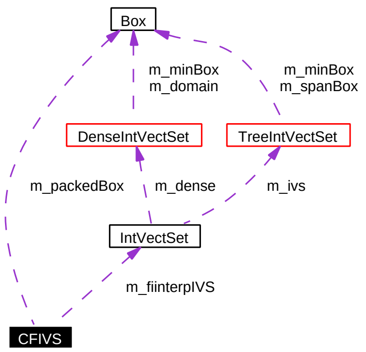
- **CCProjector.H**

7.36 CFIVS Class Reference

Internal Chombo class to get fine intvectset at CF interface.

`#include <CFIVS.H>`

Collaboration diagram for CFIVS:



Public Member Functions

- **CFIVS** ()

  Null constructor.

- **~CFIVS** ()

  Destructor.

- **CFIVS** (const **Box** &a_domain, const **Box** &a_boxIn, const **DisjointBoxLayout** &a_fineBoxes, int a_direction, **Side::LoHiSide** a_hiorlo)

  Deprecated constructor (The **ProblemDomain** *class has replaced the* **Box** *domain).*

- **CFIVS** (const **ProblemDomain** &a_domain, const **Box** &a_boxIn, const **DisjointBoxLayout** &a_fineBoxes, int a_direction, **Side::LoHiSide** a_hiorlo)

  Full constructor.

- void **define** (const **Box** &a_domain, const **Box** &a_boxIn, const **DisjointBoxLayout** &a_fineBoxes, int a_direction, **Side::LoHiSide** a_hiorlo)

  Deprecated define function.

- void **define** (const **ProblemDomain** &a_domain, const **Box** &a_boxIn, const **DisjointBoxLayout** &a_fineBoxes, int a_direction, **Side::LoHiSide** a_hiorlo)

  Full define function.

- bool **isDefined** () const

  is the face on the hi or lo side of the box? Returns "true" if the object has been defined

- bool **isPacked** () const

  Returns "true" if this CF **IntVectSet** *can be represented as a simple* **Box**.

- const **Box** & **packedBox** () const

  Returns packed box for when **isPacked()** *is true.*

- const **IntVectSet** & **getFineIVS** () const

  Get fine intvects which need to be interpolated.

Static Public Attributes

- long long **m_packCount**
- long long **m_sparseCount**

Protected Member Functions

- void **setDefaultValues** ()

Protected Attributes

- **IntVectSet m_fiinterpIVS**
- bool **m_packed**
- **Box m_packedBox**
- bool **m_isDefined**

7.36.1 Detailed Description

Internal Chombo class to get fine intvectset at CF interface.

Class to get fine intvectset at cf interface for one side of a given fine-level **Box**. This class should be considered internal to **AMRSolver** and should not be considered part of the Chombo API.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 CFIVS::CFIVS ()

Null constructor.

7.36.2.2 CFIVS::~CFIVS ()

Destructor.

7.36.2.3 CFIVS::CFIVS (const Box & *a_domain*, const Box & *a_boxIn*, const DisjointBoxLayout & *a_fineBoxes*, int *a_direction*, Side::LoHiSide *a_hiorlo*)

Deprecated constructor (The **ProblemDomain** class has replaced the **Box** domain).

Replaces **Box** domain with a **ProblemDomain** and calls define fn.

7.36.2.4 CFIVS::CFIVS (const ProblemDomain & *a_domain*, const Box & *a_boxIn*, const DisjointBoxLayout & *a_fineBoxes*, int *a_direction*, Side::LoHiSide *a_hiorlo*)

Full constructor.

Calls corresponding define function.

7.36.3 Member Function Documentation

7.36.3.1 void CFIVS::define (const ProblemDomain & *a_domain*, const Box & *a_boxIn*, const DisjointBoxLayout & *a_fineBoxes*, int *a_direction*, Side::LoHiSide *a_hiorlo*)

Full define function.

Parameters:

a_boxIn Problem domain at the fine level

a_fineBoxes fine-level grid box

a_direction fine-level grids

a_hiorlo direction of the face of interest

7.36.3.2 void CFIVS::define (const Box & *a_domain*, const Box & *a_boxIn*, const DisjointBoxLayout & *a_fineBoxes*, int *a_direction*, Side::LoHiSide *a_hiorlo*)

Deprecated define function.

Replaces **Box** domain with a **ProblemDomain** and calls define fn.

7.36.3.3 const IntVectSet& CFIVS::getFineIVS () const

Get fine intvects which need to be interpolated.

This will be empty if isEmpty() returns true

7.36.3.4 bool CFIVS::isDefined () const

is the face on the hi or lo side of the box? Returns "true" if the object has been defined

7.36.3.5 bool CFIVS::isPacked () const `[inline]`

Returns "true" if this CF **IntVectSet** can be represented as a simple **Box**.

Returns 'true' if this coarse-fine **IntVectSet** can be represented as just a **Box**, ie. a user can perform a dense computation instead of a pointwise calculation using **IVSIterator**.

7.36.3.6 const Box & CFIVS::packedBox () const `[inline]`

Returns packed box for when **isPacked()** is true.

if **isPacked()** returns 'true', then the box returned by this function is a suitable substitute for this **IntVectSet**

7.36.3.7 void CFIVS::setDefaultValues () [protected]

7.36.4 Member Data Documentation

7.36.4.1 IntVectSet CFIVS::m_fiinterpIVS [protected]

Fine intvects which need to be interpolated

7.36.4.2 bool CFIVS::m_isDefined [protected]

Has this object been defined?

7.36.4.3 long long CFIVS::m_packCount [static]

7.36.4.4 bool CFIVS::m_packed [protected]

Can the coarse-fine **IntVectSet** be represented as a **Box**?

7.36.4.5 Box CFIVS::m_packedBox [protected]

If m_packed is true, the **Box** with which the coarse-fine **IntVectSet** can be represented

7.36.4.6 long long CFIVS::m_sparseCount [static]

The documentation for this class was generated from the following file:

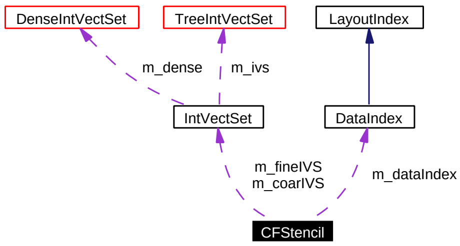
- **CFIVS.H**

7.37 CFStencil Class Reference

Class to encapsulate coarse-fine information for **AMR** operations.

`#include <CFStencil.H>`

Collaboration diagram for CFStencil:



Public Member Functions

- bool **isDefined** () const

  { Access functions}

- bool **isEmpty** () const
- const **IntVectSet** & **getFineIVS** () const
- const **IntVectSet** & **getCoarIVS** () const
- **CFStencil** & **operator=** (const **CFStencil** &cfs_in)
- **CFStencil** (const **CFStencil** &cfs_in)
- **CFStencil** (const **Box** &a_b, int a_nComp)
- void **define** (const **Box** &a_b, int a_nComp)
- **CFStencil** ()
- ~**CFStencil** ()
- **CFStencil** (const **Box** &FineDomain, const **Box** &grid, const **DisjointBoxLayout** &Levboxes, const **DisjointBoxLayout** &LevCBoxes, int RefRatio, int Direction, **Side::LoHiSide** hiorlo)
- **CFStencil** (const **ProblemDomain** &FineDomain, const **Box** &grid, const **DisjointBoxLayout** &Levboxes, const **DisjointBoxLayout** &LevCBoxes, int RefRatio, int Direction, **Side::LoHiSide** hiorlo)
- void **define** (const **Box** &FineDomain, const **Box** &grid, const **DisjointBoxLayout** &fine_boxes, const **DisjointBoxLayout** &coar_boxes, int Refratio, int Direction, **Side::LoHiSide** hiorlo)
- void **define** (const **ProblemDomain** &FineDomain, const **Box** &grid, const **DisjointBoxLayout** &fine_boxes, const **DisjointBoxLayout** &coar_boxes, int Refratio, int Direction, **Side::LoHiSide** hiorlo)
- void **clear** ()

Protected Member Functions

- void **setDefaultValues** ()

Protected Attributes

- int **m_direction**
- **Side::LoHiSide m_hiorlo**
- **DataIndex m_dataIndex**
- **IntVectSet m_fineIVS**
- **IntVectSet m_coarIVS**
- bool **m_isDefined**

7.37.1 Detailed Description

Class to encapsulate coarse-fine information for **AMR** operations.

CFStencil is a wrapper around a bunch of things that allow one to do coarse-fine interpolation. This class should be considered internal to **QuadCFInterp** and should not be considered part of the Chombo API.

7.37.2 Constructor & Destructor Documentation

7.37.2.1 CFStencil::CFStencil (const CFStencil & *cfs_in*)

7.37.2.2 CFStencil::CFStencil (const Box & *a_b*, int *a_nComp*)

7.37.2.3 CFStencil::CFStencil ()

7.37.2.4 CFStencil::~CFStencil ()

7.37.2.5 CFStencil::CFStencil (const Box & *FineDomain*, const Box & *grid*, const DisjointBoxLayout & *Levboxes*, const DisjointBoxLayout & *LevCBoxes*, int *RefRatio*, int *Direction*, Side::LoHiSide *hiorlo*)

7.37.2.6 CFStencil::CFStencil (const ProblemDomain & *FineDomain*, const Box & *grid*, const DisjointBoxLayout & *Levboxes*, const DisjointBoxLayout & *LevCBoxes*, int *RefRatio*, int *Direction*, Side::LoHiSide *hiorlo*)

7.37.3 Member Function Documentation

7.37.3.1 void CFStencil::clear ()

7.37.3.2 void CFStencil::define (const ProblemDomain & *FineDomain*, const Box & *grid*, const DisjointBoxLayout & *fine_boxes*, const DisjointBoxLayout & *coar_boxes*, int *Refratio*, int *Direction*, Side::LoHiSide *hiorlo*)

7.37.3.3 void CFStencil::define (const Box & *FineDomain*, const Box & *grid*, const DisjointBoxLayout & *fine_boxes*, const DisjointBoxLayout & *coar_boxes*, int *Refratio*, int *Direction*, Side::LoHiSide *hiorlo*)

7.37.3.4 void CFStencil::define (const Box & *a_b*, int *a_nComp*)

7.37.3.5 const IntVectSet& CFStencil::getCoarIVS () const

get coarse intvects that underly fiinterpivs. This will be empty if **isEmpty()** returns true this is the same as getCoarInterpIVS

7.37.3.6 const IntVectSet& CFStencil::getFineIVS () const

get fine intvects which need to be interpolated \ This will be empty if **isEmpty()** returns true

7.37.3.7 bool CFStencil::isDefined () const

{ Access functions}

returns true if CFStencil is defined

7.37.3.8 bool CFStencil::isEmpty () const

are there any interpolation points? \ returns false if so.

7.37.3.9 CFStencil& CFStencil::operator= (const CFStencil & *cfs_in*)

7.37.3.10 void CFStencil::setDefaultValues () [protected]

7.37.4 Member Data Documentation

7.37.4.1 IntVectSet CFStencil::m_coarIVS [protected]

7.37.4.2 DataIndex CFStencil::m_dataIndex [protected]

7.37.4.3 int CFStencil::m_direction [protected]

7.37.4.4 IntVectSet CFStencil::m_fineIVS [protected]

7.37.4.5 Side::LoHiSide CFStencil::m_hiorlo [protected]

7.37.4.6 bool CFStencil::m_isDefined [protected]

The documentation for this class was generated from the following file:

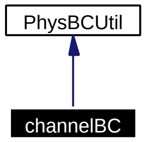
- **CFStencil.H**

7.38 channelBC Class Reference

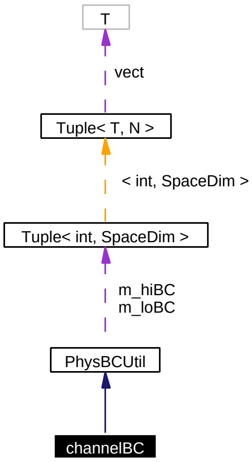
specific BC's for the convection problem (particularly for scalars)

`#include <channelBC.H>`

Inheritance diagram for channelBC:



Collaboration diagram for channelBC:



Public Member Functions

- **channelBC** ()

  constructor

- virtual ~**channelBC** ()

  destructor

- virtual **PhysBCUtil** * **newPhysBCUtil** () const

  "virtual" constructor

- void **setMaxInflowVel** (**Real** a_maxInflowVel)
- **Real maxInflowVel** () const
- virtual void **setBCs** ()

  interact with **ParmParse** *to set physical BC types*

- virtual void **computeBoundaryDt** (**Real** &a_dt, **Real** a_cfl, **Real** a_dx) const *compute effect of wall heating on dt*

- virtual **DomainGhostBC** ∗ **gradMacPressureBC** () const

  returns BC applied to MAC pressure before computing gradient

- virtual **DomainGhostBC** ∗ **gradELambdaBC** () const

  BC for grad eLambda.

- virtual **DomainGhostBC** ∗ **diffusiveBC** (int a_scalarType) const

  physical bc's for diffusion term for scalars

Protected Member Functions

- virtual **DomainGhostBC** ∗ **basicScalBC** (bool a_isHomogeneous, int a_scalType) const
- virtual void **basicCCVelBC** (**DomainGhostBC** &a_newBC, bool a_isHomogeneous, bool a_isViscous, int a_velComponent, **Interval** &a_interval) const

  returns a single-component BC for cell-centered velocity

- virtual **basicEdgeVelBC** ∗ **basicECVelBC** (bool a_isHomogeneous, bool a_isViscous, bool a_isSingleComponent) const

Protected Attributes

- **Real m_maxInflowVel**

  max inflow velocity

7.38.1 Detailed Description

specific BC's for the convection problem (particularly for scalars)

7.38.2 Constructor & Destructor Documentation

7.38.2.1 channelBC::channelBC ()

constructor

7.38.2.2 virtual channelBC::∼channelBC () `[virtual]`

destructor

7.38.3 Member Function Documentation

7.38.3.1 virtual void channelBC::basicCCVelBC (DomainGhostBC & *a_newBC*, bool *a_isHomogeneous*, bool *a_isViscous*, int *a_velComponent*, Interval & *a_interval*) const `[protected, virtual]`

returns a single-component BC for cell-centered velocity

Reimplemented from **PhysBCUtil** (p. 490).

7.38.3.2 virtual basicEdgeVelBC* channelBC::basicECVelBC (bool *a_isHomogeneous*, bool *a_isViscous*, bool *a_isSingleComponent*) const [protected, virtual]

Reimplemented from **PhysBCUtil** (p. 490).

7.38.3.3 virtual DomainGhostBC* channelBC::basicScalBC (bool *a_isHomogeneous*, int *a_scalType*) const [protected, virtual]

Reimplemented from **PhysBCUtil** (p. 490).

7.38.3.4 virtual void channelBC::computeBoundaryDt (Real & *a_dt*, Real *a_cfl*, Real *a_dx*) const [virtual]

compute effect of wall heating on dt

Reimplemented from **PhysBCUtil** (p. 490).

7.38.3.5 virtual DomainGhostBC* channelBC::diffusiveBC (int *a_scalarType*) const [virtual]

physical bc's for diffusion term for scalars

Reimplemented from **PhysBCUtil** (p. 490).

7.38.3.6 virtual DomainGhostBC* channelBC::gradELambdaBC () const [virtual]

BC for grad eLambda.

Reimplemented from **PhysBCUtil** (p. 491).

7.38.3.7 virtual DomainGhostBC* channelBC::gradMacPressureBC () const [virtual]

returns BC applied to MAC pressure before computing gradient

Reimplemented from **PhysBCUtil** (p. 491).

7.38.3.8 Real channelBC::maxInflowVel () const

7.38.3.9 virtual PhysBCUtil* channelBC::newPhysBCUtil () const [virtual]

"virtual" constructor

Reimplemented from **PhysBCUtil** (p. 492).

7.38.3.10 virtual void channelBC::setBCs () [virtual]

interact with **ParmParse** to set physical BC types

Reimplemented from **PhysBCUtil** (p. 492).

7.38.3.11 void channelBC::setMaxInflowVel (Real *a_maxInflowVel*)

7.38.4 Member Data Documentation

7.38.4.1 Real channelBC::m_maxInflowVel [protected]

max inflow velocity

The documentation for this class was generated from the following file:

- **channelBC.H**

7.39 chfptr Class Reference

#include <FORT_PROTO.H>

Public Member Functions

- **chfptr** (int a)
- **operator const int** * () const

Public Attributes

- int **i**

7.39.1 Constructor & Destructor Documentation

7.39.1.1 chfptr::chfptr (int *a*) [inline]

7.39.2 Member Function Documentation

7.39.2.1 chfptr::operator const int * () const [inline]

7.39.3 Member Data Documentation

7.39.3.1 int chfptr::i

The documentation for this class was generated from the following file:

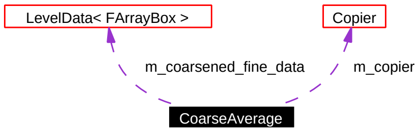
- **FORT_PROTO.H**

7.40 CoarseAverage Class Reference

replaces coarse level data with an average of fine level data.

`#include <CoarseAverage.H>`

Collaboration diagram for CoarseAverage:



Public Member Functions

- **CoarseAverage** ()
- **~CoarseAverage** ()
- **CoarseAverage** (const **DisjointBoxLayout** &a_fine_domain, int a_numcomps, int a_ref_ratio)
- **CoarseAverage** (const **DisjointBoxLayout** &a_fine_domain, const **DisjointBoxLayout** &a_crse_domain, int a_numcomps, int a_ref_ratio)
- void **define** (const **DisjointBoxLayout** &a_fine_domain, int a_numcomps, int a_ref_ratio)
- void **define** (const **DisjointBoxLayout** &a_fine_domain, const **DisjointBoxLayout** &a_crse_domain, int a_numcomps, int a_ref_ratio)
- bool **isDefined** () const
- void **averageToCoarse** (**LevelData**< **FArrayBox** > &a_coarse_data, const **LevelData**< **FArrayBox** > &a_fine_data)

Protected Member Functions

- void **averageGridData** (**BaseFab**< **Real** > &a_coarse, const **BaseFab**< **Real** > &a_fine, int a_ref_ratio) const

Protected Attributes

- bool **is_defined**
- int **m_ref_ratio**
- **LevelData**< **FArrayBox** > **m_coarsened_fine_data**
- bool **m_is_copier_defined**
- **Copier** **m_copier**

7.40.1 Detailed Description

replaces coarse level data with an average of fine level data.

This class replaces data at a coarse level of refinement with an average of data at a finer level of refinement, in areas where fine data is present. Coarse level data is not modified where fine level data is not present.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 CoarseAverage::CoarseAverage ()

Default constructor. User must subsequently call **define()**.

7.40.2.2 CoarseAverage::~CoarseAverage ()

Destructor.

7.40.2.3 CoarseAverage::CoarseAverage (const DisjointBoxLayout & *a_fine_domain*, int *a_numcomps*, int *a_ref_ratio*)

Defining constructor. Constructs a valid object. Equivalent to default construction followed by **define()**.

{ Arguments:}\ a_fine_domain (not modified): the fine level domain.\ a_numcomps (not modified): the number of components.\ a_ref_ratio (not modified): the refinement ratio.\

7.40.2.4 CoarseAverage::CoarseAverage (const DisjointBoxLayout & *a_fine_domain*, const DisjointBoxLayout & *a_crse_domain*, int *a_numcomps*, int *a_ref_ratio*)

Defining constructor. Constructs a valid object. Equivalent to default construction followed by **define()**. This version takes the coarser level as well, results in a faster averaging operation, since **Copier** can be pre-constructed.

{ Arguments:}\ a_fine_domain (not modified): the fine level domain.\ a_crse_domain (not modified): the crse level domain.\ a_numcomps (not modified): the number of components.\ a_ref_ratio (not modified): the refinement ratio.\

7.40.3 Member Function Documentation

7.40.3.1 void CoarseAverage::averageGridData (BaseFab< Real > & *a_coarse*, const BaseFab< Real > & *a_fine*, int *a_ref_ratio*) const `[protected]`

7.40.3.2 void CoarseAverage::averageToCoarse (LevelData< FArrayBox > & *a_coarse_data*, const LevelData< FArrayBox > & *a_fine_data*)

Replaces a_coarse_data with the average of a_fine_data within the coarsening of the domain of the fine level. Elsewhere, the coarse data is unchanged. It is an error to call if not this->**isDefined()**. The domain of a_fine_data should be the same as the fine domain specified in the most recent call to **define()**. It is expected that the coarse and fine level's domains are properly nested. Both a_coarse_data and a_fine_data should have the same number of components specified in the most recent call to **define()**.

{ Arguments:}\ a_coarse_data (modified): coarse data. \ a_fine_data (not modified): fine data. \

{ This:}\ Well, it's complicated. As far as the user is concerned, this object is not modified. See the design document if you care for details.

7.40.3.3 void CoarseAverage::define (const DisjointBoxLayout & *a_fine_domain*, const DisjointBoxLayout & *a_crse_domain*, int *a_numcomps*, int *a_ref_ratio*)

Defines this object. Existing information is overriden. This version takes the coarser-level grids as well, which allows for a faster averaging operation, since the **Copier** can be pre-defined

{ Arguments:}\ a_fine_domain (not modified): the fine level domain.\ a_crse_domain (not modified): the coarse level domain.\ a_numcomps (not modified): the number of components.\ a_ref_ratio (not modified): the refinement ratio.\

{ This:}\ —This object is modified.—

7.40.3.4 void CoarseAverage::define (const DisjointBoxLayout & *a_fine_domain*, int *a_numcomps*, int *a_ref_ratio*)

Defines this object. Existing information is overriden.

{ Arguments:}\ a_fine_domain (not modified): the fine level domain.\ a_numcomps (not modified): the number of components.\ a_ref_ratio (not modified): the refinement ratio.\

{ This:}\ —This object is modified.—

7.40.3.5 bool CoarseAverage::isDefined () const

Returns true if this object was created with the defining constructor or if **define()** has been called.

{ This:}\ This object is not modified.

7.40.4 Member Data Documentation

7.40.4.1 bool CoarseAverage::is_defined `[protected]`

7.40.4.2 LevelData<FArrayBox> CoarseAverage::m_coarsened_fine_data `[protected]`

7.40.4.3 Copier CoarseAverage::m_copier `[protected]`

7.40.4.4 bool CoarseAverage::m_is_copier_defined `[protected]`

7.40.4.5 int CoarseAverage::m_ref_ratio `[protected]`

The documentation for this class was generated from the following file:

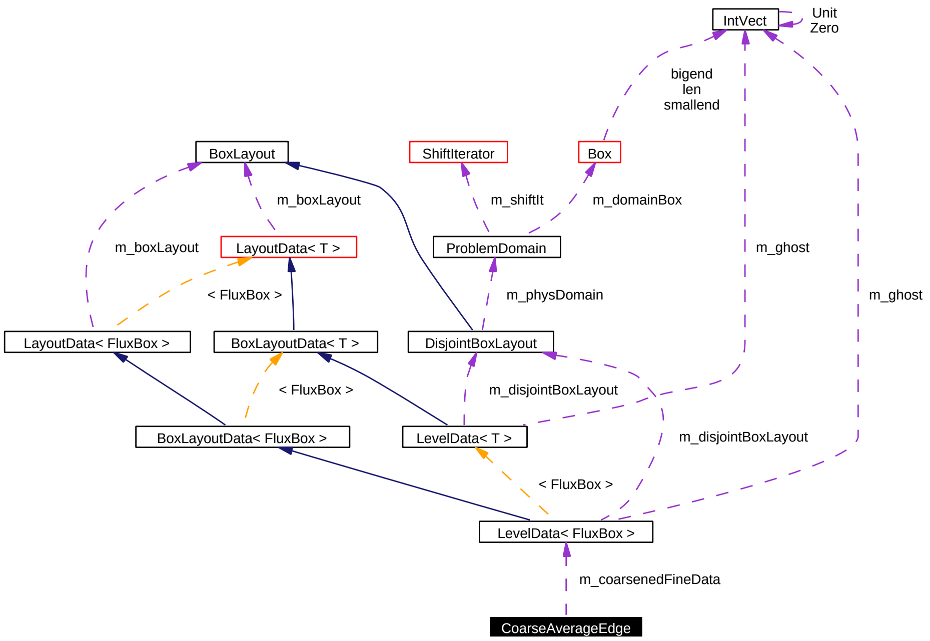
- **CoarseAverage.H**

7.41 CoarseAverageEdge Class Reference

replaces edge-centered coarse-level data w/ averaged fine-level data

`#include <CoarseAverageEdge.H>`

Collaboration diagram for CoarseAverageEdge:



Public Member Functions

- **CoarseAverageEdge** ()

  Default constructor.

- **CoarseAverageEdge** (const **DisjointBoxLayout** &a_fineGrids, int a_nComp, int a_nRef)

  defining constructor

- **~CoarseAverageEdge** ()

  destructor

- void **define** (const **DisjointBoxLayout** &a_fineGrids, int a_nComp, int a_nRef)

  defines the object

- bool **isDefined** () const
- void **averageToCoarse** (**LevelData**< **FluxBox** > &a_coarse_data, const **LevelData**< **FluxBox** > &a_fine_data)

  averages fine-level data to coarse level

Protected Member Functions

- void **averageGridData** (**FluxBox** &a_coarsenedFine, const **FluxBox** &fine) const

  averages entire single grid data from fine->crse

Protected Attributes

- bool **m_isDefined**
- int **m_nRef**
- **LevelData< FluxBox > m_coarsenedFineData**

7.41.1 Detailed Description

replaces edge-centered coarse-level data w/ averaged fine-level data

This class replaces edge-centered data on a coarse level of refinement with the average of the finer-level data which overlays the edge. This class is similar to **CoarseAverage**

7.41.2 Constructor & Destructor Documentation

7.41.2.1 CoarseAverageEdge::CoarseAverageEdge ()

Default constructor.

7.41.2.2 CoarseAverageEdge::CoarseAverageEdge (const DisjointBoxLayout & *a_fineGrids*, int *a_nComp*, int *a_nRef*)

defining constructor

7.41.2.3 CoarseAverageEdge::~CoarseAverageEdge ()

destructor

7.41.3 Member Function Documentation

7.41.3.1 void CoarseAverageEdge::averageGridData (FluxBox & *a_coarsenedFine*, const FluxBox & *fine*) const [protected]

averages entire single grid data from fine->crse

7.41.3.2 void CoarseAverageEdge::averageToCoarse (LevelData< FluxBox > & *a_coarse_data*, const LevelData< FluxBox > & *a_fine_data*)

averages fine-level data to coarse level

7.41.3.3 void CoarseAverageEdge::define (const DisjointBoxLayout & *a_fineGrids*, int *a_nComp*, int *a_nRef*)

defines the object

7.41.3.4 bool CoarseAverageEdge::isDefined () const

7.41.4 Member Data Documentation

7.41.4.1 LevelData<FluxBox> CoarseAverageEdge::m_coarsenedFineData [protected]

7.41.4.2 bool CoarseAverageEdge::m_isDefined [protected]

7.41.4.3 int CoarseAverageEdge::m_nRef [protected]

The documentation for this class was generated from the following file:

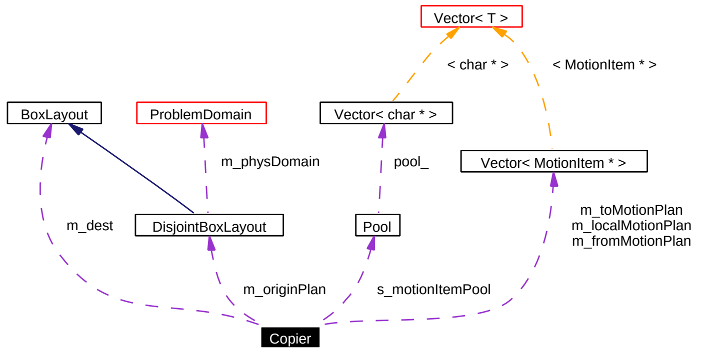
- **CoarseAverageEdge.H**

7.42 Copier Class Reference

A strange but true thing to make copying from one boxlayoutdata to another fast.

`#include <Copier.H>`

Collaboration diagram for Copier:



Public Member Functions

- **Copier** ()

  null constructor, copy constructor and operator= can be compiler defined.

- **Copier** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, bool a_exchange=false)
- **Copier** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **ProblemDomain** &a_domain, bool a_exchange=false)

  this constructor contains support for periodic BC's

- **Copier** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **IntVect** &a_destGhost, bool a_exchange=false)

  Copier to copy data into the valid and invalid regions of a_dest from a_level.

- **Copier** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **ProblemDomain** &a_domain, const **IntVect** &a_destGhost, bool a_exchange=false)

  Copier to copy data into the valid and invalid regions of a_dest from a_level with support for periodic BCs.

- virtual ~**Copier** ()
- virtual void **define** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, bool a_exchange=false)
- virtual void **define** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **ProblemDomain** &a_domain, bool a_exchange=false)

  contains support for periodic BCs

- virtual void **define** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **IntVect** &a_destGhost, bool a_exchange=false)
- virtual void **define** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **ProblemDomain** &a_domain, const **IntVect** &a_destGhost, bool a_exchange=false)

contains support for periodic BCs

- void **ghostDefine** (const **DisjointBoxLayout** &a_src, const **DisjointBoxLayout** &a_dest, const **ProblemDomain** &a_domain, const **IntVect** &a_srcGhost)

  performs the computation necessary for moving from ghost+valid cells to valid.

- virtual void **clear** ()
- bool **check** (const **DisjointBoxLayout** &from, const **BoxLayout** &to) const

  check that this Copier object corresponds to these two BoxLayouts.

- int **print** () const
- bool **bufferAllocated** () const
- void **setBufferAllocated** (bool arg) const

Protected Attributes

- **Vector< MotionItem * > m_localMotionPlan**
- **Vector< MotionItem * > m_fromMotionPlan**
- **Vector< MotionItem * > m_toMotionPlan**
- bool **buffersAllocated**

Static Protected Attributes

- **Pool s_motionItemPool**

Friends

- class **CopyIterator**
- void **dumpmemoryatexit** ()

7.42.1 Detailed Description

A strange but true thing to make copying from one boxlayoutdata to another fast.

class to handle the organization of data from a **LevelData** to a **BoxLayoutData** defined over the same index space. The idea behind this object is twofold: A) there is a significant amount of information that can be computed once and reused from one copy to the next for a **LevelData** and **BoxLayoutData** that have unchanging BoxLayouts underneath them. In these cases, **LevelData::copyTo** methods has an optional argument. That way, in cases where the operation is only performed a single time, we do not waste our time building optimizing data, etc. B) when we interface with KeLP, this class maps quite well with the existing KeLP API (class MotionPlan).

pains have been taken to not have this class be templated. Thus it does not actually perform the copying itself (which would require knowledge of the template class type). It provides an iterator interface to MotionItems that specify the required work. **LevelData** provides the copyTo methods.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 Copier::Copier () [inline]

null constructor, copy constructor and operator= can be compiler defined.

7.42.2.2 Copier::Copier (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, bool *a_exchange* = false)

7.42.2.3 Copier::Copier (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const ProblemDomain & *a_domain*, bool *a_exchange* = false)

this constructor contains support for periodic BC's

7.42.2.4 Copier::Copier (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const IntVect & *a_destGhost*, bool *a_exchange* = false)

Copier to copy data into the valid and invalid regions of a_dest from a_level.

7.42.2.5 Copier::Copier (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const ProblemDomain & *a_domain*, const IntVect & *a_destGhost*, bool *a_exchange* = false)

Copier to copy data into the valid and invalid regions of a_dest from a_level with support for periodic BCs.

7.42.2.6 virtual Copier::~Copier () [virtual]

7.42.3 Member Function Documentation

7.42.3.1 bool Copier::bufferAllocated () const

7.42.3.2 bool Copier::check (const DisjointBoxLayout & *from*, const BoxLayout & *to*) const

check that this Copier object corresponds to these two BoxLayouts.

7.42.3.3 virtual void Copier::clear () [virtual]

7.42.3.4 virtual void Copier::define (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const ProblemDomain & *a_domain*, const IntVect & *a_destGhost*, bool *a_exchange* = false) [virtual]

contains support for periodic BCs

7.42.3.5 virtual void Copier::define (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const IntVect & *a_destGhost*, bool *a_exchange* = false) [virtual]

7.42.3.6 virtual void Copier::define (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const ProblemDomain & *a_domain*, bool *a_exchange* = false) [virtual]

contains support for periodic BCs

7.42.3.7 virtual void Copier::define (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, bool *a_exchange* = false) [virtual]

7.42.3.8 void Copier::ghostDefine (const DisjointBoxLayout & *a_src*, const DisjointBoxLayout & *a_dest*, const ProblemDomain & *a_domain*, const IntVect & *a_srcGhost*)

performs the computation necessary for moving from ghost+valid cells to valid.

instead of copying from the valid of a_src to the valid+ghost of a_dest, this routine defines a Copier that moves data from the ghost+valid region of a_src to the valid region of a_dest. This is used by **LevelFluxRegister**.

7.42.3.9 int Copier::print () const

7.42.3.10 void Copier::setBufferAllocated (bool *arg*) const

7.42.4 Friends And Related Function Documentation

7.42.4.1 friend class CopyIterator [friend]

7.42.4.2 void dumpmemoryatexit () [friend]

7.42.5 Member Data Documentation

7.42.5.1 bool Copier::buffersAllocated [mutable, protected]

7.42.5.2 Vector<MotionItem*> Copier::m_fromMotionPlan [protected]

7.42.5.3 Vector<MotionItem*> Copier::m_localMotionPlan [protected]

7.42.5.4 Vector<MotionItem*> Copier::m_toMotionPlan [protected]

7.42.5.5 Pool Copier::s_motionItemPool [static, protected]

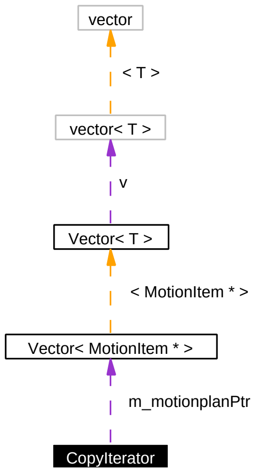
The documentation for this class was generated from the following file:

- **Copier.H**

7.43 CopyIterator Class Reference

`#include <Copier.H>`

Collaboration diagram for CopyIterator:



Public Types

- enum **local_from_to** { **LOCAL**, **FROM**, **TO** }

Public Member Functions

- **CopyIterator** (const **Copier** &a_copier, **local_from_to**)
- const **MotionItem** & **operator()** () const
- void **operator++** ()
- bool **ok** () const
- void **reset** ()

7.43.1 Member Enumeration Documentation

7.43.1.1 enum CopyIterator::local_from_to

Enumeration values:

LOCAL

FROM

TO

7.43.2 Constructor & Destructor Documentation

7.43.2.1 CopyIterator::CopyIterator (const Copier & *a_copier*, local_from_to) [inline]

7.43.3 Member Function Documentation

7.43.3.1 bool CopyIterator::ok () const [inline]

7.43.3.2 const MotionItem & CopyIterator::operator() () const [inline]

7.43.3.3 void CopyIterator::operator++ () [inline]

7.43.3.4 void CopyIterator::reset () [inline]

The documentation for this class was generated from the following file:

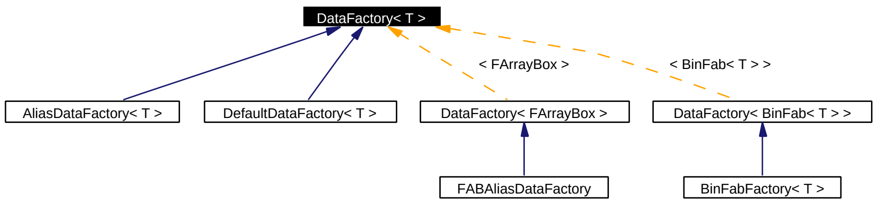
- **Copier.H**

7.44 DataFactory< T > Class Template Reference

Factory object to data members of a **BoxLayoutData** container.

`#include <BoxLayoutData.H>`

Inheritance diagram for DataFactory< T >:



Public Member Functions

- virtual T ∗ **create** (const **Box** &box, int ncomps, const **DataIndex** &a_datInd) const=0

  factory function. creates a new 'T' object

- virtual bool **callDelete** () const

7.44.1 Detailed Description

template<class T> class DataFactory< T >

Factory object to data members of a **BoxLayoutData** container.

7.44.2 Member Function Documentation

7.44.2.1 template<class T> virtual bool DataFactory< T >::callDelete () const `[inline, virtual]`

7.44.2.2 template<class T> virtual T∗ DataFactory< T >::create (const Box & *box*, int *ncomps*, const DataIndex & *a_datInd*) const `[pure virtual]`

factory function. creates a new 'T' object

creates a new 'T' object and returns a pointer to it.

Implemented in **DefaultDataFactory< T >** (p. 211), **FABAliasDataFactory** (p. 262), **AliasDataFactory< T >** (p. 24), and **BinFabFactory< T >** (p. 102).

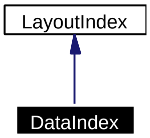
The documentation for this class was generated from the following file:

- **BoxLayoutData.H**

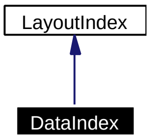
7.45 DataIndex Class Reference

`#include <DataIndex.H>`

Inheritance diagram for DataIndex:



Collaboration diagram for DataIndex:



Public Member Functions

- **DataIndex** (const **LayoutIndex** &promotion)
- **DataIndex** ()
- ~**DataIndex** ()

Friends

- class **LayoutIterator**
- class **DataIterator**
- class **BoxLayout**
- class **Copier**

7.45.1 Constructor & Destructor Documentation

7.45.1.1 DataIndex::DataIndex (const LayoutIndex & *promotion*) `[inline, explicit]`

7.45.1.2 DataIndex::DataIndex () `[inline]`

7.45.1.3 DataIndex::~DataIndex () `[inline]`

7.45.2 Friends And Related Function Documentation

7.45.2.1 friend class BoxLayout `[friend]`

Reimplemented from **LayoutIndex** (p. 349).

7.45.2.2 friend class Copier `[friend]`

7.45.2.3 friend class DataIterator `[friend]`

Reimplemented from **LayoutIndex** (p. 349).

7.45.2.4 friend class LayoutIterator `[friend]`

Reimplemented from **LayoutIndex** (p. 349).

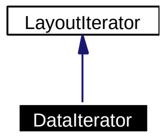
The documentation for this class was generated from the following file:

- **DataIndex.H**

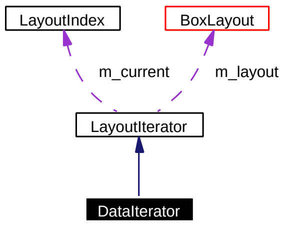
7.46 DataIterator Class Reference

`#include <DataIterator.H>`

Inheritance diagram for DataIterator:



Collaboration diagram for DataIterator:



Public Member Functions

- **DataIterator** ()
- **DataIterator** (const **BoxLayout** &a_layout)
- const **DataIndex** & **operator()** () const

  return the index that this iterator is at

- **DataIndex i** () const

  return a copy of the index that this iterator is at

Friends

- class **BoxLayout**
- class **DisjointBoxLayout**

7.46.1 Constructor & Destructor Documentation

7.46.1.1 DataIterator::DataIterator () `[inline]`

7.46.1.2 DataIterator::DataIterator (const BoxLayout & *a_layout*) `[inline]`

7.46.2 Member Function Documentation

7.46.2.1 DataIndex DataIterator::i () const `[inline]`

return a copy of the index that this iterator is at

Aborts if the iterator is not **ok()**

Reimplemented from **LayoutIterator** (p. 352).

7.46.2.2 const DataIndex& DataIterator::operator() () const [inline]

return the index that this iterator is at

Aborts if the iterator is not **ok()**

Reimplemented from **LayoutIterator** (p. 352).

7.46.3 Friends And Related Function Documentation

7.46.3.1 friend class BoxLayout [friend]

Reimplemented from **LayoutIterator** (p. 353).

7.46.3.2 friend class DisjointBoxLayout [friend]

The documentation for this class was generated from the following file:

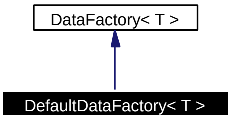
- **DataIterator.H**

7.47 DefaultDataFactory< T > Class Template Reference

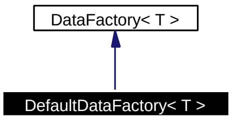
Factory object to data members of a **BoxLayoutData** container.

`#include <BoxLayoutData.H>`

Inheritance diagram for DefaultDataFactory< T >:



Collaboration diagram for DefaultDataFactory< T >:



Public Member Functions

- virtual T * **create** (const **Box** &box, int ncomps, const **DataIndex** &a_datInd) const
  factory function. creates a new 'T' object

7.47.1 Detailed Description

template<class T> class DefaultDataFactory< T >

Factory object to data members of a **BoxLayoutData** container.

creates new T object by invoking operator new

```
template <class T>
T* DefaultDataFactory<T>::create(const Box& box,
                                 int ncomps,
                                 const DataIndex& a_datInd) const
{
  return new T(box, ncomps);
}
```

7.47.2 Member Function Documentation

7.47.2.1 template<class T> T * DefaultDataFactory< T >::create (const Box & *box*, int *ncomps*, const DataIndex & *a_datInd*) const [virtual]

factory function. creates a new 'T' object

creates a new 'T' object and returns a pointer to it. Responsiblitly for calling operator 'delete' on this pointer is passed to the user.

Implements **DataFactory**$<$ **T** $>$ (p. 206).

The documentation for this class was generated from the following file:

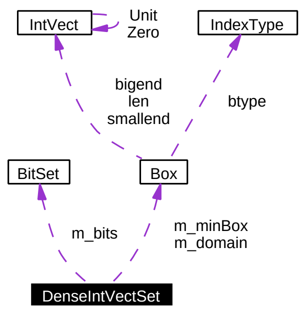
- **BoxLayoutData.H**

7.48 DenseIntVectSet Class Reference

Dense representation implementation of **IntVectSet** class.

`#include <DenseIntVectSet.H>`

Collaboration diagram for DenseIntVectSet:



Public Member Functions

- **DenseIntVectSet** ()
- **DenseIntVectSet** (const **Box** &a_domain, bool init=true)
- **DenseIntVectSet** & **operator-=** (const **Box** &box)
- **DenseIntVectSet** & **operator-=** (const **IntVect** &intvect)
- **DenseIntVectSet** & **operator-=** (const **DenseIntVectSet** &ivs)
- **DenseIntVectSet** & **operator &=** (const **Box** &box)
- **DenseIntVectSet** & **operator &=** (const **ProblemDomain** &domain)
- **DenseIntVectSet** & **operator\|=** (const **IntVect** &intvect)
- **DenseIntVectSet** & **operator\|=** (const **Box** &b)
- **DenseIntVectSet** & **operator\|=** (const **DenseIntVectSet** &b)
- **DenseIntVectSet** & **operator &=** (const **DenseIntVectSet** &ivs)
- void **shift** (const **IntVect** &iv)

  Shift every cell in the set by the input. Fast.

- bool **operator[ ]** (const **IntVect** &index) const

  O(1) time inquiry of containment.

- bool **contains** (const **Box** &box) const

  O(1) time inquiry of containment.

- void **coarsen** (int iref)
- void **refine** (int iref)
- void **grow** (int igrow)
- void **grow** (int idir, int igrow)
- **DenseIntVectSet chop** (int dir, int chop_pnt)
- const **Box** & **box** () const
- void **nestingRegion** (int a_radius, const **Box** &a_domain)
- void **nestingRegion** (int a_radius, const **ProblemDomain** &a_domain)
- bool **isEmpty** () const

cheaper than **numPts()***==0 by wide margin*

- bool **isFull** () const

  cheaper than **numPts()***==box().***numPts()***, by significant margin*

- int **numPts** () const
- bool **operator==** (const **DenseIntVectSet** &a_lhs) const
- **Vector**< **Box** > **createBoxes** () const
- void **compact** () const
- void **recalcMinBox** () const
- const **Box** & **mBox** () const

Linearization routines

- int **linearSize** () const
- void **linearIn** (const void ∗const inBuf)
- void **linearOut** (void ∗const a_outBuf) const

Friends

- class **DenseIntVectSetIterator**

7.48.1 Detailed Description

Dense representation implementation of **IntVectSet** class.

Performance-oriented class that reproduces most of the functionality of **IntVectSet**. The one function that is not supported is union (|=). This object stores a dense representation of an **IntVectSet** over the box of definition

Storage is performed with a **BitSet**. This stores 0 or 1 on a per-bit basis for each location in the **IntVectSet**.

For an explanation of undocumented functions look at **IntVectSet**

See also:
IntVectSet

7.48.2 Constructor & Destructor Documentation

7.48.2.1 DenseIntVectSet::DenseIntVectSet () `[inline]`

7.48.2.2 DenseIntVectSet::DenseIntVectSet (const Box & *a_domain*, bool *init* = true)

you can either have the domain begin all true, or all false

7.48.3 Member Function Documentation

7.48.3.1 const Box & DenseIntVectSet::box () const [inline]

7.48.3.2 DenseIntVectSet DenseIntVectSet::chop (int *dir*, int *chop_pnt*)

Chop the DenseIntVectSet at the chop_pnt in the dir direction Returns one DenseIntVectSet and modifies the object DenseIntVectSet. The union of the two is the original **IntVectSet**. The modified DenseIntVectSet is the low end, the returned DenseIntvectSet is the high end.

See also:
IntVectSet::chop

7.48.3.3 void DenseIntVectSet::coarsen (int *iref*)

coarsen the set by the input

7.48.3.4 void DenseIntVectSet::compact () const

set object to its minimum representation. changes data, but is a logically const operation

7.48.3.5 bool DenseIntVectSet::contains (const Box & *box*) const

O(1) time inquiry of containment.

returns 'true' if the entire set of points specified by the box 'b' is a member of the IntvectSet

7.48.3.6 Vector<Box> DenseIntVectSet::createBoxes () const

turn DenseIntVectSet into a **Vector** of Boxes

7.48.3.7 void DenseIntVectSet::grow (int *idir*, int *igrow*)

grow the set by the input

7.48.3.8 void DenseIntVectSet::grow (int *igrow*)

grow the set by the input

7.48.3.9 bool DenseIntVectSet::isEmpty () const

cheaper than **numPts()**==0 by wide margin

return true if the set has no points

7.48.3.10 bool DenseIntVectSet::isFull () const

cheaper than **numPts()**==box().**numPts()**, by significant margin

return true if the set consists of a set of a full box

7.48.3.11 void DenseIntVectSet::linearIn (const void *const *inBuf*)

7.48.3.12 void DenseIntVectSet::linearOut (void *const *a_outBuf*) const

7.48.3.13 int DenseIntVectSet::linearSize () const

7.48.3.14 const Box& DenseIntVectSet::mBox () const `[inline]`

7.48.3.15 void DenseIntVectSet::nestingRegion (int *a_radius*, const ProblemDomain & *a_domain*)

7.48.3.16 void DenseIntVectSet::nestingRegion (int *a_radius*, const Box & *a_domain*)

See also:
IntVectSet::nestingRegion

7.48.3.17 int DenseIntVectSet::numPts () const

return the number of points in the set

7.48.3.18 DenseIntVectSet& DenseIntVectSet::operator &= (const DenseIntVectSet & *ivs*)

make this set the intersection of itself and the input

7.48.3.19 DenseIntVectSet& DenseIntVectSet::operator &= (const ProblemDomain & *domain*)

make this set the intersection of itself and the input

7.48.3.20 DenseIntVectSet& DenseIntVectSet::operator &= (const Box & *box*)

make this set the intersection of itself and the input

7.48.3.21 DenseIntVectSet& DenseIntVectSet::operator-= (const DenseIntVectSet & *ivs*)

make this set the complement of itself and the input

7.48.3.22 DenseIntVectSet & DenseIntVectSet::operator-= (const IntVect & *intvect*) `[inline]`

make this set the complement of itself and the input

7.48.3.23 DenseIntVectSet& DenseIntVectSet::operator-= (const Box & *box*)

make this set the complement of itself and the input

7.48.3.24 bool DenseIntVectSet::operator== (const DenseIntVectSet & *a_lhs*) const

7.48.3.25 bool DenseIntVectSet::operator[] (const IntVect & *index*) const

O(1) time inquiry of containment.

7.48.3.26 DenseIntVectSet& DenseIntVectSet::operator\|= (const DenseIntVectSet & *b*)

make this set the union of itself and the input. resulting DenseIntVectSet has domain of the minBox holding both oeprator and operand

7.48.3.27 DenseIntVectSet& DenseIntVectSet::operator\|= (const Box & *b*)

make this set the union of itself and the input. b MUST be within the domain of of the DenseIntVectSet or this will throw an error

7.48.3.28 DenseIntVectSet& DenseIntVectSet::operator\|= (const IntVect & *intvect*)

make this set the union of itself and the input. intvect MUST be within the domain of of the DenseIntVectSet or this will throw an error

7.48.3.29 void DenseIntVectSet::recalcMinBox () const

7.48.3.30 void DenseIntVectSet::refine (int *iref*)

refine the set by the input

7.48.3.31 void DenseIntVectSet::shift (const IntVect & *iv*) `[inline]`

Shift every cell in the set by the input. Fast.

Shift every cell in the set by the input.

7.48.4 Friends And Related Function Documentation

7.48.4.1 friend class DenseIntVectSetIterator `[friend]`

The documentation for this class was generated from the following file:

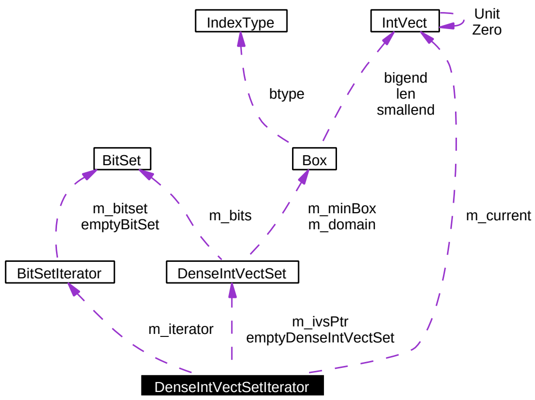
- **DenseIntVectSet.H**

7.49 DenseIntVectSetIterator Class Reference

DenseIntVectSetIterator iterates over all the 'true' members of a **DenseIntVectSet** set.

`#include <DenseIntVectSet.H>`

Collaboration diagram for DenseIntVectSetIterator:



Public Member Functions

- **DenseIntVectSetIterator** ()
- **DenseIntVectSetIterator** (const **DenseIntVectSet** &ivs)
- void **define** (const **DenseIntVectSet** &ivs)
- const **IntVect** & **operator()** () const
- bool **ok** () const
- void **operator++** ()
- void **begin** ()
- void **end** ()

Static Public Attributes

- **DenseIntVectSet emptyDenseIntVectSet**

7.49.1 Detailed Description

DenseIntVectSetIterator iterates over all the 'true' members of a **DenseIntVectSet** set.

This class is used by **IVSIterator** to implement it's iterator when **IntVectSet** is stored as a **DenseIntVectSet**

7.49.2 Constructor & Destructor Documentation

7.49.2.1 DenseIntVectSetIterator::DenseIntVectSetIterator () [inline]

7.49.2.2 DenseIntVectSetIterator::DenseIntVectSetIterator (const DenseIntVectSet & *ivs*) [inline]

7.49.3 Member Function Documentation

7.49.3.1 void DenseIntVectSetIterator::begin ()

7.49.3.2 void DenseIntVectSetIterator::define (const DenseIntVectSet & *ivs*) [inline]

7.49.3.3 void DenseIntVectSetIterator::end () [inline]

7.49.3.4 bool DenseIntVectSetIterator::ok () const [inline]

7.49.3.5 const IntVect & DenseIntVectSetIterator::operator() () const [inline]

7.49.3.6 void DenseIntVectSetIterator::operator++ () [inline]

7.49.4 Member Data Documentation

7.49.4.1 DenseIntVectSet DenseIntVectSetIterator::emptyDenseIntVectSet [static]

The documentation for this class was generated from the following file:

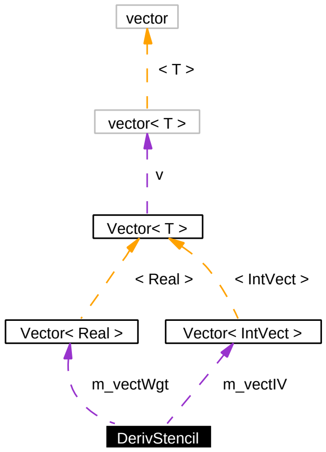
- **DenseIntVectSet.H**

7.50 DerivStencil Class Reference

class to encapsulate the operations to create derivs on irreg stencils

`#include <DerivStencil.H>`

Collaboration diagram for DerivStencil:



Public Member Functions

- void **define** ()
- bool **isDefined** () const
- **DerivStencil** ()
- **DerivStencil** (const **DerivStencil** &a_dsin)
- ~**DerivStencil** ()
- void **clear** ()
- int **size** () const
- const **IntVect** & **getIndex** (int a_ivec) const
- const **Real** & **getWeight** (int a_ivec) const
- void **accumulate** (const **IntVect** &a_iv, **Real** a_weight)
- const **DerivStencil** & **operator=** (const **DerivStencil** &a_dsin)
- const **DerivStencil** & **operator *=** (**Real** a_facin)
- const **DerivStencil** & **operator/=** (**Real** a_denom)
- const **DerivStencil** & **operator+=** (**Real** a_facin)
- const **DerivStencil** & **operator-=** (**Real** a_facin)

Protected Attributes

- **Vector**< **IntVect** > **m_vectIV**
- **Vector**< **Real** > **m_vectWgt**
- bool **isdefined**

7.50.1 Detailed Description

class to encapsulate the operations to create derivs on irreg stencils

DerivStencil is meant to be used to encapsulate the information necessary to take finite difference derivatives at a point in space. Every point in the stencil has a weight. You add them (the boxarrayindex and the weight) at the same time and you can manipulate the weights enmasse by real number operations. Stencils may not interact with each other with the same sort of arithmetic because that would bring up issues as to what to do when there is incomplete intersection between the stencils.

7.50.2 Constructor & Destructor Documentation

7.50.2.1 DerivStencil::DerivStencil ()

default constructor; creates empty vectors

7.50.2.2 DerivStencil::DerivStencil (const DerivStencil & *a_dsin*)

copy constructor; sets *this = a_dsin

7.50.2.3 DerivStencil::~DerivStencil ()

7.50.3 Member Function Documentation

7.50.3.1 void DerivStencil::accumulate (const IntVect & *a_iv*, Real *a_weight*)

add another set if the **IntVect** is not in the stencil already. \ Add the weight to the existing weight otherwise**

7.50.3.2 void DerivStencil::clear ()

make derivstencil empty

7.50.3.3 void DerivStencil::define ()

7.50.3.4 const IntVect& DerivStencil::getIndex (int *a_ivec*) const

get iv at ivec

7.50.3.5 const Real& DerivStencil::getWeight (int *a_ivec*) const

get weight at ivec

7.50.3.6 bool DerivStencil::isDefined () const `[inline]`

return true if any define function been called.

7.50.3.7 const DerivStencil& DerivStencil::operator *= (Real *a_facin*)

Multiply each weight by a_facin does nothing if vectors are of zero length

7.50.3.8 const DerivStencil& DerivStencil::operator+= (Real *a_facin*)

add a_facin to each weight does nothing if vectors are of zero length

7.50.3.9 const DerivStencil& DerivStencil::operator-= (Real *a_facin*)

subtract a_facin from each weight does nothing if vectors are of zero length

7.50.3.10 const DerivStencil& DerivStencil::operator/= (Real *a_denom*)

divide each weight by a_denom does nothing if vectors are of zero length. Be advised — This function does no checking to see if a_denom is close to zero.

7.50.3.11 const DerivStencil& DerivStencil::operator= (const DerivStencil & *a_dsin*)

assignment operator

7.50.3.12 int DerivStencil::size () const

return length of vectors

7.50.4 Member Data Documentation

7.50.4.1 bool DerivStencil::isdefined `[protected]`

7.50.4.2 Vector<IntVect> DerivStencil::m_vectIV `[protected]`

7.50.4.3 Vector<Real> DerivStencil::m_vectWgt `[protected]`

The documentation for this class was generated from the following file:

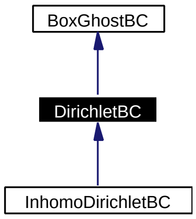
- **DerivStencil.H**

7.51 DirichletBC Class Reference

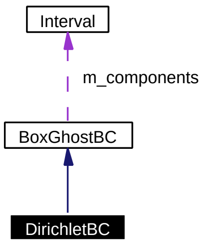
Class to enforce homogeneous Dirichlet boundary conditions.

`#include <PoissonBC.H>`

Inheritance diagram for DirichletBC:



Collaboration diagram for DirichletBC:



Public Member Functions

- **DirichletBC** ()

  Null constructor.

- **DirichletBC** (int dir, **Side::LoHiSide** sd)

  Simple constructor (sets **Interval** *to (0,0).*

- **DirichletBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)

  Full constructor.

- ~**DirichletBC** ()

  Destructor.

Friends

- class **DomainGhostBC**

7.51.1 Detailed Description

Class to enforce homogeneous Dirichlet boundary conditions.

BoxGhostBC class implentation to enforce homogeneous dirchelet boundary conditions Recall that **BoxGhostBC** is a class to encapsulate the operations of ghost-cell boundary conditions at a face. If the solution is phi and the face normal direction is x, the boundary conditions usually used can be expressed as A*phi + B*dphi/dx = C. For homogeneous Dirichlet BCs, B=C=0, A= 1 These functions are only meant to be called by **DomainGhostBC**

7.51.2 Constructor & Destructor Documentation

7.51.2.1 DirichletBC::DirichletBC ()

Null constructor.

7.51.2.2 DirichletBC::DirichletBC (int *dir*, Side::LoHiSide *sd*)

Simple constructor (sets **Interval** to (0,0).

7.51.2.3 DirichletBC::DirichletBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

Full constructor.

7.51.2.4 DirichletBC::~DirichletBC ()

Destructor.

7.51.3 Friends And Related Function Documentation

7.51.3.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 142).

Reimplemented in **InhomoDirichletBC** (p. 308).

The documentation for this class was generated from the following file:

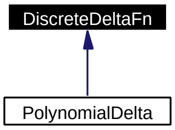
- **PoissonBC.H**

7.52 DiscreteDeltaFn Class Reference

Pure virtual base class to encapsulate discrete delta function operator.

`#include <DiscreteDeltaFn.H>`

Inheritance diagram for DiscreteDeltaFn:



Public Member Functions

- **DiscreteDeltaFn** ()

  constructor

- **DiscreteDeltaFn** (**Real** a_epsilon)

  constructor with radius – calls corresponding define fn.

- virtual ~**DiscreteDeltaFn** ()

  destructor

- virtual void **define** (**Real** a_epsilon)=0

  define function

- virtual **Real epsilon** () const=0

  returns epsilon

- virtual **DiscreteDeltaFn** * **clone** () const=0

  make a copy of this delta function

- virtual **Real evaluateDelta** (**Real** a_radius) const=0

  evaluate the delta function at the given radius

- virtual **Real integralDelta** (**Real** a_radius) const=0

  returns the integral from 0 to radius of delta

- virtual **Real computeK** (**RealVect** &a_radius, int a_idir, int a_jdir) const=0

  returns the kernel_{ij} at radius r

- virtual void **sumForce** (**FArrayBox** &a_sum, const **RealVect** &a_force, const **Box** &a_box, const **RealVect** &a_position, const **RealVect** &a_origin, **Real** a_dx)=0

  returns sum_j force_j kernel_ij over a box

- virtual void **sumForce** (**FArrayBox** &a_sum, const **RealVect** &a_force, const **Box** &a_box, int a_dir, int a_destComp, const **RealVect** &a_position, const **RealVect** &a_origin, **Real** a_dx)=0 *returns sum_j force_j kernel_ij over a box*

- virtual **Real computeLapDelta** (**Real** a_radius) const=0

  returns laplacian(delta) at radius r

7.52.1 Detailed Description

Pure virtual base class to encapsulate discrete delta function operator.

This is a pure virtual base class to encapsulate the discrete delta function which will be used to spread particle forces to the computational mesh. The parameter epsilon defines the spreading radius of the discrete delta fn.

7.52.2 Constructor & Destructor Documentation

7.52.2.1 DiscreteDeltaFn::DiscreteDeltaFn () `[inline]`

constructor

7.52.2.2 DiscreteDeltaFn::DiscreteDeltaFn (Real *a_epsilon*) `[inline]`

constructor with radius – calls corresponding define fn.

7.52.2.3 virtual DiscreteDeltaFn::~DiscreteDeltaFn () `[inline, virtual]`

destructor

7.52.3 Member Function Documentation

7.52.3.1 virtual DiscreteDeltaFn* DiscreteDeltaFn::clone () const `[pure virtual]`

make a copy of this delta function

Implemented in **PolynomialDelta** (p. 511).

7.52.3.2 virtual Real DiscreteDeltaFn::computeK (RealVect & *a_radius*, int *a_idir*, int *a_jdir*) const `[pure virtual]`

returns the kernel_{ij} at radius r

Implemented in **PolynomialDelta** (p. 511).

7.52.3.3 virtual Real DiscreteDeltaFn::computeLapDelta (Real *a_radius*) const `[pure virtual]`

returns laplacian(delta) at radius r

Implemented in **PolynomialDelta** (p. 511).

7.52.3.4 virtual void DiscreteDeltaFn::define (Real *a_epsilon*) [pure virtual]

define function

Implemented in **PolynomialDelta** (p. 511).

7.52.3.5 virtual Real DiscreteDeltaFn::epsilon () const [pure virtual]

returns epsilon

Implemented in **PolynomialDelta** (p. 511).

7.52.3.6 virtual Real DiscreteDeltaFn::evaluateDelta (Real *a_radius*) const [pure virtual]

evaluate the delta function at the given radius

Implemented in **PolynomialDelta** (p. 512).

7.52.3.7 virtual Real DiscreteDeltaFn::integralDelta (Real *a_radius*) const [pure virtual]

returns the integral from 0 to radius of delta

Implemented in **PolynomialDelta** (p. 512).

7.52.3.8 virtual void DiscreteDeltaFn::sumForce (FArrayBox & *a_sum*, const RealVect & *a_force*, const Box & *a_box*, int *a_dir*, int *a_destComp*, const RealVect & *a_position*, const RealVect & *a_origin*, Real *a_dx*) [pure virtual]

returns sum_j force_j kernel_ij over a box

this version computes a single (idir) component and places it in the destComp component of a_sum. this is done over a box for effeciency.

Implemented in **PolynomialDelta** (p. 512).

7.52.3.9 virtual void DiscreteDeltaFn::sumForce (FArrayBox & *a_sum*, const RealVect & *a_force*, const Box & *a_box*, const RealVect & *a_position*, const RealVect & *a_origin*, Real *a_dx*) [pure virtual]

returns sum_j force_j kernel_ij over a box

sum is a SpaceDim-component fab which will contain the sum this is done over a box for effeciency.

Implemented in **PolynomialDelta** (p. 512).

The documentation for this class was generated from the following file:

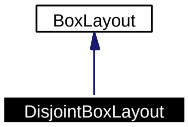
- **DiscreteDeltaFn.H**

7.53 DisjointBoxLayout Class Reference

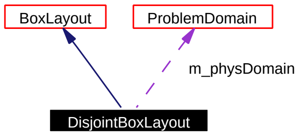
A **BoxLayout** that has a concept of disjointedness.

```
#include <DisjointBoxLayout.H>
```

Inheritance diagram for DisjointBoxLayout:



Collaboration diagram for DisjointBoxLayout:



Modification functions

- virtual void **close** ()
- virtual void **deepCopy** (const **DisjointBoxLayout** &a_source)
- virtual void **deepCopy** (const **BoxLayout** &a_source)
- virtual void **deepCopy** (const **BoxLayout** &a_source, const **ProblemDomain** &a_physDomain)
- void **coarsen** (**DisjointBoxLayout** &output, const **DisjointBoxLayout** &input, int refinement)
- void **refine** (**DisjointBoxLayout** &output, const **DisjointBoxLayout** &input, int refinement)

Public Member Functions

Constructors, destructors, defines

- **DisjointBoxLayout** ()
- **DisjointBoxLayout** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs)
- **DisjointBoxLayout** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs, const **ProblemDomain** &a_physDomain)
- virtual **~DisjointBoxLayout** ()
- virtual void **define** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs)
- void **define** (**BoxLayout** &a_layout)
- void **define** (**BoxLayout** &a_layout, const **ProblemDomain** &a_physDomain)
- virtual void **define** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs, const **ProblemDomain** &a_physDomain)
- virtual void **define_pd** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs, const **ProblemDomain** &a_physDomain)

Checks

- bool **isDisjoint** () const
- bool **checkPeriodic** (const **ProblemDomain** &a_domain) const
- bool **checkDomains** (const **DisjointBoxLayout** &a_dbl) const

Protected Member Functions

- const **ProblemDomain** & **physDomain** () const

Friends

- class **Copier**

Neighbor functions

- void **adjCellLo** (**DisjointBoxLayout** &a_output, const **DisjointBoxLayout** &a_input, int a_dir, int a_len=1)
- void **adjCellHi** (**DisjointBoxLayout** &a_output, const **DisjointBoxLayout** &a_input, int a_dir, int a_len=1)

7.53.1 Detailed Description

A **BoxLayout** that has a concept of disjointedness.

DisjointBoxLayout is-a **BoxLayout** that has a concept of disjointedness of the boxes. DisjointBoxLayout is only different from **BoxLayout** in that it has a method **isDisjoint()**. DisjointBoxLayouts also have a **ProblemDomain** member to ensure disjointness in the case of periodic boundary conditions, among other things.

7.53.2 Constructor & Destructor Documentation

7.53.2.1 DisjointBoxLayout::DisjointBoxLayout ()

Construct **BoxLayout** with no boxes.

7.53.2.2 DisjointBoxLayout::DisjointBoxLayout (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*)

Construct from a **Vector** of Boxes and optionally a **Vector** of processor assignments. If the processor assignment **Vector** is present, it must be either zero-length or the same length as the **Box Vector**. On exit, the DisjointBoxLayout will be closed.

7.53.2.3 DisjointBoxLayout::DisjointBoxLayout (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*, const ProblemDomain & *a_physDomain*)

Construct from a **Vector** of Boxes and optionally a **Vector** of processor assignments. If the processor assignment **Vector** is present, it must be either zero-length or the same length as the **Box Vector**. On exit, the DisjointBoxLayout will be closed.

7.53.2.4 virtual DisjointBoxLayout::~DisjointBoxLayout () [inline, virtual]

Ref-counted destruction. Once the last reference to the implementation of this class is destroyed, the data members are cleaned up.

7.53.3 Member Function Documentation

7.53.3.1 bool DisjointBoxLayout::checkDomains (const DisjointBoxLayout & *a_dbl*) const

Performs problem domain check when we don't have direct access to a problemDomain.

Returns `true` if this and *a_dbl* have compatible ProblemDomains.

7.53.3.2 bool DisjointBoxLayout::checkPeriodic (const ProblemDomain & *a_domain*) const

Checks to see that problem domains are compatible. To be compatible:

- *a_domain* and the **ProblemDomain** in this DisjointBoxLayout must have the same periodicity.
- The ProblemDomains must be periodic in the same dimensions.
- In all periodic directions, the period must be the same.

There are no checks in non-periodic directions, since DisjointBoxLayouts don't care about physical domains except in the periodic case.

7.53.3.3 virtual void DisjointBoxLayout::close () [virtual]

Mark this DisjointBoxLayout as complete and unchangeable.

Reimplemented from **BoxLayout** (p. 149).

7.53.3.4 virtual void DisjointBoxLayout::deepCopy (const BoxLayout & *a_source*, const ProblemDomain & *a_physDomain*) [virtual]

Create new open DisjointBoxLayout with copied data.

Checks that the source **BoxLayout** isDisjoint, throws an error if it is not. Otherwise, same as **BoxLayout::deepCopy**.

7.53.3.5 virtual void DisjointBoxLayout::deepCopy (const BoxLayout & *a_source*) [virtual]

Create new open DisjointBoxLayout with copied data.

Checks that the source **BoxLayout** isDisjoint, throws an error if it is not. Otherwise, same as **BoxLayout::deepCopy**.

Reimplemented from **BoxLayout** (p. 150).

7.53.3.6 virtual void DisjointBoxLayout::deepCopy (const DisjointBoxLayout & *a_source*) [virtual]

Create new open DisjointBoxLayout with copied data.

Doesn't check for disjointness, since *a_source* is already guaranteed to be disjoint. Also copies problem domain info.

7.53.3.7 virtual void DisjointBoxLayout::define (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*, const ProblemDomain & *a_physDomain*) [virtual]

Define this DisjointBoxLayout from a **Vector** of Boxes. This is a wrapper for a call to **BoxLayout::define**, required because **DisjointBoxLayout::define** is overloaded and the compiler will not look into the base class for name resolution.

7.53.3.8 void DisjointBoxLayout::define (BoxLayout & *a_layout*, const ProblemDomain & *a_physDomain*)

Shallow define. Only way to promote a **BoxLayout**. If **BoxLayout** has been closed, then this method checks isDisjoint and throws an error if not disjoint. If *a_layout* is disjoint, then this object becomes a closed DisjointBoxLayout object.

7.53.3.9 void DisjointBoxLayout::define (BoxLayout & *a_layout*)

Shallow define. Only way to promote a **BoxLayout**. If **BoxLayout** has been closed, then this method checks isDisjoint and throws an error if not disjoint. If *a_layout* is disjoint, then this object becomes a closed DisjointBoxLayout object.

7.53.3.10 virtual void DisjointBoxLayout::define (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*) [virtual]

Define this DisjointBoxLayout from a **Vector** of Boxes. This is a wrapper for a call to **BoxLayout::define**, required because **DisjointBoxLayout::define** is overloaded and the compiler will not look into the base class for name resolution.

Reimplemented from **BoxLayout** (p. 150).

7.53.3.11 virtual void DisjointBoxLayout::define_pd (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*, const ProblemDomain & *a_physDomain*) [inline, virtual]

Define this DisjointBoxLayout from a **Vector** of Boxes. Same as define.

7.53.3.12 bool DisjointBoxLayout::isDisjoint () const

Returns `true` if this object contains a disjoint union of Boxes.

The Disjoint testing algorithm assumes that the boxes are CELL-Centered boxes. We can expand the algorithm when someone needs it. bvs

7.53.3.13 const ProblemDomain& DisjointBoxLayout::physDomain () const [protected]

7.53.4 Friends And Related Function Documentation

7.53.4.1 void adjCellHi (DisjointBoxLayout & *a_output*, const DisjointBoxLayout & *a_input*, int *a_dir*, int *a_len* = 1) [friend]

Returns the set of boxes which result from calling **Box::adjCellHi** on each box in input. Output must be open, will be closed upon exiting this function.

7.53.4.2 void adjCellLo (DisjointBoxLayout & *a_output*, const DisjointBoxLayout & *a_input*, int *a_dir*, int *a_len* = 1) [friend]

Returns the set of boxes which you get from calling **Box::adjCellLo** on each box in input. Output must be open, is closed upon exit from this function.

7.53.4.3 void coarsen (DisjointBoxLayout & *output*, const DisjointBoxLayout & *input*, int *refinement*) [friend]

Coarsen a DisjointBoxLayout:

- *output* must be open
- *input* must be closed
- *refinement* must be a positive non-zero integer
- *output* and *input* do not share an implementation.

output is first deepCopy'ed from *input*, then coarsen(refinement) is called on each box of *output*.

output returns from this function closed.

LayoutIterators from *input* and *output* can perform interchangeably.

7.53.4.4 friend class Copier [friend]

7.53.4.5 void refine (DisjointBoxLayout & *output*, const DisjointBoxLayout & *input*, int *refinement*) [friend]

Refine a DisjointBoxLayout:

- *output* must be open
- *input* must be closed
- *refinement* must be a positive non-zero integer
- *output* and *input* do not share an implementation.

output is first deepCopy'ed from *input*, then refine(refinement) is called on each box of *output*.

output returns from this function closed.

LayoutIterators from *input* and *output* can perform interchangeably.

The documentation for this class was generated from the following file:

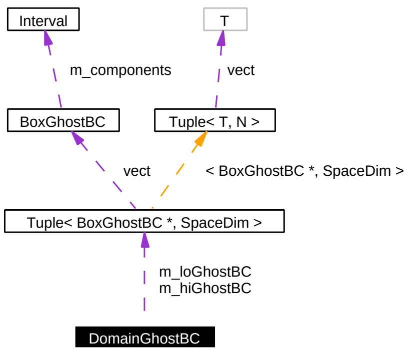
- **DisjointBoxLayout.H**

7.54 DomainGhostBC Class Reference

Class to enforce ghost-cell based boundary conditions.

`#include <GhostBC.H>`

Collaboration diagram for DomainGhostBC:



Public Member Functions

- **DomainGhostBC** ()

  Null constructor.

- **~DomainGhostBC** ()

  Destructor.

- void **setBoxGhostBC** (const **BoxGhostBC** &a_ghostBC)

  Set boundary conditions at a face to those defined in a_ghostBC.

- const **BoxGhostBC** & **operator()** (int a_direction, **Side::LoHiSide** a_side) const

  Access function – returns a reference to the BC on the given face.

- void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

  Deprecated function – use **ProblemDomain** *instead of a* **Box** *for the domain.*

- void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const

  apply boundary conditions on all sides of state which abut domain edge

- void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

  Deprecated function – use **ProblemDomain** *instead of a* **Box** *for the domain.*

- void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const

  Apply boundary conditions on all sides of state which abut the domain edge ∗.

- **DomainGhostBC** & **operator=** (const **DomainGhostBC** &a_dgbcin)

  Assignment operator.

- **DomainGhostBC** (const **DomainGhostBC** &a_dgbcin)

  Copy constructor.

Protected Member Functions

- bool **isBCDefined** (const int a_dir, const **Side::LoHiSide** a_side) const
- void **resetBoxGhostBC** (const int a_dir, const **Side::LoHiSide** a_side)

Protected Attributes

- **Tuple< BoxGhostBC ∗, SpaceDim > m_loGhostBC**
- **Tuple< BoxGhostBC ∗, SpaceDim > m_hiGhostBC**

7.54.1 Detailed Description

Class to enforce ghost-cell based boundary conditions.

This class holds a domain's worth of boxghostbcs, one for each face. The user adds a BoxGhost-BC-derived object for each face to enforce boundary conditions. If the solution is phi and the face normal direction is x, the boundary conditions at a face may often be expressed as A∗phi + B∗dphi/dx = C. The class derived from **BoxGhostBC** provides a function to to fill A B and C. DomainGhostBC calls this function and fills the ghost cells outside the domain appropriately.

7.54.2 Constructor & Destructor Documentation

7.54.2.1 DomainGhostBC::DomainGhostBC ()

Null constructor.

7.54.2.2 DomainGhostBC::∼DomainGhostBC ()

Destructor.

7.54.2.3 DomainGhostBC::DomainGhostBC (const DomainGhostBC & *a_dgbcin*)

Copy constructor.

7.54.3 Member Function Documentation

7.54.3.1 void DomainGhostBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const

apply boundary conditions on all sides of state which abut domain edge

7.54.3.2 void DomainGhostBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const

Deprecated function – use **ProblemDomain** instead of a **Box** for the domain.

7.54.3.3 void DomainGhostBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const

Apply boundary conditions on all sides of state which abut the domain edge ∗.

7.54.3.4 void DomainGhostBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const

Deprecated function – use **ProblemDomain** instead of a **Box** for the domain.

7.54.3.5 bool DomainGhostBC::isBCDefined (const int *a_dir*, const Side::LoHiSide *a_side*) const `[protected]`

Returns true if this GhostBC is defined

7.54.3.6 const BoxGhostBC& DomainGhostBC::operator() (int *a_direction*, Side::LoHiSide *a_side*) const

Access function – returns a reference to the BC on the given face.

7.54.3.7 DomainGhostBC& DomainGhostBC::operator= (const DomainGhostBC & *a_dgbcin*)

Assignment operator.

7.54.3.8 void DomainGhostBC::resetBoxGhostBC (const int *a_dir*, const Side::LoHiSide *a_side*) `[protected]`

Resets BC in the given direction and side to an unset state

7.54.3.9 void DomainGhostBC::setBoxGhostBC (const BoxGhostBC & *a_ghostBC*)

Set boundary conditions at a face to those defined in a_ghostBC.

7.54.4 Member Data Documentation

7.54.4.1 Tuple<BoxGhostBC∗, SpaceDim> DomainGhostBC::m_hiGhostBC `[protected]`

Contains the hi-side boundary conditions

7.54.4.2 Tuple<BoxGhostBC*, SpaceDim> DomainGhostBC::m_loGhostBC [protected]

Contains the lo-side boundary conditions

The documentation for this class was generated from the following file:

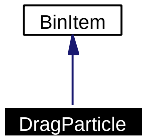
- **GhostBC.H**

7.55 DragParticle Class Reference

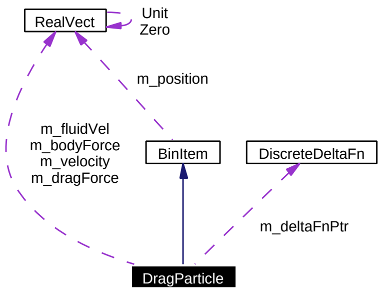
BinItem-derived object to encapsulate definition of particle in fluid.

`#include <DragParticle.H>`

Inheritance diagram for DragParticle:



Collaboration diagram for DragParticle:



Public Member Functions

- **DragParticle** ()

  default constructor

- **DragParticle** (const **RealVect** &a_position)

  partial constructor, to match the one in **BinItem**

- **DragParticle** (const **RealVect** &a_position, const **RealVect** &a_velocity, const **DiscreteDeltaFn** ∗a_deltaFnPtr)

  full constructor – calls matching define function

- **DragParticle** (const **DragParticle** &)

  copy constructor

- virtual **~DragParticle** ()

  destructor

- virtual void **define** (const **RealVect** &a_position)

  partial define function

- virtual void **define** (const **RealVect** &a_position, const **RealVect** &a_velocity, const **DiscreteDeltaFn** ∗a_deltaFnPtr) *full define function*

- virtual void **setVel** (const **RealVect** &a_vel)
  set velocity
- virtual void **setFluidVel** (const **RealVect** &a_vel)
  set local fluid velocity
- virtual void **setFluidVel** (**Real** a_vel, int a_comp)
  set local fluid velocity, single component
- virtual void **setDragForce** (const **RealVect** &a_force)
  set drag force
- virtual void **setBodyForce** (const **RealVect** &a_force)
  set force due to gravity (will be added to any computed drag force)
- virtual **RealVect velocity** () const
  get velocity
- virtual **RealVect totalForce** () const
  get total force on the particle
- virtual **RealVect dragForce** () const
  get drag force on the particle
- virtual **RealVect bodyForce** () const
  get constant body force
- virtual void **setDragCoeff** (const **Real** a_dragCoeff)
  set drag coefficient used to compute drag force
- virtual **Real mass** () const
  get mass of particle
- virtual void **setMass** (**Real** a_mass)
  set mass of particle
- virtual bool **boundaryMarker** () const
  has this particle recently hit the boundary?
- virtual void **setBoundaryMarker** (bool a_marker)
  set the boundary marker flag
- virtual **DragParticle** * **clone** () const
  create a clone of this particle
- virtual void **computeDragForce** (const **RealVect** &a_flowVelocity)
  compute force on particle from fluid

- virtual void **computeDragForce** ()

  compue force on particle from fluid, using stored fluid vel

- virtual **Real computeK** (const **RealVect** &a_x, int a_idir, int a_jdir) const

  compute K_{ij} at x for this particle

- virtual **Real computeProjForce** (const **RealVect** &a_x, int a_idir) const

  compute sum_j f_j K_{ij} for j=0,SpaceDim-1 (DRAG force on fluid)

- virtual void **computeProjForce** (**FArrayBox** &a_force, const **Box** &a_box, **Real** a_dx, **RealVect** &a_origin) const

  increment FAB with sum_j f_j K_{ij} for j=0,SpaceDim-1 (force on fluid)

- virtual void **computeProjForce** (**FArrayBox** &a_force, const **Box** &a_box, int a_idir, int a_destComp, **Real** a_dx, **RealVect** &a_origin) const

  increment FAB with sum_j f_j K_{ij} for j=0,SpaceDim-1 (force on fluid)

- virtual int **size** () const

  Number of bytes used by **linearIn()/linearOut***().*

- virtual int **linearSize** () const

  (just calls **size()** *function)*

- virtual void **linearOut** (void ∗a_buf) const

  Write a serialized (packed) representation into a_buf*.*

- virtual void **linearIn** (const void ∗const a_buf)

  Extract a serialized (packed) representation from a_buf*.*

- int **preAllocatable** ()
- **DragParticle** & **operator=** (const **DragParticle** &a_src)

  assignment operator (makes clone of delta function)

Protected Member Functions

- void **setDefaultValues** ()

  set default values for data members

Protected Attributes

- **RealVect m_velocity**

  velocity

- **RealVect m_fluidVel**

  local fluid velocity (stored here for convenience

- **RealVect m_dragForce**

drag force vector

- **RealVect m_bodyForce**

  body force vector

- **Real m_dragCoeff**

  drag coefficient

- **Real m_mass**

  particle mass

- bool **m_boundaryMarker**

  marker that we've interacted with the physical boundary

- **DiscreteDeltaFn * m_deltaFnPtr**

  local copy of discrete delta function

7.55.1 Detailed Description

BinItem-derived object to encapsulate definition of particle in fluid.

The DragParticle class encapsulates the definition of the particles used for suspensions of particles, and is derived from **BinItem** to allow it to interface with the **BinFab** infrastructure in Chombo.

The DragParticle also contains a **DiscreteDeltaFn** object to specify the spreading function which will be used for this particle.

7.55.2 Constructor & Destructor Documentation

7.55.2.1 DragParticle::DragParticle ()

default constructor

7.55.2.2 DragParticle::DragParticle (const RealVect & *a_position*)

partial constructor, to match the one in **BinItem**

Calls matching define function

7.55.2.3 DragParticle::DragParticle (const RealVect & *a_position*, const RealVect & *a_velocity*, const DiscreteDeltaFn * *a_deltaFnPtr*)

full constructor – calls matching define function

7.55.2.4 DragParticle::DragParticle (const DragParticle &)

copy constructor

7.55.2.5 virtual DragParticle::~DragParticle () [virtual]

destructor

7.55.3 Member Function Documentation

7.55.3.1 virtual RealVect DragParticle::bodyForce () const [inline, virtual]

get constant body force

7.55.3.2 virtual bool DragParticle::boundaryMarker () const [inline, virtual]

has this particle recently hit the boundary?

7.55.3.3 virtual DragParticle* DragParticle::clone () const [virtual]

create a clone of this particle

this may be a convenient way of creating a particle with the same properties (drag coeff, delta fn) as this one.

7.55.3.4 virtual void DragParticle::computeDragForce () [virtual]

compue force on particle from fluid, using stored fluid vel

7.55.3.5 virtual void DragParticle::computeDragForce (const RealVect & *a_flowVelocity*) [virtual]

compute force on particle from fluid

We assume a drag law for the force, where the drag force depends on the difference between the particle velocity and the local fluid velocity

7.55.3.6 virtual Real DragParticle::computeK (const RealVect & *a_x*, int *a_idir*, int *a_jdir*) const [virtual]

compute K_{ij} at x for this particle

Note that only the drag force is used in this computation, since body force does not act on the fluid.

7.55.3.7 virtual void DragParticle::computeProjForce (FArrayBox & *a_force*, const Box & *a_box*, int *a_idir*, int *a_destComp*, Real *a_dx*, RealVect & *a_origin*) const [virtual]

increment FAB with sum_j f_j K_{ij} for j=0,SpaceDim-1 (force on fluid)

Assumes force already computed and set. Note that this returns force exerted _by_ the particle on the surrounding fluid. Note that FAB is incremented, so you need to set to zero the first time this is called. This version computes only the idir component and puts it in the destComp component in the FAB

7.55.3.8 virtual void DragParticle::computeProjForce (FArrayBox & *a_force*, const Box & *a_box*, Real *a_dx*, RealVect & *a_origin*) const `[virtual]`

increment FAB with sum_j f_j K_{ij} for j=0,SpaceDim-1 (force on fluid)

Assumes drag force already computed and set. Note that this returns drag force exerted _by_ the particle on the surrounding fluid. Note that FAB is incremented, so you need to set to zero the first time this is called. Note that this is only the drag force, since the body force does not act on the fluid. This version computes all SpaceDim components

7.55.3.9 virtual Real DragParticle::computeProjForce (const RealVect & *a_x*, int *a_idir*) const `[virtual]`

compute sum_j f_j K_{ij} for j=0,SpaceDim-1 (DRAG force on fluid)

Assumes drag force already computed and set. Note that this returns force exerted _by_ the particle on the surrounding fluid. Note that this only takes drag force into account, since body force does not act on the fluid.

7.55.3.10 virtual void DragParticle::define (const RealVect & *a_position*, const RealVect & *a_velocity*, const DiscreteDeltaFn ∗ *a_deltaFnPtr*) `[virtual]`

full define function

7.55.3.11 virtual void DragParticle::define (const RealVect & *a_position*) `[virtual]`

partial define function

Reimplemented from **BinItem** (p. 105).

7.55.3.12 virtual RealVect DragParticle::dragForce () const `[inline, virtual]`

get drag force on the particle

7.55.3.13 virtual void DragParticle::linearIn (const void ∗const *a_buf*) `[virtual]`

Extract a serialized (packed) representation from *a_buf*.

Given a linear representation of the class data previously made using **linearIn()** in *a_buf*, set the data for this class.

7.55.3.14 virtual void DragParticle::linearOut (void ∗ *a_buf*) const `[virtual]`

Write a serialized (packed) representation into *a_buf*.

Write a linear representation of the internal data. Assumes that sufficient memory for the buffer has already been allocated by the caller.

Reimplemented from **BinItem** (p. 105).

7.55.3.15 virtual int DragParticle::linearSize () const [virtual]

(just calls **size()** function)

7.55.3.16 virtual Real DragParticle::mass () const [inline, virtual]

get mass of particle

7.55.3.17 DragParticle& DragParticle::operator= (const DragParticle & *a_src*)

assignment operator (makes clone of delta function)

7.55.3.18 int DragParticle::preAllocatable () [inline]

Reimplemented from **BinItem** (p. 105).

7.55.3.19 virtual void DragParticle::setBodyForce (const RealVect & *a_force*) [virtual]

set force due to gravity (will be added to any computed drag force)

bodyForce is a constant force which is added to any computed drag force for the total force on the particle

7.55.3.20 virtual void DragParticle::setBoundaryMarker (bool *a_marker*) [virtual]

set the boundary marker flag

7.55.3.21 void DragParticle::setDefaultValues () [protected]

set default values for data members

7.55.3.22 virtual void DragParticle::setDragCoeff (const Real *a_dragCoeff*) [virtual]

set drag coefficient used to compute drag force

7.55.3.23 virtual void DragParticle::setDragForce (const RealVect & *a_force*) [virtual]

set drag force

7.55.3.24 virtual void DragParticle::setFluidVel (Real *a_vel*, int *a_comp*) [virtual]

set local fluid velocity, single component

7.55.3.25 virtual void DragParticle::setFluidVel (const RealVect & *a_vel*) [virtual]

set local fluid velocity

7.55.3.26 virtual void DragParticle::setMass (Real *a_mass*) [virtual]

set mass of particle

7.55.3.27 virtual void DragParticle::setVel (const RealVect & *a_vel*) [virtual]

set velocity

7.55.3.28 virtual int DragParticle::size () const [virtual]

Number of bytes used by **linearIn()/linearOut**().

Returns the size, in number of bytes, of a flat linear representation of the data in this object.

Reimplemented from **BinItem** (p. 106).

7.55.3.29 virtual RealVect DragParticle::totalForce () const [inline, virtual]

get total force on the particle

7.55.3.30 virtual RealVect DragParticle::velocity () const [inline, virtual]

get velocity

7.55.4 Member Data Documentation

7.55.4.1 RealVect DragParticle::m_bodyForce [protected]

body force vector

7.55.4.2 bool DragParticle::m_boundaryMarker [protected]

marker that we've interacted with the physical boundary

7.55.4.3 DiscreteDeltaFn* DragParticle::m_deltaFnPtr [protected]

local copy of discrete delta function

7.55.4.4 Real DragParticle::m_dragCoeff [protected]

drag coefficient

7.55.4.5 RealVect DragParticle::m_dragForce [protected]

drag force vector

7.55.4.6 RealVect DragParticle::m_fluidVel [protected]

local fluid velocity (stored here for convenience

7.55.4.7 Real DragParticle::m_mass [protected]

particle mass

7.55.4.8 RealVect DragParticle::m_velocity [protected]

velocity

The documentation for this class was generated from the following file:

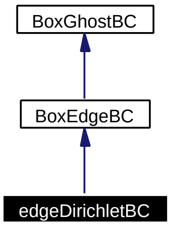
- **DragParticle.H**

7.56 edgeDirichletBC Class Reference

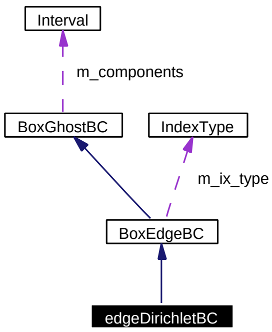
class to do Dirichlet edge-centered BC's

`#include <edgeGhostBC.H>`

Inheritance diagram for edgeDirichletBC:



Collaboration diagram for edgeDirichletBC:



Public Member Functions

- ~**edgeDirichletBC** ()
- **edgeDirichletBC** ()
- **edgeDirichletBC** (int a_dir, **Side::LoHiSide** a_sd)

    sets interval to be (0,0)

- **edgeDirichletBC** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps)
- void **setBCVal** (const **Real** a_bcVal)

    set edge-BC value (single-valued)

- **Real BCVal** () const

Protected Member Functions

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

deprecated interface

- virtual **BoxGhostBC** ∗ **new_boxghostbc** () const
  virtual constructor workaround

Protected Attributes

- **Real m_bcVal**

Friends

- class **DomainGhostBC**

7.56.1 Detailed Description

class to do Dirichlet edge-centered BC's

this class implements the edge-centered instantiation of the edge-centered homogeneous BC's

7.56.2 Constructor & Destructor Documentation

7.56.2.1 edgeDirichletBC::∼edgeDirichletBC ()

7.56.2.2 edgeDirichletBC::edgeDirichletBC ()

7.56.2.3 edgeDirichletBC::edgeDirichletBC (int *a_dir*, Side::LoHiSide *a_sd*)

sets interval to be (0,0)

7.56.2.4 edgeDirichletBC::edgeDirichletBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*)

7.56.3 Member Function Documentation

7.56.3.1 Real edgeDirichletBC::BCVal () const

7.56.3.2 virtual void edgeDirichletBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const `[protected, virtual]`

deprecated interface

Reimplemented from **BoxEdgeBC** (p. 132).

7.56.3.3 virtual void edgeDirichletBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const `[protected, virtual]`

Pure-virtual Worker function to fill arrays which define the boundary condition. Essentially, if the boundary condition is represented as A∗phi + B∗dphi/dn = C, then a_dircfac is A, a_neumfac is

B, and a_inhmval is C.

Reimplemented from **BoxEdgeBC** (p. 132).

7.56.3.4 virtual BoxGhostBC* edgeDirichletBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround

Reimplemented from **BoxEdgeBC** (p. 132).

7.56.3.5 void edgeDirichletBC::setBCVal (const Real *a_bcVal*)

set edge-BC value (single-valued)

7.56.4 Friends And Related Function Documentation

7.56.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 142).

7.56.5 Member Data Documentation

7.56.5.1 Real edgeDirichletBC::m_bcVal [protected]

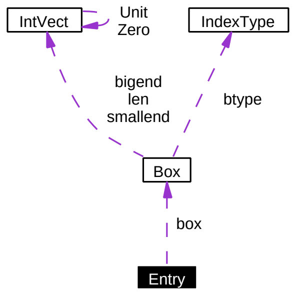
The documentation for this class was generated from the following file:

- **edgeGhostBC.H**

7.57 Entry Struct Reference

`#include <BoxLayout.H>`

Collaboration diagram for Entry:



Public Member Functions

- **Entry** ()
- **Entry** (const **Box** &a_box)
- **Entry** (const **Box** &a_box, const unsigned int a_index)
- bool **operator<** (const **Entry** &rhs) const

Public Attributes

- **Box box**
- unsigned int **index**
- unsigned int **m_procID**

7.57.1 Constructor & Destructor Documentation

7.57.1.1 Entry::Entry () `[inline]`

7.57.1.2 Entry::Entry (const Box & *a_box*) `[inline]`

7.57.1.3 Entry::Entry (const Box & *a_box*, const unsigned int *a_index*) `[inline]`

7.57.2 Member Function Documentation

7.57.2.1 bool Entry::operator< (const Entry & *rhs*) const `[inline]`

7.57.3 Member Data Documentation

7.57.3.1 Box Entry::box

7.57.3.2 unsigned int Entry::index

7.57.3.3 unsigned int Entry::m_procID

The documentation for this struct was generated from the following file:

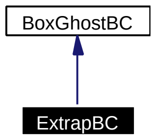
- **BoxLayout.H**

7.58 ExtrapBC Class Reference

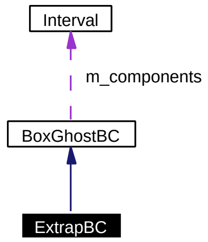
Class to enforce homogeneous neumann boundary conditions.

`#include <ExtrapBC.H>`

Inheritance diagram for ExtrapBC:



Collaboration diagram for ExtrapBC:



Public Member Functions

- **~ExtrapBC** ()
- **ExtrapBC** ()
- **ExtrapBC** (int dir, **Side::LoHiSide** sd)
- **ExtrapBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)

Protected Member Functions

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const
- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyExtrapBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

    deprecated interface

- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

    deprecated interface

- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const
  deprecated interface
- virtual void **applyExtrapBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const
  deprecated interface
- **BoxGhostBC** * **new_boxghostbc** () const

Friends

- class **DomainGhostBC**

7.58.1 Detailed Description

Class to enforce homogeneous neumann boundary conditions.

BoxGhostBC class implentation to enforce homogeneous neumann boundary conditions Recall that **BoxGhostBC** is a class to encapsulate the operations of ghost-cell boundary conditions at a face. If the solution is phi and the face normal direction is x, the boundary conditions usually used can be expressed as A*phi + B*dphi/dx = C. For homogeneous Neumann BCs, A=C=0, B= 1 These functions are only meant to be called by **DomainGhostBC**

7.58.2 Constructor & Destructor Documentation

7.58.2.1 ExtrapBC::~ExtrapBC ()

7.58.2.2 ExtrapBC::ExtrapBC ()

7.58.2.3 ExtrapBC::ExtrapBC (int *dir*, Side::LoHiSide *sd*)

7.58.2.4 ExtrapBC::ExtrapBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.58.3 Member Function Documentation

7.58.3.1 virtual void ExtrapBC::applyExtrapBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

7.58.3.2 virtual void ExtrapBC::applyExtrapBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

7.58.3.3 virtual void ExtrapBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

Reimplemented from **BoxGhostBC** (p. 140).

7.58.3.4 virtual void ExtrapBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.58.3.5 virtual void ExtrapBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const [protected, virtual]

deprecated interface

Reimplemented from **BoxGhostBC** (p. 141).

7.58.3.6 virtual void ExtrapBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.58.3.7 virtual void ExtrapBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const [protected, virtual]

deprecated interface

Implements **BoxGhostBC** (p. 141).

7.58.3.8 virtual void ExtrapBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const [protected, virtual]

Pure-virtual Worker function to fill arrays which define the boundary condition. Essentially, if the boundary condition is represented as A*phi + B*dphi/dn = C, then a_dircfac is A, a_neumfac is B, and a_inhmval is C.

Implements **BoxGhostBC** (p. 141).

7.58.3.9 BoxGhostBC* ExtrapBC::new_boxghostbc () const [protected, virtual]

Virtual constructor workaround.

Implements **BoxGhostBC** (p. 141).

7.58.4 Friends And Related Function Documentation

7.58.4.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 142).

The documentation for this class was generated from the following file:

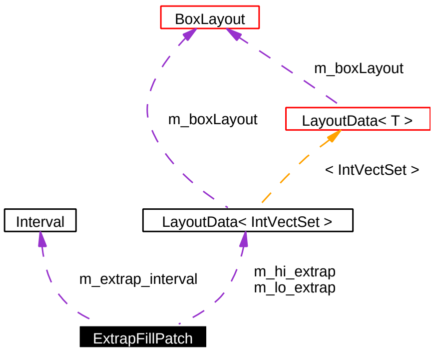
- **ExtrapBC.H**

7.59 ExtrapFillPatch Class Reference

fills outer ghost cells by copying from inner ghost cells.

`#include <ExtrapFillPatch.H>`

Collaboration diagram for ExtrapFillPatch:



Public Member Functions

- **ExtrapFillPatch** ()
- ~**ExtrapFillPatch** ()
- **ExtrapFillPatch** (const **DisjointBoxLayout** &a_level_domain, const **Box** &a_problem_domain, const **Interval** &a_extrap_interval)
- **ExtrapFillPatch** (const **DisjointBoxLayout** &a_level_domain, const **ProblemDomain** &a_problem_domain, const **Interval** &a_extrap_interval)
- void **define** (const **DisjointBoxLayout** &a_level_domain, const **Box** &a_problem_domain, const **Interval** &a_extrap_interval)
- void **define** (const **DisjointBoxLayout** &a_level_domain, const **ProblemDomain** &a_problem_domain, const **Interval** &a_extrap_interval)
- bool **isDefined** () const
- void **fillExtrap** (**LevelData**< **FArrayBox** > &a_data, int a_dir, int a_dest_comp, int a_num_comp)

Protected Attributes

- bool **m_is_defined**
- **LayoutData**< **IntVectSet** > **m_lo_extrap** [**SpaceDim**]
- **LayoutData**< **IntVectSet** > **m_hi_extrap** [**SpaceDim**]
- **Interval m_extrap_interval**

7.59.1 Detailed Description

fills outer ghost cells by copying from inner ghost cells.

The picture below shows existing ghost cell data in a border around the valid region (including corners). These values are extrapolated to the ghost cells in the outer "flaps." The coordinate direction of extrapolation is a parameter (in this example, it is 0).

```
                   +-----+----------------------------------+-----+
                   |     | +-----------------------------+ |     |
                   |     | |                             | |     |
                   |     | |                             | |     |
                   |     | |                             | |     |
                   |     | |    InteriorBox              | |     |
                   |     | |                             |%| x   |
                   |     | |                             | |/    |
                   |  x  |%|                             | /     |
                   | /   | |                             |/|     |
                   |/    | |                             / |     |
                   /     | |                 -----------/| |     |
                  /|     | |                /            | |     |
                 / |     | |               /             | |     |
                /  |     | |              /              | |     |
               /   |     | +-------------/---------------+x|     |
              /    +-----+--------------/----------------/-+-----+
             /                         /                /
            /                         /                /
           /                         /                /
          /                         /                /
  ------/-----------------------  /                /
  | Ghost cells in outer flaps  |/                /
  | (locations x) are filled by |                /
  | extrapolating from existing |   ------------/------------------------
  | ghost cells (location ).    |  | Border of exisiting ghost cells.  |
  |-----------------------------|   | Assumed to be filled correctly    |
                                    | on input.                         |
                                    |-----------------------------------|
```

Limitation on grid configuration

The picture below shows the ghost cells of grid 1 for extrapolation in direction 0. The cells marked X were filled before extrapolation. The cells marked > are extrapolated to from the first X to their right. The cells marked ? want to be filled from an X to the right, but there isn't any. This algorithm is not defined for such a grid configuration, in which the outline of the level has a "step" of width (in this case 2) less than the extrapolation radius (in this case 4). Using this algorithm with grids that violate this condition may result in undefined behavior.

```
+ - - - - - - - - - - - +
 $>$ $>$ $>$ X X X X X X $<$ $<$ $<$
|       +-------+       |
 $>$ $>$ $>$ X|       |X $<$ $<$ $<$
|       |       |       |
 $>$ $>$ $>$ X|       |X $<$ $<$ $<$
|       | grid 1|       |
 $>$ $>$ $>$ X|       |X $<$ $<$ $<$
|       |       |       |
 $>$ $>$ $>$ X|       |X $<$ $<$ $<$
|   +---+-------+       |
 ? ?|^ ^ ^ ^ ^|X $<$ $<$ $<$
+ - - - - - - - - - - - +
    |           |
    |           |
    |  grid 0   |
    |           |
    |           |
    +-----------+
```

7.59.2 Constructor & Destructor Documentation

7.59.2.1 ExtrapFillPatch::ExtrapFillPatch ()

Default constructor. User must subsequently call **define()**.

7.59.2.2 ExtrapFillPatch::~ExtrapFillPatch ()

Destructor.

7.59.2.3 ExtrapFillPatch::ExtrapFillPatch (const DisjointBoxLayout & *a_level_domain*, const Box & *a_problem_domain*, const Interval & *a_extrap_interval*)

Defining constructor. Constructs a valid ExtrapFillPatch object. Equivalent to default construction followed by **define()**. It is a fatal error if any grid in a_level_domain has any dimension less than a_extrap_interval.end().

There is also a restriction on grids with "jagged steps" which is described in "Limitation on grid configuration" in the class documentation above (with the ASCII art). It is not checked whether the input domain complies with this restriction. Using this algorithm with grids that violate this condition may result in undefined behavior.

{ Arguments:}\ a_level_domain (not modified): domain of the level. \ a_problem_domain (not modified): problem domain at this level. \ a_extrap_interval (not modified): the interval of ghost cells to fill by extrapolation. \

7.59.2.4 ExtrapFillPatch::ExtrapFillPatch (const DisjointBoxLayout & *a_level_domain*, const ProblemDomain & *a_problem_domain*, const Interval & *a_extrap_interval*)

Defining constructor. Constructs a valid ExtrapFillPatch object. Equivalent to default construction followed by **define()**. It is a fatal error if any grid in a_level_domain has any dimension less than a_extrap_interval.end().

There is also a restriction on grids with "jagged steps" which is described in "Limitation on grid configuration" in the class documentation above (with the ASCII art). It is not checked whether the input domain complies with this restriction. Using this algorithm with grids that violate this condition may result in undefined behavior.

{ Arguments:}\ a_level_domain (not modified): domain of the level. \ a_problem_domain (not modified): problem domain at this level. \ a_extrap_interval (not modified): the interval of ghost cells to fill by extrapolation. \

7.59.3 Member Function Documentation

7.59.3.1 void ExtrapFillPatch::define (const DisjointBoxLayout & *a_level_domain*, const ProblemDomain & *a_problem_domain*, const Interval & *a_extrap_interval*)

Defines this ExtrapFillPatch. Existing definition is overridden. The user may call **define()** once and call fillInterp() multiple times with different valid data sets.

It is a fatal error if any grid in a_level_domain has any dimension less than a_extrap_interval.end().

There is also a restriction on grids with "jagged steps" which is described in "Limitation on grid configuration" in the class documentation above (with the ASCII art). It is not checked whether the input domain complies with this restriction. Using this algorithm with grids that violate this condition may result in undefined behavior.

{ Arguments:}\ a_level_domain (not modified): domain of the level. \ a_problem_domain (not modified): problem domain at this level. \ a_extrap_interval (not modified): the interval of ghost cells to fill by extrapolation. \

{ This:}\ —This object is modified.—

7.59.3.2 void ExtrapFillPatch::define (const DisjointBoxLayout & *a_level_domain*, const Box & *a_problem_domain*, const Interval & *a_extrap_interval*)

Defines this ExtrapFillPatch. Existing definition is overridden. The user may call **define()** once and call fillInterp() multiple times with different valid data sets.

It is a fatal error if any grid in a_level_domain has any dimension less than a_extrap_interval.end().

There is also a restriction on grids with "jagged steps" which is described in "Limitation on grid configuration" in the class documentation above (with the ASCII art). It is not checked whether the input domain complies with this restriction. Using this algorithm with grids that violate this condition may result in undefined behavior.

{ Arguments:}\ a_level_domain (not modified): domain of the level. \ a_problem_domain (not modified): problem domain at this level. \ a_extrap_interval (not modified): the interval of ghost cells to fill by extrapolation. \

{ This:}\ —This object is modified.—

7.59.3.3 void ExtrapFillPatch::fillExtrap (LevelData< FArrayBox > & *a_data*, int *a_dir*, int *a_dest_comp*, int *a_num_comp*)

Fills ghost cells in the outer flaps, by interpolation from data in the inner ghost cells. It is an error to call if not this->**isDefined()**.\

{ Arguments:}\ a_data (modified): data at this level, both the source and destination for extrapolation. \ a_dir (not modified): zero-based coordinate direction of extrapolation. \ a_dest_comp (not modified): the first component for extrapolation. \ a_num_comp (not modified): the number of components to extrapolate. \

{ This:}\ This object is not modified.

7.59.3.4 bool ExtrapFillPatch::isDefined () const

Returns true if this object was created with the defining constructor or if **define()** has called.

{ This:}\ This object is not modified.

7.59.4 Member Data Documentation

7.59.4.1 Interval ExtrapFillPatch::m_extrap_interval [protected]

7.59.4.2 LayoutData<IntVectSet> ExtrapFillPatch::m_hi_extrap[SpaceDim] [protected]

7.59.4.3 bool ExtrapFillPatch::m_is_defined [protected]

7.59.4.4 LayoutData<IntVectSet> ExtrapFillPatch::m_lo_extrap[SpaceDim] [protected]

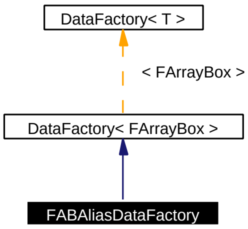
The documentation for this class was generated from the following file:

- **ExtrapFillPatch.H**

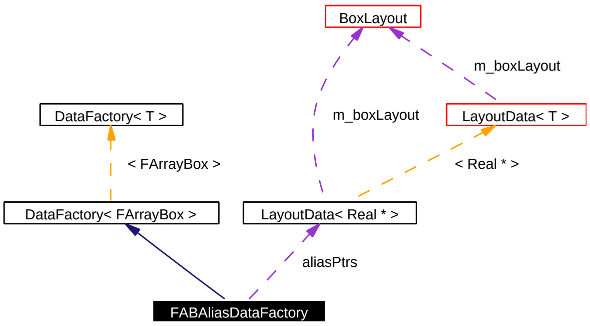
7.60 FABAliasDataFactory Class Reference

`#include <BoxLayoutData.H>`

Inheritance diagram for FABAliasDataFactory:



Collaboration diagram for FABAliasDataFactory:



Public Member Functions

- virtual ~**FABAliasDataFactory** ()
- **FABAliasDataFactory** (const **LayoutData**< **Real** ∗ > &aliases)
- void **define** (const **LayoutData**< **Real** ∗ > &aliases)
- virtual **FArrayBox** ∗ **create** (const **Box** &box, int ncomps, const **DataIndex** &a_datInd) const

  factory function. creates a new 'T' object using an aliased dataPtr for T

Protected Attributes

- **LayoutData**< **Real** ∗ > **aliasPtrs**

7.60.1 Constructor & Destructor Documentation

7.60.1.1 virtual FABAliasDataFactory::~FABAliasDataFactory () `[inline, virtual]`

7.60.1.2 FABAliasDataFactory::FABAliasDataFactory (const LayoutData< Real * > & *aliases*)

7.60.2 Member Function Documentation

7.60.2.1 virtual FArrayBox* FABAliasDataFactory::create (const Box & *box*, int *ncomps*, const DataIndex & *a_datInd*) const `[virtual]`

factory function. creates a new 'T' object using an aliased dataPtr for T

creates a new 'T' object and returns a pointer to it. Responsiblitly for calling operator 'delete' on this pointer is passed to the user.

Implements **DataFactory< FArrayBox >** (p. 206).

7.60.2.2 void FABAliasDataFactory::define (const LayoutData< Real * > & *aliases*)

7.60.3 Member Data Documentation

7.60.3.1 LayoutData<Real*> FABAliasDataFactory::aliasPtrs `[protected]`

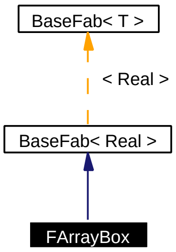
The documentation for this class was generated from the following file:

- **BoxLayoutData.H**

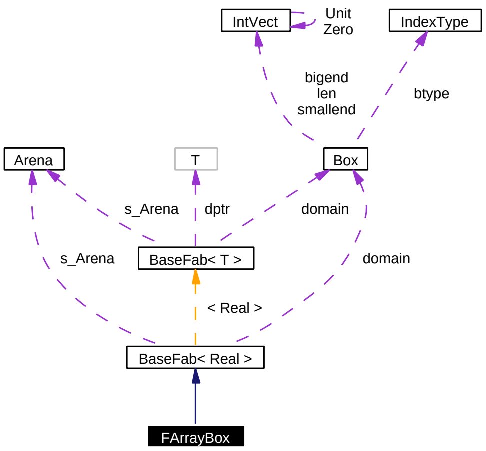
7.61 FArrayBox Class Reference

`#include <FArrayBox.H>`

Inheritance diagram for FArrayBox:



Collaboration diagram for FArrayBox:



Public Member Functions

- **FArrayBox** ()
- **FArrayBox** (const **Box** &b, int ncomp, **Real** ∗alias=NULL)
- void **define** (const **Box** &b, int ncomp, **Real** ∗alias=NULL)
- **FArrayBox** (const **Interval** &a_comps, **FArrayBox** &a_original)
- void **define** (const **Interval** &a_comps, **FArrayBox** &a_original)
- **FArrayBox** (std::istream &is)
- virtual **∼FArrayBox** ()
- virtual **Real norm** (int p=2, int comp=0, int numcomp=1) const
- virtual **Real norm** (const **Box** &subbox, int p=2, int comp=0, int numcomp=1) const
- virtual **Real sumPow** (const **Box** &subbox, int p=2, int comp=0, int numcomp=1) const
- **Real dotProduct** (const **FArrayBox** &fab2) const

    Return the dot product of this FArrayBox with another.

- **Real min** (int comp=0) const

- **Real min** (const **Box** &subbox, int comp=0) const
- **Real max** (int comp=0) const
- **Real max** (const **Box** &subbox, int comp=0) const
- **IntVect minIndex** (int comp=0) const
- **IntVect minIndex** (const **Box** &subbox, int comp=0) const
- **IntVect maxIndex** (int comp=0) const
- **IntVect maxIndex** (const **Box** &subbox, int comp=0) const
- int **maskLT** (**BaseFab**< int > &mask, **Real** val, int comp=0) const
- int **maskLE** (**BaseFab**< int > &mask, **Real** val, int comp=0) const
- int **maskEQ** (**BaseFab**< int > &mask, **Real** val, int comp=0) const
- int **maskGT** (**BaseFab**< int > &mask, **Real** val, int comp=0) const
- int **maskGE** (**BaseFab**< int > &mask, **Real** val, int comp=0) const
- void **abs** ()
- void **abs** (int comp, int numcomp=1)
- void **abs** (const **Box** &subbox, int comp=0, int numcomp=1)
- **Real sum** (int comp, int numcomp=1) const
- **Real sum** (const **Box** &subbox, int comp, int numcomp=1) const
- **FArrayBox** & **invert** (**Real** v)
- **FArrayBox** & **invert** (**Real** v, const **Box** &subbox, int comp=0, int numcomp=1)
- **FArrayBox** & **invert** (**Real** v, int comp, int numcomp=1)
- **FArrayBox** & **negate** (const **Box** &subbox, int comp=0, int numcomp=1)
- **FArrayBox** & **negate** (int comp, int numcomp=1)
- **FArrayBox** & **negate** ()
- **FArrayBox** & **plus** (**Real** r, const **Box** &b, int comp=0, int numcomp=1)
- **FArrayBox** & **plus** (**Real** r, int comp, int numcomp=1)
- **FArrayBox** & **operator+=** (**Real** r)
- **FArrayBox** & **operator+=** (const **FArrayBox** &f)
- **FArrayBox** & **plus** (const **FArrayBox** &src)
- **FArrayBox** & **plus** (**Real** r)
- **FArrayBox** & **plus_real** (**Real** r)
- **FArrayBox** & **plus** (const **FArrayBox** &src, const **Real** &scale)
- **FArrayBox** & **plus** (const **FArrayBox** &src, const **Real** &scale, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **plus** (const **FArrayBox** &src, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **plus** (const **FArrayBox** &src, const **Box** &subbox, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **plus** (const **FArrayBox** &src, const **Box** &srcbox, const **Box** &destbox, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **plus** (const **FArrayBox** &src, const **Box** &srcbox, const **Box** &destbox, const **Real** &scale, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **operator-=** (**Real** r)
- **FArrayBox** & **operator-=** (const **FArrayBox** &f)
- **FArrayBox** & **minus** (const **FArrayBox** &src)
- **FArrayBox** & **minus** (const **FArrayBox** &src, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **minus** (const **FArrayBox** &src, const **Box** &subbox, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **minus** (const **FArrayBox** &src, const **Box** &srcbox, const **Box** &destbox, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **operator *=** (**Real** r)
- **FArrayBox** & **mult** (**Real** r)

- **FArrayBox** & **mult** (**Real** r, int comp, int numcomp=1)
- **FArrayBox** & **mult** (**Real** r, const **Box** &b, int comp=0, int numcomp=1)
- **FArrayBox** & **operator** *= (const **FArrayBox** &f)
- **FArrayBox** & **mult** (const **FArrayBox** &src)
- **FArrayBox** & **mult** (const **FArrayBox** &src, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **mult** (const **FArrayBox** &src, const **Box** &subbox, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **mult** (const **FArrayBox** &src, const **Box** &srcbox, const **Box** &destbox, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **operator/=** (**Real** r)
- **FArrayBox** & **divide** (**Real** r)
- **FArrayBox** & **divide** (**Real** r, int comp, int numcomp=1)
- **FArrayBox** & **divide** (**Real** r, const **Box** &b, int comp=0, int numcomp=1)
- **FArrayBox** & **operator/=** (const **FArrayBox** &src)
- **FArrayBox** & **divide** (const **FArrayBox** &src)
- **FArrayBox** & **divide** (const **FArrayBox** &src, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **divide** (const **FArrayBox** &src, const **Box** &subbox, int srccomp, int destcomp, int numcomp=1)
- **FArrayBox** & **divide** (const **FArrayBox** &src, const **Box** &srcbox, const **Box** &destbox, int srccomp, int destcomp, int numcomp=1)
- **Real get** (const **IntVect** &iv, int comp) const
- void **set** (const **IntVect** &iv, int comp, **Real** val)

Protected Member Functions

- virtual void **performCopy** (const **BaseFab**< **Real** > &src, const **Box** &srcbox, int srccomp, const **Box** &destbox, int destcomp, int numcomp)

7.61.1 Detailed Description

Fortran Array Boxes (generally called FABs) are objects constructed to interface with arrays in Fortran. Useful operations can be performed upon FABs in C++, and they provide a convenient interface to Fortran when it is necessary to retreat into that language for doing arithmetic operations when performance matters.

FArrayBox is derived from **BaseFab<Real>**. FArrayBox adds additional useful capabilities which make sense for Real types, such as I/O and L**p norms.

The C pre-processor macro 'CH' must be defined to use this class. The internal precision of FArrayBox objects is set by defining either 'CH' or 'CH'

This class does NOT provide a copy constructor or assignment operator.

7.61.2 Constructor & Destructor Documentation

7.61.2.1 FArrayBox::FArrayBox ()

Constructs an invalid FArrayBox with no memory.

7.61.2.2 FArrayBox::FArrayBox (const Box & *b*, int *ncomp*, Real ∗ *alias* = NULL)

: Constructs an initial FArrayBox with the data space allocated but not inititialized. ncomp is the number of components (variables) at each data point in the **Box**.

7.61.2.3 FArrayBox::FArrayBox (const Interval & *a_comps*, FArrayBox & *a_original*) [inline]

Construct an aliaed FArrayBox. See **BaseFab** class for details.

7.61.2.4 FArrayBox::FArrayBox (std::istream & *is*) [explicit]

Constructs FArrayBox by reading it from istream.

7.61.2.5 virtual FArrayBox::~FArrayBox () [virtual]

The (virtual) destructor.

7.61.3 Member Function Documentation

7.61.3.1 void FArrayBox::abs (const Box & *subbox*, int *comp* = 0, int *numcomp* = 1)

Modifies this FArrayBox by replacing eahc value with its absolute value, for components (comp: comp+numcomp-1) and within the subbox.

7.61.3.2 void FArrayBox::abs (int *comp*, int *numcomp* = 1)

Modifies this FArrayBox by replacing eahc value with its absolute value, for components (comp: comp+numcomp-1).

7.61.3.3 void FArrayBox::abs ()

Modifies this FArrayBox by replacing each value with its absolute value.

7.61.3.4 void FArrayBox::define (const Interval & *a_comps*, FArrayBox & *a_original*) [inline]

7.61.3.5 void FArrayBox::define (const Box & *b*, int *ncomp*, Real ∗ *alias* = NULL) [inline, virtual]

: Make **BaseFab** with desired domain and number of components. existing data is lost. Data is in uninialized state.

Reimplemented from **BaseFab< Real >** (p. 75).

7.61.3.6 FArrayBox& FArrayBox::divide (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise division by values in the argument FArrayBox. Divides this FArrayBox's components (destcomp:destcomp+numcomp-1) in the **Box** destbox by src's components (srccomp:srccomp+numcomp-1) in the **Box** srcbox. Corresponding locations within the two FArrayBoxes are indexed relative to srcbox and destbox, and will in general not be the same. The srcbox and destbox must be same size. The results are UNDEFINED if the src and dest FArrayBoxes are the same and the srcbox and destbox overlap.

7.61.3.7 FArrayBox& FArrayBox::divide (const FArrayBox & *src*, const Box & *subbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise division by values in the argument FArrayBox. Divides this FArrayBox's components (destcomp:destcomp+numcomp-1) by src's components (srccomp:srccomp+numcomp-1) where the domain of this FArrayBox intersects the subbox. NOTE: subbox must be contained in this FAB.

7.61.3.8 FArrayBox& FArrayBox::divide (const FArrayBox & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise division by values in the argument FArrayBox. Divides this FArrayBox's components (destcomp:destcomp+numcomp-1) by src's components (srccomp:srccomp+numcomp-1) where the domains of the two FArrayBoxes intersect.

7.61.3.9 FArrayBox& FArrayBox::divide (const FArrayBox & *src*)

Modifies this FArrayBox by pointwise division by the values in the argument FArrayBox. You might come to grief if the domains of the FArrayBoxes don't match. The same as the /= operator.

7.61.3.10 FArrayBox& FArrayBox::divide (Real *r*, const Box & *b*, int *comp* = 0, int *numcomp* = 1)

Modifies this FArrayBox by dividing all values by the scalar Real r. For given range of components and within given subbox.

7.61.3.11 FArrayBox& FArrayBox::divide (Real *r*, int *comp*, int *numcomp* = 1)

Modifies this FArrayBox by dividing all values by the scalar Real r. For given range of components.

7.61.3.12 FArrayBox& FArrayBox::divide (Real *r*)

Modifies this FArrayBox by dividing all values by the scalar Real r.

7.61.3.13 Real FArrayBox::dotProduct (const FArrayBox & *fab2*) const

Return the dot product of this FArrayBox with another.

Return the dot product of this FArrayBox and "fab2" over their common box and all components.

7.61.3.14 Real FArrayBox::get (const IntVect & *iv*, int *comp*) const [inline]

7.61.3.15 FArrayBox& FArrayBox::invert (Real *v*, int *comp*, int *numcomp* = 1)

Modifies this FArrayBox by replacing each value x with v/x. For given range of components.

7.61.3.16 FArrayBox& FArrayBox::invert (Real *v*, const Box & *subbox*, int *comp* = 0, int *numcomp* = 1)

Modifies this FArrayBox by replacing each value x with v/x. For given range of components and within given subbox.

7.61.3.17 FArrayBox& FArrayBox::invert (Real *v*)

Modifies this FArrayBox by replacing each value x with v/x.

7.61.3.18 int FArrayBox::maskEQ (BaseFab< int > & *mask*, Real *val*, int *comp* = 0) const

Computes mask array with value of 1 in cells where this FArrayBox has value equal to val, 0 otherwise. mask is resized by this function. The number of cells marked with 1 returned.

7.61.3.19 int FArrayBox::maskGE (BaseFab< int > & *mask*, Real *val*, int *comp* = 0) const

Computes mask array with value of 1 in cells where this FArrayBox has value greater than or equal to val, 0 otherwise. mask is resized by this function. The number of cells marked with 1 returned.

7.61.3.20 int FArrayBox::maskGT (BaseFab< int > & *mask*, Real *val*, int *comp* = 0) const

Computes mask array with value of 1 in cells where this FArrayBox has value greater than val, 0 otherwise. mask is resized by this function. The number of cells marked with 1 returned.

7.61.3.21 int FArrayBox::maskLE (BaseFab< int > & *mask*, Real *val*, int *comp* = 0) const

Computes mask array with value of 1 in cells where this FArrayBox has value less than or equal to val, 0 otherwise. mask is resized by this function. The number of cells marked with 1 returned.

7.61.3.22 int FArrayBox::maskLT (BaseFab< int > & *mask*, Real *val*, int *comp* = 0) const

Computes mask array with value of 1 in cells where this FArrayBox has value less than val, 0 otherwise. mask is resized by this function. The number of cells marked with 1 returned.

7.61.3.23 Real FArrayBox::max (const Box & *subbox*, int *comp* = 0) const

Returns the maximum value of given component of this FArrayBox in given subbox.

7.61.3.24 Real FArrayBox::max (int *comp* = 0) const

Returns the maximum value of given component of this FArrayBox.

7.61.3.25 IntVect FArrayBox::maxIndex (const Box & *subbox*, int *comp* = 0) const

: Returns location of maximum value in given component of this FArrayBox in given subbox.

7.61.3.26 IntVect FArrayBox::maxIndex (int *comp* = 0) const

Returns location of maximum value in given component of this FArrayBox.

7.61.3.27 Real FArrayBox::min (const Box & *subbox*, int *comp* = 0) const

Returns the minimum value of given component of this FArrayBox in given subbox.

7.61.3.28 Real FArrayBox::min (int *comp* = 0) const

Returns the minimum value of given component of this FArrayBox.

7.61.3.29 IntVect FArrayBox::minIndex (const Box & *subbox*, int *comp* = 0) const

: Returns location of minimum value in given component of this FArrayBox in given subbox.

7.61.3.30 IntVect FArrayBox::minIndex (int *comp* = 0) const

Finds location of minimum value in given component of this FArrayBox.

7.61.3.31 FArrayBox& FArrayBox::minus (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise subtraction of values in the argument FArrayBox. Subtracts src's components (srccomp:srccomp+numcomp-1) in the **Box** srcbox from this FArrayBox's components (destcomp:destcomp+numcomp-1) in the **Box** destbox. Corresponding locations within the two FArrayBoxes are indexed relative to srcbox and destbox, and will in general not be the same. The srcbox and destbox must be same size. The results are UNDEFINED if the src and dest FArrayBoxes are the same and the srcbox and destbox overlap.

7.61.3.32 FArrayBox& FArrayBox::minus (const FArrayBox & *src*, const Box & *subbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise subtraction of values in the argument FArrayBox. Subtracts src's components (srccomp:srccomp+numcomp-1) from this FArrayBox's components (destcomp:destcomp+numcomp-1) where the domain of this FArrayBox intersects the subbox. NOTE: subbox must be contained in this FAB.

7.61.3.33 FArrayBox& FArrayBox::minus (const FArrayBox & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise subtraction of values in the argument FArrayBox. Subtracts src's components (srccomp:srccomp+numcomp-1) from this FArrayBox's components (destcomp:destcomp+numcomp-1) where the domains of the two FArrayBoxes intersect.

7.61.3.34 FArrayBox& FArrayBox::minus (const FArrayBox & *src*)

Modifies this FArrayBox by pointwise subtraction of the values of the argument FArrayBox. You might come to grief if the domains of the FArrayBoxes don't match. The same as -= operator.

7.61.3.35 FArrayBox& FArrayBox::mult (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise multiplication by values in the argument FArrayBox. Multiplies src's components (srccomp:srccomp+numcomp-1) in the **Box** srcbox by this FArrayBox's components (destcomp:destcomp+numcomp-1) in the **Box** destbox. Corresponding locations within the two FArrayBoxes are indexed relative to srcbox and destbox, and will in general not be the same. The srcbox and destbox must be same size. The results are UNDEFINED if the src and dest FArrayBoxes are the same and the srcbox and destbox overlap.

7.61.3.36 FArrayBox& FArrayBox::mult (const FArrayBox & *src*, const Box & *subbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise multiplication by values in the argument FArrayBox. Multiplies src's components (srccomp:srccomp+numcomp-1) by this FArrayBox's components (destcomp:destcomp+numcomp-1) where the domain of this FArrayBox intersects the subbox. NOTE: subbox must be contained in this FAB.

7.61.3.37 FArrayBox& FArrayBox::mult (const FArrayBox & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise multiplication by values in the argument FArrayBox. Multiplies src's components (srccomp:srccomp+numcomp-1) by this FArrayBox's components (destcomp:destcomp+numcomp-1) where the domains of the two FArrayBoxes intersect.

7.61.3.38 FArrayBox& FArrayBox::mult (const FArrayBox & *src*)

Modifies this FArrayBox by pointwise multiplication by the values in the argument FArrayBox. You might come to grief if the domains of the FArrayBoxes don't match. The same as the *=

operator.

7.61.3.39 FArrayBox& FArrayBox::mult (Real *r*, const Box & *b*, int *comp* = 0, int *numcomp* = 1)

Modifies this FArrayBox by multiplying all values by the scalar Real r. For given range of components and within given subbox.

7.61.3.40 FArrayBox& FArrayBox::mult (Real *r*, int *comp*, int *numcomp* = 1)

Modifies this FArrayBox by multiplying all values by the scalar Real r. For given range of components.

7.61.3.41 FArrayBox& FArrayBox::mult (Real *r*)

Modifies this FArrayBox by multiplying all values by the scalar Real r.

7.61.3.42 FArrayBox& FArrayBox::negate ()

Modifies this FArrayBox by replacing each value with its additive inverse.

7.61.3.43 FArrayBox& FArrayBox::negate (int *comp*, int *numcomp* = 1)

Modifies this FArrayBox by replacing each value with its additive inverse. For given range of components.

7.61.3.44 FArrayBox& FArrayBox::negate (const Box & *subbox*, int *comp* = 0, int *numcomp* = 1)

Modifies this FArrayBox by replacing each value with its additive inverse. For given range of components and within given subbox.

7.61.3.45 virtual Real FArrayBox::norm (const Box & *subbox*, int *p* = 2, int *comp* = 0, int *numcomp* = 1) const `[virtual]`

: Returns the Lp-norm of this FAB using components (comp : comp+numcomp-1) and within the subbox. p < 0 -> ERROR. p = 0 -> infinity norm (max norm). p = 1 -> sum of ABS(FAB) p > 1 -> Lp-norm

7.61.3.46 virtual Real FArrayBox::norm (int *p* = 2, int *comp* = 0, int *numcomp* = 1) const `[virtual]`

: Returns the Lp-norm of this FAB using components (comp : comp+numcomp-1). p < 0 -> ERROR. p = 0 -> infinity norm (max norm). p = 1 -> sum of ABS(FAB) p > 1 -> Lp-norm

7.61.3.47 FArrayBox& FArrayBox::operator *= (const FArrayBox & *f*)

Modifies this FArrayBox by pointwise multiplication of the values by the argument FArrayBox. You might come to grief if the domains of the FArrayBoxes don't match.

7.61.3.48 FArrayBox& FArrayBox::operator *= (Real *r*)

Modifies this FArrayBox by multiplying all values by the scalar Real r.

7.61.3.49 FArrayBox& FArrayBox::operator+= (const FArrayBox & *f*)

Modifies this FArrayBox by pointwise addition of the values of the argument FArrayBox. You might come to grief if the domains of the FArrayBoxes don't match.

7.61.3.50 FArrayBox& FArrayBox::operator+= (Real *r*)

Modifies this FArrayBox by adding the scalar Real r to all values.

7.61.3.51 FArrayBox& FArrayBox::operator-= (const FArrayBox & *f*)

Modifies this FArrayBox by pointwise subtraction of the values of the argument FArrayBox. You might come to grief if the domains of the FArrayBoxes don't match.

7.61.3.52 FArrayBox& FArrayBox::operator-= (Real *r*)

Modifies this FArrayBox by subtracting the scalar Real r to all values. Note: use plus(-r) for more general operations.

7.61.3.53 FArrayBox& FArrayBox::operator/= (const FArrayBox & *src*)

Modifies this FArrayBox by pointwise division of the values by the argument FArrayBox. You might come to grief if the domains of the FArrayBoxes don't match.

7.61.3.54 FArrayBox& FArrayBox::operator/= (Real *r*)

Modifies this FArrayBox by dividing all values by the scalar Real r.

7.61.3.55 virtual void FArrayBox::performCopy (const BaseFab< Real > & *src*, const Box & *srcbox*, int *srccomp*, const Box & *destbox*, int *destcomp*, int *numcomp*) `[protected, virtual]`

Reimplemented from **BaseFab< Real >** (p. 77).

7.61.3.56 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, const Real & *scale*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

7.61.3.57 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise addition of values in the argument FArrayBox. Adds src's components (srccomp:srccomp+numcomp-1) in the **Box** srcbox to this FArrayBox's components (destcomp:destcomp+numcomp-1) in the **Box** destbox. Corresponding locations within the two FArrayBoxes are indexed relative to srcbox and destbox, and will in general not be the same. The srcbox and destbox must be same size. The results are UNDEFINED if the src and dest FArrayBoxes are the same and the srcbox and destbox overlap.

7.61.3.58 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, const Box & *subbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise addition of values in the argument FArrayBox. Adds src's components (srccomp:srccomp+numcomp-1) to this FArrayBox's components (destcomp:destcomp+numcomp-1) where the domain of this FArrayBox intersects the subbox. NOTE: subbox must be contained in this FAB.

7.61.3.59 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

: Modifies this FArrayBox by pointwise addition of values in the argument FArrayBox. Adds src's components (srccomp:srccomp+numcomp-1) to this FArrayBox's components (destcomp:destcomp+numcomp-1) where the domains of the two FArrayBoxes intersect.

7.61.3.60 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, const Real & *scale*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

Modifies this FArrayBox by pointwise scaled addition of the argument FArrayBox (a[i] <- a[i] + scale ∗ src[i]). Uses domain of the intersection of these two FArrayBoxes.

7.61.3.61 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, const Real & *scale*)

Modifies this FArrayBox by pointwise scaled addition of the argument FArrayBox (a[i] <- a[i] + scale ∗ src[i]). Uses domain of the intersection of these two FArrayBoxes.

7.61.3.62 FArrayBox& FArrayBox::plus (Real *r*)

Modifies this FArrayBox by adding the scalar Real r to all values.

7.61.3.63 FArrayBox& FArrayBox::plus (const FArrayBox & *src*)

Modifies this FArrayBox by pointwise addition of the values of the argument FArrayBox. You might come to grief if the domains of the FArrayBoxes don't match. The same as += operator.

7.61.3.64 FArrayBox& FArrayBox::plus (Real *r*, int *comp*, int *numcomp* = 1)

Modifies this FArrayBox by adding the scalar Real r to all values. For given range of components.

7.61.3.65 FArrayBox& FArrayBox::plus (Real *r*, const Box & *b*, int *comp* = 0, int *numcomp* = 1)

Modifies this FArrayBox by adding the scalar Real r to all values. For given range of components and within given subbox.

7.61.3.66 FArrayBox& FArrayBox::plus_real (Real *r*) `[inline]`

7.61.3.67 void FArrayBox::set (const IntVect & *iv*, int *comp*, Real *val*) `[inline]`

7.61.3.68 Real FArrayBox::sum (const Box & *subbox*, int *comp*, int *numcomp* = 1) const

: Returns sum of component of this FArrayBox in given subbox.

7.61.3.69 Real FArrayBox::sum (int *comp*, int *numcomp* = 1) const

Returns sum of given component of FArrayBox.

7.61.3.70 virtual Real FArrayBox::sumPow (const Box & *subbox*, int *p* = 2, int *comp* = 0, int *numcomp* = 1) const `[virtual]`

Returns sum of pow(fab[i,c],p): i in subbox, comp<=c<comp+numcomp p>=2 only

The documentation for this class was generated from the following file:

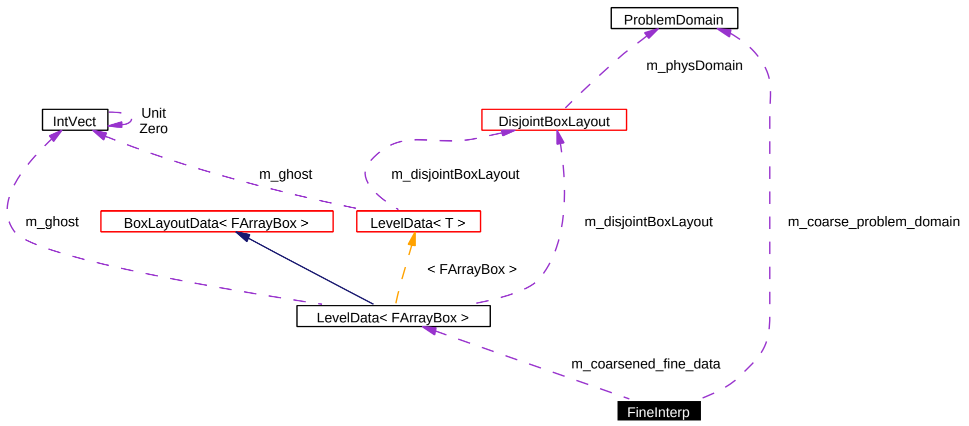
- **FArrayBox.H**

7.62 FineInterp Class Reference

replaces fine level data with interpolation of coarse level data.

`#include <FineInterp.H>`

Collaboration diagram for FineInterp:



Public Member Functions

- **FineInterp** ()
- ~**FineInterp** ()
- **FineInterp** (const **DisjointBoxLayout** &a_fine_domain, const int &a_numcomps, const int &a_ref_ratio, const **Box** &a_fine_problem_domain)
- **FineInterp** (const **DisjointBoxLayout** &a_fine_domain, const int &a_numcomps, const int &a_ref_ratio, const **ProblemDomain** &a_fine_problem_domain)
- void **define** (const **DisjointBoxLayout** &a_fine_domain, const int &a_numcomps, const int &a_ref_ratio, const **Box** &a_fine_problem_domain)
- void **define** (const **DisjointBoxLayout** &a_fine_domain, const int &a_numcomps, const int &a_ref_ratio, const **ProblemDomain** &a_fine_problem_domain)
- bool **isDefined** () const
- void **interpToFine** (**LevelData**< **FArrayBox** > &a_fine_data, const **LevelData**< **FArrayBox** > &a_coarse_data)

Protected Member Functions

- void **interpGridData** (**BaseFab**< **Real** > &a_fine, const **BaseFab**< **Real** > &a_coarse, const **Box** &a_coarsened_fine_box, int a_ref_ratio) const

Protected Attributes

- bool **is_defined**
- int **m_ref_ratio**
- **LevelData**< **FArrayBox** > **m_coarsened_fine_data**
- **ProblemDomain m_coarse_problem_domain**

7.62.1 Detailed Description

replaces fine level data with interpolation of coarse level data.

This class replaces data at a fine level of refinement with data interpolated from a coarser level of refinement. Interpolation is piecewise bi(tri)linear, with van Leer slopes if there is room for the stencil, with lower-order slopes if there isn't. See the design document.

7.62.2 Constructor & Destructor Documentation

7.62.2.1 FineInterp::FineInterp ()

Default constructor. User must subsequently call **define()**.

7.62.2.2 FineInterp::~FineInterp ()

Destructor.

7.62.2.3 FineInterp::FineInterp (const DisjointBoxLayout & *a_fine_domain*, const int & *a_numcomps*, const int & *a_ref_ratio*, const Box & *a_fine_problem_domain*)

Defining constructor. Constructs a valid object. Equivalent to default construction followed by **define()**.

{ Arguments:}\ a_fine_domain (not modified): the fine level domain.\ a_numcomps (not modified): the number of components.\ a_ref_ratio (not modified): the refinement ratio.\ a_fine_problem_domain (not modified): problem domain at the fine level.\

7.62.2.4 FineInterp::FineInterp (const DisjointBoxLayout & *a_fine_domain*, const int & *a_numcomps*, const int & *a_ref_ratio*, const ProblemDomain & *a_fine_problem_domain*)

Defining constructor. Constructs a valid object. Equivalent to default construction followed by **define()**.

{ Arguments:}\ a_fine_domain (not modified): the fine level domain.\ a_numcomps (not modified): the number of components.\ a_ref_ratio (not modified): the refinement ratio.\ a_fine_problem_domain (not modified): problem domain at the fine level.\

7.62.3 Member Function Documentation

7.62.3.1 void FineInterp::define (const DisjointBoxLayout & *a_fine_domain*, const int & *a_numcomps*, const int & *a_ref_ratio*, const ProblemDomain & *a_fine_problem_domain*)

Defines this object. Existing information is overriden.

{ Arguments:}\ a_fine_domain (not modified): the fine level domain.\ a_numcomps (not modified): the number of components.\ a_ref_ratio (not modified): the refinement ratio.\ a_fine_problem_domain (not modified): problem domain at the fine level.\

{ This:}\ —This object is modified.—

7.62.3.2 void FineInterp::define (const DisjointBoxLayout & *a_fine_domain*, const int & *a_numcomps*, const int & *a_ref_ratio*, const Box & *a_fine_problem_domain*)

Defines this object. Existing information is overriden.

{ Arguments:}\ a_fine_domain (not modified): the fine level domain.\ a_numcomps (not modified): the number of components.\ a_ref_ratio (not modified): the refinement ratio.\ a_fine_problem_domain (not modified): problem domain at the fine level.\

{ This:}\ —This object is modified.—

7.62.3.3 void FineInterp::interpGridData (BaseFab< Real > & *a_fine*, const BaseFab< Real > & *a_coarse*, const Box & *a_coarsened_fine_box*, int *a_ref_ratio*) const `[protected]`

7.62.3.4 void FineInterp::interpToFine (LevelData< FArrayBox > & *a_fine_data*, const LevelData< FArrayBox > & *a_coarse_data*)

Replaces a_fine_data with data interpolated from a_coarse_data. It is an error to call if not this->**isDefined()**. The domain of a_fine_data should be the same as the fine domain specified in the most recent call to **define()**. It is expected that the coarse and fine level's domains are properly nested. Both a_coarse_data and a_fine_data should have the same number of components specified in the most recent call to **define()**.

{ Arguments:}\ a_fine_data (modified): fine data. \ a_coarse_data (not modified): coarse data. \

{ This:}\ Well, it's complicated. As far as the user is concerned, this object is not modified. See the design document if you care for details.

7.62.3.5 bool FineInterp::isDefined () const

Returns true if this object was created with the defining constructor or if **define()** has been called.

{ This:}\ This object is not modified.

7.62.4 Member Data Documentation

7.62.4.1 bool FineInterp::is_defined `[protected]`

7.62.4.2 ProblemDomain FineInterp::m_coarse_problem_domain `[protected]`

7.62.4.3 LevelData<FArrayBox> FineInterp::m_coarsened_fine_data `[protected]`

7.62.4.4 int FineInterp::m_ref_ratio `[protected]`

The documentation for this class was generated from the following file:

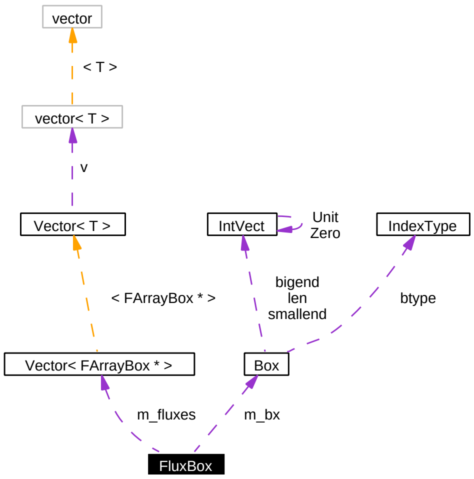
- **FineInterp.H**

7.63 FluxBox Class Reference

A FArrayBox-like container for face-centered fluxes.

`#include <FluxBox.H>`

Collaboration diagram for FluxBox:



Public Member Functions

- **FluxBox** ()
  Default constructor.
- **FluxBox** (const **Box** &bx, int n=1)
  Constructs FluxBox on cell-centered box with n components.
- ~**FluxBox** ()
  Destructor.
- void **resize** (const **Box** &bx, int n=1)
  Resize FluxBox similar to **BaseFab::resize()**.
- void **define** (const **Box** &bx, int n=1)
  Define function.
- void **clear** ()
  Returns the FluxBox to the undefined state.
- int **nComp** () const
  Number of components.
- const **Box** & **box** () const
  Returns cell-centered box which defines fluxBox.

- **FArrayBox** & **getFlux** (const int dir)

  Returns face-centered flux in direction dir.

- const **FArrayBox** & **getFlux** (const int dir) const

  Returns const reference to face-centered flux in direction dir.

- **FArrayBox** & **operator**[ ] (const int dir)

  Returns **FArrayBox** *in direction dir.*

- const **FArrayBox** & **operator**[ ] (const int dir) const

  Constant version.

- void **setVal** (const **Real** val)

  Set all fluxes to val.

- void **setVal** (const **Real** val, const int dir)

  Set fluxes in direction dir to val.

- void **setVal** (const **Real** val, const int dir, const int startComp, const int nComp)

  More specific setVal.

- void **setVal** (const **Real** val, const **Box** &bx)

  Sets fluxes on faces surrounding cell-centered box bx.

- void **setVal** (const **Real** val, const **Box** &bx, const int dir, const int startComp, const int nComp)

  Most specific setVal.

- void **copy** (const **FluxBox** &src)

  Copy from src to this FluxBox – sizes must be identical.

- void **copy** (const **FluxBox** &src, const int srcComp, const int destComp, const int numComp)

  Copy on overlap, for all directions.

- void **copy** (const **FluxBox** &src, const int dir, const int srcComp, const int destComp, const int numComp)

  Copy on overlap of FluxBoxes, in direction dir.

- void **copy** (const **Box** &R, const **Interval** &Cdest, const **FluxBox** &src, const **Interval** &Csrc)

  Copies from a subsection of one box into another.

- void **copy** (const **Box** &srcbox, const **Interval** &srccomps, const **Box** &destbox, const **FluxBox** &src, const **Interval** &destcomps)

  Modifies this FluxBox by copying the contents of src *into it.*

- **FluxBox** & **shift** (const **IntVect** &v)

  Modifies the domain of this FluxBox by shifting.

- int **size** (const **Box** &bx, const **Interval** &comps) const

  Returns size of linearized data over bx.

- void **linearOut** (void ∗buf, const **Box** &R, const **Interval** &comps) const

  Writes a linear representation of this FluxBox.

- void **linearIn** (void ∗buf, const **Box** &R, const **Interval** &comps)

  Read a linear representation of the data over the **Box** *R.*

Static Public Member Functions

- int **preAllocatable** ()

  Helper function for linearization.

Protected Attributes

- **Box m_bx**
- int **m_nvar**
- **Vector< FArrayBox ∗ > m_fluxes**

7.63.1 Detailed Description

A FArrayBox-like container for face-centered fluxes.

This is a class to contain face-centered fluxes on a box.

7.63.2 Constructor & Destructor Documentation

7.63.2.1 FluxBox::FluxBox ()

Default constructor.

7.63.2.2 FluxBox::FluxBox (const Box & *bx*, int $n = 1$)

Constructs FluxBox on cell-centered box with n components.

7.63.2.3 FluxBox::~FluxBox ()

Destructor.

7.63.3 Member Function Documentation

7.63.3.1 const Box& FluxBox::box () const

Returns cell-centered box which defines fluxBox.

7.63.3.2 void FluxBox::clear ()

Returns the FluxBox to the undefined state.

7.63.3.3 void FluxBox::copy (const Box & *srcbox*, const Interval & *srccomps*, const Box & *destbox*, const FluxBox & *src*, const Interval & *destcomps*)

Modifies this FluxBox by copying the contents of *src* into it.

This, the most general form of copy, specifies the contents of any sub-box srcbox in 'FluxBox' src may be copied into a (possibly different) destbox in the destination 'FluxBox'. Note that although the srcbox and the destbox may be disjoint, they must be the same size and shape. If the sizes differ, the copy is undefined and a runtime error results. This copy function is the only one of the copy functions to allow a copy between differing boxes. The user also specifies how many components are copied, starting at component srccomp in src and stored starting at component destcomp. The results are UNDEFINED if the src and dest FluxBoxes are the same and the srcbox and destbox overlap.

7.63.3.4 void FluxBox::copy (const Box & *R*, const Interval & *Cdest*, const FluxBox & *src*, const Interval & *Csrc*)

Copies from a subsection of one box into another.

Assumes the boxes are both in the same index space, and that box R is completely contained in both the src and destination boxes. This function required by **BoxLayoutData**

7.63.3.5 void FluxBox::copy (const FluxBox & *src*, const int *dir*, const int *srcComp*, const int *destComp*, const int *numComp*)

Copy on overlap of FluxBoxes, in direction dir.

7.63.3.6 void FluxBox::copy (const FluxBox & *src*, const int *srcComp*, const int *destComp*, const int *numComp*)

Copy on overlap, for all directions.

7.63.3.7 void FluxBox::copy (const FluxBox & *src*)

Copy from src to this FluxBox – sizes must be identical.

7.63.3.8 void FluxBox::define (const Box & *bx*, int $n = 1$)

Define function.

7.63.3.9 const FArrayBox& FluxBox::getFlux (const int *dir*) const

Returns const reference to face-centered flux in direction *dir*.

7.63.3.10 FArrayBox& FluxBox::getFlux (const int *dir*)

Returns face-centered flux in direction *dir*.

7.63.3.11 void FluxBox::linearIn (void ∗ *buf*, const Box & *R*, const Interval & *comps*)

Read a linear representation of the data over the **Box** R.

Reads in the output of linearOut

7.63.3.12 void FluxBox::linearOut (void ∗ *buf*, const Box & *R*, const Interval & *comps*) const

Writes a linear representation of this FluxBox.

Write a linear representaion of the internal data for the faces surrounding cell-centered box R. Assumes that sufficient memory for the buffer has already been allocated by the caller

7.63.3.13 int FluxBox::nComp () const

Number of components.

7.63.3.14 const FArrayBox& FluxBox::operator[] (const int *dir*) const

Constant version.

7.63.3.15 FArrayBox& FluxBox::operator[] (const int *dir*)

Returns **FArrayBox** in direction dir.

7.63.3.16 int FluxBox::preAllocatable () `[inline, static]`

Helper function for linearization.

7.63.3.17 void FluxBox::resize (const Box & *bx*, int *n* = 1)

Resize FluxBox similar to **BaseFab::resize()**.

7.63.3.18 void FluxBox::setVal (const Real *val*, const Box & *bx*, const int *dir*, const int *startComp*, const int *nComp*)

Most specific setVal.

Sets fluxes on faces surrounding cell-centered box bx

7.63.3.19 void FluxBox::setVal (const Real *val*, const Box & *bx*)

Sets fluxes on faces surrounding cell-centered box bx.

7.63.3.20 void FluxBox::setVal (const Real *val*, const int *dir*, const int *startComp*, const int *nComp*)

More specific setVal.

7.63.3.21 void FluxBox::setVal (const Real *val*, const int *dir*)

Set fluxes in direction dir to val.

7.63.3.22 void FluxBox::setVal (const Real *val*)

Set all fluxes to val.

7.63.3.23 FluxBox& FluxBox::shift (const IntVect & *v*)

Modifies the domain of this FluxBox by shifting.

7.63.3.24 int FluxBox::size (const Box & *bx*, const Interval & *comps*) const

Returns size of linearized data over bx.

Returns size, in number of bytes, of a flat linear representation of data in components comps in faces around cell-centered box R

7.63.4 Member Data Documentation

7.63.4.1 Box FluxBox::m_bx `[protected]`

Cell-centered **Box** over which this FluxBox is defined

7.63.4.2 Vector<FArrayBox∗> FluxBox::m_fluxes `[protected]`

BL_SPACEDIM FArrayBoxes which hold fluxes

7.63.4.3 int FluxBox::m_nvar `[protected]`

Number of variables on each face

The documentation for this class was generated from the following file:

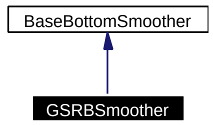
- **FluxBox.H**

7.64 GSRBSmoother Class Reference

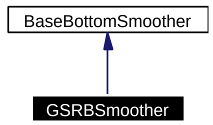
GSRB smoothing as a bottom smoother.

`#include <GSRBSmoother.H>`

Inheritance diagram for GSRBSmoother:



Collaboration diagram for GSRBSmoother:



Public Member Functions

- **GSRBSmoother** ()

  Null constructor.

- virtual ~**GSRBSmoother** ()

  Destructor.

- virtual **BaseBottomSmoother** ∗ **new_bottomSmoother** () const
- virtual void **doBottomSmooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs, **LevelOp** ∗a_levelopPtr)

  Do GSRB smoothing.

- virtual void **setNumSmooth** (int a_numSmooth)

  Set the number of smoother passes to perform.

- virtual int **numSmooth** () const

  Returns the number of smoother passes which are performed.

- virtual void **setConvergenceMetric** (**Real** a_metric, int a_comp=0)

  sets metric to judge convergence (normally norm(rhs) of outer problem

- virtual void **setSolverTol** (**Real** a_tol)

  this doesn't really do anything, but is required by **BaseBottomSmoother**

Protected Attributes

- int **m_numSmooth**

  Number of GSRB smoother passes to do.

7.64.1 Detailed Description

GSRB smoothing as a bottom smoother.

The GSRBSmoother class is a bottom smoother for LevelOp-derived classes which does nothing but a whole lotta GSRB smoothing

7.64.2 Constructor & Destructor Documentation

7.64.2.1 GSRBSmoother::GSRBSmoother ()

Null constructor.

7.64.2.2 virtual GSRBSmoother::~GSRBSmoother () `[virtual]`

Destructor.

7.64.3 Member Function Documentation

7.64.3.1 virtual void GSRBSmoother::doBottomSmooth (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*, LevelOp ∗ *a_levelopPtr*) `[virtual]`

Do GSRB smoothing.

True to its name, this class does GSRB smoothing in this function

Implements **BaseBottomSmoother** (p. 69).

7.64.3.2 virtual BaseBottomSmoother∗ GSRBSmoother::new_bottomSmoother () const `[virtual]`

This gets around the "no virtual constructor" rule

Implements **BaseBottomSmoother** (p. 69).

7.64.3.3 virtual int GSRBSmoother::numSmooth () const `[virtual]`

Returns the number of smoother passes which are performed.

7.64.3.4 virtual void GSRBSmoother::setConvergenceMetric (Real *a_metric*, int *a_comp* = 0) `[inline, virtual]`

sets metric to judge convergence (normally norm(rhs) of outer problem

To make the iterative solver convergence more consistent, the solver will consider itself "solved" if the residual is reduced to epsilon∗metric, where epsilon is the solver tolerance (set by setSolverTol). A good starting point for the metric is the norm of the RHS of the original problem (i.e. the rhs sent in to **LevelSolver** or **AMRSolver**). Default value is 1.0.

Implements **BaseBottomSmoother** (p. 69).

7.64.3.5 virtual void GSRBSmoother::setNumSmooth (int *a_numSmooth*) [virtual]

Set the number of smoother passes to perform.

7.64.3.6 virtual void GSRBSmoother::setSolverTol (Real *a_tol*) [inline, virtual]

this doesn't really do anything, but is required by **BaseBottomSmoother**

Implements **BaseBottomSmoother** (p. 69).

7.64.4 Member Data Documentation

7.64.4.1 int GSRBSmoother::m_numSmooth [protected]

Number of GSRB smoother passes to do.

The documentation for this class was generated from the following file:

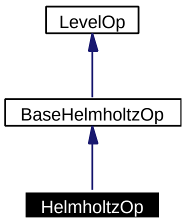
- **GSRBSmoother.H**

7.65 HelmholtzOp Class Reference

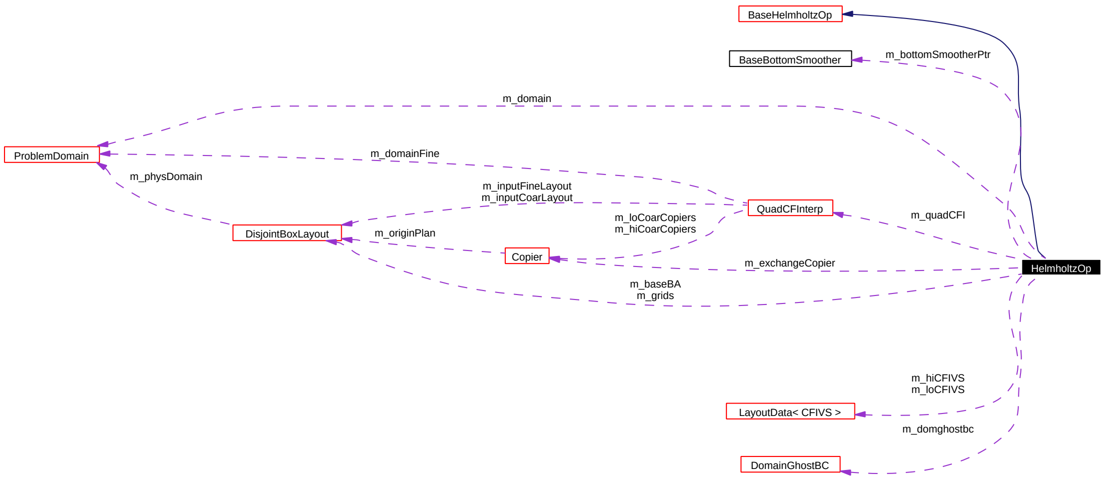
Helmholtz operator class.

`#include <HelmholtzOp.H>`

Inheritance diagram for HelmholtzOp:



Collaboration diagram for HelmholtzOp:



Public Member Functions

- void **setHelmCoeff** (**Real** a_helmcoeff)

  *Set helmholtz coefficient beta in (alpha*I +beta*laplacian)phi = rho.*

- void **scaleHelmCoeff** (**Real** a_scale)

  *Scale the helmholtz coefficient beta in (alpha*I + beta*laplacian)phi.*

- void **setAlpha** (**Real** a_alpha)

  *Set helmholtz coefficient alpha in (alpha*I +beta*laplacian)phi = rho.*

- void **scaleAlpha** (**Real** a_scale)

  Function to scale the coefficient alpha.

- void **scaleBeta** (**Real** a_scale)

*Scale the helmholtz coefficient beta in (alpha*I + beta*laplacian)phi.*

- **LevelOp** ∗ **new_levelop** () const

  Virtual constructor workaround.

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_baseBAPtr, **Real** a_dxLevel, int a_refRatio, const **Box** &a_domain, bool a_homogeneousOnly=false, int a_ncomp=1)

  Full define function (Deprecated – use **ProblemDomain***).*

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_baseBAPtr, **Real** a_dxLevel, int a_refRatio, const **ProblemDomain** &a_domain, bool a_homogeneousOnly=false, int a_ncomp=1)

  Full define function.

- void **define** (const **LevelOp** ∗a_opfine, int a_refToFine)

  Define from finer levelop.

- **HelmholtzOp** ()

  Default constructor.

- **~HelmholtzOp** ()

  Destructor.

- void **smooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs)

  Smoother.

- virtual void **levelPreconditioner** (**LevelData**< **FArrayBox** > &a_phihat, const **LevelData**< **FArrayBox** > &a_rhshat)

  Apply preconditioning to the solution on this level.

- void **applyOpI** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > ∗phiCoarse, **LevelData**< **FArrayBox** > &LOfPhi)

  Evaluate Operator with inhomogeneous BCs.

- void **applyOpIcfHphys** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > ∗phiCoarse, **LevelData**< **FArrayBox** > &LOfPhi)

  Evaluate operator.

- void **applyOpH** (**LevelData**< **FArrayBox** > &phi, **LevelData**< **FArrayBox** > &LOfPhi)

  Evaluate operator with homogeneous boundary conditions.

- void **applyOpHcfIphys** (**LevelData**< **FArrayBox** > &phi, **LevelData**< **FArrayBox** > &LOfPhi)

  Evaluate Operator.

- void **CFInterp** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > &phiCoarse)

  Coarse / Fine interpolation operator. fills ghost cells of phi.

- void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &phi)

  homogeneous coarse/fine interpolation operator

- void **setDomainGhostBC** (const **DomainGhostBC** &a_dombcin)

  Sets the physical boundary conditions for the operator.

- void **setBottomSmoother** (const **BaseBottomSmoother** &a_bottomSmoother)

  Sets the bottom smoother which will be used.

- void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &phif, const **DataIndex** &a_datIndex, int idir, **Side::LoHiSide** hiorlo)

  does homogeneous coarse/fine interpolation

- void **bottomSmoother** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > &rhs)

  This sends the problem to the bottom smoother.

- bool **isDefined** () const

  Has full define function been called?

- void **getFlux** (**FArrayBox** &flux, const **FArrayBox** &data, const **DataIndex** &a_datInd, int dir)

  Fills flux with the flux at this level.

- virtual void **setConvergenceMetric** (**Real** a_metric, int a_comp)

  pass-through to bottom smoother

Protected Attributes

- **DomainGhostBC m_domghostbc**
- **ProblemDomain m_domain**
- **DisjointBoxLayout m_grids**
- **Copier m_exchangeCopier**
- **DisjointBoxLayout m_baseBA**
- int **m_refRatio**
- **Real m_dxLevel**
- **Real m_dxCrse**
- int **m_ncomp**
- **QuadCFInterp m_quadCFI**
- **BaseBottomSmoother** ∗ **m_bottomSmootherPtr**
- bool **m_isDefined**
- bool **m_isBCDefined**
- bool **m_ihcfiEnabled**
- bool **m_hcoeffDefined**
- **LayoutData**< **CFIVS** > **m_loCFIVS** [**SpaceDim**]
- **LayoutData**< **CFIVS** > **m_hiCFIVS** [**SpaceDim**]
- **Real m_helmCoeff**
- **Real m_alphaCoeff**

7.65.1 Detailed Description

Helmholtz operator class.

Operator to solve Helmholtz equation (ie (alpha*I+beta*laplacian)phi = rho)

7.65.2 Constructor & Destructor Documentation

7.65.2.1 HelmholtzOp::HelmholtzOp ()

Default constructor.

7.65.2.2 HelmholtzOp::~HelmholtzOp ()

Destructor.

7.65.3 Member Function Documentation

7.65.3.1 void HelmholtzOp::applyOpH (LevelData< FArrayBox > & *phi*, LevelData< FArrayBox > & *LOfPhi*) `[virtual]`

Evaluate operator with homogeneous boundary conditions.

Uses homogeneous form of both C/F boundary conditions and phys boundary conditions.

Implements **LevelOp** (p. 378).

7.65.3.2 void HelmholtzOp::applyOpHcfIphys (LevelData< FArrayBox > & *phi*, LevelData< FArrayBox > & *LOfPhi*) `[virtual]`

Evaluate Operator.

Uses homogeneous C/F boundary conditions. inhomogeneous domain boundary conditions

Implements **LevelOp** (p. 378).

7.65.3.3 void HelmholtzOp::applyOpI (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > * *phiCoarse*, LevelData< FArrayBox > & *LOfPhi*) `[virtual]`

Evaluate Operator with inhomogeneous BCs.

Uses inhomogeneous form of both C/F boundary conditions and phys boundary conditions.

Implements **LevelOp** (p. 378).

7.65.3.4 void HelmholtzOp::applyOpIcfHphys (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > * *phiCoarse*, LevelData< FArrayBox > & *LOfPhi*) `[virtual]`

Evaluate operator.

Uses inhomogeneous C/F boundary conditions. homogeneous domain boundary conditions.

Implements **LevelOp** (p. 378).

7.65.3.5 void HelmholtzOp::bottomSmoother (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > & *rhs*) [virtual]

This sends the problem to the bottom smoother.

Implements **LevelOp** (p. 378).

7.65.3.6 void HelmholtzOp::CFInterp (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > & *phiCoarse*) [virtual]

Coarse / Fine interpolation operator. fills ghost cells of phi.

Implements **LevelOp** (p. 379).

7.65.3.7 void HelmholtzOp::define (const LevelOp * *a_opfine*, int *a_refToFine*) [virtual]

Define from finer levelop.

Implements **LevelOp** (p. 379).

7.65.3.8 void HelmholtzOp::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout * *a_baseBAPtr*, Real *a_dxLevel*, int *a_refRatio*, const ProblemDomain & *a_domain*, bool *a_homogeneousOnly* = false, int *a_ncomp* = 1) [virtual]

Full define function.

Makes all coarse-fine information and sets internal variables

Parameters:

a_grids Grids at this level

a_baseBAPtr Coarser-level grids

a_dxLevel Cell spacing at this level

a_refRatio Refinement ratio between this level and coarser

a_domain Problem domain at this level

a_homogeneousOnly Only need homogeneous BCs?

a_ncomp number of components

Implements **LevelOp** (p. 379).

7.65.3.9 void HelmholtzOp::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout * *a_baseBAPtr*, Real *a_dxLevel*, int *a_refRatio*, const Box & *a_domain*, bool *a_homogeneousOnly* = false, int *a_ncomp* = 1) [virtual]

Full define function (Deprecated – use **ProblemDomain**).

Makes all coarse-fine information and sets internal variables

Implements **LevelOp** (p. 379).

7.65.3.10 void HelmholtzOp::getFlux (FArrayBox & *flux*, const FArrayBox & *data*, const DataIndex & *a_datInd*, int *dir*) [virtual]

Fills flux with the flux at this level.

The fluxes live on the cell faces with direction dir. Fluxes are computed for all interior edges of data. The flux fab is resized inside the routine.

Implements **LevelOp** (p. 380).

7.65.3.11 void HelmholtzOp::homogeneousCFInterp (LevelData< FArrayBox > & *phif*, const DataIndex & *a_datIndex*, int *idir*, Side::LoHiSide *hiorlo*)

does homogeneous coarse/fine interpolation

7.65.3.12 void HelmholtzOp::homogeneousCFInterp (LevelData< FArrayBox > & *phi*) [virtual]

homogeneous coarse/fine interpolation operator

Implements **LevelOp** (p. 380).

7.65.3.13 bool HelmholtzOp::isDefined () const [virtual]

Has full define function been called?

Implements **LevelOp** (p. 380).

7.65.3.14 virtual void HelmholtzOp::levelPreconditioner (LevelData< FArrayBox > & *a_phihat*, const LevelData< FArrayBox > & *a_rhshat*) [virtual]

Apply preconditioning to the solution on this level.

Given rhshat, returns phihat from M(phihat) = rhshat

Implements **LevelOp** (p. 380).

7.65.3.15 LevelOp* HelmholtzOp::new_levelop () const [virtual]

Virtual constructor workaround.

Coefficient and bcs must be defined before called.

Implements **LevelOp** (p. 380).

7.65.3.16 void HelmholtzOp::scaleAlpha (Real *a_scale*) [virtual]

Function to scale the coefficient alpha.

Scale the helmholtz coefficient alpha in (alpha*I + beta*laplacian)phi.

Implements **BaseHelmholtzOp** (p. 81).

7.65.3.17 void HelmholtzOp::scaleBeta (Real *a_scale*) [virtual]

Scale the helmholtz coefficient beta in (alpha*I + beta*laplacian)phi.

Implements **BaseHelmholtzOp** (p. 81).

7.65.3.18 void HelmholtzOp::scaleHelmCoeff (Real *a_scale*)

Scale the helmholtz coefficient beta in (alpha*I + beta*laplacian)phi.

beta := a_scale*beta. This function exists for backward compatibility with the old helmholtzOp. in reality, this just calls scaleBeta

7.65.3.19 void HelmholtzOp::setAlpha (Real *a_alpha*)

Set helmholtz coefficient alpha in (alpha*I +beta*laplacian)phi = rho.

7.65.3.20 void HelmholtzOp::setBottomSmoother (const BaseBottomSmoother & *a_bottomSmoother*)

Sets the bottom smoother which will be used.

Default is **BiCGStabSmoother**

7.65.3.21 virtual void HelmholtzOp::setConvergenceMetric (Real *a_metric*, int *a_comp*) [virtual]

pass-through to bottom smoother

Implements **LevelOp** (p. 381).

7.65.3.22 void HelmholtzOp::setDomainGhostBC (const DomainGhostBC & *a_dombcin*) [virtual]

Sets the physical boundary conditions for the operator.

Implements **BaseHelmholtzOp** (p. 81).

7.65.3.23 void HelmholtzOp::setHelmCoeff (Real *a_helmcoeff*)

Set helmholtz coefficient beta in (alpha*I +beta*laplacian)phi = rho.

7.65.3.24 void HelmholtzOp::smooth (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*) [virtual]

Smoother.

Assumes that problem has already been put in residual correction form, so that C/F boundary conditions are homogeneous.

Implements **LevelOp** (p. 381).

7.65.4 Member Data Documentation

7.65.4.1 Real HelmholtzOp::m_alphaCoeff [protected]

The alpha in (alpha*I +beta*laplacian)phi = rho

7.65.4.2 DisjointBoxLayout HelmholtzOp::m_baseBA [protected]

DisjointBoxLayout for next coarser level this is == NULL if there is no coarser level

7.65.4.3 BaseBottomSmoother* HelmholtzOp::m_bottomSmootherPtr [protected]

Bottom smoother object

7.65.4.4 ProblemDomain HelmholtzOp::m_domain [protected]

Domain of fine grid

7.65.4.5 DomainGhostBC HelmholtzOp::m_domghostbc [protected]

Boundary conditons

7.65.4.6 Real HelmholtzOp::m_dxCrse [protected]

Mesh spacing at the next coarser level

7.65.4.7 Real HelmholtzOp::m_dxLevel [protected]

Mesh spacing at this level

7.65.4.8 Copier HelmholtzOp::m_exchangeCopier [protected]

Copier object that knows how to perform exchange operations on **LevelData** objects defined on m_grids

7.65.4.9 DisjointBoxLayout HelmholtzOp::m_grids [protected]

The grids at the current level

7.65.4.10 bool HelmholtzOp::m_hcoeffDefined [protected]

Is the helmholtz coefficient defined?

7.65.4.11 Real HelmholtzOp::m_helmCoeff [protected]

The beta in (alpha*I+beta*laplacian)phi = rho

7.65.4.12 LayoutData<CFIVS> HelmholtzOp::m_hiCFIVS[SpaceDim] [protected]

Intvectset for coarse-fine interpolation

7.65.4.13 bool HelmholtzOp::m_ihcfiEnabled [protected]

Is inhomogeneous CF interpolation possible?

7.65.4.14 bool HelmholtzOp::m_isBCDefined [protected]

Has the boundary condition been set?

7.65.4.15 bool HelmholtzOp::m_isDefined [protected]

Has full define function been called?

7.65.4.16 LayoutData<CFIVS> HelmholtzOp::m_loCFIVS[SpaceDim] [protected]

Intvectset for coarse-fine interpolation

7.65.4.17 int HelmholtzOp::m_ncomp [protected]

Number of components

7.65.4.18 QuadCFInterp HelmholtzOp::m_quadCFI [protected]

Coarse-fine information between this level and the next coarser level

7.65.4.19 int HelmholtzOp::m_refRatio [protected]

Refinement ratio between this and the next coarser level

The documentation for this class was generated from the following file:

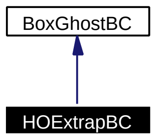
- **HelmholtzOp.H**

7.66 HOExtrapBC Class Reference

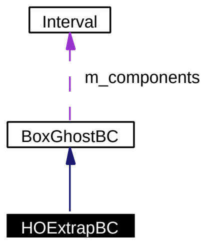
class to do extrapolation BC's

`#include <HOExtrapBC.H>`

Inheritance diagram for HOExtrapBC:



Collaboration diagram for HOExtrapBC:



Public Member Functions

- **HOExtrapBC** ()
- **~HOExtrapBC** ()
- **HOExtrapBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- **HOExtrapBC** (int dir, **Side::LoHiSide** sd)

Protected Member Functions

- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyHOExtrapBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const

  this doesn't do anything since we don't use the factors

- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

  deprecated interface

- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

deprecated interface

- virtual void **applyHOExtrapBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

  deprecated interface

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

  this doesn't do anything since we don't use the factors

- **BoxGhostBC** ∗ **new_boxghostbc** () const

Friends

- class **DomainGhostBC**

7.66.1 Detailed Description

class to do extrapolation BC's

7.66.2 Constructor & Destructor Documentation

7.66.2.1 HOExtrapBC::HOExtrapBC ()

7.66.2.2 HOExtrapBC::∼HOExtrapBC ()

7.66.2.3 HOExtrapBC::HOExtrapBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.66.2.4 HOExtrapBC::HOExtrapBC (int *dir*, Side::LoHiSide *sd*)

7.66.3 Member Function Documentation

7.66.3.1 virtual void HOExtrapBC::applyHOExtrapBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

7.66.3.2 virtual void HOExtrapBC::applyHOExtrapBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

7.66.3.3 virtual void HOExtrapBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

Reimplemented from **BoxGhostBC** (p. 140).

7.66.3.4 virtual void HOExtrapBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.66.3.5 virtual void HOExtrapBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const [protected, virtual]

deprecated interface

Reimplemented from **BoxGhostBC** (p. 141).

7.66.3.6 virtual void HOExtrapBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.66.3.7 virtual void HOExtrapBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const [protected, virtual]

this doesn't do anything since we don't use the factors

Implements **BoxGhostBC** (p. 141).

7.66.3.8 virtual void HOExtrapBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const [protected, virtual]

this doesn't do anything since we don't use the factors

Implements **BoxGhostBC** (p. 141).

7.66.3.9 BoxGhostBC* HOExtrapBC::new_boxghostbc () const [protected, virtual]

Virtual constructor workaround.

Implements **BoxGhostBC** (p. 141).

7.66.4 Friends And Related Function Documentation

7.66.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 142).

The documentation for this class was generated from the following file:

- **HOExtrapBC.H**

7.67 IndexType Class Reference

Cell-Based or Node-Based Indices.

`#include <Box.H>`

Public Types

- enum **CellIndex** { **CELL** = 0, **NODE** = 1 }

Public Member Functions

- **IndexType** ()
- **IndexType** (const **IndexType** &rhs)
- **IndexType** (const **IntVect** &iv)
- **IndexType** & **operator=** (const **IndexType** &rhs)
- **IndexType** (D_DECL(**CellIndex** i, **CellIndex** j, **CellIndex** k))
- void **set** (int dir)
- void **unset** (int dir)
- bool **test** (int dir) const
- void **setall** ()
- void **clear** ()
- bool **any** () const
- bool **ok** () const
- void **flip** (int i)
- bool **operator==** (const **IndexType** &t) const
- bool **operator!=** (const **IndexType** &t) const
- bool **cellCentered** () const
- bool **nodeCentered** () const
- void **setType** (int dir, **CellIndex** t)
- **CellIndex ixType** (int dir) const
- int **operator**[ ] (int dir) const
- **IntVect ixType** () const

Static Public Member Functions

- **IndexType TheCellType** ()
- **IndexType TheNodeType** ()

Friends

- std::ostream & **operator**<< (std::ostream &os, const **IndexType** &itype)
- std::istream & **operator**>> (std::istream &is, **IndexType** &itype)

7.67.1 Detailed Description

Cell-Based or Node-Based Indices.

The class IndexType defines an index as being cell based or node (face) based in each of the CH directions. This class defines an enumerated type CellIndex to be either CELL or NODE; i.e. each of the CH dimensions must be either CELL or NODE.

7.67.2 Member Enumeration Documentation

7.67.2.1 enum IndexType::CellIndex

The cell index type: one of CELL or NODE.

Enumeration values:

- **CELL**
- **NODE**

7.67.3 Constructor & Destructor Documentation

7.67.3.1 IndexType::IndexType () `[inline]`

The default constructor.

7.67.3.2 IndexType::IndexType (const IndexType & *rhs*) `[inline]`

The copy constructor.

7.67.3.3 IndexType::IndexType (const IntVect & *iv*) `[inline, explicit]`

Constructs an IndexType identical to an **IntVect**.

7.67.3.4 IndexType::IndexType (D_DECL(CellIndex i, CellIndex j, CellIndex k)) `[inline]`

Constructs an IndexType given an explicit CellIndex for each direction. D is a macro that sets the constructor to take BL arguments.

7.67.4 Member Function Documentation

7.67.4.1 bool IndexType::any () const `[inline]`

True if this IndexType is NODE based in any direction.

7.67.4.2 bool IndexType::cellCentered () const `[inline]`

True if this IndexType is CELL based in all directions.

7.67.4.3 void IndexType::clear () `[inline]`

Set this IndexType to be CELL based in all directions.

7.67.4.4 void IndexType::flip (int *i*) `[inline]`

Change this IndexType from CELL to NODE or NODE to CELL in direction dir.

7.67.4.5 IntVect IndexType::ixType () const [inline]

Convert this IndexType to an **IntVect** via component-wise conversion of enum CellIndex to int.

7.67.4.6 IndexType::CellIndex IndexType::ixType (int *dir*) const [inline]

Returns the CellIndex of this IndexType in direction dir.

7.67.4.7 bool IndexType::nodeCentered () const [inline]

True if this IndexType is NODE based in all directions.

7.67.4.8 bool IndexType::ok () const [inline]

True if this IndexType is valid.

7.67.4.9 bool IndexType::operator!= (const IndexType & *t*) const [inline]

True if IndexTypes are not identical.

7.67.4.10 IndexType & IndexType::operator= (const IndexType & *rhs*) [inline]

The assignment operator.

7.67.4.11 bool IndexType::operator== (const IndexType & *t*) const [inline]

True if IndexTypes are identical.

7.67.4.12 int IndexType::operator[] (int *dir*) const [inline]

Return an integer representing this IndexType in direction dir.

7.67.4.13 void IndexType::set (int *dir*) [inline]

Sets this IndexType to be NODE based in direction dir.

7.67.4.14 void IndexType::setall () [inline]

Set this IndexType to be NODE based in all directions.

7.67.4.15 void IndexType::setType (int *dir*, CellIndex *t*) [inline]

Set this IndexType to CellIndex type t in direction dir.

7.67.4.16 bool IndexType::test (int *dir*) const [inline]

True if this IndexType is NODE based in direction dir.

7.67.4.17 IndexType IndexType::TheCellType () [static]

This static member function returns an IndexType object all of whose components are of value **IndexType::CELL**. It is provided as a convenience to our users when defining a **Box** all of whose faces should be of type **IndexType::CELL**.

7.67.4.18 IndexType IndexType::TheNodeType () [static]

This static member function returns an IndexType object all of whose components are of value **IndexType::NODE**. It is provided as a convenience to our users when defining a **Box** all of whose faces should be of type **IndexType::NODE**.

7.67.4.19 void IndexType::unset (int *dir*) [inline]

Sets this IndexType to be CELL based in direction dir.

7.67.5 Friends And Related Function Documentation

7.67.5.1 std::ostream& operator<< (std::ostream & *os*, const IndexType & *itype*) [friend]

Write an IndexType to an ostream in ASCII.

7.67.5.2 std::istream& operator>> (std::istream & *is*, IndexType & *itype*) [friend]

Read an IndexType from an istream.

The documentation for this class was generated from the following file:

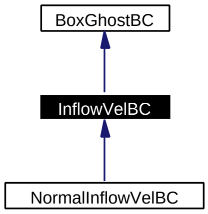
- **Box.H**

7.68 InflowVelBC Class Reference

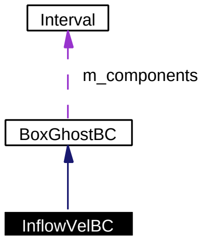
does inflow

`#include <basicVelBC.H>`

Inheritance diagram for InflowVelBC:



Collaboration diagram for InflowVelBC:



Public Member Functions

- **InflowVelBC** ()
- virtual ~**InflowVelBC** ()
- **InflowVelBC** (int dir, **Side::LoHiSide** sd)
- **InflowVelBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- void **isHomogeneous** (bool a_isHomogeneous)

  is the BC homogeneous?

- bool **isHomogeneous** () const
- void **setBCVal** (const **Real** a_bcVal)

  for single-valued inflow...

- **Real BCVal** () const
- void **Time** (const **Real** m_time)

  set time (for time-varying inflow

- **Real Time** () const

Protected Member Functions

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dirfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dirfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

  deprecated interface

- virtual **BoxGhostBC** * **new_boxghostbc** () const

Friends

- class **DomainGhostBC**

7.68.1 Detailed Description

does inflow

7.68.2 Constructor & Destructor Documentation

7.68.2.1 InflowVelBC::InflowVelBC ()

7.68.2.2 virtual InflowVelBC::~InflowVelBC () `[virtual]`

7.68.2.3 InflowVelBC::InflowVelBC (int *dir*, Side::LoHiSide *sd*)

7.68.2.4 InflowVelBC::InflowVelBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.68.3 Member Function Documentation

7.68.3.1 Real InflowVelBC::BCVal () const

7.68.3.2 virtual void InflowVelBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dirfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const `[protected, virtual]`

deprecated interface

Implements **BoxGhostBC** (p. 141).

Reimplemented in **NormalInflowVelBC** (p. 430).

7.68.3.3 virtual void InflowVelBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dirfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const `[protected, virtual]`

Pure-virtual Worker function to fill arrays which define the boundary condition. Essentially, if the boundary condition is represented as A*phi + B*dphi/dn = C, then a_dircfac is A, a_neumfac is B, and a_inhmval is C.

Implements **BoxGhostBC** (p. 141).

7.68.3.4 bool InflowVelBC::isHomogeneous () const

7.68.3.5 void InflowVelBC::isHomogeneous (bool *a_isHomogeneous*)

is the BC homogeneous?

7.68.3.6 virtual BoxGhostBC* InflowVelBC::new_boxghostbc () const [protected, virtual]

Virtual constructor workaround.

Implements **BoxGhostBC** (p. 141).

Reimplemented in **NormalInflowVelBC** (p. 430).

7.68.3.7 void InflowVelBC::setBCVal (const Real *a_bcVal*)

for single-valued inflow...

7.68.3.8 Real InflowVelBC::Time () const

7.68.3.9 void InflowVelBC::Time (const Real *m_time*)

set time (for time-varying inflow

7.68.4 Friends And Related Function Documentation

7.68.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 142).

Reimplemented in **NormalInflowVelBC** (p. 430).

The documentation for this class was generated from the following file:

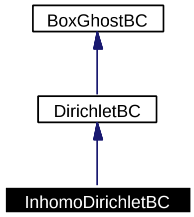
- **basicVelBC.H**

7.69 InhomoDirichletBC Class Reference

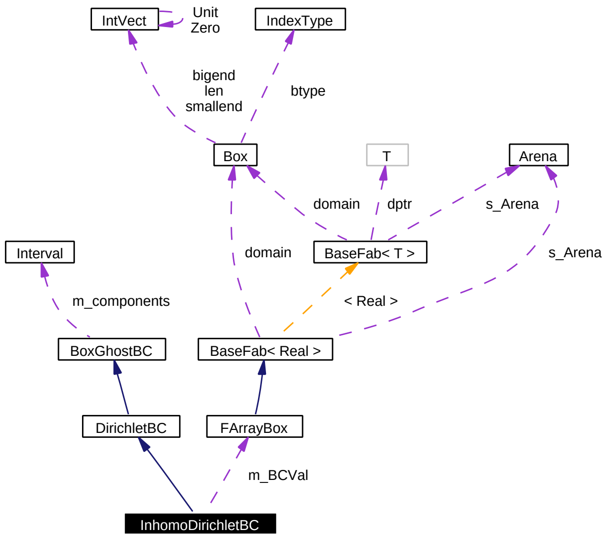
implements dirichlet BCs for scalars

`#include <InhomoDirichletBC.H>`

Inheritance diagram for InhomoDirichletBC:



Collaboration diagram for InhomoDirichletBC:



Public Member Functions

- **InhomoDirichletBC** ()

  Null constructor.

- **InhomoDirichletBC** (int dir, **Side::LoHiSide** sd)

  Simple constructor (sets **Interval** *to (0,0).*

- **InhomoDirichletBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)

  Full constructor.

- ~**InhomoDirichletBC** ()

  Destructor.

- void **setBCVal** (const **FArrayBox** &a_BCVal)
  bc val should be the face-centered value, but in a cell-centered FAB

Friends

- class **DomainGhostBC**

7.69.1 Detailed Description

implements dirichlet BCs for scalars

7.69.2 Constructor & Destructor Documentation

7.69.2.1 InhomoDirichletBC::InhomoDirichletBC ()

Null constructor.

7.69.2.2 InhomoDirichletBC::InhomoDirichletBC (int *dir*, Side::LoHiSide *sd*)

Simple constructor (sets **Interval** to (0,0).

7.69.2.3 InhomoDirichletBC::InhomoDirichletBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

Full constructor.

7.69.2.4 InhomoDirichletBC::~InhomoDirichletBC ()

Destructor.

7.69.3 Member Function Documentation

7.69.3.1 void InhomoDirichletBC::setBCVal (const FArrayBox & *a_BCVal*)

bc val should be the face-centered value, but in a cell-centered FAB

values in the bcVal fab should be the inhomogeneous BC values at the actual domain boundary. However, the actual FAB should be cell-centered, to overlie the ghost cells just outside the domain.

7.69.4 Friends And Related Function Documentation

7.69.4.1 friend class DomainGhostBC `[friend]`

Reimplemented from **DirichletBC** (p. 224).

The documentation for this class was generated from the following file:

- **InhomoDirichletBC.H**

7.70 Interval Struct Reference

Structure for passing component ranges in code.

`#include <Interval.H>`

Public Member Functions

- **Interval** ()
- **Interval** (int a_firstComp, int a_lastComp)
- void **define** (int a_firstComp, int a_lastComp)
- int **begin** () const
- int **end** () const
- int **size** () const
- bool **contains** (int a_val) const

7.70.1 Detailed Description

Structure for passing component ranges in code.

A class to encapsulate component ranges

7.70.2 Constructor & Destructor Documentation

7.70.2.1 Interval::Interval () `[inline]`

7.70.2.2 Interval::Interval (int *a_firstComp*, int *a_lastComp*) `[inline]`

initialize with first and last component numbers

7.70.3 Member Function Documentation

7.70.3.1 int Interval::begin () const `[inline]`

return first component number

7.70.3.2 bool Interval::contains (int *a_val*) const `[inline]`

7.70.3.3 void Interval::define (int *a_firstComp*, int *a_lastComp*) `[inline]`

7.70.3.4 int Interval::end () const `[inline]`

return last component number

7.70.3.5 int Interval::size () const `[inline]`

return last - first + 1

The documentation for this struct was generated from the following file:

- **Interval.H**

7.71 IntVect Class Reference

An integer **Vector** in SpaceDim-dimensional space.

`#include <IntVect.H>`

Collaboration diagram for IntVect:

Addition operators

- **IntVect** & **operator+=** (int s)
- **IntVect** & **operator+=** (const **IntVect** &p)
- **IntVect operator+** (const **IntVect** &p) const
- **IntVect operator+** (int s) const
- **IntVect operator+** (int s, const **IntVect** &p)

Subtraction operators

- **IntVect** & **operator-=** (int s)
- **IntVect** & **operator-=** (const **IntVect** &p)
- **IntVect operator-** (const **IntVect** &p) const
- **IntVect operator-** (int s) const
- **IntVect operator-** (int s, const **IntVect** &p)

Multiplication operators

- **IntVect** & **operator** ∗= (int s)
- **IntVect** & **operator** ∗= (const **IntVect** &p)
- **IntVect operator** ∗ (const **IntVect** &p) const
- **IntVect operator** ∗ (int s) const
- **IntVect operator** ∗ (int s, const **IntVect** &p)

Other arithmetic operators

- **IntVect** & **min** (const **IntVect** &p)
- **IntVect** & **max** (const **IntVect** &p)
- **IntVect** & **scale** (int s)
- **IntVect** & **reflect** (int ref_ix, int idir)
- **IntVect** & **shift** (int coord, int s)
- **IntVect** & **shift** (const **IntVect** &iv)
- **IntVect** & **diagShift** (int s)
- **IntVect** & **coarsen** (const **IntVect** &p)
- **IntVect** & **coarsen** (int p)
- **IntVect min** (const **IntVect** &p1, const **IntVect** &p2)
- **IntVect max** (const **IntVect** &p1, const **IntVect** &p2)
- **IntVect scale** (const **IntVect** &p, int s)

- **IntVect reflect** (const **IntVect** &a, int ref_ix, int idir)
- **IntVect diagShift** (const **IntVect** &p, int s)
- **IntVect coarsen** (const **IntVect** &p, int s)
- **IntVect coarsen** (const **IntVect** &p1, const **IntVect** &p2)

I/O Functions

- void **printOn** (std::ostream &os) const
- void **p** () const
- void **dumpOn** (std::ostream &os) const
- std::ostream & **operator<<** (std::ostream &os, const **IntVect** &iv)
- std::istream & **operator>>** (std::istream &os, **IntVect** &iv)

Constants

- const **IntVect TheZeroVector** ()
- const **IntVect TheUnitVector** ()
- const **IntVect Zero**
- const **IntVect Unit**
- **IntVect BASISV** (int dir)

Public Member Functions

Constructors and Accessors

- **IntVect** ()
- **~IntVect** ()
- **IntVect** (D_DECL(int i, int j, int k))
- **IntVect** (const int *a)
- **IntVect** (const **IntVect** &rhs)
- **IntVect copy** () const
- **IntVect** & **operator=** (const **IntVect** &rhs)
- int & **operator**[ ] (int i)
- int **operator**[ ] (int i) const
- void **setVal** (int i, int val)

Data pointer functions

- const int * **getVect** () const
- const int * **dataPtr** () const
- int * **dataPtr** ()

Comparison Operators

- bool **operator==** (const **IntVect** &p) const
- bool **operator!=** (const **IntVect** &p) const
- bool **operator<** (const **IntVect** &p) const
- bool **operator<=** (const **IntVect** &p) const
- bool **operator>** (const **IntVect** &p) const
- bool **operator>=** (const **IntVect** &p) const
- bool **lexLT** (const **IntVect** &s) const
- bool **lexGT** (const **IntVect** &s) const

Unary operators

- **IntVect operator+** () const
- **IntVect operator-** () const

Division operators

- **IntVect** & **operator/=** (int s)
- **IntVect** & **operator/=** (const **IntVect** &p)
- **IntVect operator/** (const **IntVect** &p) const
- **IntVect operator/** (int s) const

Protected Attributes

- int **vect** [**SpaceDim**]

Static Protected Attributes

- const size_t **IntVectSize**

Friends

- class **Box**
- class **HDF5Handle**
- class **VolIndex**
- class **FaceIndex**

7.71.1 Detailed Description

An integer **Vector** in SpaceDim-dimensional space.

The class IntVect is an implementation of an integer vector in a SpaceDim-dimensional space. It represents a point in a discrete space. IntVect values are accessed using the operator[] function, as for a normal C++ array. In addition, the basic arithmetic operators have been overloaded to implement scaling and translation operations.

7.71.2 Constructor & Destructor Documentation

7.71.2.1 IntVect::IntVect () `[inline]`

Construct an IntVect whose components are uninitialized.

7.71.2.2 IntVect::~IntVect () `[inline]`

Destructor.

7.71.2.3 IntVect::IntVect (D_DECL(int i, int j, int k)) `[inline]`

Construct an IntVect given the specific values for its coordinates. D_DECL is a macro that sets the constructor to take CH_SPACEDIM arguments.

7.71.2.4 IntVect::IntVect (const int * *a*) [inline, explicit]

Construct an IntVect setting the coordinates to the corresponding values in the integer array *a*.

7.71.2.5 IntVect::IntVect (const IntVect & *rhs*) [inline]

The copy constructor.

7.71.3 Member Function Documentation

7.71.3.1 IntVect& IntVect::coarsen (int *p*)

Modify IntVect by component-wise integer projection.

7.71.3.2 IntVect& IntVect::coarsen (const IntVect & *p*)

Modify IntVect by component-wise integer projection.

7.71.3.3 IntVect IntVect::copy () const [inline]

7.71.3.4 int * IntVect::dataPtr () [inline]

Only for sending to Fortran

7.71.3.5 const int * IntVect::dataPtr () const [inline]

Only for sending to Fortran

7.71.3.6 IntVect & IntVect::diagShift (int *s*) [inline]

Modifies this IntVect by adding a scalar *s* to each component.

7.71.3.7 void IntVect::dumpOn (std::ostream & *os*) const

Print an IntVect to the ostream a bit more verbosely.

7.71.3.8 const int * IntVect::getVect () const [inline]

Returns a const pointer to an array of coordinates of the IntVect. Useful for arguments to FORTRAN calls.

7.71.3.9 bool IntVect::lexGT (const IntVect & *s*) const [inline]

Returns true if this IntVect is lexically greater than the argument. An IntVect MUST BE either lexically less than, lexically greater than, or equal to another IntVect.

iv1 is lexically less than iv2 if:

in 2-D:

(iv1[0] > iv2[0]) || ((iv1[0] == iv2[0]) && (iv1[1] > iv2[1]));

in 3-D:

(iv1[0] > iv2[0]) || (iv1[0]==iv2[0] && ((iv1[1] > iv2[1] || ((iv1[1] == iv2[1]) && (iv1[2] > iv2[2])))));

7.71.3.10 bool IntVect::lexLT (const IntVect & *s*) const [inline]

Returns true if this IntVect is lexically less than the argument. An IntVect MUST BE either lexically less than, lexically greater than, or equal to another IntVect.

iv1 is lexically less than iv2 if:

in 2-D:

(iv1[0] < iv2[0]) || ((iv1[0] == iv2[0]) && (iv1[1] < iv2[1]));

in 3-D:

(iv1[0] < iv2[0]) || (iv1[0]==iv2[0] && ((iv1[1] < iv2[1] || ((iv1[1] == iv2[1]) && (iv1[2] < iv2[2])))));

7.71.3.11 IntVect & IntVect::max (const IntVect & *p*) [inline]

Modifies this IntVect by taking component-wise max with IntVect argument.

7.71.3.12 IntVect & IntVect::min (const IntVect & *p*) [inline]

Modifies this IntVect by taking component-wise min with IntVect argument.

7.71.3.13 IntVect IntVect::operator * (int *s*) const [inline]

Returns an IntVect that is this IntVect with each component multiplied by a scalar.

7.71.3.14 IntVect IntVect::operator * (const IntVect & *p*) const [inline]

Returns component-wise product of this IntVect with argument.

7.71.3.15 IntVect & IntVect::operator *= (const IntVect & *p*) [inline]

Modifies this IntVect by component-wise multiplication by argument.

7.71.3.16 IntVect & IntVect::operator *= (int *s*) [inline]

Modifies this IntVect by multiplication of each component by a scalar.

7.71.3.17 bool IntVect::operator!= (const IntVect & *p*) const [inline]

Returns true if this IntVect is different from argument IntVect. All comparisons between analogous components must be satisfied.

7.71.3.18 IntVect IntVect::operator+ (int *s*) const [inline]

Return an IntVect that is this IntVect with a scalar added to each component.

7.71.3.19 IntVect IntVect::operator+ (const IntVect & *p*) const [inline]

Returns component-wise sum of this IntVect and argument.

7.71.3.20 IntVect IntVect::operator+ () const [inline]

Unary plus – for completeness.

7.71.3.21 IntVect & IntVect::operator+= (const IntVect & *p*) [inline]

Modifies this IntVect by component-wise addition with argument.

7.71.3.22 IntVect & IntVect::operator+= (int *s*) [inline]

Modifies this IntVect by addition of a scalar to each component.

7.71.3.23 IntVect IntVect::operator- (int *s*) const [inline]

Returns an IntVect that is this IntVect with a scalar s subtracted from each component.

7.71.3.24 IntVect IntVect::operator- (const IntVect & *p*) const [inline]

Returns an IntVect that is this IntVect with p subtracted from it component-wise.

7.71.3.25 IntVect IntVect::operator- () const [inline]

Unary minus – negates all components of this IntVect.

7.71.3.26 IntVect & IntVect::operator-= (const IntVect & *p*) [inline]

Modifies this IntVect by component-wise subtraction by argument.

7.71.3.27 IntVect & IntVect::operator-= (int *s*) [inline]

Modifies this IntVect by subtraction of a scalar from each component.

7.71.3.28 IntVect IntVect::operator/ (int *s*) const [inline]

Returns an IntVect that is this IntVect with each component divided by a scalar.

7.71.3.29 IntVect IntVect::operator/ (const IntVect & *p*) const [inline]

Returns component-wise quotient of this IntVect by argument.

7.71.3.30 IntVect & IntVect::operator/= (const IntVect & *p*) [inline]

Modifies this IntVect by component-wise division by IntVect argument.

7.71.3.31 IntVect & IntVect::operator/= (int *s*) [inline]

Modifies this IntVect by division of each component by a scalar.

7.71.3.32 bool IntVect::operator< (const IntVect & *p*) const [inline]

Returns true if this IntVect is less than argument IntVect. All comparisons between analogous components must be satisfied. Note that, since the comparison is component-wise, it is possible for an IntVect to be neither greater than, less than, nor equal to another.

7.71.3.33 bool IntVect::operator<= (const IntVect & *p*) const [inline]

Returns true if this IntVect is less than or equal to argument IntVect. All comparisons between analogous components must be satisfied. Note that, since the comparison is component-wise, it is possible for an IntVect to be neither greater than or equal to, less than or equal to, nor equal to another.

7.71.3.34 IntVect & IntVect::operator= (const IntVect & *rhs*) [inline]

The assignment operator.

7.71.3.35 bool IntVect::operator== (const IntVect & *p*) const [inline]

Returns true if this IntVect is equivalent to argument IntVect. All comparisons between analogous components must be satisfied.

7.71.3.36 bool IntVect::operator> (const IntVect & *p*) const [inline]

Returns true if this IntVect is greater than argument IntVect. All comparisons between analogous components must be satisfied. Note that, since the comparison is component-wise, it is possible for an IntVect to be neither greater than, less than, nor equal to another.

7.71.3.37 bool IntVect::operator>= (const IntVect & *p*) const [inline]

Returns true if this IntVect is greater than or equal to argument IntVect. All comparisons between analogous components must be satisfied. Note that, since the comparison is component-wise, it is possible for an IntVect to be neither greater than or equal to, less than or equal to, nor equal to another.

7.71.3.38 int IntVect::operator[] (int *i*) const [inline]

Returns the i'th coordinate of the IntVect.

7.71.3.39 int & IntVect::operator[] (int *i*) [inline]

Returns a modifiable lvalue reference to the *i*'th coordinate of the IntVect.

7.71.3.40 void IntVect::p () const

Print an IntVect to the **pout()**.

7.71.3.41 void IntVect::printOn (std::ostream & *os*) const

Print an IntVect to the ostream.

7.71.3.42 IntVect & IntVect::reflect (int *ref_ix*, int *idir*) [inline]

Modifies IntVect by reflecting it in the plane defined by the index *ref_ix* and with normal in the direction of *idir*. Directions are based at zero.

7.71.3.43 IntVect & IntVect::scale (int *s*) [inline]

Modifies this IntVect by multiplying each component by a scalar.

7.71.3.44 void IntVect::setVal (int *i*, int *val*) [inline]

Set *i*'th coordinate of IntVect to *val*.

7.71.3.45 IntVect & IntVect::shift (const IntVect & *iv*) [inline]

Modifies this IntVect by component-wise addition with IntVect argument.

7.71.3.46 IntVect & IntVect::shift (int *coord*, int *s*) [inline]

Modifies this IntVect by adding *s* to component in given coordinate direction.

7.71.3.47 const IntVect IntVect::TheUnitVector () [inline, static]

TheUnitVector is out of favor. Use **IntVect::Unit**.

7.71.3.48 const IntVect IntVect::TheZeroVector () [inline, static]

TheZeroVector is out of favor. Use **IntVect::Zero**.

7.71.4 Friends And Related Function Documentation

7.71.4.1 IntVect BASISV (int *dir*) [friend]

Returns a basis vector in the given coordinate direction.

In 3-D: BASISV(0) == (1,0,0); BASISV(1) == (0,1,0); BASISV(2) == (0,0,1).

In 2-D: BASISV(0) == (1,0); BASISV(1) == (0,1).

Note that the coordinate directions are based at zero.

7.71.4.2 friend class Box [friend]

7.71.4.3 IntVect coarsen (const IntVect & *p1*, const IntVect & *p2*) [friend]

Returns an IntVect which is the component-wise integer projection of IntVect *p1* by IntVect *p2*.

7.71.4.4 IntVect coarsen (const IntVect & *p*, int *s*) [friend]

Returns an IntVect that is the component-wise integer projection of *p* by *s*.

7.71.4.5 IntVect diagShift (const IntVect & *p*, int *s*) [friend]

Returns IntVect obtained by adding a scalar to each of the components of the given IntVect.

7.71.4.6 friend class FaceIndex [friend]

7.71.4.7 friend class HDF5Handle [friend]

7.71.4.8 IntVect max (const IntVect & *p1*, const IntVect & *p2*) [friend]

Returns the IntVect that is the component-wise maximum of two argument IntVects.

7.71.4.9 IntVect min (const IntVect & *p1*, const IntVect & *p2*) [friend]

Returns the IntVect that is the component-wise minimum of two argument IntVects.

7.71.4.10 IntVect operator * (int *s*, const IntVect & *p*) [friend]

Returns an IntVect that is an IntVect *p* with each component multiplied by a scalar *s*.

7.71.4.11 IntVect operator+ (int *s*, const IntVect & *p*) [friend]

Returns an IntVect that is an IntVect *p* with a scalar *s* added to each component.

7.71.4.12 IntVect operator- (int *s*, const IntVect & *p*) [friend]

Returns *s* - *p*.

7.71.4.13 std::ostream& operator<< (std::ostream & *os*, const IntVect & *iv*) [friend]

Print the IntVect to given output stream in ASCII.

7.71.4.14 std::istream& operator>> (std::istream & *os*, IntVect & *iv*) [friend]

Read next IntVect from given input stream.

7.71.4.15 IntVect reflect (const IntVect & *a*, int *ref_ix*, int *idir*) [friend]

Returns an IntVect that is the reflection of the given IntVect in the plane which passes through *ref_ix* and normal to the coordinate direction idir.

7.71.4.16 IntVect scale (const IntVect & *p*, int *s*) [friend]

Returns an IntVect obtained by multiplying each of the components of the given IntVect by a scalar.

7.71.4.17 friend class VolIndex [friend]

7.71.5 Member Data Documentation

7.71.5.1 const size_t IntVect::IntVectSize [static, protected]

Number of bytes of storage used by this IntVect.

7.71.5.2 const IntVect IntVect::Unit [static]

This is an IntVect all of whose components are equal to one.

7.71.5.3 int IntVect::vect[SpaceDim] [protected]

The individual components of this IntVect.

7.71.5.4 const IntVect IntVect::Zero [static]

This is an IntVect all of whose components are equal to zero.

The documentation for this class was generated from the following file:

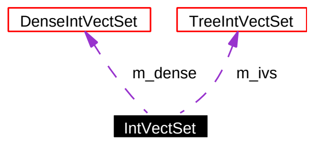
- **IntVect.H**

7.72 IntVectSet Class Reference

An irregular domain on an integer lattice.

`#include <IntVectSet.H>`

Collaboration diagram for IntVectSet:



Constructors, Destructor, and defines

- **~IntVectSet** ()
- **IntVectSet** ()

    default constructor: defines an empty IntVectSet.

- void **define** ()

    default define: modifies this IntVectSet to be empty.

- **IntVectSet** (const **IntVectSet** &ivs)

    copy constructor

- void **define** (const **IntVectSet** &ivs)

    copy define: modifies this IntVectSet into a copy of ivs

- void **define_intvectset** (const **IntVectSet** &ivs)

    copy define: same as `define(IntVectSet)`

- **IntVectSet copy** () const

    return a copy of this IntVectSet

- **IntVectSet** (const **DenseIntVectSet** &a_dense)

    conversion constructor

- void **define** (const **DenseIntVectSet** &a_dense)

    conversion define

- **IntVectSet** (const **TreeIntVectSet** &a_tree)

    conversion constructor

- void **define** (const **TreeIntVectSet** &a_tree)

    conversion define

- **IntVectSet** (const **IntVect** &iv)

    IntVect *constructor.*

- void **define** (const **IntVect** &iv)
  IntVect *define: modifies this IntVectSet to have just one* **IntVect**.

- void **define_intvect** (const **IntVect** &iv)
  IntVect *define: same as* `define(IntVect)`.

- **IntVectSet** (const **Box** &b)
  Box *constructor.*

- void **define** (const **Box** &b)
  Box *define.*

- void **define_box** (const **Box** &b)
  Box *define: same as* `define(Box)`.

Union operators

- **IntVectSet** & **operator**|= (const **IntVectSet** &ivs)
  unions another IntVectSet ivs *into this IntVectSet*

- void **or_intvectset** (const **IntVectSet** &ivs)
  *same as operator|= except it doesn't return *this*

- **IntVectSet** & **operator**|= (const **IntVect** &iv)
  unions a single **IntVect** iv *into this IntVectSet*

- void **or_intvect** (const **IntVect** &ivs)
  *same as operator|= except it doesn't return *this*

- **IntVectSet** & **operator**|= (const **Box** &b)
  unions the IntVects in the **Box into** *this IntVectSet*

- void **or_box** (const **Box** &b)
  *same as operator|= except it doesn't return *this*

- **IntVectSet operator**| (const **IntVectSet** &ivs1, const **IntVectSet** &ivs2)
  Returns a new IntVectSet that is the union of two IntVectSets.

- **IntVectSet operator**| (const **IntVectSet** &ivs, const **IntVect** &iv)
  Returns a new IntVectSet that is the union of an IntVectSet and an **IntVect**.

- **IntVectSet operator**| (const **IntVect** &iv, const **IntVectSet** &ivs)
  Returns a new IntVectSet that is the union of an IntVectSet and an **IntVect**.

- **IntVectSet operator**| (const **IntVectSet** &ivs, const **Box** &b)
  Returns a new IntVectSet that is the union of an IntVectSet and a **Box**.

- **IntVectSet operator**| (const **Box** &b, const **IntVectSet** &ivs)
  Returns a new IntVectSet that is the union of an IntVectSet and a **Box**.

Complement operators

- **IntVectSet operator-** (const **IntVectSet** &ivs) const
  Returns the complement of the IntVectSet ivs *within this IntVectSet.*

- **IntVectSet operator-** (const **Box** &b) const
  Returns the complement of the **Box** b *within this IntVectSet.*

- **IntVectSet operator-** (const **IntVect** &iv) const
  Returns the complement of the **IntVect** iv *within this IntVectSet.*

- **IntVectSet** & **operator-=** (const **IntVectSet** &ivs)
  Remove the IntVects in the IntVectSet ivs *from this IntVectSet.*

- void **minus** (const **IntVectSet** &ivs)
  *same as operator-= except it doesn't return *this*

- **IntVectSet** & **operator-=** (const **Box** &b)
  Remove the IntVects in the **Box** b *from this IntVectSet.*

- void **minus_box** (const **Box** &b)
  *same as operator-= except it doesn't return *this*

- **IntVectSet** & **operator-=** (const **IntVect** &iv)
  Remove the **IntVect** iv *from this IntVectSet.*

- void **minus_intvect** (const **IntVect** &iv)
  *same as operator-= except it doesn't return *this*

Intersection operators

- **IntVectSet** & **operator &=** (const **IntVectSet** &ivs)
  Modifies this IntVectSet to its intersection with another IntVectSet.

- void **and_intvectset** (const **IntVectSet** &ivs)
  *same as operator&= except it doesn't return *this*

- **IntVectSet** & **operator &=** (const **Box** &b)
  Modifies this IntVectSet to be its intersection with a **Box**.

- void **and_box** (const **Box** &b)
  *same as operator&= except it doesn't return *this*

- **IntVectSet** & **operator &=** (const **ProblemDomain** &domain)
  Modifies this IntVectSet to be its intersection with the **Box** *in a* **ProblemDomain**.

- void **and_domain** (const **ProblemDomain** &d)
  *same as operator&= except it doesn't return *this*

- **IntVectSet operator &** (const **IntVectSet** &ivs1, const **IntVectSet** &ivs2)

  Returns a new IntVectSet that is the intersection of two IntVectSets.

- **IntVectSet operator &** (const **IntVectSet** &ivs, const **Box** &b)

  Returns a new IntVectSet that is the intersection of an IntVectSet and a **Box**.

- **IntVectSet operator &** (const **Box** &b, const **IntVectSet** &ivs)

  Returns a new IntVectSet that is the intersection of an IntVectSet and a **Box**.

Modification functions

- void **grow** (int igrow)

  Add IntVects to this IntVectSet in all directions.

- **IntVectSet** & **grow** (int idir, int igrow)

  Add IntVects to this IntVectSet in one direction.

- void **grow_dir** (int idir, int igrow)

  same as `grow(idir,igrow)` *except doesn't return *this*

- **IntVectSet** & **refine** (int iref=2)

  Refine all the IntVects in this IntVecSet.

- **IntVectSet** & **coarsen** (int iref=2)

  Coarsen all the IntVects in this IntVectSet.

- void **shift** (const **IntVect** &iv)

  Increment all the IntVects in this IntVectSet by the **IntVect** iv.

- void **nestingRegion** (int radius, const **Box** &domain, int granularity=1)

  Make this IntVectSet be properly nested.

- void **nestingRegion** (int radius, const **ProblemDomain** &probdomain, int granularity=1)

  Make this IntVectSet be properly nested.

- void **nestingRegion_prob** (int radius, const **ProblemDomain** &probdomain)

  same as **nestingRegion()** *except it doesn't return *this*

- void **makeEmpty** ()

  Modifies this IntVectSet to be empty.

- **IntVectSet chop** (int dir, int chop_pnt)

  Chop the IntVectSet at the chop_pnt *index in the* dir *direction.*

- void **setMaxDense** (const int &a_maxDense)
- **IntVectSet grow** (const **IntVectSet** &ivs, int igrow)

  Add IntVects to an IntVectSet in all directions.

- **IntVectSet refine** (const **IntVectSet** &ivs, int iref=2)

  Refine all the IntVects in an IntVectSet.

- **IntVectSet coarsen** (const **IntVectSet** &ivs, int iref=2)

  Coarsen all the IntVects in an IntVectSet.

Accessor and Inquiry functions

- int **numPts** () const

  Returns the number of IntVects in this IntVectSet.

- const **Box** & **minBox** () const

  Returns the minimum enclosing box of this IntVectSet.

- void **recalcMinBox** () const

  Forces recalculation of the minimum enclosing box of this IntVectSet.

- bool **isEmpty** () const

  Returns true if no IntVects are in this IntVectSet.

- bool **isDense** () const

  Returns true if this IntVectSet is currently being represented in a dense fashion.

- bool **contains** (const **IntVect** &iv) const

  Returns true if this IntVectSet contains iv*.*

- bool **contains** (const **IntVectSet** &ivs) const

  Returns true if this IntVectSet contains all the IntVects in ivs.

- bool **contains_intvectset** (const **IntVectSet** &ivs) const

  same as `contains(ivs)`

- bool **contains** (const **Box** &box) const

  Returns true if this IntVectSet contains all the IntVects in box.

- bool **contains_box** (const **Box** &b) const

  same as `contains(b)`

- **Vector**< **Box** > **boxes** () const

  Returns a **Vector<Box>** *representation of this IntVectSet.*

- void **compact** () const

  Optimize memory usage of this IntVectSet.

Linearization functions

- int **linearSize** () const

  Returns the number of bytes in a linear representation of ∗this.

- void **linearIn** (const void ∗const a_inBuf)

  Modify ∗this using the data in the linear representation in a_inBuf*.*

- void **linearOut** (void ∗const a_outBuf) const

  Write a linear representation of ∗this to a_outBuf*.*

Public Member Functions

- bool **operator==** (const **IntVectSet** &a_ivs) const

  Returns true if this IntVectSet has the same IntVects as a_ivs*.*

- void **printBoxes** (std::ostream &a_ostream) const

  Writes a **Vector<Box>** *representation to an output stream.*

- void **p** () const

  Writes a **Vector<Box>** *representation to the parallel stdout.*

- void **convert** () const

Static Public Attributes

- long int **count**
- long int **peakcount**
- int **s_maxDense**

Friends

- class **IVSIterator**
- std::ostream & **operator<<** (std::ostream &os, const **IntVectSet** &ivs)

  Writes a text representation of an IntVectSet to an output stream.

- void **dumpmemoryatexit** ()

7.72.1 Detailed Description

An irregular domain on an integer lattice.

IntVectSet represents an irregular region in an integer lattice SpaceDim-dimensional index space as an arbitrary collection of IntVects. A full set calculus is defined. Any **IntVect** or cell-centered **Box** can be fully represented as an IntVectSet. There is an iterator that provides access to the contents of an IntVectSet. IntVectSets are implicitly cell-centered. Intersection, union, and complement operations are defined for pairs of IntVectSets (and, by extension, an IntVectSet and an **IntVect** or an IntVectSet and a **Box**). The minimum **Box** of an IntVectSet is defined as the smallest **Box** that contains every **IntVect** in the IntVectSet.

The IntVects in an IntVectSet do not have a canonical ordering.

7.72.2 Constructor & Destructor Documentation

7.72.2.1 IntVectSet::~IntVectSet ()

7.72.2.2 IntVectSet::IntVectSet () `[inline]`

default constructor: defines an empty IntVectSet.

7.72.2.3 IntVectSet::IntVectSet (const IntVectSet & *ivs*) `[inline]`

copy constructor

7.72.2.4 IntVectSet::IntVectSet (const DenseIntVectSet & *a_dense*) `[explicit]`

conversion constructor

7.72.2.5 IntVectSet::IntVectSet (const TreeIntVectSet & *a_tree*) `[explicit]`

conversion constructor

7.72.2.6 IntVectSet::IntVectSet (const IntVect & *iv*) `[explicit]`

IntVect constructor.

construct this to be an IntVectSet with just one **IntVect**.

7.72.2.7 IntVectSet::IntVectSet (const Box & *b*) `[explicit]`

Box constructor.

construct this to be an IntVectSet with all the IntVects in the **Box** .

7.72.3 Member Function Documentation

7.72.3.1 void IntVectSet::and_box (const Box & *b*) `[inline]`

same as operator&= except it doesn't return *this

7.72.3.2 void IntVectSet::and_domain (const ProblemDomain & *d*) `[inline]`

same as operator&= except it doesn't return *this

7.72.3.3 void IntVectSet::and_intvectset (const IntVectSet & *ivs*) `[inline]`

same as operator&= except it doesn't return *this

7.72.3.4 Vector<Box> IntVectSet::boxes () const

Returns a **Vector<Box>** representation of this IntVectSet.

The returned Boxes contain only the IntVects in this IntVectSet, so the union of the Boxes in the **Vector** == this IntVectSet.

7.72.3.5 IntVectSet IntVectSet::chop (int *dir*, int *chop_pnt*)

Chop the IntVectSet at the *chop_pnt* index in the *dir* direction.

Returns one IntVectSet and modifies this IntVectSet. The union of the two is the original IntVectSet. This IntVectSet gets the IntVects with *dir* index >= *chop_pnt*, and the returned IntVectSet gets the IntVects with *dir* index < *chop_pnt*. It is an error if *dir* is invalid.

7.72.3.6 IntVectSet& IntVectSet::coarsen (int *iref* = 2)

Coarsen all the IntVects in this IntVectSet.

Modifies this IntVectSet by coarsening it by the factor *iref*. It is an error if *iref* <= 0. Definition of coarsening: for each **IntVect** `iv` in the original IntVectSet, the refined IntVectSet will contain the **IntVect** defined by **coarsen(iv, iref)**.

7.72.3.7 void IntVectSet::compact () const

Optimize memory usage of this IntVectSet.

Attempts to optimize storage for IntVectSet. This will also optimize iteration through the IntVectSet later. Best to call it when you are no longer modifying the IntVectSet

7.72.3.8 bool IntVectSet::contains (const Box & *box*) const

Returns true if this IntVectSet contains all the IntVects in *box*.

Note: this is done using an algorithm that is much faster than looping through all the IntVects.

7.72.3.9 bool IntVectSet::contains (const IntVectSet & *ivs*) const

Returns true if this IntVectSet contains all the IntVects in *ivs*.

7.72.3.10 bool IntVectSet::contains (const IntVect & *iv*) const

Returns true if this IntVectSet contains *iv*.

7.72.3.11 bool IntVectSet::contains_box (const Box & *b*) const [inline]

same as `contains(b)`

7.72.3.12 bool IntVectSet::contains_intvectset (const IntVectSet & *ivs*) const [inline]

same as `contains(ivs)`

7.72.3.13 void IntVectSet::convert () const

7.72.3.14 IntVectSet IntVectSet::copy () const [inline]

return a copy of this IntVectSet

7.72.3.15 void IntVectSet::define (const Box & *b*)

Box define.

modifies this **IntVect** to have all the IntVects in the **Box** .

7.72.3.16 void IntVectSet::define (const IntVect & *iv*)

IntVect define: modifies this IntVectSet to have just one **IntVect**.

7.72.3.17 void IntVectSet::define (const TreeIntVectSet & *a_tree*)

conversion define

7.72.3.18 void IntVectSet::define (const DenseIntVectSet & *a_dense*)

conversion define

7.72.3.19 void IntVectSet::define (const IntVectSet & *ivs*) [inline]

copy define: modifies this IntVectSet into a copy of *ivs*

7.72.3.20 void IntVectSet::define ()

default define: modifies this IntVectSet to be empty.

7.72.3.21 void IntVectSet::define_box (const Box & *b*) [inline]

Box define: same as `define(Box)`.

7.72.3.22 void IntVectSet::define_intvect (const IntVect & *iv*) [inline]

IntVect define: same as `define(IntVect)`.

7.72.3.23 void IntVectSet::define_intvectset (const IntVectSet & *ivs*) [inline]

copy define: same as `define(IntVectSet)`

7.72.3.24 IntVectSet& IntVectSet::grow (int *idir*, int *igrow*)

Add IntVects to this IntVectSet in one direction.

Modifies this IntVectSet by growing it by the specified number of IntVects *igrow* in the specified coordinate direction *idir*. Directions are zero-based.

7.72.3.25 void IntVectSet::grow (int *igrow*)

Add IntVects to this IntVectSet in all directions.

Modifies this IntVectSet by growing it by the specified number of IntVects in all directions, including diagonal. It is an error if *igrow* < 0.

7.72.3.26 void IntVectSet::grow_dir (int *idir*, int *igrow*) `[inline]`

same as `grow(idir,igrow)` except doesn't return *this

7.72.3.27 bool IntVectSet::isDense () const `[inline]`

Returns true if this IntVectSet is currently being represented in a dense fashion.

7.72.3.28 bool IntVectSet::isEmpty () const

Returns true if no IntVects are in this IntVectSet.

7.72.3.29 void IntVectSet::linearIn (const void *const *a_inBuf*)

Modify *this using the data in the linear representation in *a_inBuf*.

7.72.3.30 void IntVectSet::linearOut (void *const *a_outBuf*) const

Write a linear representation of *this to *a_outBuf*.

Assumes a_outBuf is at least **linearSize()** bytes long.

7.72.3.31 int IntVectSet::linearSize () const

Returns the number of bytes in a linear representation of *this.

7.72.3.32 void IntVectSet::makeEmpty ()

Modifies this IntVectSet to be empty.

7.72.3.33 const Box& IntVectSet::minBox () const

Returns the minimum enclosing box of this IntVectSet.

7.72.3.34 void IntVectSet::minus (const IntVectSet & *ivs*) `[inline]`

same as operator-= except it doesn't return *this

7.72.3.35 void IntVectSet::minus_box (const Box & *b*) `[inline]`

same as operator-= except it doesn't return *this

7.72.3.36 void IntVectSet::minus_intvect (const IntVect & *iv*) `[inline]`

same as operator-= except it doesn't return *this

7.72.3.37 void IntVectSet::nestingRegion (int *radius*, const ProblemDomain & *probdomain*, int *granularity* = 1)

Make this IntVectSet be properly nested.

Modifies this IntVectSet to remove all IntVects that are not at least *radius* from the the edge of the IntVectSet in any direction. *radius* must be positive. IntVects that border non-periodic boundaries are spared from this trimming. Radius extends across periodic images.

7.72.3.38 void IntVectSet::nestingRegion (int *radius*, const Box & *domain*, int *granularity* = 1)

Make this IntVectSet be properly nested.

Modifies this IntVectSet to remove all IntVects that are not at least *radius* from the the edge of the IntVectSet in any direction. *radius* must be positive. IntVects that border the **Box** *domain* are spared from this trimming.

7.72.3.39 void IntVectSet::nestingRegion_prob (int *radius*, const ProblemDomain & *probdomain*) `[inline]`

same as **nestingRegion()** except it doesn't return *this

7.72.3.40 int IntVectSet::numPts () const

Returns the number of IntVects in this IntVectSet.

7.72.3.41 IntVectSet& IntVectSet::operator &= (const ProblemDomain & *domain*)

Modifies this IntVectSet to be its intersection with the **Box** in a **ProblemDomain**.

7.72.3.42 IntVectSet& IntVectSet::operator &= (const Box & *b*)

Modifies this IntVectSet to be its intersection with a **Box**.

7.72.3.43 IntVectSet& IntVectSet::operator &= (const IntVectSet & *ivs*)

Modifies this IntVectSet to its intersection with another IntVectSet.

7.72.3.44 IntVectSet IntVectSet::operator- (const IntVect & *iv*) const

Returns the complement of the **IntVect** *iv* within this IntVectSet.

7.72.3.45 IntVectSet IntVectSet::operator- (const Box & *b*) const

Returns the complement of the **Box** *b* within this IntVectSet.

7.72.3.46 IntVectSet IntVectSet::operator- (const IntVectSet & *ivs*) const

Returns the complement of the IntVectSet *ivs* within this IntVectSet.

7.72.3.47 IntVectSet& IntVectSet::operator-= (const IntVect & *iv*)

Remove the **IntVect** *iv* from this IntVectSet.

Modifies this IntVectSet to be the complement of the argument **IntVect** within this IntVectSet.

7.72.3.48 IntVectSet& IntVectSet::operator-= (const Box & *b*)

Remove the IntVects in the **Box** *b* from this IntVectSet.

Modifies this IntVectSet to be the complement of the argument **Box** within this IntVectSet.

7.72.3.49 IntVectSet& IntVectSet::operator-= (const IntVectSet & *ivs*)

Remove the IntVects in the IntVectSet *ivs* from this IntVectSet.

Modifies this IntVectSet to be the complement of the IntVectSet *ivs* within this IntVectSet.

7.72.3.50 bool IntVectSet::operator== (const IntVectSet & *a_ivs*) const

Returns true if this IntVectSet has the same IntVects as *a_ivs*.

7.72.3.51 IntVectSet& IntVectSet::operator\|= (const Box & *b*)

unions the IntVects in the **Box into** this IntVectSet

7.72.3.52 IntVectSet& IntVectSet::operator\|= (const IntVect & *iv*)

unions a single **IntVect** *iv* into this IntVectSet

7.72.3.53 IntVectSet& IntVectSet::operator\|= (const IntVectSet & *ivs*)

unions another IntVectSet *ivs* into this IntVectSet

7.72.3.54 void IntVectSet::or_box (const Box & *b*) [inline]

same as operator|= except it doesn't return *this

7.72.3.55 void IntVectSet::or_intvect (const IntVect & *ivs*) [inline]

same as operator|= except it doesn't return *this

7.72.3.56 void IntVectSet::or_intvectset (const IntVectSet & *ivs*) [inline]

same as operator|= except it doesn't return *this

7.72.3.57 void IntVectSet::p () const [inline]

Writes a **Vector<Box>** representation to the parallel stdout.

7.72.3.58 void IntVectSet::printBoxes (std::ostream & *a_ostream*) const

Writes a **Vector<Box>** representation to an output stream.

7.72.3.59 void IntVectSet::recalcMinBox () const

Forces recalculation of the minimum enclosing box of this IntVectSet.

7.72.3.60 IntVectSet& IntVectSet::refine (int *iref* = 2)

Refine all the IntVects in this IntVecSet.

Modifies this IntVectSet by refining it by the factor *iref*. It is an error if *iref* <= 0. Definition of refinement: for each **IntVect** `iv` in the original IntVectSet, the refined IntVectSet will contain the **Box** defined by **refine(Box(iv,iv), iref)**.

7.72.3.61 void IntVectSet::setMaxDense (const int & *a_maxDense*)

Set the max box size for IntVectSet::define(Box) which will make the IntVectSet dense. Default is 10^4.

7.72.3.62 void IntVectSet::shift (const IntVect & *iv*)

Increment all the IntVects in this IntVectSet by the **IntVect** *iv*.

7.72.4 Friends And Related Function Documentation

7.72.4.1 IntVectSet coarsen (const IntVectSet & *ivs*, int *iref* = 2) [friend]

Coarsen all the IntVects in an IntVectSet.

Creates a new IntVectSet that is a copy of the argument IntVectSet *ivs* coarsened by the factor *iref*. It is an error if *iref* <= 0. Definition of coarsening: for each **IntVect** `iv` in the original IntVectSet, the refined IntVectSet will contain the **IntVect** defined by **coarsen(iv, iref)**.

7.72.4.2 void dumpmemoryatexit () `[friend]`

7.72.4.3 IntVectSet grow (const IntVectSet & *ivs*, int *igrow*) `[friend]`

Add IntVects to an IntVectSet in all directions.

Creates a new IntVectSet that is a copy of the argument IntVectSet *ivs* grown by the specified number of IntVects in all directions *igrow*, including diagonal. It is an error if *igrow* < 0.

7.72.4.4 friend class IVSIterator `[friend]`

7.72.4.5 IntVectSet operator & (const Box & *b*, const IntVectSet & *ivs*) `[friend]`

Returns a new IntVectSet that is the intersection of an IntVectSet and a **Box**.

The result may be empty.

7.72.4.6 IntVectSet operator & (const IntVectSet & *ivs*, const Box & *b*) `[friend]`

Returns a new IntVectSet that is the intersection of an IntVectSet and a **Box**.

The result may be empty.

7.72.4.7 IntVectSet operator & (const IntVectSet & *ivs1*, const IntVectSet & *ivs2*) `[friend]`

Returns a new IntVectSet that is the intersection of two IntVectSets.

The result may be empty.

7.72.4.8 std::ostream& operator<< (std::ostream & *os*, const IntVectSet & *ivs*) `[friend]`

Writes a text representation of an IntVectSet to an output stream.

7.72.4.9 IntVectSet operator| (const Box & *b*, const IntVectSet & *ivs*) `[friend]`

Returns a new IntVectSet that is the union of an IntVectSet and a **Box**.

7.72.4.10 IntVectSet operator| (const IntVectSet & *ivs*, const Box & *b*) `[friend]`

Returns a new IntVectSet that is the union of an IntVectSet and a **Box**.

7.72.4.11 IntVectSet operator| (const IntVect & *iv*, const IntVectSet & *ivs*) `[friend]`

Returns a new IntVectSet that is the union of an IntVectSet and an **IntVect**.

7.72.4.12 IntVectSet operator| (const IntVectSet & *ivs*, const IntVect & *iv*) `[friend]`

Returns a new IntVectSet that is the union of an IntVectSet and an **IntVect**.

7.72.4.13 IntVectSet operator| (const IntVectSet & *ivs1*, const IntVectSet & *ivs2*) `[friend]`

Returns a new IntVectSet that is the union of two IntVectSets.

7.72.4.14 IntVectSet refine (const IntVectSet & *ivs*, int *iref* = 2) `[friend]`

Refine all the IntVects in an IntVectSet.

Creates a new IntVectSet that is a copy of the argument IntVectSet *ivs* refined by the factor *iref*. It is an error if *iref* <= 0. Definition of refinement: for each **IntVect** `iv` in the original IntVectSet *ivs*, the refined IntVectSet will contain the **Box** defined by **refine(Box(iv,iv), iref).**

7.72.5 Member Data Documentation

7.72.5.1 long int IntVectSet::count `[static]`

7.72.5.2 long int IntVectSet::peakcount `[static]`

7.72.5.3 int IntVectSet::s_maxDense `[static]`

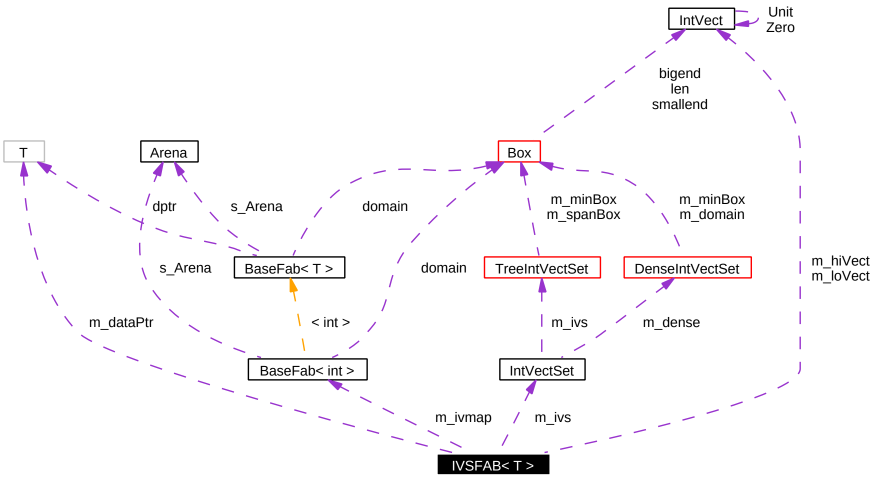
The documentation for this class was generated from the following file:

- **IntVectSet.H**

7.73 IVSFAB< T > Class Template Reference

#include <IVSFAB.H>

Collaboration diagram for IVSFAB< T >:



Public Member Functions

- **IVSFAB** ()
- **IVSFAB** (const **IntVectSet** &a_region, const int &a_nvarin)
- **~IVSFAB** ()
- void **define** (const **IntVectSet** &a_region, const int &a_nvarin)
- void **setVal** (const T &value)
- void **copy** (const **Box** &a_fromBox, const **Interval** &a_destInterval, const **Box** &a_toBox, const **IVSFAB**< T > &a_src, const **Interval** &a_srcInterval)
- int **size** (const **Box** &R, const **Interval** &comps) const
- void **linearOut** (void *buf, const **Box** &R, const **Interval** &comps) const
- void **linearIn** (void *buf, const **Box** &R, const **Interval** &comps)
- void **clear** ()
- bool **isDefined** () const
- int **numIvs** () const
- int **nComp** () const
- const **IntVectSet** & **getIVS** () const
- T & **operator()** (const **IntVect** &a_iv, const int &varlocin)
- const T & **operator()** (const **IntVect** &a_iv, const int &varlocin) const
- const T * **dataPtr** (const int &a_comp) const
- T * **dataPtr** (const int &a_comp)
- **IVSFAB** (const **Box** &a_region, int a_nVar)

  invalid but necessary for leveldata to compile

- const int * **loVect** () const
- const int * **hiVect** () const

Static Public Member Functions

- int **preAllocatable** ()
- void **setVerbose** (bool a_verbose)

Protected Member Functions

- int **getIndex** (const **IntVect** &a_iv, const int &a_comp) const

Protected Attributes

- T * **m_dataPtr**
- int **m_nComp**
- int **m_nIvs**
- **IntVect m_loVect**
- **IntVect m_hiVect**
- **BaseFab**< int > **m_ivmap**
- **IntVectSet m_ivs**
- bool **m_isDefined**

Static Protected Attributes

- bool **s_verbose** = false

7.73.1 Detailed Description

template<class T> class IVSFAB< T >

IVSFAB is a templated data holder defined at the Ivs of an irregular domain.

7.73.2 Constructor & Destructor Documentation

7.73.2.1 template<class T> IVSFAB< T >::IVSFAB () `[inline]`

Default constructor. Constructs an uninitialized IVFAB. You must subsequently call { define} before using this IVFAB.

7.73.2.2 template<class T> IVSFAB< T >::IVSFAB (const IntVectSet & *a_region*, const int & *a_nvarin*) `[inline]`

7.73.2.3 template<class T> IVSFAB< T >::~IVSFAB () `[inline]`

7.73.2.4 template<class T> IVSFAB< T >::IVSFAB (const Box & *a_region*, int *a_nVar*) `[inline]`

invalid but necessary for leveldata to compile

7.73.3 Member Function Documentation

7.73.3.1 template<class T> void IVSFAB< T >::clear () [inline]

Remove all data from this IVSFAB. You must call { define} again in order to use it.

7.73.3.2 template<class T> void IVSFAB< T >::copy (const Box & *a_fromBox*, const Interval & *a_destInterval*, const Box & *a_toBox*, const IVSFAB< T > & *a_src*, const Interval & *a_srcInterval*) [inline]

7.73.3.3 template<class T> T * IVSFAB< T >::dataPtr (const int & *a_comp*) [inline]

Return a pointer to the internal data storage of this IVFAB.

7.73.3.4 template<class T> const T * IVSFAB< T >::dataPtr (const int & *a_comp*) const [inline]

Return a const pointer to the internal data storage of this IVFAB. \

7.73.3.5 template<class T> void IVSFAB< T >::define (const IntVectSet & *a_region*, const int & *a_nvarin*) [inline]

7.73.3.6 template<class T> int IVSFAB< T >::getIndex (const IntVect & *a_iv*, const int & *a_comp*) const [inline, protected]

7.73.3.7 template<class T> const IntVectSet & IVSFAB< T >::getIVS () const [inline]

Return the irregular domain of the IVSFAB.

7.73.3.8 template<class T> const int * IVSFAB< T >::hiVect () const [inline]

Return a fake "upper right corner of the domain" for use with Fortran.

7.73.3.9 template<class T> bool IVSFAB< T >::isDefined () const [inline]

Tells whether this IVSFAB has been defined, either with a constructor or with { define}. It must be initialized in order to access its data.

7.73.3.10 template<class T> void IVSFAB< T >::linearIn (void * *buf*, const Box & *R*, const Interval & *comps*) [inline]

7.73.3.11 template<class T> void IVSFAB< T >::linearOut (void * *buf*, const Box & *R*, const Interval & *comps*) const [inline]

7.73.3.12 template<class T> const int * IVSFAB< T >::loVect () const [inline]

Return a fake "lower left corner of the domain" for use with Fortran.

7.73.3.13 template<class T> int IVSFAB< T >::nComp () const [inline]

Return the number of data components of this IVSFAB.

7.73.3.14 template<class T> int IVSFAB< T >::numIvs () const [inline]

Return the number of Ivs in this IVSFAB.

7.73.3.15 template<class T> const T & IVSFAB< T >::operator() (const IntVect & *a_iv*, const int & *varlocin*) const [inline]

7.73.3.16 template<class T> T & IVSFAB< T >::operator() (const IntVect & *a_iv*, const int & *varlocin*) [inline]

Indexing operator. Return a reference to the contents of this IVFAB, at the specified Iv and data component. The first component is zero, the last is {*nvar-1*}. The returned object is a modifiable lvalue.

7.73.3.17 template<class T> int IVSFAB< T >::preAllocatable () [inline, static]

This stuff required by **LevelData** in parallel:

7.73.3.18 template<class T> void IVSFAB< T >::setVal (const T & *value*) [inline]

Set a value at every data location.

7.73.3.19 template<class T> void IVSFAB< T >::setVerbose (bool *a_verbose*) [static]

7.73.3.20 template<class T> int IVSFAB< T >::size (const Box & *R*, const Interval & *comps*) const [inline]

7.73.4 Member Data Documentation

7.73.4.1 template<class T> T* IVSFAB< T >::m_dataPtr [protected]

7.73.4.2 template<class T> IntVect IVSFAB< T >::m_hiVect [protected]

7.73.4.3 template<class T> bool IVSFAB< T >::m_isDefined [protected]

7.73.4.4 template<class T> BaseFab<int> IVSFAB< T >::m_ivmap [protected]

7.73.4.5 template<class T> IntVectSet IVSFAB< T >::m_ivs [protected]

7.73.4.6 template<class T> IntVect IVSFAB< T >::m_loVect [protected]

7.73.4.7 template<class T> int IVSFAB< T >::m_nComp [protected]

7.73.4.8 template<class T> int IVSFAB< T >::m_nIvs [protected]

7.73.4.9 template<class T> bool IVSFAB< T >::s_verbose = false [static, protected]

The documentation for this class was generated from the following files:

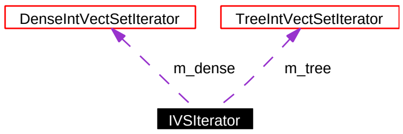
- **IVSFAB.H**
- **IVSFABI.H**

7.74 IVSIterator Class Reference

Iterator for an **IntVectSet**.

`#include <IntVectSet.H>`

Collaboration diagram for IVSIterator:



Public Member Functions

- **IVSIterator** ()
- **IVSIterator** (const **IntVectSet** &ivs)
- **~IVSIterator** ()
- void **define** (const **IntVectSet** &ivs)
- const **IntVect** & **operator()** () const

  returns the **IntVect** *that this iterator is at*

- const **IntVect** & **iv** () const

  same as operator()

- bool **ok** () const

  returns true if this iterator is still in its **IntVectSet**

- void **operator++** ()

  move iterator to the next **IntVect** *in its* **IntVectSet**

- void **incr** ()

  same as operator++

- void **begin** ()

  initialize this iterator to the first **IntVect** *in its* **IntVectSet**

- void **reset** ()

  same as **begin()**

- void **end** ()

  move this iterator to after the last **IntVect** *in the set*

7.74.1 Detailed Description

Iterator for an **IntVectSet**.

IVSIterator iterates over every point (**IntVect**) in an **IntVectSet**. It has exactly the same syntax and sematic as **BoxIterator**. Typical usage:

```
IntVectSet ivs;
...
IVSIterator ivsit (ivs);
for (ivsit.begin(); ivsit.ok(); ++ivsit)
{
  IntVect iv = ivsit();
  (do operations involving iv)
}
```

7.74.2 Constructor & Destructor Documentation

7.74.2.1 IVSIterator::IVSIterator () [inline]

A default constructed iterator iterates over an empty **IntVectSet**. It starts in the **begin()** state, and is never **ok()**.

7.74.2.2 IVSIterator::IVSIterator (const IntVectSet & *ivs*)

Iterates over **IntVectSet** *ivs*. There is no canonical ordering. Starts in the **begin()** state.

7.74.2.3 IVSIterator::~IVSIterator () [inline]

7.74.3 Member Function Documentation

7.74.3.1 void IVSIterator::begin () [inline]

initialize this iterator to the first **IntVect** in its **IntVectSet**

7.74.3.2 void IVSIterator::define (const IntVectSet & *ivs*)

Modifies this IVSIterator to iterate over the **IntVectSet** *ivs*. There is no canonical ordering. Starts in the **begin()** state.

7.74.3.3 void IVSIterator::end () [inline]

move this iterator to after the last **IntVect** in the set

The iterator will be !ok() afterwards.

7.74.3.4 void IVSIterator::incr () [inline]

same as operator++

7.74.3.5 const IntVect& IVSIterator::iv () const [inline]

same as operator()

7.74.3.6 bool IVSIterator::ok () const [inline]

returns true if this iterator is still in its **IntVectSet**

7.74.3.7 const IntVect & IVSIterator::operator() () const [inline]

returns the **IntVect** that this iterator is at

7.74.3.8 void IVSIterator::operator++ () [inline]

move iterator to the next **IntVect** in its **IntVectSet**

7.74.3.9 void IVSIterator::reset () [inline]

same as **begin()**

The documentation for this class was generated from the following file:

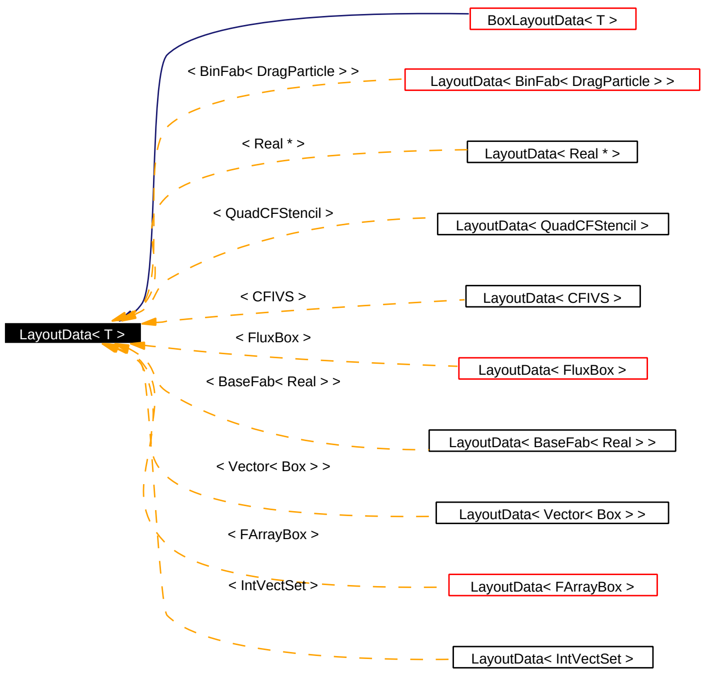
- **IntVectSet.H**

7.75 LayoutData< T > Class Template Reference

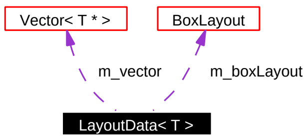
Data that maintains a one-to-one mapping to the boxes in a **BoxLayout**.

`#include <LayoutData.H>`

Inheritance diagram for LayoutData< T >:



Collaboration diagram for LayoutData< T >:



Public Member Functions

- **LayoutData** ()
- **LayoutData** (const **BoxLayout** &dp)
- virtual ~**LayoutData** ()
- virtual void **define** (const **BoxLayout** &dp)
- **DataIterator dataIterator** () const
- const T & **operator**[] (const **DataIndex** &a_index) const

    const accessor function

- const T & **operator**[] (const **DataIterator** &a_iterator) const

    const accessor function

- T & **operator**[ ] (const **DataIndex** &a_index)

    non-const accessor function

- T & **operator**[ ] (const **DataIterator** &a_iterator)

    non-const accessor function

- **Box box** (const **DataIndex** &a_index) const
- **Box box** (const **DataIterator** &a_iterator) const
- const **BoxLayout** & **boxLayout** () const

Protected Attributes

- **BoxLayout m_boxLayout**
- **Vector**< T * > **m_vector**
- bool **m_callDelete**

Friends

- class **ArrayViewData**

7.75.1 Detailed Description

template<class T> class LayoutData< T >

Data that maintains a one-to-one mapping to the boxes in a **BoxLayout**.

A collection of Box-oriented objects. The arrangement of the T objects is given by the underlying **BoxLayout** object. LayoutData does not attempt to prevent users from manipulating the template class T to become out of synch with the boxes in the **BoxLayout**. Caveat emptor.

Non-local (off-processor access) to a LayoutData index is an error. you can assure that access to the data stored in a LayoutData is local to your processor by using the **DataIterator** object for indexing. (In fact, you are pretty much forced to by the

Data in a LayoutData CANNOT be communicated to other processors using the API presented here.

class T must provide the following methods:

```
{
  T()
}
```

This class is useful for processor-local data that needs to be indexable along with a **BoxLayoutData**. Auxillary data objects, etc.

7.75.2 Constructor & Destructor Documentation

7.75.2.1 template<class T> LayoutData< T >::LayoutData () `[inline]`

7.75.2.2 template<class T> LayoutData< T >::LayoutData (const BoxLayout & *dp*) `[inline]`

constructor. allocates a T object for every box in the **BoxLayout** dp using the T null constructor. dp must be closed.

7.75.2.3 template<class T> LayoutData< T >::~LayoutData () `[inline, virtual]`

7.75.3 Member Function Documentation

7.75.3.1 template<class T> Box LayoutData< T >::box (const DataIterator & *a_iterator*) const `[inline]`

7.75.3.2 template<class T> Box LayoutData< T >::box (const DataIndex & *a_index*) const `[inline]`

7.75.3.3 template<class T> const BoxLayout& LayoutData< T >::boxLayout () const `[inline]`

7.75.3.4 template<class T> DataIterator LayoutData< T >::dataIterator () const `[inline]`

7.75.3.5 template<class T> void LayoutData< T >::define (const BoxLayout & *dp*) `[inline, virtual]`

constructor. allocates a T object for every box in the **BoxLayout** dp using the T null constructor. dp must be closed. previously stored T objects are removed

Reimplemented in **BoxLayoutData< T >** (p. 156), **LevelData< T >** (p. 358), **BoxLayoutData< BinFab< DragParticle > >** (p. 156), **BoxLayoutData< FluxBox >** (p. 156), **BoxLayoutData< FArrayBox >** (p. 156), **LevelData< BinFab< DragParticle > >** (p. 358), **LevelData< FluxBox >** (p. 358), and **LevelData< FArrayBox >** (p. 358).

7.75.3.6 template<class T> T & LayoutData< T >::operator[] (const DataIterator & *a_iterator*) `[inline]`

non-const accessor function

7.75.3.7 template<class T> T & LayoutData< T >::operator[] (const DataIndex & *a_index*) `[inline]`

non-const accessor function

7.75.3.8 template<class T> const T & LayoutData< T >::operator[] (const DataIterator & *a_iterator*) const `[inline]`

const accessor function

returns member for which **DataIterator** is currently indexed to. equivalent to layout[iterator()]

7.75.3.9 template<class T> const T & LayoutData< T >::operator[] (const DataIndex & *a_index*) const [inline]

const accessor function

7.75.4 Friends And Related Function Documentation

7.75.4.1 template<class T> friend class ArrayViewData [friend]

7.75.5 Member Data Documentation

7.75.5.1 template<class T> BoxLayout LayoutData< T >::m_boxLayout [protected]

7.75.5.2 template<class T> bool LayoutData< T >::m_callDelete [protected]

7.75.5.3 template<class T> Vector<T*> LayoutData< T >::m_vector [protected]

The documentation for this class was generated from the following files:

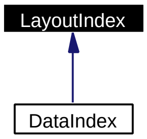
- **LayoutData.H**
- **LayoutDataI.H**

7.76 LayoutIndex Class Reference

An index for **LayoutIterator**.

`#include <DataIndex.H>`

Inheritance diagram for LayoutIndex:



Public Member Functions

- **LayoutIndex** ()
- **~LayoutIndex** ()
- bool **operator==** (const **LayoutIndex** &rhs) const
- bool **eq** (const **LayoutIndex** &rhs) const
- bool **operator!=** (const **LayoutIndex** &rhs) const
- bool **isNull** () const

  returns 'true' if this **DataIndex** *has been null constructed.*

- int **intCode** () const

Friends

- class **DataIterator**
- class **LayoutIterator**
- class **BoxLayout**
- class **DataIndex**

7.76.1 Detailed Description

An index for **LayoutIterator**.

7.76.2 Constructor & Destructor Documentation

7.76.2.1 LayoutIndex::LayoutIndex () `[inline]`

7.76.2.2 LayoutIndex::~LayoutIndex () `[inline]`

7.76.3 Member Function Documentation

7.76.3.1 bool LayoutIndex::eq (const LayoutIndex & *rhs*) const `[inline]`

7.76.3.2 int LayoutIndex::intCode () const `[inline]`

7.76.3.3 bool LayoutIndex::isNull () const `[inline]`

returns 'true' if this **DataIndex** has been null constructed.

7.76.3.4 bool LayoutIndex::operator!= (const LayoutIndex & *rhs*) const `[inline]`

7.76.3.5 bool LayoutIndex::operator== (const LayoutIndex & *rhs*) const `[inline]`

7.76.4 Friends And Related Function Documentation

7.76.4.1 friend class BoxLayout `[friend]`

Reimplemented in **DataIndex** (p. 207).

7.76.4.2 friend class DataIndex `[friend]`

7.76.4.3 friend class DataIterator `[friend]`

Reimplemented in **DataIndex** (p. 208).

7.76.4.4 friend class LayoutIterator `[friend]`

Reimplemented in **DataIndex** (p. 208).

The documentation for this class was generated from the following file:

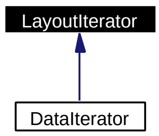
- **DataIndex.H**

7.77 LayoutIterator Class Reference

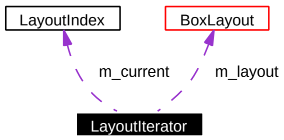
An Iterator based on a **BoxLayout** object.

#include <LayoutIterator.H>

Inheritance diagram for LayoutIterator:



Collaboration diagram for LayoutIterator:



Public Member Functions

- **LayoutIterator** ()

    a null constructed LayoutIterator will return false on **ok()**

- **LayoutIterator** (const **BoxLayout** &a_layout)
- **~LayoutIterator** ()
- const **LayoutIndex** & **operator()** () const

    return the index that this iterator is at

- **LayoutIndex i** () const

    return a copy of the index that this iterator is at

- void **operator++** ()

    move the iterator to the next **Box** *in the layout*

- void **incr** ()

    move the iterator to the next **Box** *in the layout*

- bool **ok** () const

    return true if this iterator is still in its Layout

- void **begin** ()

    initialize this iterator to the first **Box** *in its Layout*

- void **reset** ()

    same as **begin()**

- void **end** ()

  move this iterator to after the last **Box** *in the layout*

Protected Member Functions

- **LayoutIterator** (const **BoxLayout** &boxlayout, const int *layoutID)

Protected Attributes

- **BoxLayout m_layout**
- unsigned int **m_index**
- **LayoutIndex m_current**

Friends

- class **BoxLayout**

7.77.1 Detailed Description

An Iterator based on a **BoxLayout** object.

An Iterator based on a **BoxLayout** object. It does not support a dereferencing operation(1), since it is intended to work with all of BoxLayouts, DisjointBoxLayouts, BoxLayoutDatas LevelData's, and any object that is built on top of a **BoxLayout** object. LayoutIterator accesses the data in a BoxLayout-based object in a NON-data-parallel manner (i.e. every processor iterates through **all** the Boxes in the **BoxLayout**). This differs from the **DataIterator** class.

BoxLayout-based objects can act as the Factory for the LayoutIterator.

(1) STL-speak. not critical for comprehension, but can help people familiar with STL iterators and expecting similar behaviour.

7.77.2 Constructor & Destructor Documentation

7.77.2.1 LayoutIterator::LayoutIterator () `[inline]`

a null constructed LayoutIterator will return false on **ok()**

7.77.2.2 LayoutIterator::LayoutIterator (const BoxLayout & *a_layout*) [inline]

7.77.2.3 LayoutIterator::~LayoutIterator () [inline]

7.77.2.4 LayoutIterator::LayoutIterator (const BoxLayout & *boxlayout*, const int * *layoutID*) [protected]

7.77.3 Member Function Documentation

7.77.3.1 void LayoutIterator::begin () [inline]

initialize this iterator to the first **Box** in its Layout

7.77.3.2 void LayoutIterator::end ()

move this iterator to after the last **Box** in the layout

The iterator will be !ok() afterwards.

7.77.3.3 LayoutIndex LayoutIterator::i () const [inline]

return a copy of the index that this iterator is at

Reimplemented in **DataIterator** (p. 209).

7.77.3.4 void LayoutIterator::incr () [inline]

move the iterator to the next **Box** in the layout

7.77.3.5 bool LayoutIterator::ok () const [inline]

return true if this iterator is still in its Layout

7.77.3.6 const LayoutIndex & LayoutIterator::operator() () const [inline]

return the index that this iterator is at

Reimplemented in **DataIterator** (p. 210).

7.77.3.7 void LayoutIterator::operator++ () [inline]

move the iterator to the next **Box** in the layout

7.77.3.8 void LayoutIterator::reset ()

same as **begin()**

7.77.4 Friends And Related Function Documentation

7.77.4.1 friend class BoxLayout [friend]

Reimplemented in **DataIterator** (p. 210).

7.77.5 Member Data Documentation

7.77.5.1 LayoutIndex LayoutIterator::m_current [protected]

7.77.5.2 unsigned int LayoutIterator::m_index [protected]

7.77.5.3 BoxLayout LayoutIterator::m_layout [protected]

The documentation for this class was generated from the following file:

- **LayoutIterator.H**

7.78 LDOperator< T > Class Template Reference

`#include <LevelData.H>`

Public Member Functions

- virtual int **size** (const T &arg, const **Box** &b, const **Interval** &comps) const
- virtual void **linearOut** (const T &arg, void *buf, const **Box** &R, const **Interval** &comps) const
- virtual void **linearIn** (T &arg, void *buf, const **Box** &R, const **Interval** &comps) const
- virtual void **op** (T &dest, const **Box** &RegionFrom, const **Interval** &Cdest, const **Box** &RegionTo, const T &src, const **Interval** &Csrc) const

template<class T> class LDOperator< T >

7.78.1 Member Function Documentation

7.78.1.1 template<class T> virtual void LDOperator< T >::linearIn (T & *arg*, void * *buf*, const Box & *R*, const Interval & *comps*) const `[inline, virtual]`

7.78.1.2 template<class T> virtual void LDOperator< T >::linearOut (const T & *arg*, void * *buf*, const Box & *R*, const Interval & *comps*) const `[inline, virtual]`

7.78.1.3 template<class T> virtual void LDOperator< T >::op (T & *dest*, const Box & *RegionFrom*, const Interval & *Cdest*, const Box & *RegionTo*, const T & *src*, const Interval & *Csrc*) const `[inline, virtual]`

7.78.1.4 template<class T> virtual int LDOperator< T >::size (const T & *arg*, const Box & *b*, const Interval & *comps*) const `[inline, virtual]`

The documentation for this class was generated from the following file:

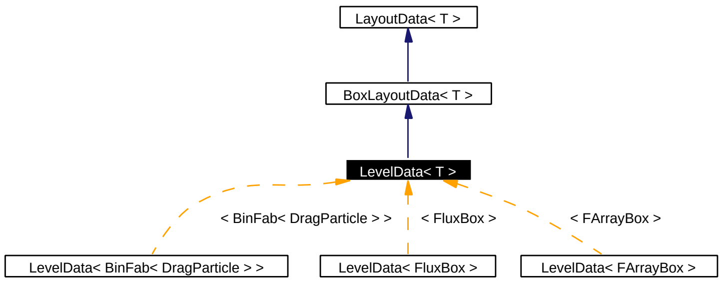
- **LevelData.H**

7.79 LevelData< T > Class Template Reference

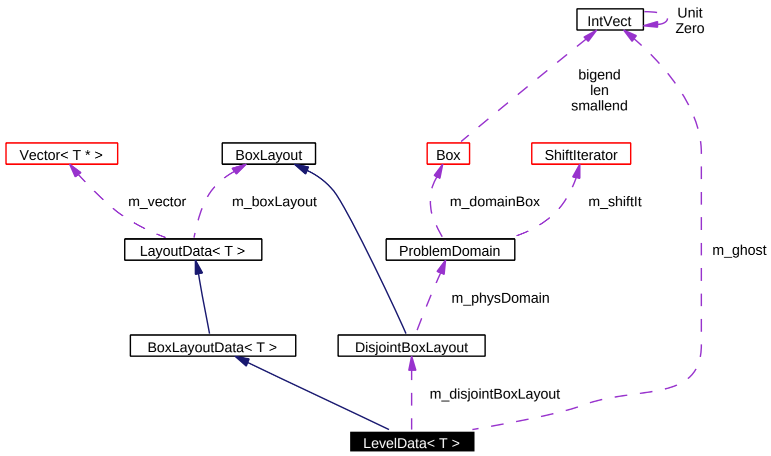
Data over a disjoint union of rectangles.

`#include <LevelData.H>`

Inheritance diagram for LevelData< T >:



Collaboration diagram for LevelData< T >:



Public Member Functions

- **LevelData** ()
- **LevelData** (const **DisjointBoxLayout** &dp, int comps, const **IntVect** &ghost=IntVect::TheZeroVector(), const **DataFactory**< T > &a_factory=**DefaultDataFactory**< T >())
- virtual ~**LevelData** ()
- virtual void **define** (const **DisjointBoxLayout** &dp, int comps, const **IntVect** &ghost=IntVect::TheZeroVector(), const **DataFactory**< T > &a_factory=**DefaultDataFactory**< T >())
- virtual void **define** (const **LevelData**< T > &da, const **DataFactory**< T > &a_factory=**DefaultDataFactory**< T >())
- virtual void **define** (const **LevelData**< T > &da, const **Interval** &comps, const **DataFactory**< T > &a_factory=**DefaultDataFactory**< T >())

- virtual void **copyTo** (const **Interval** &srcComps, **BoxLayoutData**< T > &dest, const **Interval** &destComps) const
- virtual void **copyTo** (const **Interval** &srcComps, **BoxLayoutData**< T > &dest, const **Interval** &destComps, const **Copier** &copier) const
- virtual void **copyTo** (const **Interval** &srcComps, **LevelData**< T > &dest, const **Interval** &destComps) const
- virtual void **copyTo** (const **Interval** &srcComps, **LevelData**< T > &dest, const **Interval** &destComps, const **Copier** &copier, const **LDOperator**< T > &a_op=**LDOperator**< T >()) const
- virtual void **exchange** (const **Interval** &comps)
- virtual void **exchange** (const **Interval** &comps, const **Copier** &copier)
- const **IntVect** & **ghostVect** () const
- const **DisjointBoxLayout** & **getBoxes** () const
- const **DisjointBoxLayout** & **disjointBoxLayout** () const
- virtual void **apply** (void(*a_Function)(const **Box** &box, int comps, T &t))

overidden virtual functions

*These functions will invoke error messages when invoked. C++ will ensure that constructors are not called for the base class by a user, but a 'define' function has no such protection, hence the need to prevent such usage. A runtime error is not a perfect solution...(strong construction gets around this *sigh*). classes that derive from* **LevelData** *will have to turn it's valid defines into runtime errors also and make it's own defines. Thus taking over the job of the compiler.*

- virtual void **define** (const **BoxLayout** &dp, int comps, const **DataFactory**< T > &factory)
- virtual void **define** (const **BoxLayoutData**< T > &da, const **DataFactory**< T > &factory=**DefaultDataFactory**< T >())
- virtual void **define** (const **BoxLayoutData**< T > &da, const **Interval** &comps, const **DataFactory**< T > &factory=**DefaultDataFactory**< T >())
- virtual void **define** (const **BoxLayout** &deadFunction)

  overridden and throws an error.

Protected Member Functions

- void **makeItSo** (const **Interval** &a_srcComps, const **LevelData**< T > &a_src, **BoxLayoutData**< T > &a_dest, const **Interval** &a_destComps, const **Copier** &a_copier, const **LDOperator**< T > &a_op=**LDOperator**< T >()) const
- void **completePendingSends** () const
- void **allocateBuffers** (const **LevelData**< T > &a_src, const **Interval** &a_srcComps, const **BoxLayoutData**< T > &a_dest, const **Interval** &a_destComps, const **Copier** &a_copier, const **LDOperator**< T > &a_op) const
- void **writeSendDataFromMeIntoBuffers** (const **LevelData**< T > &a_src, const **Interval** &a_srcComps, const **LDOperator**< T > &a_op) const
- void **postSendsFromMe** () const
- void **postReceivesToMe** () const
- void **unpackReceivesToMe** (**BoxLayoutData**< T > &a_dest, const **Interval** &a_destComps, const **LDOperator**< T > &a_op) const

Protected Attributes

- **DisjointBoxLayout m_disjointBoxLayout**
- **IntVect m_ghost**

Parallel messaging members

- void * **m_sendbuffer**
- size_t **m_sendcapacity**
- void * **m_recbuffer**
- size_t **m_reccapacity**

7.79.1 Detailed Description

template<class T> class LevelData< T >

Data over a disjoint union of rectangles.

LevelData is-a **BoxLayoutData**. It is built from a **DisjointBoxLayout** instead though, thus making the data in a **DisjointBoxLayout** uniquely defined for all spatial indices defined by the underlying **DisjointBoxLayout**. It carries within it an inherent concept of ghost cells (which the user can choose to be empty).

Since LevelData is-a **BoxLayoutData**, all the methods required of the template class T for **BoxLayoutData** are also required for LevelData. LevelData does not have any extra requirements.

7.79.2 Constructor & Destructor Documentation

7.79.2.1 template<class T> LevelData< T >::LevelData ()

7.79.2.2 template<class T> LevelData< T >::LevelData (const DisjointBoxLayout & *dp*, int *comps*, const IntVect & *ghost* = IntVect::TheZeroVector(), const DataFactory< T > & *a_factory* = DefaultDataFactory< T >())

7.79.2.3 template<class T> LevelData< T >::~LevelData () `[virtual]`

7.79.3 Member Function Documentation

7.79.3.1 template<class T> void LevelData< T >::allocateBuffers (const LevelData< T > & *a_src*, const Interval & *a_srcComps*, const BoxLayoutData< T > & *a_dest*, const Interval & *a_destComps*, const Copier & *a_copier*, const LDOperator< T > & *a_op*) const `[protected]`

7.79.3.2 template<class T> void LevelData< T >::apply (void(* *a_Function*)(const Box &box, int comps, T &t)) `[virtual]`

User writes a function with the signature:

```
void myfunction(const Box& box, int comps, T& t){ your code here;}
```

They can then hand this off to LayoutData::apply. This class then cycles through all the T objects and invokes this function. Function must not be inline. (I'm still trying to figure out a nice way to send in non-static member functions).

for example:

```
LevelData<FArrayBox> data(layout, 3, IntVect::Unit);
struct val{
static void set1(const Box& box, int comps, const FArrayBox& f)
{
  f.setVal(box, box.smallEnd(0));
}

data.apply(val::set1);
```

Reimplemented from **BoxLayoutData< T >** (p. 156).

7.79.3.3 template<class T> void LevelData< T >::completePendingSends () const [protected]

7.79.3.4 template<class T> void LevelData< T >::copyTo (const Interval & *srcComps*, LevelData< T > & *dest*, const Interval & *destComps*, const Copier & *copier*, const LDOperator< T > & *a_op* = LDOperator< T >()) const [virtual]

same as copyTo that takes a **BoxLayoutData**, except that it fills the ghost cells of 'dest' with data from 'this' also. USer passes in a prebuilt **Copier** object

7.79.3.5 template<class T> void LevelData< T >::copyTo (const Interval & *srcComps*, LevelData< T > & *dest*, const Interval & *destComps*) const [virtual]

same as copyTo that takes a **BoxLayoutData**, except that it fills the ghost cells of 'dest' with data from 'this' also.

7.79.3.6 template<class T> void LevelData< T >::copyTo (const Interval & *srcComps*, BoxLayoutData< T > & *dest*, const Interval & *destComps*, const Copier & *copier*) const [virtual]

7.79.3.7 template<class T> void LevelData< T >::copyTo (const Interval & *srcComps*, BoxLayoutData< T > & *dest*, const Interval & *destComps*) const [virtual]

7.79.3.8 template<class T> void LevelData< T >::define (const BoxLayout & *deadFunction*) [virtual]

overridden and throws an error.

Reimplemented from **BoxLayoutData< T >** (p. 156).

7.79.3.9 template<class T> void LevelData< T >::define (const BoxLayoutData< T > & *da*, const Interval & *comps*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) [virtual]

Define this **BoxLayoutData** to be the subset of the data in da defined by the **Interval** comps.

Reimplemented from **BoxLayoutData< T >** (p. 156).

7.79.3.10 template<class T> void LevelData< T >::define (const BoxLayoutData< T > & *da*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) `[virtual]`

Reimplemented from **BoxLayoutData< T >** (p. 156).

7.79.3.11 template<class T> void LevelData< T >::define (const BoxLayout & *dp*, int *comps*, const DataFactory< T > & *factory*) `[virtual]`

Reimplemented from **BoxLayoutData< T >** (p. 157).

7.79.3.12 template<class T> void LevelData< T >::define (const LevelData< T > & *da*, const Interval & *comps*, const DataFactory< T > & *a_factory* = DefaultDataFactory< T >()) `[virtual]`

Copy definer. 'this' LevelData thrown away and da's LevelData copied. This LevelData is now defined for the range [0, comps.size()] and filled with the data in da from [comps.begin(), comps.end()]

7.79.3.13 template<class T> void LevelData< T >::define (const LevelData< T > & *da*, const DataFactory< T > & *a_factory* = DefaultDataFactory< T >()) `[virtual]`

copy definer. this LevelData thrown away and da's LevelData copied

7.79.3.14 template<class T> void LevelData< T >::define (const DisjointBoxLayout & *dp*, int *comps*, const IntVect & *ghost* = IntVect::TheZeroVector(), const DataFactory< T > & *a_factory* = DefaultDataFactory< T >()) `[virtual]`

7.79.3.15 template<class T> const DisjointBoxLayout& LevelData< T >::disjointBoxLayout () const `[inline]`

7.79.3.16 template<class T> void LevelData< T >::exchange (const Interval & *comps*, const Copier & *copier*) `[virtual]`

7.79.3.17 template<class T> void LevelData< T >::exchange (const Interval & *comps*) `[virtual]`

7.79.3.18 template<class T> const DisjointBoxLayout& LevelData< T >::getBoxes () const `[inline]`

7.79.3.19 template<class T> const IntVect& LevelData< T >::ghostVect () const `[inline]`

7.79.3.20 template<class T> void LevelData< T >::makeItSo (const Interval & *a_srcComps*, const LevelData< T > & *a_src*, BoxLayoutData< T > & *a_dest*, const Interval & *a_destComps*, const Copier & *a_copier*, const LDOperator< T > & *a_op* = LDOperator< T >()) const `[protected]`

7.79.3.21 template<class T> void LevelData< T >::postReceivesToMe () const `[protected]`

7.79.3.22 template<class T> void LevelData< T >::postSendsFromMe () const `[protected]`

7.79.3.23 template<class T> void LevelData< T >::unpackReceivesToMe (BoxLayoutData< T > & *a_dest*, const Interval & *a_destComps*, const LDOperator< T > & *a_op*) const `[protected]`

7.79.3.24 template<class T> void LevelData< T >::writeSendDataFromMeIntoBuffers (const LevelData< T > & *a_src*, const Interval & *a_srcComps*, const LDOperator< T > & *a_op*) const `[protected]`

7.79.4 Member Data Documentation

7.79.4.1 template<class T> DisjointBoxLayout LevelData< T >::m_disjointBoxLayout `[protected]`

7.79.4.2 template<class T> IntVect LevelData< T >::m_ghost `[protected]`

7.79.4.3 template<class T> void* LevelData< T >::m_recbuffer `[mutable, protected]`

7.79.4.4 template<class T> size_t LevelData< T >::m_reccapacity `[mutable, protected]`

7.79.4.5 template<class T> void* LevelData< T >::m_sendbuffer `[mutable, protected]`

7.79.4.6 template<class T> size_t LevelData< T >::m_sendcapacity `[mutable, protected]`

The documentation for this class was generated from the following files:

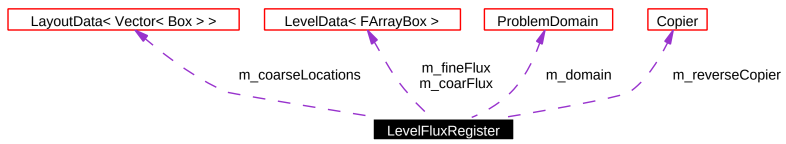
- **LevelData.H**
- **LevelDataI.H**

7.80 LevelFluxRegister Class Reference

LevelFluxRegister-A class to encapsulate a levels worth of flux registers.

`#include <LevelFluxRegister.H>`

Collaboration diagram for LevelFluxRegister:



Public Member Functions

- **LevelFluxRegister** ()
- **LevelFluxRegister** (const **DisjointBoxLayout** &a_dbl, const **DisjointBoxLayout** &a_dblCoarse, const **Box** &a_dProblem, int a_nRefine, int a_nComp)
- **LevelFluxRegister** (const **DisjointBoxLayout** &a_dbl, const **DisjointBoxLayout** &a_dblCoarse, const **ProblemDomain** &a_dProblem, int a_nRefine, int a_nComp)
- **~LevelFluxRegister** ()
- void **define** (const **DisjointBoxLayout** &a_dbl, const **DisjointBoxLayout** &a_dblCoarse, const **Box** &a_dProblem, int a_nRefine, int a_nComp)
- void **define** (const **DisjointBoxLayout** &a_dbl, const **DisjointBoxLayout** &a_dblCoarse, const **ProblemDomain** &a_dProblem, int a_nRefine, int a_nComp)
- void **undefine** ()
- void **setToZero** ()
- void **incrementCoarse** (**FArrayBox** &a_coarseFlux, **Real** a_scale, const **DataIndex** &a_coarseDataIndex, const **Interval** &a_srcInterval, const **Interval** &a_dstInterval, int a_dir)
- void **incrementCoarse** (**FArrayBox** &a_coarseFlux, **Real** a_scale, const **DataIndex** &a_coarseDataIndex, const **Interval** &a_srcInterval, const **Interval** &a_dstInterval, int a_dir, **Side::LoHiSide** a_sd)
- void **incrementFine** (**FArrayBox** &a_fineFlux, **Real** a_scale, const **DataIndex** &a_fineDataIndex, const **Interval** &a_srcInterval, const **Interval** &a_dstInterval, int a_dir, **Side::LoHiSide** a_sd)
- void **reflux** (**LevelData**< **FArrayBox** > &a_uCoarse, **Real** a_scale)
- void **reflux** (**LevelData**< **FArrayBox** > &a_uCoarse, const **Interval** &a_coarse_interval, const **Interval** &a_flux_interval, **Real** a_scale)
- bool **isDefined** () const

Protected Attributes

- **LevelData< FArrayBox > m_coarFlux**
- **LevelData< FArrayBox > m_fineFlux**
- **LayoutData< Vector< Box > > m_coarseLocations** [CH_SPACEDIM *2]
- **ProblemDomain m_domain**
- bool **m_isDefined**
- int **m_nRefine**
- **Copier m_reverseCopier**

7.80.1 Detailed Description

LevelFluxRegister-A class to encapsulate a levels worth of flux registers.

A LevelFluxRegister handles all the data choreography necessary to create a levels worth of flux registers. You send a pointer to a valid LevelCFStencil to its constructor or define function and it parsels out all the necessary information to the grid flux registers which hands it to the flux registers. The actual functions to manipulate data in the flux registers live in the class FluxRegister.

7.80.2 Constructor & Destructor Documentation

7.80.2.1 LevelFluxRegister::LevelFluxRegister ()

Default constructor. Creates an uninitialized levelfluxregister.

7.80.2.2 LevelFluxRegister::LevelFluxRegister (const DisjointBoxLayout & *a_dbl*, const DisjointBoxLayout & *a_dblCoarse*, const Box & *a_dProblem*, int *a_nRefine*, int *a_nComp*)

Full constructor. Calls the define function which creates a levels worth of flux registers. a_dProblem is fine-level domain

7.80.2.3 LevelFluxRegister::LevelFluxRegister (const DisjointBoxLayout & *a_dbl*, const DisjointBoxLayout & *a_dblCoarse*, const ProblemDomain & *a_dProblem*, int *a_nRefine*, int *a_nComp*)

Full constructor. Calls the define function which creates a levels worth of flux registers. a_dProblem is fine-level domain

7.80.2.4 LevelFluxRegister::~LevelFluxRegister ()

7.80.3 Member Function Documentation

7.80.3.1 void LevelFluxRegister::define (const DisjointBoxLayout & *a_dbl*, const DisjointBoxLayout & *a_dblCoarse*, const ProblemDomain & *a_dProblem*, int *a_nRefine*, int *a_nComp*)

Full define function. Creates a levels worth of flux registers. The values in the flux registers are still undefined, however. To zero the fluxregisters, you must call **setToZero()**.

7.80.3.2 void LevelFluxRegister::define (const DisjointBoxLayout & *a_dbl*, const DisjointBoxLayout & *a_dblCoarse*, const Box & *a_dProblem*, int *a_nRefine*, int *a_nComp*)

Full define function. Creates a levels worth of flux registers. The values in the flux registers are still undefined, however. To zero the fluxregisters, you must call **setToZero()**.

7.80.3.3 void LevelFluxRegister::incrementCoarse (FArrayBox & *a_coarseFlux*, Real *a_scale*, const DataIndex & *a_coarseDataIndex*, const Interval & *a_srcInterval*, const Interval & *a_dstInterval*, int *a_dir*, Side::LoHiSide *a_sd*)

Same as above, except you only increment the a_sd side.

7.80.3.4 void LevelFluxRegister::incrementCoarse (FArrayBox & *a_coarseFlux*, Real *a_scale*, const DataIndex & *a_coarseDataIndex*, const Interval & *a_srcInterval*, const Interval & *a_dstInterval*, int *a_dir*)

increments the register with data from coarseFlux, multiplied by scale. coarseFlux must contain the coarse fluxes in the dir direction for the grid m_coarseLayout[coarseDataIndex]. By convention, only the low side flux is used to avoid double-counting at coarse-fine interfaces. This operation is local.

7.80.3.5 void LevelFluxRegister::incrementFine (FArrayBox & *a_fineFlux*, Real *a_scale*, const DataIndex & *a_fineDataIndex*, const Interval & *a_srcInterval*, const Interval & *a_dstInterval*, int *a_dir*, Side::LoHiSide *a_sd*)

increments the register with data from fineFlux, multiplied by scale. fineFlux must contain the coarse fluxes in the dir direction for the grid m_fineLayout[fineDataIndex]. This operation is local.

7.80.3.6 bool LevelFluxRegister::isDefined () const

has full define function been called? return true if so.

7.80.3.7 void LevelFluxRegister::reflux (LevelData< FArrayBox > & *a_uCoarse*, const Interval & *a_coarse_interval*, const Interval & *a_flux_interval*, Real *a_scale*)

increments uCoarse with the reflux divergence of the contents of the flux register. This operation is global and blocking.

7.80.3.8 void LevelFluxRegister::reflux (LevelData< FArrayBox > & *a_uCoarse*, Real *a_scale*)

increments uCoarse with the reflux divergence of the contents of the flux register. Note that there is no srccomp etc crap here. this is done for all components so uCoarse has to have the same number of components as input nComp. This operation is global and blocking.

7.80.3.9 void LevelFluxRegister::setToZero ()

Initialize values of registers to zero.

7.80.3.10 void LevelFluxRegister::undefine ()

Modifies this LevelFluxRegister so that it is returned to the uninitialized state. User must now call the full **define()** before using it.

7.80.4 Member Data Documentation

7.80.4.1 LevelData<FArrayBox> LevelFluxRegister::m_coarFlux [protected]

7.80.4.2 LayoutData< Vector<Box> > LevelFluxRegister::m_coarseLocations[CH_SPACEDIM*2] [protected]

7.80.4.3 ProblemDomain LevelFluxRegister::m_domain [protected]

7.80.4.4 LevelData<FArrayBox> LevelFluxRegister::m_fineFlux [protected]

7.80.4.5 bool LevelFluxRegister::m_isDefined [protected]

7.80.4.6 int LevelFluxRegister::m_nRefine [protected]

7.80.4.7 Copier LevelFluxRegister::m_reverseCopier [protected]

The documentation for this class was generated from the following file:

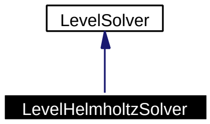
- **LevelFluxRegister.H**

7.81 LevelHelmholtzSolver Class Reference

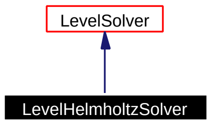
special case **LevelSolver** to solve Helmholtz eqn

`#include <LevelHelmholtzSolver.H>`

Inheritance diagram for LevelHelmholtzSolver:



Collaboration diagram for LevelHelmholtzSolver:



Public Member Functions

- **LevelHelmholtzSolver** ()

    Default constructor.

- virtual ~**LevelHelmholtzSolver** ()
- **LevelHelmholtzSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse, int a_nComp=1, bool a_limitCoarsening=false)

    constructor – note that there is no need to pass in a **LevelOp***,*

- **LevelHelmholtzSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse, int a_nComp=1, bool a_limitCoarsening=false)

    (deprecated) constructor – note there is no need to pass in a **LevelOp***,*

- **LevelHelmholtzSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **HelmholtzOp** *const a_opin, int a_nComp=1, bool a_limitCoarsening=false)
- **LevelHelmholtzSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **HelmholtzOp** *const a_opin, int a_nComp=1, bool a_limitCoarsening=false)

    deprecated interface using **Box** *instead of* **ProblemDomain**

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse, int a_nComp=1, bool a_limitCoarsening=false)
- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse, int a_nComp=1, bool a_limitCoarsening=false)

deprecated interface using **Box** *instead of* **ProblemDomain**

- virtual void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_-baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **HelmholtzOp** *const a_opin, int a_nComp=1, bool a_limitCoarsening=false)
- virtual void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_-baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **HelmholtzOp** *const a_opin, int a_nComp=1, bool a_limitCoarsening=false)

  (deprecated) interface using **Box** *instead of* **ProblemDomain**

- virtual void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_-baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **LevelOp** *const a_opin, int a_nComp=1, bool a_limitCoarsening=false)

  this function hides the **LevelSolver** *version – should never be called*

- virtual void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_-baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **LevelOp** *const a_opin, int a_nComp=1, bool a_limitCoarsening=false)

  this function hides the **LevelSolver** *version – should never be called*

- void **setBeta** (**Real** a_beta)
- void **setBC** (const **DomainGhostBC** &a_BC)

7.81.1 Detailed Description

special case **LevelSolver** to solve Helmholtz eqn

LevelHelmholtzSolver is a special case of **LevelSolver** for solving the Helmholtz equation on a level. It allows the coefficients to be changed without allocating an entire new **LevelSolver**.

7.81.2 Constructor & Destructor Documentation

7.81.2.1 LevelHelmholtzSolver::LevelHelmholtzSolver ()

Default constructor.

7.81.2.2 virtual LevelHelmholtzSolver::~LevelHelmholtzSolver () `[virtual]`

7.81.2.3 LevelHelmholtzSolver::LevelHelmholtzSolver (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout * *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, int *a_nComp* = 1, bool *a_limitCoarsening* = false)

constructor – note that there is no need to pass in a **LevelOp**,

7.81.2.4 LevelHelmholtzSolver::LevelHelmholtzSolver (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout * *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, int *a_nComp* = 1, bool *a_limitCoarsening* = false)

(deprecated) constructor – note there is no need to pass in a **LevelOp**,

7.81.2.5 LevelHelmholtzSolver::LevelHelmholtzSolver (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const HelmholtzOp ∗const *a_opin*, int *a_nComp* = 1, bool *a_limitCoarsening* = false)

7.81.2.6 LevelHelmholtzSolver::LevelHelmholtzSolver (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const HelmholtzOp ∗const *a_opin*, int *a_nComp* = 1, bool *a_limitCoarsening* = false)

deprecated interface using **Box** instead of **ProblemDomain**

7.81.3 Member Function Documentation

7.81.3.1 virtual void LevelHelmholtzSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const LevelOp ∗const *a_opin*, int *a_nComp* = 1, bool *a_limitCoarsening* = false) `[virtual]`

this function hides the **LevelSolver** version – should never be called

Reimplemented from **LevelSolver** (p. 385).

7.81.3.2 virtual void LevelHelmholtzSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const LevelOp ∗const *a_opin*, int *a_nComp* = 1, bool *a_limitCoarsening* = false) `[virtual]`

this function hides the **LevelSolver** version – should never be called

Reimplemented from **LevelSolver** (p. 386).

7.81.3.3 virtual void LevelHelmholtzSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const HelmholtzOp ∗const *a_opin*, int *a_nComp* = 1, bool *a_limitCoarsening* = false) `[virtual]`

(deprecated) interface using **Box** instead of **ProblemDomain**

7.81.3.4 virtual void LevelHelmholtzSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const HelmholtzOp ∗const *a_opin*, int *a_nComp* = 1, bool *a_limitCoarsening* = false) `[virtual]`

7.81.3.5 void LevelHelmholtzSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, int *a_nComp* = 1, bool *a_limitCoarsening* = false)

deprecated interface using **Box** instead of **ProblemDomain**

7.81.3.6 void LevelHelmholtzSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout * *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, int *a_nComp* = 1, bool *a_limitCoarsening* = false)

7.81.3.7 void LevelHelmholtzSolver::setBC (const DomainGhostBC & *a_BC*)

7.81.3.8 void LevelHelmholtzSolver::setBeta (Real *a_beta*)

The documentation for this class was generated from the following file:

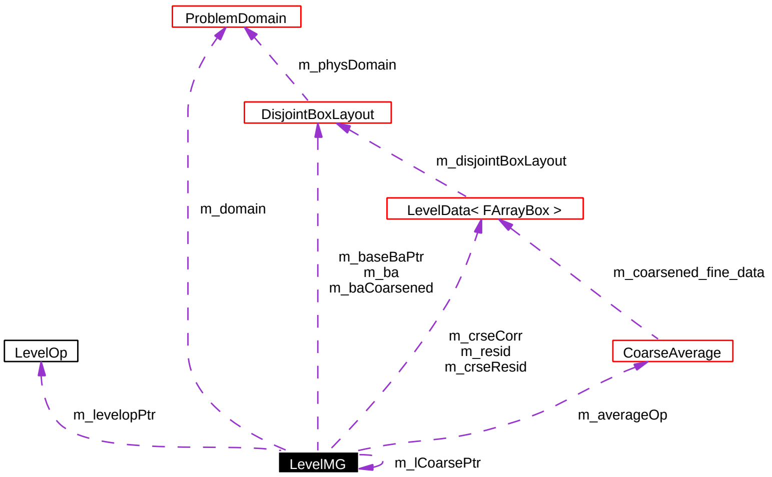
- **LevelHelmholtzSolver.H**

7.82 LevelMG Class Reference

Internal Chombo class to manage multigrid solve on a level.

`#include <LevelMG.H>`

Collaboration diagram for LevelMG:



Public Member Functions

- bool **isDefined** () const

  Has this object been fully defined?

- **LevelMG** ()

  Null Constructor.

- **LevelMG** (const **DisjointBoxLayout** &a_ba, const **DisjointBoxLayout** ∗a_baseBaPtr, **Real** a_dxLevel, int a_refRatio, const **Box** &a_domain, int a_nCoarserLevels, const **LevelOp** ∗const a_opin, int a_ncomp=1)

  Deprecated full constructor (use **ProblemDomain** *instead of* **Box**).

- **LevelMG** (const **DisjointBoxLayout** &a_ba, const **DisjointBoxLayout** ∗a_baseBaPtr, **Real** a_dxLevel, int a_refRatio, const **ProblemDomain** &a_domain, int a_nCoarserLevels, const **LevelOp** ∗const a_opin, int a_ncomp=1)

  Full constructor – calls matching define function.

- void **define** (const **DisjointBoxLayout** &a_ba, const **DisjointBoxLayout** ∗a_baseBaPtr, **Real** a_dxLevel, int a_refRatio, const **Box** &a_domain, int a_nCoarserLevels, const **LevelOp** ∗const a_opin, int a_ncomp=1)

  Deprecated full define function (use **ProblemDomain** *instead of* **Box**).

- void **define** (const **DisjointBoxLayout** &a_ba, const **DisjointBoxLayout** ∗a_baseBaPtr, **Real** a_dxLevel, int a_refRatio, const **ProblemDomain** &a_domain, int a_nCoarserLevels, const **LevelOp** ∗const a_opin, int a_ncomp)

  Full Define Function.

- void **define** (const **LevelMG** &a_level, int a_refCoarse, const **LevelOp** *const a_opin)

  Constructor for coarsened version of object.

- **LevelMG** (const **LevelMG** &a_level, int a_refCoarse, const **LevelOp** *a_opin)

  Constructor for coarsened version of object.

- **~LevelMG** ()

  Destructor.

- void **mgRelax** (**LevelData**< **FArrayBox** > &a_soln, const **LevelData**< **FArrayBox** > &a_rhs, bool bottomsolveflag)

  Invoke multigrid relaxation step.

- void **setNumBottomGSRB** (int a_numBottomGSRB)

  Set number of smoothing steps at bottom.

- void **setNumSmoothUp** (int a_numSmoothUp)

  Set number of smoothing steps on the way up V-cycle.

- void **setNumSmoothDown** (int a_numSmoothDown)

  Set number of smoothing steps on the way down V-cycle.

- **LevelOp** * **levelOpPtr** ()

  this is a dangerous access function that should not generally be used.

- **LevelMG** * **lCoarsePtr** ()

  this is another access function that should not be generally used.

- void **setConvergenceMetric** (**Real** a_metric, int a_comp)

  sets convergence metric – essentially a pass through to **LevelOp**

Protected Member Functions

- void **setDefaultValues** ()
- void **clearMemory** ()

Protected Attributes

- bool **m_isDefined**
- **LevelData< FArrayBox > m_resid**
- **LevelData< FArrayBox > m_crseResid**
- **LevelData< FArrayBox > m_crseCorr**
- **DisjointBoxLayout m_ba**
- **DisjointBoxLayout m_baCoarsened**
- const **DisjointBoxLayout** * **m_baseBaPtr**
- **Real m_dxLevel**
- **ProblemDomain m_domain**

- int **m_nCoarserLevels**
- **LevelMG** ∗ **m_lCoarsePtr**
- **LevelOp** ∗ **m_levelopPtr**
- int **m_numBottomGSRB**
- int **m_numSmoothUp**
- int **m_numSmoothDown**
- int **m_refToCoar**
- **CoarseAverage m_averageOp**

7.82.1 Detailed Description

Internal Chombo class to manage multigrid solve on a level.

Multigrid solver on a single multigrid level. This class should be considered internal to **AMRSolver** and not a part of the Chombo API.

7.82.2 Constructor & Destructor Documentation

7.82.2.1 LevelMG::LevelMG ()

Null Constructor.

7.82.2.2 LevelMG::LevelMG (const DisjointBoxLayout & *a_ba*, const DisjointBoxLayout ∗ *a_baseBaPtr*, Real *a_dxLevel*, int *a_refRatio*, const Box & *a_domain*, int *a_nCoarserLevels*, const LevelOp ∗const *a_opin*, int *a_ncomp* = 1)

Deprecated full constructor (use **ProblemDomain** instead of **Box**).

7.82.2.3 LevelMG::LevelMG (const DisjointBoxLayout & *a_ba*, const DisjointBoxLayout ∗ *a_baseBaPtr*, Real *a_dxLevel*, int *a_refRatio*, const ProblemDomain & *a_domain*, int *a_nCoarserLevels*, const LevelOp ∗const *a_opin*, int *a_ncomp* = 1)

Full constructor – calls matching define function.

7.82.2.4 LevelMG::LevelMG (const LevelMG & *a_level*, int *a_refCoarse*, const LevelOp ∗ *a_opin*)

Constructor for coarsened version of object.

7.82.2.5 LevelMG::∼LevelMG ()

Destructor.

7.82.3 Member Function Documentation

7.82.3.1 void LevelMG::clearMemory () [protected]

7.82.3.2 void LevelMG::define (const LevelMG & *a_level*, int *a_refCoarse*, const LevelOp *const *a_opin*)

Constructor for coarsened version of object.

7.82.3.3 void LevelMG::define (const DisjointBoxLayout & *a_ba*, const DisjointBoxLayout * *a_baseBaPtr*, Real *a_dxLevel*, int *a_refRatio*, const ProblemDomain & *a_domain*, int *a_nCoarserLevels*, const LevelOp *const *a_opin*, int *a_ncomp*)

Full Define Function.

Parameters:

a_ba Grids at this multigrid level

a_baseBaPtr Coarser grids

a_dxLevel Cell spacing at this level

a_refRatio Refinement ratio

a_domain Physical domain of problem

a_nCoarserLevels Number of multigrid levels coarser than this one

a_opin **LevelOp** which encapsulates operator

a_ncomp number of components

7.82.3.4 void LevelMG::define (const DisjointBoxLayout & *a_ba*, const DisjointBoxLayout * *a_baseBaPtr*, Real *a_dxLevel*, int *a_refRatio*, const Box & *a_domain*, int *a_nCoarserLevels*, const LevelOp *const *a_opin*, int *a_ncomp* = 1)

Deprecated full define function (use **ProblemDomain** instead of **Box**).

7.82.3.5 bool LevelMG::isDefined () const

Has this object been fully defined?

7.82.3.6 LevelMG* LevelMG::lCoarsePtr ()

this is another access function that should not be generally used.

7.82.3.7 LevelOp* LevelMG::levelOpPtr ()

this is a dangerous access function that should not generally be used.

7.82.3.8 void LevelMG::mgRelax (LevelData< FArrayBox > & *a_soln*, const LevelData< FArrayBox > & *a_rhs*, bool *bottomsolveflag*)

Invoke multigrid relaxation step.

Default is pure MG V-cycle, suitable for use in multilevel solver application; otherwise, use approximate solver such as CG at bottom level. It is assumed that the problem has already been put in residual-correction form. In particular, only the homogeneous form of the physical and coarse-fine boundary conditions need be invoked.

7.82.3.9 void LevelMG::setConvergenceMetric (Real *a_metric*, int *a_comp*)

sets convergence metric – essentially a pass through to **LevelOp**

7.82.3.10 void LevelMG::setDefaultValues () `[protected]`

7.82.3.11 void LevelMG::setNumBottomGSRB (int *a_numBottomGSRB*) `[inline]`

Set number of smoothing steps at bottom.

7.82.3.12 void LevelMG::setNumSmoothDown (int *a_numSmoothDown*)

Set number of smoothing steps on the way down V-cycle.

7.82.3.13 void LevelMG::setNumSmoothUp (int *a_numSmoothUp*)

Set number of smoothing steps on the way up V-cycle.

7.82.4 Member Data Documentation

7.82.4.1 CoarseAverage LevelMG::m_averageOp [protected]

7.82.4.2 DisjointBoxLayout LevelMG::m_ba [protected]

7.82.4.3 DisjointBoxLayout LevelMG::m_baCoarsened [protected]

7.82.4.4 const DisjointBoxLayout* LevelMG::m_baseBaPtr [protected]

7.82.4.5 LevelData<FArrayBox> LevelMG::m_crseCorr [protected]

7.82.4.6 LevelData<FArrayBox> LevelMG::m_crseResid [protected]

7.82.4.7 ProblemDomain LevelMG::m_domain [protected]

7.82.4.8 Real LevelMG::m_dxLevel [protected]

7.82.4.9 bool LevelMG::m_isDefined [protected]

7.82.4.10 LevelMG* LevelMG::m_lCoarsePtr [protected]

7.82.4.11 LevelOp* LevelMG::m_levelopPtr [protected]

7.82.4.12 int LevelMG::m_nCoarserLevels [protected]

7.82.4.13 int LevelMG::m_numBottomGSRB [protected]

7.82.4.14 int LevelMG::m_numSmoothDown [protected]

7.82.4.15 int LevelMG::m_numSmoothUp [protected]

7.82.4.16 int LevelMG::m_refToCoar [protected]

7.82.4.17 LevelData<FArrayBox> LevelMG::m_resid [protected]

The documentation for this class was generated from the following file:

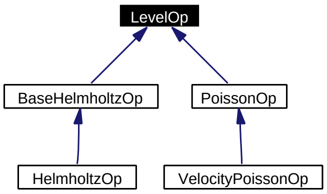
- **LevelMG.H**

7.83 LevelOp Class Reference

Pure base class to encapsulate level operations API for elliptic solvers.

`#include <LevelOp.H>`

Inheritance diagram for LevelOp:



Public Member Functions

- **LevelOp** ()

  Null constructor.

- virtual ~**LevelOp** ()

  Destructor.

- virtual void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseBAPtr, **Real** a_dxLevel, int a_refRatio, const **ProblemDomain** &a_domain, bool a_homogeneousOnly, int a_ncomp=1)=0

  Full define function.

- virtual void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseBAPtr, **Real** a_dxLevel, int a_refRatio, const **Box** &a_domain, bool a_homogeneousOnly, int a_ncomp=1)=0

  Full define function. (Deprecated – use the **ProblemDomain** *version).*

- virtual void **define** (const **LevelOp** *a_opfine, int a_reftoFine)=0

  Full define function.

- virtual void **CFInterp** (**LevelData**< **FArrayBox** > &a_phif, const **LevelData**< **FArrayBox** > &a_phic)=0

  Coarse / Fine interpolation operator.

- virtual void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &a_phif)=0

  Homogeneous coarse/fine interpolation operator.

- virtual **LevelOp** * **new_levelop** () const=0

  "Virtual constructor" workaround

- virtual bool **isDefined** () const=0

  Has this object been fully defined?

- virtual void **smooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs)=0

  Smoother.

- virtual void **levelPreconditioner** (**LevelData**< **FArrayBox** > &a_phihat, const **LevelData**< **FArrayBox** > &a_rhshat)=0

  Apply preconditioning to the solution on this level.

- virtual void **applyOpI** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phicPtr, **LevelData**< **FArrayBox** > &a_lofPhi)=0

  Evaluate operator, inhomogeneous boundary conditions.

- virtual void **applyOpIcfHphys** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phicPtr, **LevelData**< **FArrayBox** > &a_lofPhi)=0

  Evaluate operator.

- virtual void **applyOpHcfIphys** (**LevelData**< **FArrayBox** > &a_phi, **LevelData**< **FArrayBox** > &a_lofPhi)=0

  Evaluate operator.

- virtual void **applyOpH** (**LevelData**< **FArrayBox** > &a_phi, **LevelData**< **FArrayBox** > &a_lofPhi)=0

  Evaluate Operator with homogeneous boundary conditions.

- virtual void **bottomSmoother** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs)=0

  Smoother at bottom level.

- virtual void **getFlux** (**FArrayBox** &a_fineFlux, const **FArrayBox** &a_data, const **DataIndex** &a_datInd, int a_dir)=0

  Get flux at this level in the direction dir over the given **Box**.

- virtual void **setConvergenceMetric** (**Real** a_metric, int a_comp)=0

  set the convergence metrics (if appropriate) for this problem

7.83.1 Detailed Description

Pure base class to encapsulate level operations API for elliptic solvers.

LevelOp is a pure base class to encapsulate level operations API for elliptic solvers. All elliptic operators used by **AMRSolver** inherit the LevelOp interface.

7.83.2 Constructor & Destructor Documentation

7.83.2.1 LevelOp::LevelOp () `[inline]`

Null constructor.

7.83.2.2 virtual LevelOp::~LevelOp () [inline, virtual]

Destructor.

7.83.3 Member Function Documentation

7.83.3.1 virtual void LevelOp::applyOpH (LevelData< FArrayBox > & *a_phi*, LevelData< FArrayBox > & *a_lofPhi*) [pure virtual]

Evaluate Operator with homogeneous boundary conditions.

Uses homogeneous form of both C/F and physical boundary conditions.

Implemented in **HelmholtzOp** (p. 290), and **PoissonOp** (p. 504).

7.83.3.2 virtual void LevelOp::applyOpHcfIphys (LevelData< FArrayBox > & *a_phi*, LevelData< FArrayBox > & *a_lofPhi*) [pure virtual]

Evaluate operator.

Uses homogeneous form of C/F boundary conditions and inhomogeneous form of physical boundary conditions.

Implemented in **HelmholtzOp** (p. 290), and **PoissonOp** (p. 504).

7.83.3.3 virtual void LevelOp::applyOpI (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > * *a_phicPtr*, LevelData< FArrayBox > & *a_lofPhi*) [pure virtual]

Evaluate operator, inhomogeneous boundary conditions.

Uses inhomogeneous form of both C/F and physical boundary conditions.

Implemented in **HelmholtzOp** (p. 290), and **PoissonOp** (p. 504).

7.83.3.4 virtual void LevelOp::applyOpIcfHphys (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > * *a_phicPtr*, LevelData< FArrayBox > & *a_lofPhi*) [pure virtual]

Evaluate operator.

Uses inhomogeneous form of C/F boundary conditions. homogeneous form of phys boundary conditions.

Implemented in **HelmholtzOp** (p. 290), and **PoissonOp** (p. 504).

7.83.3.5 virtual void LevelOp::bottomSmoother (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*) [pure virtual]

Smoother at bottom level.

Implemented in **HelmholtzOp** (p. 291), and **PoissonOp** (p. 504).

7.83.3.6 virtual void LevelOp::CFInterp (LevelData< FArrayBox > & *a_phif*, const LevelData< FArrayBox > & *a_phic*) [pure virtual]

Coarse / Fine interpolation operator.

Fill ghost cells of phi by interpolating using coarse-level and fine-level data.

Parameters:

a_phif Data at this level

a_phic Coarser-level data

Implemented in **HelmholtzOp** (p. 291), and **PoissonOp** (p. 504).

7.83.3.7 virtual void LevelOp::define (const LevelOp ∗ *a_opfine*, int *a_reftoFine*) [pure virtual]

Full define function.

Define from finer levelop. reftofine is the refinement ratio to get from this LevelOp to *opfine*

Parameters:

a_opfine Finer levelOp from which this will be defined

a_reftoFine Amount this LevelOp will be coarser than *opfine*

Implemented in **HelmholtzOp** (p. 291), and **PoissonOp** (p. 505).

7.83.3.8 virtual void LevelOp::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseBAPtr*, Real *a_dxLevel*, int *a_refRatio*, const Box & *a_domain*, bool *a_homogeneousOnly*, int *a_ncomp* = 1) [pure virtual]

Full define function. (Deprecated – use the **ProblemDomain** version).

Makes all coarse-fine information and sets internal variables

Implemented in **HelmholtzOp** (p. 291), and **PoissonOp** (p. 505).

7.83.3.9 virtual void LevelOp::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseBAPtr*, Real *a_dxLevel*, int *a_refRatio*, const ProblemDomain & *a_domain*, bool *a_homogeneousOnly*, int *a_ncomp* = 1) [pure virtual]

Full define function.

Makes all coarse-fine information and sets internal variables

Parameters:

a_grids Grids at this level

a_baseBAPtr Coarser-level grids

a_dxLevel Cell spacing at this level

a_refRatio Refinement ratio btwn this level and coarser level

a_domain Physical problem domain

a_homogeneousOnly Will we only need homogeneous C/F BC's?

a_ncomp Number of components

Implemented in **HelmholtzOp** (p. 291), and **PoissonOp** (p. 505).

7.83.3.10 virtual void LevelOp::getFlux (FArrayBox & *a_fineFlux*, const FArrayBox & *a_data*, const DataIndex & *a_datInd*, int *a_dir*) `[pure virtual]`

Get flux at this level in the direction dir over the given **Box**.

The fluxes live on the cell faces with direction dir. Fluxes are computed for all interior edges of *data*. The flux fab is resized inside the routine.

Parameters:

a_fineFlux Face-centered Fluxes in direction dir (resized in function)

a_data Cell-centered box over which fluxes will be computed

a_datInd Data index of grid where flux is located

a_dir Direction of fluxes to be computed

Implemented in **HelmholtzOp** (p. 292), and **PoissonOp** (p. 505).

7.83.3.11 virtual void LevelOp::homogeneousCFInterp (LevelData< FArrayBox > & *a_phif*) `[pure virtual]`

Homogeneous coarse/fine interpolation operator.

Implemented in **HelmholtzOp** (p. 292), and **PoissonOp** (p. 505).

7.83.3.12 virtual bool LevelOp::isDefined () const `[pure virtual]`

Has this object been fully defined?

Implemented in **VelocityPoissonOp** (p. 629), **HelmholtzOp** (p. 292), and **PoissonOp** (p. 505).

7.83.3.13 virtual void LevelOp::levelPreconditioner (LevelData< FArrayBox > & *a_phihat*, const LevelData< FArrayBox > & *a_rhshat*) `[pure virtual]`

Apply preconditioning to the solution on this level.

Given rhshat, returns phihat from M(phihat) = rhshat

Implemented in **HelmholtzOp** (p. 292), and **PoissonOp** (p. 506).

7.83.3.14 virtual LevelOp* LevelOp::new_levelop () const `[pure virtual]`

"Virtual constructor" workaround"

Implemented in **VelocityPoissonOp** (p. 629), **HelmholtzOp** (p. 292), and **PoissonOp** (p. 506).

7.83.3.15 virtual void LevelOp::setConvergenceMetric (Real *a_metric*, int *a_comp*) [pure virtual]

set the convergence metrics (if appropriate) for this problem

If this LevelOp is being used in the context of a solve, we may wish to set a metric against which to define convergence (for example, the norm of the RHS). In many cases, this will simply be a pass-through to a bottom solver.

Implemented in **HelmholtzOp** (p. 293), and **PoissonOp** (p. 506).

7.83.3.16 virtual void LevelOp::smooth (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*) [pure virtual]

Smoother.

Assumes that problem has already been put in residual correction form, so that C/F boundary conditions are homogeneous.

Parameters:

a_phi Current guess at solution

a_rhs Right-hand side of problem we're solving

Implemented in **HelmholtzOp** (p. 293), and **PoissonOp** (p. 506).

The documentation for this class was generated from the following file:

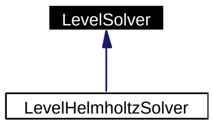
- **LevelOp.H**

7.84 LevelSolver Class Reference

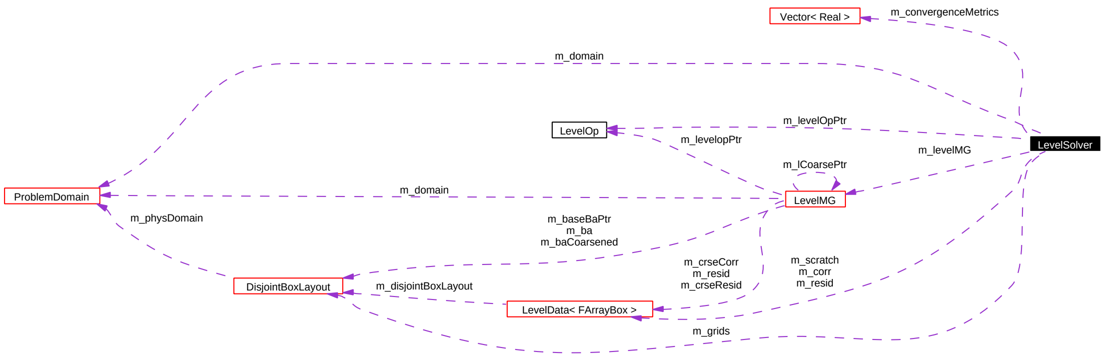
Solves elliptic equations on a level.

`#include <LevelSolver.H>`

Inheritance diagram for LevelSolver:



Collaboration diagram for LevelSolver:



Public Member Functions

- **LevelSolver** ()

  Null constructor, leaves LevelSolver undefined.

- virtual **~LevelSolver** ()

  Destructor.

- **LevelSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **LevelOp** *const a_opin, int a_ncomp, bool a_limitCoarsening=false)

  Full constructor. Calls full define function.

- **LevelSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **LevelOp** *const a_opin, int a_ncomp=1, bool a_limitCoarsening=false)

  Full constructor (Deprecated – use **ProblemDomain** *instead).*

- virtual void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **LevelOp** *const a_opin, int a_ncomp=1, bool a_limitCoarsening=false)

Full define function.

- virtual void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_-baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **LevelOp** *const a_opin, int a_ncomp=1, bool a_limitCoarsening=false)

  Full define function. (Deprecated – use **ProblemDomain** *version instead).*

- bool **isDefined** () const

  Has this object been defined already?

- void **clearMemory** ()

  Clears storage.

- void **setDefaultValues** ()

  Internal use function to set default values for solver parameters.

- void **levelSolveH** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs, bool a_initializePhiToZero=true)

  Do a level solve with homogeneous BCs on this level using multigrid.

- void **levelSolve** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phic, const **LevelData**< **FArrayBox** > &a_rhs, bool a_initializePhiToZero=true)

  Do a level solve with inhomogeneous BCs on this level using multigrid.

- void **setNumSmoothUp** (int a_numSmoothUp)

  Set number of multigrid smoothings on way up v-cycle. Default is 4.

- void **setNumSmoothDown** (int a_numSmoothDown)

  Set number of multigrid smoothings on way down v-cycle; Default is 4.

- void **setTolerance** (**Real** a_tolerance)

  Set solution tolerance.

- void **setOperatorTolerance** (**Real** a_operatorTolerance)

  Set "operator tolerance" of iterative solution.

- void **setVerbose** (bool a_verbose)

  Set how much info is printed out during solve.

- void **setMaxIter** (int a_maxIter)

  Set max number multigrid vcycles to use in solution.

- void **setMinIter** (int a_minIter)

  Set min number of iterations.

- virtual void **setConvergenceMetric** (**Real** a_metric, int a_comp)

  set the convergence metrics (if appropriate) for this problem

- virtual void **resetConvergenceMetrics** ()

  rest the convergence metrics to an undefined state (for solver re-use)

Protected Attributes

- **LevelMG m_levelMG**
- **LevelOp * m_levelOpPtr**
- bool **m_isDefined**
- **DisjointBoxLayout m_grids**
- **ProblemDomain m_domain**
- **LevelData< FArrayBox > m_resid**
- **LevelData< FArrayBox > m_scratch**
- **LevelData< FArrayBox > m_corr**
- int **m_nRefCrse**
- **Real m_dxLevel**
- int **m_maxIter**
- int **m_minIter**
- int **m_numSmoothUp**
- int **m_numSmoothDown**
- **Real m_tolerance**
- **Vector< Real > m_convergenceMetrics**
- **Real m_operatorTolerance**
- bool **m_verbose**

7.84.1 Detailed Description

Solves elliptic equations on a level.

LevelSolver Solves elliptic equations on a level using multigrid.

7.84.2 Constructor & Destructor Documentation

7.84.2.1 LevelSolver::LevelSolver ()

Null constructor, leaves LevelSolver undefined.

7.84.2.2 virtual LevelSolver::~LevelSolver () `[virtual]`

Destructor.

7.84.2.3 LevelSolver::LevelSolver (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout * *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const LevelOp *const *a_opin*, int *a_ncomp*, bool *a_limitCoarsening* = false)

Full constructor. Calls full define function.

Calls corresponding full define function.

Inputs:

- **a_grids**: Grid on which solution will be computed.
- **a_baseGrids**: pointer to next coarser grids. Should be == NULL if there is no coarser level.

- **a_domain**: domain of solution level
- **a_dxLevel**: grid spacing at solution level
- **a_nRefCrse**: refinement ratio between two levels. Not used if there is no finer level
- **a_opin**: operator L used in solution of Lphi = rho
- **a_limitCoarsening**: if true, only do multigrid coarsening down to next coarser **AMR** level (only coarsen by a_nRefCrse). If false, coarsen as far as possible.

7.84.2.4 LevelSolver::LevelSolver (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const LevelOp ∗const *a_opin*, int *a_ncomp* = 1, bool *a_limitCoarsening* = false)

Full constructor (Deprecated – use **ProblemDomain** instead).

Calls corresponding full define function.

Inputs:

- **a_grids**: Grid on which solution will be computed.
- **a_baseGrids**: pointer to next coarser grids. Should be == NULL if there is no coarser level.
- **a_domain**: domain of solution level
- **a_dxLevel**: grid spacing at solution level
- **a_nRefCrse**: refinement ratio between two levels. Not used if there is no finer level
- **a_opin**: operator L used in solution of Lphi = rho
- **a_limitCoarsening**: if true, only do multigrid coarsening down to next coarser **AMR** level (only coarsen by a_nRefCrse). If false, coarsen as far as possible.

7.84.3 Member Function Documentation

7.84.3.1 void LevelSolver::clearMemory ()

Clears storage.

7.84.3.2 virtual void LevelSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const LevelOp ∗const *a_opin*, int *a_ncomp* = 1, bool *a_limitCoarsening* = false) `[virtual]`

Full define function. (Deprecated – use **ProblemDomain** version instead).

Inputs:

- **a_grids**: Grid on which solution will be computed.
- **a_baseGrids**: pointer to next coarser grids. Should be == NULL if there is no coarser level.

- **a_domain**: domain of solution level
- **a_dxLevel**: grid spacing at solution level
- **a_nRefCrse**: refinement ratio between two levels. Not used if there is no finer level
- **a_opin**: operator L used in solution of Lphi = rho
- **a_limitCoarsening**: if true, only do multigrid coarsening down to next coarser **AMR** level (only coarsen by a_nRefCrse). If false, coarsen as far as possible.

Reimplemented in **LevelHelmholtzSolver** (p. 368).

7.84.3.3 virtual void LevelSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*, const LevelOp ∗const *a_opin*, int *a_ncomp* = 1, bool *a_limitCoarsening* = false) `[virtual]`

Full define function.

Inputs:

- **a_grids**: Grid on which solution will be computed.
- **a_baseGrids**: pointer to next coarser grids. Should be == NULL if there is no coarser level.
- **a_domain**: domain of solution level
- **a_dxLevel**: grid spacing at solution level
- **a_nRefCrse**: refinement ratio between two levels. Not used if there is no finer level
- **a_opin**: operator L used in solution of Lphi = rho
- **a_limitCoarsening**: if true, only do multigrid coarsening down to next coarser **AMR** level (only coarsen by a_nRefCrse). If false, coarsen as far as possible.

Reimplemented in **LevelHelmholtzSolver** (p. 368).

7.84.3.4 bool LevelSolver::isDefined () const

Has this object been defined already?

7.84.3.5 void LevelSolver::levelSolve (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phic*, const LevelData< FArrayBox > & *a_rhs*, bool *a_initializePhiToZero* = true)

Do a level solve with inhomogeneous BCs on this level using multigrid.

Uses inhomogeneous bcs at interfaces. phic should be set to NULL if there is no coarser level.

Parameters:

a_phi initial guess at solution, returns solution

a_phic solution at next coarser level

a_rhs right-hand side

a_initializePhiToZero if true, set phi to 0 at start (o/w use initial guess)

7.84.3.6 void LevelSolver::levelSolveH (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*, bool *a_initializePhiToZero* = true)

Do a level solve with homogeneous BCs on this level using multigrid.

Uses homogeneous bcs at coarse/fine interfaces

Parameters:

a_phi initial guess at solution, returns solution

a_rhs right-hand side

a_initializePhiToZero if true, set phi to 0 at start (o/w use initial guess)

7.84.3.7 virtual void LevelSolver::resetConvergenceMetrics () `[virtual]`

rest the convergence metrics to an undefined state (for solver re-use)

7.84.3.8 virtual void LevelSolver::setConvergenceMetric (Real *a_metric*, int *a_comp*) `[virtual]`

set the convergence metrics (if appropriate) for this problem

sets metric against which we define convergence (for example, the norm of the RHS). Also passes a_metric through to the **LevelMG** (and eventually to a bottom solver.)

7.84.3.9 void LevelSolver::setDefaultValues ()

Internal use function to set default values for solver parameters.

7.84.3.10 void LevelSolver::setMaxIter (int *a_maxIter*)

Set max number multigrid vcycles to use in solution.

7.84.3.11 void LevelSolver::setMinIter (int *a_minIter*)

Set min number of iterations.

Only relevant when residual is not decreasing fast enough to satisfy the "operator tolerance". Default is 4.

7.84.3.12 void LevelSolver::setNumSmoothDown (int *a_numSmoothDown*)

Set number of multigrid smoothings on way down v-cycle; Default is 4.

7.84.3.13 void LevelSolver::setNumSmoothUp (int *a_numSmoothUp*)

Set number of multigrid smoothings on way up v-cycle. Default is 4.

7.84.3.14 void LevelSolver::setOperatorTolerance (Real *a_operatorTolerance*)

Set "operator tolerance" of iterative solution.

Iteration will stop if (new_residual/old_residual) > 1-operatorTolerance (and at least minIter iterations have been performed). Default is 1.0e-5.

7.84.3.15 void LevelSolver::setTolerance (Real *a_tolerance*)

Set solution tolerance.

7.84.3.16 void LevelSolver::setVerbose (bool *a_verbose*)

Set how much info is printed out during solve.

7.84.4 Member Data Documentation

7.84.4.1 Vector<Real> LevelSolver::m_convergenceMetrics `[protected]`

metrics used to judge convergence

7.84.4.2 LevelData<FArrayBox> LevelSolver::m_corr `[protected]`

Correction

7.84.4.3 ProblemDomain LevelSolver::m_domain `[protected]`

Problem domain at this level

7.84.4.4 Real LevelSolver::m_dxLevel `[protected]`

Cell spacing on this level

7.84.4.5 DisjointBoxLayout LevelSolver::m_grids `[protected]`

Grids at this level

7.84.4.6 bool LevelSolver::m_isDefined `[protected]`

Has this LevelSolver been defined yet?

7.84.4.7 LevelMG LevelSolver::m_levelMG `[protected]`

LevelMG level solver object to relax on this level

7.84.4.8 LevelOp* LevelSolver::m_levelOpPtr `[protected]`

LevelMG level solver object to relax on this level

7.84.4.9 int LevelSolver::m_maxIter [protected]

Maximum number of solver iterations

7.84.4.10 int LevelSolver::m_minIter [protected]

Minimum number of iterations

7.84.4.11 int LevelSolver::m_nRefCrse [protected]

Refinement ratio to coarser level

7.84.4.12 int LevelSolver::m_numSmoothDown [protected]

Number of smoother iterations on down cycle

7.84.4.13 int LevelSolver::m_numSmoothUp [protected]

Number of smoother iterations on up cycle

7.84.4.14 Real LevelSolver::m_operatorTolerance [protected]

Tolerance used to stop solver iteration if convergence hangs.

7.84.4.15 LevelData<FArrayBox> LevelSolver::m_resid [protected]

Residual

7.84.4.16 LevelData<FArrayBox> LevelSolver::m_scratch [protected]

Useful scratch space.

7.84.4.17 Real LevelSolver::m_tolerance [protected]

Solver tolerance

7.84.4.18 bool LevelSolver::m_verbose [protected]

How verbose shold the solve be?

The documentation for this class was generated from the following file:

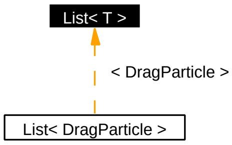
- **LevelSolver.H**

7.85 List< T > Class Template Reference

A Doubly-Linked List Class.

`#include <List.H>`

Inheritance diagram for List< T >:



Public Member Functions

- **List** ()

  Construct an empty List<T>.

- **List** (const **List**< T > &rhs)

  The copy constructor.

- **List**< T > & **operator=** (const **List**< T > &rhs)

  The assignment operator.

- ~**List** ()

  The destructor.

- void **prepend** (const T &value)

  Adds a copy of the value to the beginning of the List<T>.

- void **append** (const T &value)

  Adds a copy of the value to the end of the List<T>.

- void **add** (const T &value)

  Adds a copy of the value to the end of the List<T>.

- void **join** (const **List**< T > &src)

  Appends a copy of all items in List<T> src to this List<T>.

- void **catenate** (**List**< T > &src)

  Appends a copy of all items in List<T> src to this List<T>.

- void **clear** ()

  Removes all objects from the List<T>.

- **List**< T > * **copy** () const

  Returns a copy of this List<T> on the heap.

- T & **firstElement** () const

Returns a reference to the first element in the List<T>.

- T & **lastElement** () const

  Returns a reference to the last element in the List<T>.

- bool **includes** (const T &value) const

  Returns true if the List<T> contains an object identical to value.

- bool **operator==** (const **List**< T > &rhs) const

  Returns true if the this and rhs are memberwise equal.

- bool **operator!=** (const **List**< T > &rhs) const

  Returns true if the this and rhs are not equal.

- bool **isEmpty** () const

  Returns true if the List<T> is empty.

- bool **isNotEmpty** () const

  Returns true if the List<T> is not empty.

- int **length** () const

  Returns the number of objects in the List<T>.

- void **removeFirst** ()

  Removes the first element in the List<T>.

- void **removeLast** ()

  Removes the last element in the List<T>.

- const T & **operator[]** (const **ListIterator**< T > &li) const

  Returns reference to object pointed to by the ListIterator<T>.

- T & **operator[]** (const **ListIterator**< T > &li)

  Returns reference to object pointed to by the ListIterator<T>.

- void **remove** (const T &value)

  Removes all objects in the List<T> equal to value.

- void **remove** (const **List**< T > &lst)

  Removes all objects in the List<T> equal to any of the values in lst.

- void **remove** (**ListIterator**< T > &lit)

  Removes the object pointed to by the ListIterator<T>.

- void **replace** (**ListIterator**< T > &li, const T &val)

  Replace the value pointed to by the ListIterator<T> by val.

- void **addAfter** (**ListIterator**< T > &lit, const T &val)

  Insert val into List<T> after the object pointed to by lit.

- void **addBefore** (**ListIterator**< T > &lit, const T &val)

  Insert val into List<T> before the object pointed to by lit.

- **ListIterator**< T > **listIterator** () const

  Returns a ListIterator<T> to the first object in this List<T>.

- **ListIterator**< T > **first** () const

  Returns a ListIterator<T> to the first object in this List<T>.

- **ListIterator**< T > **last** () const

  Returns a ListIterator<T> to the last object in this List<T>.

Protected Member Functions

- void **remove** (**ListLink**< T > *ln)
- **ListLink**< T > * **addBefore** (**ListLink**< T > *ln, const T &val)
- **ListLink**< T > * **addAfter** (**ListLink**< T > *ln, const T &val)

Protected Attributes

- **ListLink**< T > * **head**
- **ListLink**< T > * **tail**

Friends

- class **ListIterator**< **T** >

7.85.1 Detailed Description

template<class T> class List< T >

A Doubly-Linked List Class.

The List<T> class is a template class that implements a doubly-linked list of objects. A List<T> is a useful container class when the number of objects in the collection is not known ahead of time. A List<T> can contain an arbitrary number of elements; operations such as insertion, deletion, and catenation are easily implemented and inexpensive.

The only difficulty when defining a list class is devising a mechanism to access the elements. In an array, an element is accessed using an integer index. Since the elements in a List<T> are ordered by position, we could define an integer indexing operation that walks along the List<T> links from the beginning until the numbered element is found. Unfortunately, this would be very inefficient when accessing elements near the end of a long list. Another solution is to allow user access to the individual link objects that contain the element as well as the forward and backward pointers. This is not a satisfactory solution since it allows user access to the internal representation of the class. The solution chosen is to define a ListIterator<T> template class.

Think of a ListIterator<T> as a pointer to an object in the List<T>. You can access the element currently pointed to by the iterator, move the iterator forward and backward through the List<T>, and use it as a mechanism to define where elements should be inserted and deleted. If the iterator is moved off the end of the list it behaves as a null pointer.

This is a concrete class, not a polymorphic one.

7.85.2 Constructor & Destructor Documentation

7.85.2.1 template<class T> List< T >::List () [inline]

Construct an empty List<T>.

7.85.2.2 template<class T> List< T >::List (const List< T > & *rhs*)

The copy constructor.

7.85.2.3 template<class T> List< T >::~List () [inline]

The destructor.

7.85.3 Member Function Documentation

7.85.3.1 template<class T> void List< T >::add (const T & *value*)

Adds a copy of the value to the end of the List<T>.

7.85.3.2 template<class T> ListLink< T > * List< T >::addAfter (ListLink< T > * *ln*, const T & *val*) [protected]

A helper function for adding nodes.

7.85.3.3 template<class T> void List< T >::addAfter (ListIterator< T > & *lit*, const T & *val*) [inline]

Insert val into List<T> after the object pointed to by *lit*.

7.85.3.4 template<class T> ListLink< T > * List< T >::addBefore (ListLink< T > * *ln*, const T & *val*) [protected]

A helper function for adding nodes.

7.85.3.5 template<class T> void List< T >::addBefore (ListIterator< T > & *lit*, const T & *val*) [inline]

Insert val into List<T> before the object pointed to by *lit*.

7.85.3.6 template<class T> void List< T >::append (const T & *value*) [inline]

Adds a copy of the value to the end of the List<T>.

7.85.3.7 template<class T> void List< T >::catenate (List< T > & *src*)

Appends a copy of all items in List<T> src to this List<T>.

This differs from **join()** in that it unlinks the objects from the List<T> src and glues them to the end of this List<T>, leaving List<T> src empty. This is more efficient that **join()** if src is no longer needed.

7.85.3.8 template<class T> void List< T >::clear ()

Removes all objects from the List<T>.

7.85.3.9 template<class T> List< T > * List< T >::copy () const [inline]

Returns a copy of this List<T> on the heap.

It is the user's responsibility to delete this when no longer needed.

7.85.3.10 template<class T> ListIterator< T > List< T >::first () const [inline]

Returns a ListIterator<T> to the first object in this List<T>.

7.85.3.11 template<class T> T & List< T >::firstElement () const [inline]

Returns a reference to the first element in the List<T>.

7.85.3.12 template<class T> bool List< T >::includes (const T & *value*) const

Returns true if the List<T> contains an object identical to *value*.

Type T must have an **operator==()** defined, or be an intrinsic type.

7.85.3.13 template<class T> bool List< T >::isEmpty () const [inline]

Returns true if the List<T> is empty.

7.85.3.14 template<class T> bool List< T >::isNotEmpty () const [inline]

Returns true if the List<T> is not empty.

7.85.3.15 template<class T> void List< T >::join (const List< T > & *src*)

Appends a copy of all items in List<T> src to this List<T>.

7.85.3.16 template<class T> ListIterator< T > List< T >::last () const [inline]

Returns a ListIterator<T> to the last object in this List<T>.

7.85.3.17 template<class T> T & List< T >::lastElement () const `[inline]`

Returns a reference to the last element in the List<T>.

7.85.3.18 template<class T> int List< T >::length () const

Returns the number of objects in the List<T>.

7.85.3.19 template<class T> ListIterator< T > List< T >::listIterator () const `[inline]`

Returns a ListIterator<T> to the first object in this List<T>.

7.85.3.20 template<class T> bool List< T >::operator!= (const List< T > & *rhs*) const

Returns true if the this and rhs are not equal.

7.85.3.21 template<class T> List< T > & List< T >::operator= (const List< T > & *rhs*)

The assignment operator.

7.85.3.22 template<class T> bool List< T >::operator== (const List< T > & *rhs*) const

Returns true if the this and rhs are memberwise equal.

Lists are memberwise equal if he two lists are the same size and each of the elements in the list compare equal. Type T must have an **operator==()** defined, or be an intrinsic type.

7.85.3.23 template<class T> T & List< T >::operator[] (const ListIterator< T > & *li*) `[inline]`

Returns reference to object pointed to by the ListIterator<T>.

7.85.3.24 template<class T> const T & List< T >::operator[] (const ListIterator< T > & *li*) const `[inline]`

Returns reference to object pointed to by the ListIterator<T>.

7.85.3.25 template<class T> void List< T >::prepend (const T & *value*) `[inline]`

Adds a copy of the value to the beginning of the List<T>.

7.85.3.26 template<class T> void List< T >::remove (ListLink< T > * *ln*) `[protected]`

A helper function for removing nodes.

7.85.3.27 template<class T> void List< T >::remove (ListIterator< T > & *lit*)

Removes the object pointed to by the ListIterator<T>.

7.85.3.28 template<class T> void List< T >::remove (const List< T > & *lst*)

Removes all objects in the List<T> equal to any of the values in lst.

7.85.3.29 template<class T> void List< T >::remove (const T & *value*)

Removes all objects in the List<T> equal to value.

7.85.3.30 template<class T> void List< T >::removeFirst () `[inline]`

Removes the first element in the List<T>.

7.85.3.31 template<class T> void List< T >::removeLast () `[inline]`

Removes the last element in the List<T>.

7.85.3.32 template<class T> void List< T >::replace (ListIterator< T > & *li*, const T & *val*) `[inline]`

Replace the value pointed to by the ListIterator<T> by val.

7.85.4 Friends And Related Function Documentation

7.85.4.1 template<class T> friend class ListIterator< T > `[friend]`

Our good and trusted friend.

7.85.5 Member Data Documentation

7.85.5.1 template<class T> ListLink<T>* List< T >::head `[protected]`

The head of the list.

7.85.5.2 template<class T> ListLink<T>* List< T >::tail `[protected]`

The tail of the list.

The documentation for this class was generated from the following files:

- **List.H**
- **ListImplem.H**

7.86 ListIterator< T > Class Template Reference

Iterator over a **List**.

```
#include <List.H>
```

Public Member Functions

- **ListIterator** (const **List**< T > &aList)

  Construct a ListIterator<T> to first element of aList.

- **ListIterator** (const **ListIterator**< T > &rhs)

  The copy constructor.

- void **rewind** ()

  Reset this ListIterator<T> to point to the first element in the List<T>.

- void **begin** ()

  Reset this ListIterator<T> to point to the first element in the List<T>.

- const T & **operator()** () const

  Return a constant reference to the object in the List<T> currently pointed to by this ListIterator<T>.

- T & **operator()** ()
- const T & **operator** ∗ () const

  Return a constant reference to the object in the List<T> currently pointed to by this ListIterator<T>.

- **operator bool** () const

  This is a conversion operator to makes the iterator look like a pointer.

- bool **ok** () const

  Return true if the iterator is not past the end of the list.

- bool **operator!** () const

  Returns true if ListIterator<T> doesn't point to any element on the List<T>.

- const T & **value** () const

  Return a constant reference to the object in the List<T> currently pointed to by the iterator.

- const T & **value** ()

  Return a constant reference to the object in the List<T> currently pointed to by the iterator.

- **ListIterator**< T > & **operator++** ()

  The prefix auto-increment operator.

- **ListIterator**< T > & **operator–** ()

  The prefix auto-decrement operator.

- **ListIterator**< T > **operator--** (int)

  The postfix auto-decrement operator.

- **ListIterator**< T > **operator++** (int)

  The postfix auto-increment operator.

- bool **operator==** (const **ListIterator**< T > &) const

  Equality test for two ListIterator<T>s.

- bool **operator!=** (const **ListIterator**< T > &) const

  Are the ListIterator<T>s not equal?

Protected Member Functions

- **ListIterator** (const **List**< T > &_list, **ListLink**< T > *_p)

Protected Attributes

- const **List**< T > & **list**
- **ListLink**< T > * **p**

Friends

- class **List**< **T** >

7.86.1 Detailed Description

template<class T> class ListIterator< T >

Iterator over a **List**.

The class ListIterator<T> is an iterator over class List<T>. This class does NOT provide a default constructor or an assignment operator.

7.86.2 Constructor & Destructor Documentation

7.86.2.1 template<class T> ListIterator< T >::ListIterator (const List< T > & *aList*) `[inline]`

Construct a ListIterator<T> to first element of aList.

7.86.2.2 template<class T> ListIterator< T >::ListIterator (const ListIterator< T > & *rhs*) `[inline]`

The copy constructor.

7.86.2.3 template<class T> ListIterator< T >::ListIterator (const List< T > & *_list*, ListLink< T > * *_p*) [inline, protected]

Construct a ListIterator<T> to a List<T> and object in that List<T>.

7.86.3 Member Function Documentation

7.86.3.1 template<class T> void ListIterator< T >::begin () [inline]

Reset this ListIterator<T> to point to the first element in the List<T>.

Same as **rewind()**, but included to be consistent with other iterators.

7.86.3.2 template<class T> bool ListIterator< T >::ok () const [inline]

Return true if the iterator is not past the end of the list.

Same as bool(), but included to be consistent with other iterators.

7.86.3.3 template<class T> const T & ListIterator< T >::operator * () const [inline]

Return a constant reference to the object in the List<T> currently pointed to by this ListIterator<T>.

7.86.3.4 template<class T> ListIterator< T >::operator bool () const [inline]

This is a conversion operator to makes the iterator look like a pointer.

This operator makes it easy to check if the iterator is pointing to an element on the List<T>. If the iterator has been moved off the List<T> or if the List<T> is empty, this conversion returns the NULL pointer.

7.86.3.5 template<class T> bool ListIterator< T >::operator! () const [inline]

Returns true if ListIterator<T> doesn't point to any element on the List<T>.

7.86.3.6 template<class T> bool ListIterator< T >::operator!= (const ListIterator< T > &) const [inline]

Are the ListIterator<T>s not equal?

7.86.3.7 template<class T> T & ListIterator< T >::operator() () [inline]

7.86.3.8 template<class T> const T & ListIterator< T >::operator() () const [inline]

Return a constant reference to the object in the List<T> currently pointed to by this ListIterator<T>.

7.86.3.9 template<class T> ListIterator< T > ListIterator< T >::operator++ (int) [inline]

The postfix auto-increment operator.

This advances the ListIterator<T> to point to the next element on the List<T>. It then returns a ListIterator<T> that points to the old element to allow for chaining with other operators.

7.86.3.10 template<class T> ListIterator< T > & ListIterator< T >::operator++ () [inline]

The prefix auto-increment operator.

Advances the ListIterator<T> to point to the next element on the List<T>. It then returns a reference to itself to allow for chaining with other operators.

7.86.3.11 template<class T> ListIterator< T > ListIterator< T >::operator-- (int) [inline]

The postfix auto-decrement operator.

Moves the ListIterator<T> to point to the previous element on the List<T>. It then returns a ListIterator<T> that points to the old element to allow for chaining with other operators.

7.86.3.12 template<class T> ListIterator< T > & ListIterator< T >::operator-- () [inline]

The prefix auto-decrement operator.

Moves theListIterator<T> to point to the previous element on the List<T>. It then returns a reference to itself to allow for chaining with other operators.

7.86.3.13 template<class T> bool ListIterator< T >::operator== (const ListIterator< T > &) const [inline]

Equality test for two ListIterator<T>s.

Do the two ListIterator<T>s point to the same List<T> and the same element within the List<T>?

7.86.3.14 template<class T> void ListIterator< T >::rewind () [inline]

Reset this ListIterator<T> to point to the first element in the List<T>.

7.86.3.15 template<class T> const T& ListIterator< T >::value ()

Return a constant reference to the object in the List<T> currently pointed to by the iterator.

7.86.3.16 template<class T> const T & ListIterator< T >::value () const [inline]

Return a constant reference to the object in the List<T> currently pointed to by the iterator.

7.86.4 Friends And Related Function Documentation

7.86.4.1 template<class T> friend class List< T > [friend]

7.86.5 Member Data Documentation

7.86.5.1 template<class T> const List<T>& ListIterator< T >::list [protected]

A reference to the List<T> to which we point.

7.86.5.2 template<class T> ListLink<T>* ListIterator< T >::p [protected]

A pointer to the element in the List<T> to which we point.

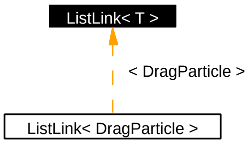
The documentation for this class was generated from the following file:

- **List.H**

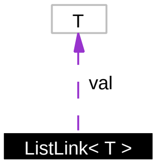
7.87 ListLink< T > Class Template Reference

`#include <List.H>`

Inheritance diagram for ListLink< T >:



Collaboration diagram for ListLink< T >:



Friends

- class **List< T >**
- class **ListIterator< T >**

template<class T> class ListLink< T >

7.87.1 Friends And Related Function Documentation

7.87.1.1 template<class T> friend class List< T > `[friend]`

7.87.1.2 template<class T> friend class ListIterator< T > `[friend]`

The documentation for this class was generated from the following file:

- **List.H**

7.88 Mask Class Reference

`#include <Mask.H>`

Public Types

- enum **cellType** { **maskCovered** = -2, **maskPhysical** = -1, **maskCopy** = 0, **maskCoarse** = 1 }

Public Member Functions

- void **buildMask** (**BaseFab**< int > &a_mask, const **ProblemDomain** &a_dProblem, const **BoxLayout** &a_grids, const **BoxLayout** *a_fineGridsPtr=NULL, int a_nRefFine=-1)
- void **buildMasks** (**LevelData**< **BaseFab**< int > > &a_masks, const **ProblemDomain** &a_dProblem, const **BoxLayout** &a_grids, const **BoxLayout** *a_fineGridsPtr=NULL, int a_nRefFine=-1)
- void **buildMask** (**BaseFab**< int > &a_mask, const **Box** &a_dProblem, const **BoxLayout** &a_grids, const **BoxLayout** *a_fineGridsPtr=NULL, int a_nRefFine=-1)
- void **buildMasks** (**LevelData**< **BaseFab**< int > > &a_masks, const **Box** &a_dProblem, const **BoxLayout** &a_grids, const **BoxLayout** *a_fineGridsPtr=NULL, int a_nRefFine=-1)

7.88.1 Member Enumeration Documentation

7.88.1.1 enum Mask::cellType

Enumeration values:

maskCovered

maskPhysical

maskCopy

maskCoarse

7.88.2 Member Function Documentation

7.88.2.1 void Mask::buildMask (BaseFab< int > & *a_mask*, const Box & *a_dProblem*, const BoxLayout & *a_grids*, const BoxLayout * *a_fineGridsPtr* = NULL, int *a_nRefFine* = -1)

7.88.2.2 void Mask::buildMask (BaseFab< int > & *a_mask*, const ProblemDomain & *a_dProblem*, const BoxLayout & *a_grids*, const BoxLayout * *a_fineGridsPtr* = NULL, int *a_nRefFine* = -1)

7.88.2.3 void Mask::buildMasks (LevelData< BaseFab< int > > & *a_masks*, const Box & *a_dProblem*, const BoxLayout & *a_grids*, const BoxLayout * *a_fineGridsPtr* = NULL, int *a_nRefFine* = -1)

7.88.2.4 void Mask::buildMasks (LevelData< BaseFab< int > > & *a_masks*, const ProblemDomain & *a_dProblem*, const BoxLayout & *a_grids*, const BoxLayout * *a_fineGridsPtr* = NULL, int *a_nRefFine* = -1)

The documentation for this class was generated from the following file:

- **Mask.H**

7.89 MayDay Class Reference

Error-reporting Functions.

`#include <MayDay.H>`

Static Public Member Functions

- void **Error** (const char *msg=**nullString**)

  Print out message to cerr and exit via **MayDay::Abort()**.

- void **Abort** (const char *msg=**nullString**)

  Print out message to cerr and exit via abort().

- void **Warning** (const char *msg=**nullString**)

  Issues warning to stderr, but does not halt program execution.

Static Protected Attributes

- const char * **nullString**

  A null string used as a default argument by the other member functions.

7.89.1 Detailed Description

Error-reporting Functions.

This class consists entirely of static error-reporting functions that are used throughout the MayDay implementation together with a version string. In the best of all possible worlds we'd use a namespace, but we aren't there, yet :-) Note that all the output functions add a terminating exclamation mark, so there is no need to add any punction to sentences that get output explicitly.

7.89.2 Member Function Documentation

7.89.2.1 void MayDay::Abort (const char * *msg* = nullString) `[static]`

Print out message to cerr and exit via abort().

7.89.2.2 void MayDay::Error (const char * *msg* = nullString) `[static]`

Print out message to cerr and exit via **MayDay::Abort()**.

7.89.2.3 void MayDay::Warning (const char * *msg* = nullString) `[static]`

Issues warning to stderr, but does not halt program execution.

7.89.3 Member Data Documentation

7.89.3.1 const char* MayDay::nullString [static, protected]

A null string used as a default argument by the other member functions.

The documentation for this class was generated from the following file:

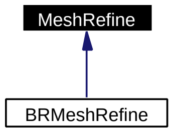
- **MayDay.H**

7.90 MeshRefine Class Reference

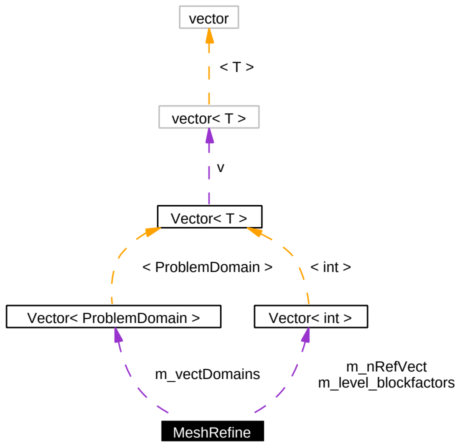
Class which manages grid generation.

`#include <MeshRefine.H>`

Inheritance diagram for MeshRefine:



Collaboration diagram for MeshRefine:



Public Member Functions

- **MeshRefine** ()

  default constructor – leaves object in an unusable state

- **MeshRefine** (const **Box** &a_baseDomain, const **Vector**< int > &a_refRatios, const **Real** a_fillRatio, const int a_blockFactor, const int a_bufferSize, const int a_maxSize)

  full constructor – leaves object in usable state

- **MeshRefine** (const **ProblemDomain** &a_baseDomain, const **Vector**< int > &a_refRatios, const **Real** a_fillRatio, const int a_blockFactor, const int a_bufferSize, const int a_maxSize)

  full constructor – leaves object in usable state

- virtual ~**MeshRefine** ()

  destructor

- void **define** (const **Box** &a_baseDomain, const **Vector**< int > &a_refRatios, const **Real** a_fillRatio, const int a_blockFactor, const int a_bufferSize, const int a_maxSize) *define function – size of RefRatios will define maximum number of levels*

- void **define** (const **ProblemDomain** &a_baseDomain, const **Vector**< int > &a_refRatios, const **Real** a_fillRatio, const int a_blockFactor, const int a_bufferSize, const int a_maxSize)

  define function – size of RefRatios will define maximum number of levels

- virtual int **regrid** (**Vector**< **Vector**< **Box** > > &a_newmeshes, const **IntVectSet** &a_tags, const int a_baseLevel, const int a_topLevel, const **Vector**< **Vector**< **Box** > > &a_oldMeshes) const

  create hierarchy of grids from a single level of tags

- virtual int **regrid** (**Vector**< **Vector**< **Box** > > &a_newmeshes, const **Vector**< **IntVectSet** > &a_tags, const int a_baseLevel, const int a_topLevel, const **Vector**< **Vector**< **Box** > > &a_oldMeshes) const

  create hierarchy of grids from tags at all levels

- const **Vector**< int > & **refRatios** () const

  returns vector of refinement ratios

- **Real fillRatio** () const

  returns fillRatio

- int **blockFactor** () const

  returns blocking factor

- int **bufferSize** () const

  returns proper nesting buffer size

- int **maxSize** () const

  returns maximum box size in any dimension – 0 means no limit

- void **refRatios** (const **Vector**< int > &a_nRefVect)

  sets vector of refinement ratios

- void **fillRatio** (const **Real** a_fill_ratio)

  sets fillRatio

- void **blockFactor** (const int a_block_factor)

  sets blocking factor

- void **bufferSize** (const int a_buffer_size)

  sets proper nesting buffer size

- void **maxSize** (const int a_max_size)

  sets maximum box size in any dimension – 0 means no limit

- bool **isDefined** () const

  has this object been defined properly?

- void **granularity** (int a_granularity) *sets proper nesting region granularity.*

- virtual void **makeBoxes** (**Vector**< **Box** > &a_mesh, const **IntVectSet** &a_tags, const **IntVectSet** &a_pnd, const **ProblemDomain** &a_domain, const int a_maxSize) const=0

  constructs a set of boxes which covers a set of tagged cells

Protected Member Functions

- virtual void **computeLocalBlockFactors** ()

  computes local blockFactors used internally to enforce the BlockFactor

- virtual void **makePNDs** (**Vector**< **IntVectSet** > &a_pnds, const int a_baseLevel, const int a_topLevel, const **Vector**< **ProblemDomain** > &a_domains, const **IntVectSet** &a_baseMesh, const **Vector**< int > &a_bufferSize) const

  Computes proper nesting domains.

Protected Attributes

- bool **m_isDefined**
- **Vector**< **ProblemDomain** > **m_vectDomains**
- **Vector**< int > **m_nRefVect**
- **Real m_fillRatio**
- int **m_blockFactor**
- **Vector**< int > **m_level_blockfactors**
- int **m_bufferSize**
- int **m_maxSize**
- int **m_granularity**

7.90.1 Detailed Description

Class which manages grid generation.

This class manages grid generation from sets of tagged cells. It is designed to be a pure virtual base class from which another class may be derived with a specific grid-generation algorithm (for example, the **BRMeshRefine** class).

There are two ways grids can be defined based on tagged cells. one takes a single **IntVectSet** of tags defined on the BaseLevel mesh and uses that set of tags for every level to be refined; the other takes a Vector<IntVectSet> of tags defined on all the mesh levels to be refined and uses those.

Long Description:

Create new meshes based on tagged cells on a range of levels of a mesh hierarchy. Each level of tagged cells is used to generate a new mesh at the next finer level. The finest level in the output mesh will be one level higher than the top of the range of levels given as input. As a special case, use the same tags (appropriately refined) for all levels.

Usage:

Call the regrid functions after computing error estimates and tagging cells. To add a new mesh level, set TopLevel to the index of the finest level in the existing mesh and define tags on the finest level. To keep the existing number of mesh levels, set TopLevel to one less than the index of the finest level and don't define any tags on the finest level. If a single **IntVectSet** of tags is passed (instead of a Vector<IntVectSet>) then the same tags (properly refined) will be used for all the new meshes up to level TopLevel+1. In any case, the meshes at levels BaseLevel and below are not modified. The output argument newmeshes will be reallocated to the necessary size before being used. When this function returns, the elements of the newmeshes vector corresponding to the unchanged levels will be filled in with copies of the levels from the old mesh vector. The variable tags is modified in an undefined way, so its contents should not be relied upon. The variable BlockFactor specifies the amount by which each box will be coarsenable. Every grid box will have an integral multiple of BlockFactor cells in each dimension and also lower index values that are integral multiples. As a side effect, the minimum box size will be BlockFactor.

Expensive validations are done only when debugging is enabled (i.e. the DEBUG make variable is "TRUE").

Usage Notes:

All the input vectors should be defined with max index >= TopLevel. They should have values for indices [BaseLevel:TopLevel]. (except for OldMeshes, which must be defined for all indices). The new mesh vector newmeshes will be redefined up to index TopLevel+1. RefRatios should be defined such that RefRatios[L] is the value to use to refine the level L mesh to produce the level L+1 mesh. The tags vector is modified in an undefined manner. The output variable newmeshes may not be completely defined if an exception occurs. The BlockFactor can be used to force a minimum box size.

7.90.2 Constructor & Destructor Documentation

7.90.2.1 MeshRefine::MeshRefine ()

default constructor – leaves object in an unusable state

7.90.2.2 MeshRefine::MeshRefine (const Box & *a_baseDomain*, const Vector< int > & *a_refRatios*, const Real *a_fillRatio*, const int *a_blockFactor*, const int *a_bufferSize*, const int *a_maxSize*)

full constructor – leaves object in usable state

Parameters:

a_baseDomain level 0 domain

a_refRatios refinement ratios – refRatio[0] is btwn levels 0 and 1

a_fillRatio measure of how efficiently tagged cells will be covered

a_blockFactor amount by which grids are guaranteed to be coarsenable

a_bufferSize proper nesting buffer amount

a_maxSize maximum grid length in any direction – 0 means no limit.

7.90.2.3 MeshRefine::MeshRefine (const ProblemDomain & *a_baseDomain*, const Vector< int > & *a_refRatios*, const Real *a_fillRatio*, const int *a_blockFactor*, const int *a_bufferSize*, const int *a_maxSize*)

full constructor – leaves object in usable state

Parameters:

a_baseDomain level 0 domain

a_refRatios refinement ratios – refRatio[0] is btwn levels 0 and 1

a_fillRatio measure of how efficiently tagged cells will be covered

a_blockFactor amount by which grids are guaranteed to be coarsenable

a_bufferSize proper nesting buffer amount

a_maxSize maximum grid length in any direction – 0 means no limit.

7.90.2.4 virtual MeshRefine::~MeshRefine () `[virtual]`

destructor

7.90.3 Member Function Documentation

7.90.3.1 void MeshRefine::blockFactor (const int *a_block_factor*)

sets blocking factor

7.90.3.2 int MeshRefine::blockFactor () const

returns blocking factor

7.90.3.3 void MeshRefine::bufferSize (const int *a_buffer_size*)

sets proper nesting buffer size

7.90.3.4 int MeshRefine::bufferSize () const

returns proper nesting buffer size

7.90.3.5 virtual void MeshRefine::computeLocalBlockFactors () `[protected, virtual]`

computes local blockFactors used internally to enforce the BlockFactor

This function computes values for m_local_blockfactors array, which is the amount that tags on a level are coarsened in order to guarantee that the grids on the next finer level are coarsenable by the BlockFactor.

7.90.3.6 void MeshRefine::define (const ProblemDomain & *a_baseDomain*, const Vector< int > & *a_refRatios*, const Real *a_fillRatio*, const int *a_blockFactor*, const int *a_bufferSize*, const int *a_maxSize*)

define function – size of RefRatios will define maximum number of levels

Parameters:

a_baseDomain level 0 domain

a_refRatios refinement ratios – refRatio[0] is btwn levels 0 and 1

a_fillRatio measure of how efficiently tagged cells will be covered

a_blockFactor amount by which grids are guaranteed to be coarsenable

a_bufferSize proper nesting buffer amount

a_maxSize maximum grid length in any direction – 0 means no limit

Reimplemented in **BRMeshRefine** (p. 163).

7.90.3.7 void MeshRefine::define (const Box & *a_baseDomain*, const Vector< int > & *a_refRatios*, const Real *a_fillRatio*, const int *a_blockFactor*, const int *a_bufferSize*, const int *a_maxSize*)

define function – size of RefRatios will define maximum number of levels

Parameters:

a_baseDomain level 0 domain

a_refRatios refinement ratios – refRatio[0] is btwn levels 0 and 1

a_fillRatio measure of how efficiently tagged cells will be covered

a_blockFactor amount by which grids are guaranteed to be coarsenable

a_bufferSize proper nesting buffer amount

a_maxSize maximum grid length in any direction – 0 means no limit

Reimplemented in **BRMeshRefine** (p. 163).

7.90.3.8 void MeshRefine::fillRatio (const Real *a_fill_ratio*)

sets fillRatio

7.90.3.9 Real MeshRefine::fillRatio () const

returns fillRatio

7.90.3.10 void MeshRefine::granularity (int *a_granularity*)

sets proper nesting region granularity.

7.90.3.11 bool MeshRefine::isDefined () const

has this object been defined properly?

7.90.3.12 virtual void MeshRefine::makeBoxes (Vector< Box > & *a_mesh*, const IntVectSet & *a_tags*, const IntVectSet & *a_pnd*, const ProblemDomain & *a_domain*, const int *a_maxSize*) const `[pure virtual]`

constructs a set of boxes which covers a set of tagged cells

constructs a set of boxes which covers a set of tagged cells by using the algorithm of choice. Everything should be on the same level, and blocking factor is not applied. Boxes will be on the same refinement level as the tags. This would normally be a protected function, but it can be useful to call it on its own, so it has been left public.

Parameters:

a_mesh output: refined boxes at each new level

a_tags input: set of tagged cells to cover

a_pnd input: proper nesting domain in which mesh boxes must live

a_domain input: physical domain

a_maxSize input: largest number of cells in any dimension for any box

Implemented in **BRMeshRefine** (p. 164).

7.90.3.13 virtual void MeshRefine::makePNDs (Vector< IntVectSet > & *a_pnds*, const int *a_baseLevel*, const int *a_topLevel*, const Vector< ProblemDomain > & *a_domains*, const IntVectSet & *a_baseMesh*, const Vector< int > & *a_bufferSize*) const `[protected, virtual]`

Computes proper nesting domains.

This should only be called by refine. it assumes that everything has already been coarsened by the local blocking factor

Parameters:

a_pnds output: proper nesting domains at each level

a_baseLevel input: index of highest **AMR** level not to be refined

a_topLevel input: index of highest **AMR** level in output

a_domains input: (same as in *meshRefine*)

a_baseMesh input: boxes at mesh level *BaseLevel*

a_bufferSize input: (similar to *meshRefine*; but with level-dependent coarsening factors)

7.90.3.14 void MeshRefine::maxSize (const int *a_max_size*)

sets maximum box size in any dimension – 0 means no limit

7.90.3.15 int MeshRefine::maxSize () const

returns maximum box size in any dimension – 0 means no limit

7.90.3.16 void MeshRefine::refRatios (const Vector< int > & *a_nRefVect*)

sets vector of refinement ratios

7.90.3.17 const Vector<int>& MeshRefine::refRatios () const

returns vector of refinement ratios

7.90.3.18 virtual int MeshRefine::regrid (Vector< Vector< Box > > & *a_newmeshes*, const Vector< IntVectSet > & *a_tags*, const int *a_baseLevel*, const int *a_topLevel*, const Vector< Vector< Box > > & *a_oldMeshes*) const `[virtual]`

create hierarchy of grids from tags at all levels

This function creates a hierarchy of grids from tags at all refinement levels. It is possible that not all levels will return with grids, since there may not be tags at all levels. Returns the new finest level of grids.

Parameters:

a_newmeshes new set of grids at every level

a_tags tagged cells on each existing level

a_baseLevel index of base mesh level (finest unchanged level)

a_topLevel top level to refine (one less than finest possible level)

a_oldMeshes existing grids (if no previous grids, set to domains)

7.90.3.19 virtual int MeshRefine::regrid (Vector< Vector< Box > > & *a_newmeshes*, const IntVectSet & *a_tags*, const int *a_baseLevel*, const int *a_topLevel*, const Vector< Vector< Box > > & *a_oldMeshes*) const `[virtual]`

create hierarchy of grids from a single level of tags

This function creates a hierarchy of grids from a single level of tags on BaseLevel. If tags exist, then all levels will have grids. Returns the new finest level of grids.

Parameters:

a_newmeshes new set of grids at every level

a_tags tagged cells on baseLevel

a_baseLevel index of base mesh level (finest unchanged level)

a_topLevel top level to refine (one less than finest possible level)

a_oldMeshes existing grids (if no previous grids, set to domains)

7.90.4 Member Data Documentation

7.90.4.1 int MeshRefine::m_blockFactor [protected]

7.90.4.2 int MeshRefine::m_bufferSize [protected]

7.90.4.3 Real MeshRefine::m_fillRatio [protected]

7.90.4.4 int MeshRefine::m_granularity [protected]

7.90.4.5 bool MeshRefine::m_isDefined [protected]

7.90.4.6 Vector<int> MeshRefine::m_level_blockfactors [protected]

7.90.4.7 int MeshRefine::m_maxSize [protected]

7.90.4.8 Vector<int> MeshRefine::m_nRefVect [protected]

7.90.4.9 Vector<ProblemDomain> MeshRefine::m_vectDomains [protected]

The documentation for this class was generated from the following file:

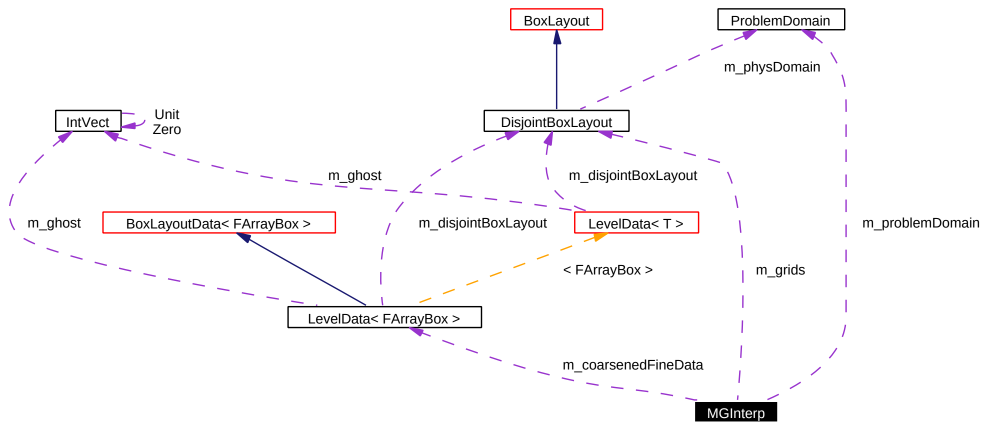
- **MeshRefine.H**

7.91 MGInterp Class Reference

Internal Chombo class to interpolate coarse level data to a fine level.

`#include <MGInterp.H>`

Collaboration diagram for MGInterp:



Public Member Functions

- **MGInterp** ()

  Null constructor.

- ~**MGInterp** ()

  Destructor.

- **MGInterp** (const **DisjointBoxLayout** &a_fineDomain, int a_numcomps, int a_refRatio, const **Box** &a_problemDomain)

  Full constructor (calls full define).

- **MGInterp** (const **DisjointBoxLayout** &a_fineDomain, int a_numcomps, int a_refRatio, const **ProblemDomain** &a_problemDomain)

  Full constructor (calls full define).

- void **define** (const **DisjointBoxLayout** &a_fineDomain, int a_numcomps, int a_refRatio, const **Box** &a_problemDomain)

  Full define.

- void **define** (const **DisjointBoxLayout** &a_fineDomain, int a_numcomps, int a_refRatio, const **ProblemDomain** &a_problemDomain)

  Full define.

- void **interpToFine** (**LevelData**< **FArrayBox** > &a_fineData, const **LevelData**< **FArrayBox** > &a_coarseData)
- bool **isDefined** () const

  Is the object defined.

Protected Attributes

- bool **m_isDefined**
- int **m_refRatio**
- **LevelData< FArrayBox > m_coarsenedFineData**
- **DisjointBoxLayout m_grids**
- **ProblemDomain m_problemDomain**

7.91.1 Detailed Description

Internal Chombo class to interpolate coarse level data to a fine level.

Class to interpolate coarse level data to a fine level data for multigrid. This class should be considered internal to **AMRSolver** and not a part of the Chombo API.

7.91.2 Constructor & Destructor Documentation

7.91.2.1 MGInterp::MGInterp ()

Null constructor.

7.91.2.2 MGInterp::~MGInterp ()

Destructor.

7.91.2.3 MGInterp::MGInterp (const DisjointBoxLayout & *a_fineDomain*, int *a_numcomps*, int *a_refRatio*, const Box & *a_problemDomain*)

Full constructor (calls full define).

7.91.2.4 MGInterp::MGInterp (const DisjointBoxLayout & *a_fineDomain*, int *a_numcomps*, int *a_refRatio*, const ProblemDomain & *a_problemDomain*)

Full constructor (calls full define).

7.91.3 Member Function Documentation

7.91.3.1 void MGInterp::define (const DisjointBoxLayout & *a_fineDomain*, int *a_numcomps*, int *a_refRatio*, const ProblemDomain & *a_problemDomain*)

Full define.

Specify the fine level domain and refinement ratio.

7.91.3.2 void MGInterp::define (const DisjointBoxLayout & *a_fineDomain*, int *a_numcomps*, int *a_refRatio*, const Box & *a_problemDomain*)

Full define.

Specify the fine level domain and refinement ratio.

7.91.3.3 void MGInterp::interpToFine (LevelData< FArrayBox > & *a_fineData*, const LevelData< FArrayBox > & *a_coarseData*)

Interpolate coarse level data to fine level

7.91.3.4 bool MGInterp::isDefined () const

Is the object defined.

7.91.4 Member Data Documentation

7.91.4.1 LevelData<FArrayBox> MGInterp::m_coarsenedFineData [protected]

7.91.4.2 DisjointBoxLayout MGInterp::m_grids [protected]

7.91.4.3 bool MGInterp::m_isDefined [protected]

7.91.4.4 ProblemDomain MGInterp::m_problemDomain [protected]

7.91.4.5 int MGInterp::m_refRatio [protected]

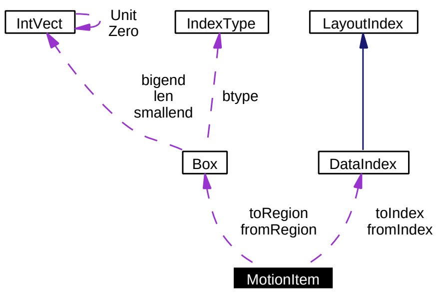
The documentation for this class was generated from the following file:

- **MGInterp.H**

7.92 MotionItem Class Reference

`#include <Copier.H>`

Collaboration diagram for MotionItem:



Public Member Functions

- **MotionItem** (const **DataIndex** &a_from, const **DataIndex** &a_to, const **Box** &a_region)
- **MotionItem** (const **DataIndex** &a_from, const **DataIndex** &a_to, const **Box** &a_fromRegion, const **Box** &a_toRegion)
- void **reverse** ()

Public Attributes

- **DataIndex fromIndex**
- **DataIndex toIndex**
- **Box fromRegion**
- **Box toRegion**
- int **procID**

7.92.1 Constructor & Destructor Documentation

7.92.1.1 MotionItem::MotionItem (const DataIndex & *a_from*, const DataIndex & *a_to*, const Box & *a_region*) `[inline]`

7.92.1.2 MotionItem::MotionItem (const DataIndex & *a_from*, const DataIndex & *a_to*, const Box & *a_fromRegion*, const Box & *a_toRegion*) `[inline]`

7.92.2 Member Function Documentation

7.92.2.1 void MotionItem::reverse () `[inline]`

7.92.3 Member Data Documentation

7.92.3.1 DataIndex MotionItem::fromIndex

7.92.3.2 Box MotionItem::fromRegion

7.92.3.3 int MotionItem::procID

7.92.3.4 DataIndex MotionItem::toIndex

7.92.3.5 Box MotionItem::toRegion

The documentation for this class was generated from the following file:

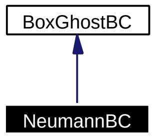
- **Copier.H**

7.93 NeumannBC Class Reference

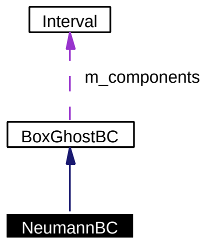
Class to enforce homogeneous Neumann boundary conditions.

`#include <PoissonBC.H>`

Inheritance diagram for NeumannBC:



Collaboration diagram for NeumannBC:



Public Member Functions

- **NeumannBC** ()

  Null constructor.

- **NeumannBC** (int dir, **Side::LoHiSide** sd)

  Basic constructor – sets **Interval** *to (0,0).*

- **NeumannBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)

  Full constructor.

- **~NeumannBC** ()

  Destructor.

Protected Member Functions

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const
- **BoxGhostBC** ∗ **new_boxghostbc** () const

Friends

- class **DomainGhostBC**

7.93.1 Detailed Description

Class to enforce homogeneous Neumann boundary conditions.

BoxGhostBC class implentation to enforce homogeneous neumann boundary conditions Recall that **BoxGhostBC** is a class to encapsulate the operations of ghost-cell boundary conditions at a face. If the solution is phi and the face normal direction is x, the boundary conditions usually used can be expressed as A∗phi + B∗dphi/dx = C. For homogeneous Neumann BCs, A=C=0, B= 1 These functions are only meant to be called by **DomainGhostBC**

7.93.2 Constructor & Destructor Documentation

7.93.2.1 NeumannBC::NeumannBC ()

Null constructor.

7.93.2.2 NeumannBC::NeumannBC (int *dir*, Side::LoHiSide *sd*)

Basic constructor – sets **Interval** to (0,0).

7.93.2.3 NeumannBC::NeumannBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

Full constructor.

7.93.2.4 NeumannBC::∼NeumannBC ()

Destructor.

7.93.3 Member Function Documentation

7.93.3.1 virtual void NeumannBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const `[protected, virtual]`

Fills coefficient arrays – fills neumann factor with 1.0 and other two fields with 0.0.

Implements **BoxGhostBC** (p. 141).

7.93.3.2 virtual void NeumannBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const `[protected, virtual]`

Fills coefficient arrays – fills neumann factor with 1.0 and other two fields with 0.0.

Implements **BoxGhostBC** (p. 141).

7.93.3.3 BoxGhostBC* NeumannBC::new_boxghostbc () const [protected, virtual]

Virtual constructor workaround.

Implements **BoxGhostBC** (p. 141).

7.93.4 Friends And Related Function Documentation

7.93.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 142).

The documentation for this class was generated from the following file:

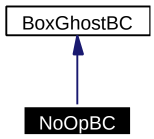
- **PoissonBC.H**

7.94 NoOpBC Class Reference

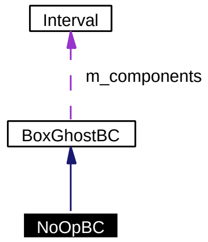
class to do nothing on an edge

`#include <edgeGhostBC.H>`

Inheritance diagram for NoOpBC:



Collaboration diagram for NoOpBC:



Public Member Functions

- **NoOpBC** ()
- **NoOpBC** (int a_dir, **Side::LoHiSide** a_sd)
- **NoOpBC** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps)
- virtual ~**NoOpBC** ()

Protected Member Functions

- virtual **BoxGhostBC** ∗ **new_boxghostbc** () const

    virtual constructor workaround

- void **define** (int a_dir, **Side::LoHiSide** a_sd)

    sets interval to be (0,0)

- void **define** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps)
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

    deprecated interface using **Box** *instead of* **ProblemDomain**

- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const

- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const

  deprecated interface using **Box** *instead of* **ProblemDomain**

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

7.94.1 Detailed Description

class to do nothing on an edge

This class functions as a placeholder – if you don't want to change the values on this edge, but it's a physical boundary, it's the clas you want!

7.94.2 Constructor & Destructor Documentation

7.94.2.1 NoOpBC::NoOpBC ()

7.94.2.2 NoOpBC::NoOpBC (int *a_dir*, Side::LoHiSide *a_sd*)

7.94.2.3 NoOpBC::NoOpBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*)

7.94.2.4 virtual NoOpBC::~NoOpBC () `[virtual]`

7.94.3 Member Function Documentation

7.94.3.1 virtual void NoOpBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface using **Box** instead of **ProblemDomain**

Reimplemented from **BoxGhostBC** (p. 140).

7.94.3.2 virtual void NoOpBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.94.3.3 virtual void NoOpBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface using **Box** instead of **ProblemDomain**

Reimplemented from **BoxGhostBC** (p. 141).

7.94.3.4 virtual void NoOpBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_-domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.94.3.5 void NoOpBC::define (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*) [protected]

Full define function

Reimplemented from **BoxGhostBC** (p. 141).

7.94.3.6 void NoOpBC::define (int *a_dir*, Side::LoHiSide *a_sd*) [protected]

sets interval to be (0,0)

Reimplemented from **BoxGhostBC** (p. 141).

7.94.3.7 virtual void NoOpBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const [protected, virtual]

Deprecated function – use ProplemDomain version instead

Implements **BoxGhostBC** (p. 141).

7.94.3.8 virtual void NoOpBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const [protected, virtual]

Pure-virtual Worker function to fill arrays which define the boundary condition. Essentially, if the boundary condition is represented as A*phi + B*dphi/dn = C, then a_dircfac is A, a_neumfac is B, and a_inhmval is C.

Implements **BoxGhostBC** (p. 141).

7.94.3.9 virtual BoxGhostBC* NoOpBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround

Implements **BoxGhostBC** (p. 141).

The documentation for this class was generated from the following file:

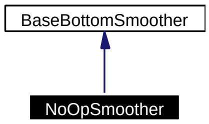
- **edgeGhostBC.H**

7.95 NoOpSmoother Class Reference

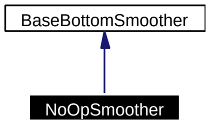
Does nothing as a bottom smoother.

`#include <NoOpSmoother.H>`

Inheritance diagram for NoOpSmoother:



Collaboration diagram for NoOpSmoother:



Public Member Functions

- **NoOpSmoother** ()

  Null constructor.

- virtual ~**NoOpSmoother** ()

  Destructor.

- virtual **BaseBottomSmoother** ∗ **new_bottomSmoother** () const

  This gets around the "no virtual constructor" rule.

- virtual void **doBottomSmooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs, **LevelOp** ∗a_levelopPtr)

  Do (no) smoothing.

- virtual void **setConvergenceMetric** (**Real** a_metric, int a_comp=0)

  sets metric to judge convergence (normally norm(rhs) of outer problem

- virtual void **setSolverTol** (**Real** a_tol)

  this doesn't really do anything, but is required by **BaseBottomSmoother**

7.95.1 Detailed Description

Does nothing as a bottom smoother.

The NoOpSmoother class is a bottom smoother for LevelOp-derived classes which does nothing (mostly as a debugging test option)

7.95.2 Constructor & Destructor Documentation

7.95.2.1 NoOpSmoother::NoOpSmoother ()

Null constructor.

7.95.2.2 virtual NoOpSmoother::~NoOpSmoother () `[virtual]`

Destructor.

7.95.3 Member Function Documentation

7.95.3.1 virtual void NoOpSmoother::doBottomSmooth (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*, LevelOp * *a_levelopPtr*) `[virtual]`

Do (no) smoothing.

True to its name, this class does nothing in this functino

Implements **BaseBottomSmoother** (p. 69).

7.95.3.2 virtual BaseBottomSmoother* NoOpSmoother::new_bottomSmoother () const `[virtual]`

This gets around the "no virtual constructor" rule.

Implements **BaseBottomSmoother** (p. 69).

7.95.3.3 virtual void NoOpSmoother::setConvergenceMetric (Real *a_metric*, int *a_comp* = 0) `[inline, virtual]`

sets metric to judge convergence (normally norm(rhs) of outer problem

To make the iterative solver convergence more consistent, the solver will consider itself "solved" if the residual is reduced to epsilon*metric, where epsilon is the solver tolerance (set by setSolverTol). A good starting point for the metric is the norm of the RHS of the original problem (i.e. the rhs sent in to **LevelSolver** or **AMRSolver**). Default value is 1.0.

Implements **BaseBottomSmoother** (p. 69).

7.95.3.4 virtual void NoOpSmoother::setSolverTol (Real *a_tol*) `[inline, virtual]`

this doesn't really do anything, but is required by **BaseBottomSmoother**

Implements **BaseBottomSmoother** (p. 69).

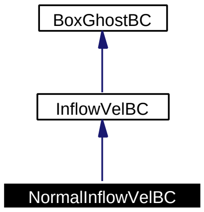
The documentation for this class was generated from the following file:

- **NoOpSmoother.H**

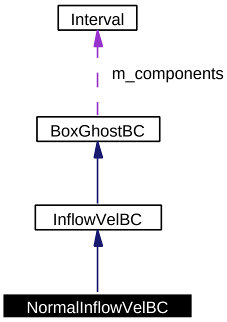
7.96 NormalInflowVelBC Class Reference

`#include <channelBC.H>`

Inheritance diagram for NormalInflowVelBC:



Collaboration diagram for NormalInflowVelBC:



Public Member Functions

- **NormalInflowVelBC** ()
- virtual ~**NormalInflowVelBC** ()
- **NormalInflowVelBC** (int dir, **Side::LoHiSide** sd)
- **NormalInflowVelBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)

Protected Member Functions

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dirfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

  deprecated interface

- virtual **BoxGhostBC** * **new_boxghostbc** () const

Friends

- class **DomainGhostBC**

7.96.1 Constructor & Destructor Documentation

7.96.1.1 NormalInflowVelBC::NormalInflowVelBC ()

7.96.1.2 virtual NormalInflowVelBC::~NormalInflowVelBC () `[virtual]`

7.96.1.3 NormalInflowVelBC::NormalInflowVelBC (int *dir*, Side::LoHiSide *sd*)

7.96.1.4 NormalInflowVelBC::NormalInflowVelBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.96.2 Member Function Documentation

7.96.2.1 virtual void NormalInflowVelBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dirfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const `[protected, virtual]`

deprecated interface

Reimplemented from **InflowVelBC** (p. 305).

7.96.2.2 virtual BoxGhostBC* NormalInflowVelBC::new_boxghostbc () const `[protected, virtual]`

Virtual constructor workaround.

Reimplemented from **InflowVelBC** (p. 306).

7.96.3 Friends And Related Function Documentation

7.96.3.1 friend class DomainGhostBC `[friend]`

Reimplemented from **InflowVelBC** (p. 306).

The documentation for this class was generated from the following file:

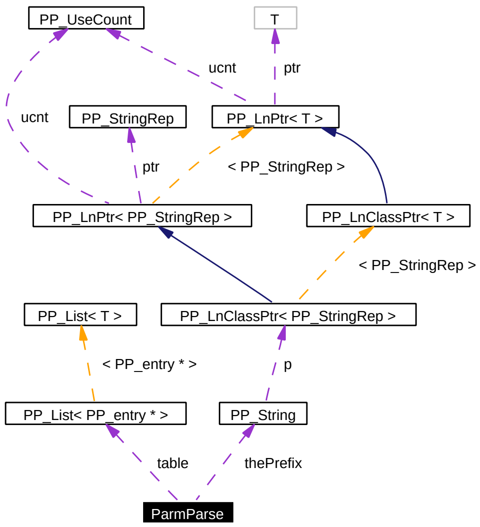
- **channelBC.H**

7.97 ParmParse Class Reference

Parse Parameters From Command Line and Input Files.

`#include <ParmParse.H>`

Collaboration diagram for ParmParse:



Public Types

- enum **PPType** {
  ppDefn, **ppOption**, **ppInt**, **ppFloat**,
  ppDouble, **ppString**, **ppEQ_sign**, **ppEOF** }

Public Member Functions

- **ParmParse** (int argc, char ∗∗argv, const char ∗prefix=0, const char ∗parfile=0)
- **ParmParse** (const char ∗prefix=0)
- **~ParmParse** ()
- bool **contains** (const char ∗name)
- bool **contains** (const std::string &name)
- int **countval** (const char ∗name, int n=-1)
- int **countname** (const char ∗name)
- int **countname** (const std::string &name)
- void **dumpTable** (std::ostream &os)
- void **get** (const char ∗name, int &ref, int ival=0)

  access single object

- int **query** (const char ∗name, int &ref, int ival=0)
- void **get** (const char ∗name, float &ref, int ival=0)
- int **query** (const char ∗name, float &ref, int ival=0)
- void **get** (const char ∗name, double &ref, int ival=0)

- int **query** (const char *name, double &ref, int ival=0)
- void **get** (const char *name, std::string &ref, int ival=0)

  Get the ival'th value of last occurrence of the requested name.

- int **query** (const char *name, std::string &ref, int ival=0)

  Get the ival'th value of last occurrence of the requested name.

- void **getarr** (const char *name, **Vector**< int > &ref, int start_ix, int num_val)

  access an array of objects

- void **getarr** (const char *name, std::vector< int > &ref, int start_ix, int num_val)

  access an array of objects

- int **queryarr** (const char *name, **Vector**< int > &ref, int start_ix, int num_val)

  access an array of objects

- int **queryarr** (const char *name, std::vector< int > &ref, int start_ix, int num_val)

  access an array of objects

- void **getarr** (const char *name, **Vector**< float > &ref, int start_ix, int num_val)

  access an array

- void **getarr** (const char *name, std::vector< float > &ref, int start_ix, int num_val)

  access an array

- int **queryarr** (const char *name, **Vector**< float > &ref, int start_ix, int num_val)
- int **queryarr** (const char *name, std::vector< float > &ref, int start_ix, int num_val)
- void **getarr** (const char *name, **Vector**< double > &ref, int start_ix, int num_val)
- void **getarr** (const char *name, std::vector< double > &ref, int start_ix, int num_val)
- int **queryarr** (const char *name, **Vector**< double > &ref, int start_ix, int num_val)
- int **queryarr** (const char *name, std::vector< double > &ref, int start_ix, int num_val)
- void **getarr** (const char *name, **Vector**< std::string > &ref, int start_ix, int num_val)
- void **getarr** (const char *name, std::vector< std::string > &ref, int start_ix, int num_val)
- int **queryarr** (const char *name, **Vector**< std::string > &ref, int start_ix, int num_val)
- int **queryarr** (const char *name, std::vector< std::string > &ref, int start_ix, int num_val)

Protected Member Functions

- void **bldTable** (const char *str, int lenstr, **PP_List**< **PP_entry** * > &tab)
- void **addDefn** (**PP_String** &def, **PP_List**< **PP_String** > &val, **PP_List**< **PP_entry** * > &tab)
- void **read_file** (const char *fname, **PP_List**< **PP_entry** * > &tab)
- **PPType getToken** (const char *, int &, int, char *)
- void **rmTable** ()
- void **ppinit** (const char *parfile)
- const **PP_entry** * **ppindex** (int n, const char *name) const
- void **getval** (const char *name, const **PPType** type, void *ptr, int ival, int k=-1)
- void **getarr** (const char *name, const **PPType** type, void *ptr, int start_ix, int num_val, int k=-1)
- int **queryval** (const char *name, const **PPType** type, void *ptr, int ival, int k=-1)

- int **queryarr** (const char *name, const **PPType** type, void *ptr, int start_ix, int num_val, int k=-1)
- bool **isInteger** (const **PP_String** &str, int &val)
- int **isDouble** (const **PP_String** &str, double &val)

Protected Attributes

- **PP_String thePrefix**

Static Protected Attributes

- **PP_List< PP_entry * > table**
- int **xargc**
- char ** **xargv**
- int **num_obj**

Friends

- class **PP_entry**

7.97.1 Detailed Description

Parse Parameters From Command Line and Input Files.

The ParmParse class is used to interpret parameters passed in to a program from the command line and an arbitrary collection of input files. The parameters are stored in static table that can be queried by any object of type ParmParse. A parameter can be either an "option" (usually specified on the command line) or a "definition". An option is of the form "-<name>" and is stored in the table without the hyphen. A definition is of the form "<name> = <value><value>...<value>". It is stored in the table as a name, value-list pair.

In the following example, verbose and no_opt are stored in the table as options. niter is a definition with the single integer value 10; name is a definition with the string value "big code" and dx is a definition with the two floating point values 0.5 and 0.75.

prog -verbose -no_opt niter = 10 name = "big code" dx = 0.5 0.75

The ParmParse class has two constructors. The first is responsible for building the table and is usually called by the main routine of an application. It has arguments for the command line argc and argv parameters, as well as an optional filename argument for reading definitions from an input file. The table is built by reading the input file first (if it exists) with the command line arguments added to the end of the table. The order of a definition in the table is significant, so command line parameters can be used to override definitions in the input file. A definition of the explicit form: FILE= is not added to the table but is a directive to include the named file at that point in the table.

The second constructor is generally used by other classes in the code. It permits access to the table via a large collection of query functions. Both constructors have an optional prefix argument that narrows the search to entries in the table with the same prefix. For example, let PlanR be a ParmParse object with code prefix "ope". PlanR.get("val",v) will look for an entry in the parameter list of the form: ope.val==, and will reject all entries not starting with the correct code prefix.

The query functions search the table for definition names that match a given string (and prefix) and return values from the corresponding value list. The values can be returned as ints, Array<int>s, floats, std::vector<float>s, doubles, std::vector<double>s, std::strings, or std::vector<std::sring>s. All values in the table are stored as **PP_String** objects, but if an int, float, or double is requested, the translation is done automatically. In the previous example, the value of niter could be returned as either an std::string, an int, a double, or a float. The values of dx can be returned as std::strings, floats, or doubles, but the value of name can be returned only as an std::string.

Comments in an input file include all text from a '#' character to the end of the line. Here is a sample input file:

-no_garbage

niter = 100

title = "Double Wammy"

cell_size = 0.5 0.75

plot.var = Density 1 10

plot.var = Energy 5 12

bigarray = 1 2 3 4 5 6 7 8

9 10 11 12

test = apple "boy blue" 10 20 30 40

FILE = prob_file

7.97.2 Member Enumeration Documentation

7.97.2.1 enum ParmParse::PPType

Enumeration values:

- **ppDefn**
- **ppOption**
- **ppInt**
- **ppFloat**
- **ppDouble**
- **ppString**
- **ppEQ_sign**
- **ppEOF**

7.97.3 Constructor & Destructor Documentation

7.97.3.1 ParmParse::ParmParse (int *argc*, char ** *argv*, const char * *prefix* = 0, const char * *parfile* = 0)

Construct an initial ParmParse object from the argc and argv passed in to main(). An error will be signalled if another ParmParse object currently exists. If parfile is specified, read the parameters in from that file first and then append those derived from argv to the table. If prefix is specified, load this string as the code prefix for this particular ParmParse object.

7.97.3.2 ParmParse::ParmParse (const char * *prefix* = 0)

Construct an additional ParmParse object sharing the same internal table as any other such objects in existence. If prefix is specified, load this string as the code prefix for this particular ParmParse object.

7.97.3.3 ParmParse::~ParmParse ()

The destructor. The internal static table will only be deleted if there are no other ParmParse objects in existence.

7.97.4 Member Function Documentation

7.97.4.1 void ParmParse::addDefn (PP_String & *def*, PP_List< PP_String > & *val*, PP_List< PP_entry * > & *tab*) `[protected]`

7.97.4.2 void ParmParse::bldTable (const char * *str*, int *lenstr*, PP_List< PP_entry * > & *tab*) `[protected]`

7.97.4.3 bool ParmParse::contains (const std::string & *name*)

Returns true if name is in table.

7.97.4.4 bool ParmParse::contains (const char * *name*)

Returns true if name is in table.

7.97.4.5 int ParmParse::countname (const std::string & *name*) `[inline]`

Returns the number of times the given name (prepended with prefix) appears in the table.

7.97.4.6 int ParmParse::countname (const char * *name*)

Returns the number of times the given name (prepended with prefix) appears in the table.

7.97.4.7 int ParmParse::countval (const char * *name*, int *n* = -1) `[inline]`

Returns the number of values associated with nth occurence of name (prepended with the prefix) in the table. n == -1 implies the last occurence.

7.97.4.8 void ParmParse::dumpTable (std::ostream & *os*)

Write the contents of the table in ASCII to the ostream.

7.97.4.9 void ParmParse::get (const char * *name*, std::string & *ref*, int *ival* = 0) `[inline]`

Get the ival'th value of last occurrence of the requested name.

Get the ival'th value of last occurrence of the requested name. If successful, the value is converted to a string and stored in reference ref. If the name does not exist or ival'th value does not exist, or if the printed representation of the value cannot be converted to a string, an error message is output and the program halts. Note that ival == 0 is the first value in the list.

7.97.4.10 void ParmParse::get (const char * *name*, double & *ref*, int *ival* = 0) [inline]

Get the ival'th value of last occurrence of the requested name. If successful, the value is converted to a double and stored in reference ref. If the name does not exist or ival'th value does not exist, or if the printed representation of the value cannot be converted to a double, an error message is output and the program halts. Note that ival == 0 is the first value in the list.

7.97.4.11 void ParmParse::get (const char * *name*, float & *ref*, int *ival* = 0) [inline]

Get the ival'th value of last occurrence of the requested name. If successful, the value is converted to a float and stored in reference ref. If the name does not exist or ival'th value does not exist, or if the printed representation of the value cannot be converted to a float, an error message is output and the program halts. Note that ival == 0 is the first value in the list.

7.97.4.12 void ParmParse::get (const char * *name*, int & *ref*, int *ival* = 0) [inline]

access single object

Get the ival'th value of last occurrence of the requested name. If successful, the value is converted to an int and stored in reference ref. If the name does not exist or ival'th value does not exist, or if the printed representation of the value cannot be converted to an int, an error message is output and the program halts. Note that ival == 0 is the first value in the list.

7.97.4.13 void ParmParse::getarr (const char * *name*, const PPType *type*, void * *ptr*, int *start_ix*, int *num_val*, int *k* = -1) [protected]

7.97.4.14 void ParmParse::getarr (const char * *name*, std::vector< std::string > & *ref*, int *start_ix*, int *num_val*) [inline]

Gets a std::vector<string> of num_val values from last occurrence of given name. If successful, the values are converted to a string and stored in the std::vector<string> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<string>[0], std::vector<string>[1] holds start_ix+1, etc. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a string, an error message is reported and the program halts.

7.97.4.15 void ParmParse::getarr (const char * *name*, Vector< std::string > & *ref*, int *start_ix*, int *num_val*) [inline]

Gets a std::vector<string> of num_val values from last occurrence of given name. If successful, the values are converted to a string and stored in the std::vector<string> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<string>[0], std::vector<string>[1] holds start_ix+1, etc. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a string, an error message is reported and the program halts.

7.97.4.16 void ParmParse::getarr (const char * *name*, std::vector< double > & *ref*, int *start_ix*, int *num_val*) [inline]

Gets a std::vector<double> of num_val values from last occurrence of given name. If successful, the values are converted to a double and stored in the std::vector<double> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<double>[0], std::vector<double>[1] holds start_ix+1, etc. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a double, an error message is reported and the program halts.

7.97.4.17 void ParmParse::getarr (const char * *name*, Vector< double > & *ref*, int *start_ix*, int *num_val*) [inline]

Gets a std::vector<double> of num_val values from last occurrence of given name. If successful, the values are converted to a double and stored in the std::vector<double> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<double>[0], std::vector<double>[1] holds start_ix+1, etc. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a double, an error message is reported and the program halts.

7.97.4.18 void ParmParse::getarr (const char * *name*, std::vector< float > & *ref*, int *start_ix*, int *num_val*) [inline]

access an array

Gets a std::vector<float> of num_val values from last occurrence of given name. If successful, the values are converted to a float and stored in the std::vector<float> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<float>[0], std::vector<float>[1] holds start_ix+1, etc. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a float, an error message is reported and the program halts.

7.97.4.19 void ParmParse::getarr (const char * *name*, Vector< float > & *ref*, int *start_ix*, int *num_val*) [inline]

access an array

Gets a std::vector<float> of num_val values from last occurrence of given name. If successful, the values are converted to a float and stored in the std::vector<float> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<float>[0], std::vector<float>[1] holds start_ix+1, etc. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a float, an error message is reported and the program halts.

7.97.4.20 void ParmParse::getarr (const char * *name*, std::vector< int > & *ref*, int *start_ix*, int *num_val*) [inline]

access an array of objects

Gets a std::vector<int> of num_val values from last occurrence of given name. If successful, the values are converted to an int and stored in the std::vector<int> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<int>[0], std::vector<int>[1] holds start_ix+1, etc. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to an int, an error message is reported and the program halts.

7.97.4.21 void ParmParse::getarr (const char ∗ *name*, Vector< int > & *ref*, int *start_ix*, int *num_val*) `[inline]`

access an array of objects

Gets a std::vector<int> of num_val values from last occurrence of given name. If successful, the values are converted to an int and stored in the std::vector<int> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<int>[0], std::vector<int>[1] holds start_ix+1, etc. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to an int, an error message is reported and the program halts.

7.97.4.22 PPType ParmParse::getToken (const char ∗, int &, int, char ∗) `[protected]`

7.97.4.23 void ParmParse::getval (const char ∗ *name*, const PPType *type*, void ∗ *ptr*, int *ival*, int *k* = -1) `[protected]`

7.97.4.24 int ParmParse::isDouble (const PP_String & *str*, double & *val*) `[inline, protected]`

7.97.4.25 bool ParmParse::isInteger (const PP_String & *str*, int & *val*) `[inline, protected]`

7.97.4.26 const PP_entry∗ ParmParse::ppindex (int *n*, const char ∗ *name*) const `[protected]`

7.97.4.27 void ParmParse::ppinit (const char ∗ *parfile*) `[protected]`

7.97.4.28 int ParmParse::query (const char ∗ *name*, std::string & *ref*, int *ival* = 0) `[inline]`

Get the ival'th value of last occurrence of the requested name.

Get the ival'th value of last occurrence of the requested name. If successful, the value is converted to a string and stored in reference ref. Returns 1 if successful. Returns 0 if the name does not exist. If ival'th value does not exist, or if the printed representation of the value cannot be converted to a string, an error message is output and the program halts. Note that ival == 0 is the first value in the list.

7.97.4.29 int ParmParse::query (const char ∗ *name*, double & *ref*, int *ival* = 0) `[inline]`

Get the ival'th value of last occurrence of the requested name. If successful, the value is converted to a double and stored in reference ref. Returns 1 if successful. Returns 0 if the name does not exist. If ival'th value does not exist, or if the printed representation of the value cannot be converted to a double, an error message is output and the program halts. Note that ival == 0 is the first value in the list.

7.97.4.30 int ParmParse::query (const char * *name*, float & *ref*, int *ival* = 0) [inline]

Get the ival'th value of last occurrence of the requested name. If successful, the value is converted to a float and stored in reference ref. Returns 1 if successful. Returns 0 if the name does not exist. If ival'th value does not exist, or if the printed representation of the value cannot be converted to a float, an error message is output and the program halts. Note that ival == 0 is the first value in the list.

7.97.4.31 int ParmParse::query (const char * *name*, int & *ref*, int *ival* = 0) [inline]

Get the ival'th value of last occurrence of the requested name. If successful, the value is converted to an int and stored in reference ref. Returns 1 if successful. Returns 0 if the name does not exist. If ival'th value does not exist, or if the printed representation of the value cannot be converted to an int, an error message is output and the program halts. Note that ival == 0 is the first value in the list.

7.97.4.32 int ParmParse::queryarr (const char * *name*, const PPType *type*, void * *ptr*, int *start_ix*, int *num_val*, int *k* = -1) [protected]

7.97.4.33 int ParmParse::queryarr (const char * *name*, std::vector< std::string > & *ref*, int *start_ix*, int *num_val*) [inline]

Gets a std::vector<string> of num_val values from last occurrence of given name. If successful, the values are converted to a string and stored in the std::vector<string> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<string>[0], std::vector<string>[1] holds start_ix+1, etc. Returns 0 if the name does not exist. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a string, an error message is reported and the program halts.

7.97.4.34 int ParmParse::queryarr (const char * *name*, Vector< std::string > & *ref*, int *start_ix*, int *num_val*) [inline]

Gets a std::vector<string> of num_val values from last occurrence of given name. If successful, the values are converted to a string and stored in the std::vector<string> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<string>[0], std::vector<string>[1] holds start_ix+1, etc. Returns 0 if the name does not exist. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a string, an error message is reported and the program halts.

7.97.4.35 int ParmParse::queryarr (const char * *name*, std::vector< double > & *ref*, int *start_ix*, int *num_val*) [inline]

Gets a std::vector<double> of num_val values from last occurrence of given name. If successful, the values are converted to a double and stored in the std::vector<double> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<double>[0], std::vector<double>[1] holds start_ix+1, etc. Returns 0 if the name does not exist. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a double, an error message is reported and the program halts.

7.97.4.36 int ParmParse::queryarr (const char * *name*, Vector< double > & *ref*, int *start_ix*, int *num_val*) `[inline]`

Gets a std::vector<double> of num_val values from last occurrence of given name. If successful, the values are converted to a double and stored in the std::vector<double> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<double>[0], std::vector<double>[1] holds start_ix+1, etc. Returns 0 if the name does not exist. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a double, an error message is reported and the program halts.

7.97.4.37 int ParmParse::queryarr (const char * *name*, std::vector< float > & *ref*, int *start_ix*, int *num_val*) `[inline]`

Gets a std::vector<float> of num_val values from last occurrence of given name. If successful, the values are converted to a float and stored in the std::vector<float> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<float>[0], std::vector<float>[1] holds start_ix+1, etc. Returns 0 if the name does not exist. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a float, an error message is reported and the program halts.

7.97.4.38 int ParmParse::queryarr (const char * *name*, Vector< float > & *ref*, int *start_ix*, int *num_val*) `[inline]`

Gets a std::vector<float> of num_val values from last occurrence of given name. If successful, the values are converted to a float and stored in the std::vector<float> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<float>[0], std::vector<float>[1] holds start_ix+1, etc. Returns 0 if the name does not exist. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to a float, an error message is reported and the program halts.

7.97.4.39 int ParmParse::queryarr (const char * *name*, std::vector< int > & *ref*, int *start_ix*, int *num_val*) `[inline]`

access an array of objects

Gets a std::vector<int> of num_val values from last occurrence of given name. If successful, the values are converted to an int and stored in the std::vector<int> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<int>[0], std::vector<int>[1] holds start_ix+1, etc. Returns 0 if the name does not exist. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to an int, an error message is reported and the program halts.

7.97.4.40 int ParmParse::queryarr (const char * *name*, Vector< int > & *ref*, int *start_ix*, int *num_val*) [inline]

access an array of objects

Gets a std::vector<int> of num_val values from last occurrence of given name. If successful, the values are converted to an int and stored in the std::vector<int> object ref. ref is resized (if necessary) to hold num_val values. The value in the list indexed by start_ix is copied into std::vector<int>[0], std::vector<int>[1] holds start_ix+1, etc. Returns 0 if the name does not exist. If there are fewer than start_ix + num_val values associated with the last occurrence, or if some of the values cannot be converted to an int, an error message is reported and the program halts.

7.97.4.41 int ParmParse::queryval (const char * *name*, const PPType *type*, void * *ptr*, int *ival*, int *k* = -1) [protected]

7.97.4.42 void ParmParse::read_file (const char * *fname*, PP_List< PP_entry * > & *tab*) [protected]

7.97.4.43 void ParmParse::rmTable () [protected]

7.97.5 Friends And Related Function Documentation

7.97.5.1 friend class PP_entry [friend]

7.97.6 Member Data Documentation

7.97.6.1 int ParmParse::num_obj [static, protected]

7.97.6.2 PP_List<PP_entry*> ParmParse::table [static, protected]

7.97.6.3 PP_String ParmParse::thePrefix [protected]

7.97.6.4 int ParmParse::xargc [static, protected]

7.97.6.5 char** ParmParse::xargv [static, protected]

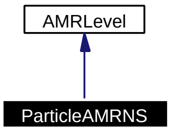
The documentation for this class was generated from the following file:

- **ParmParse.H**

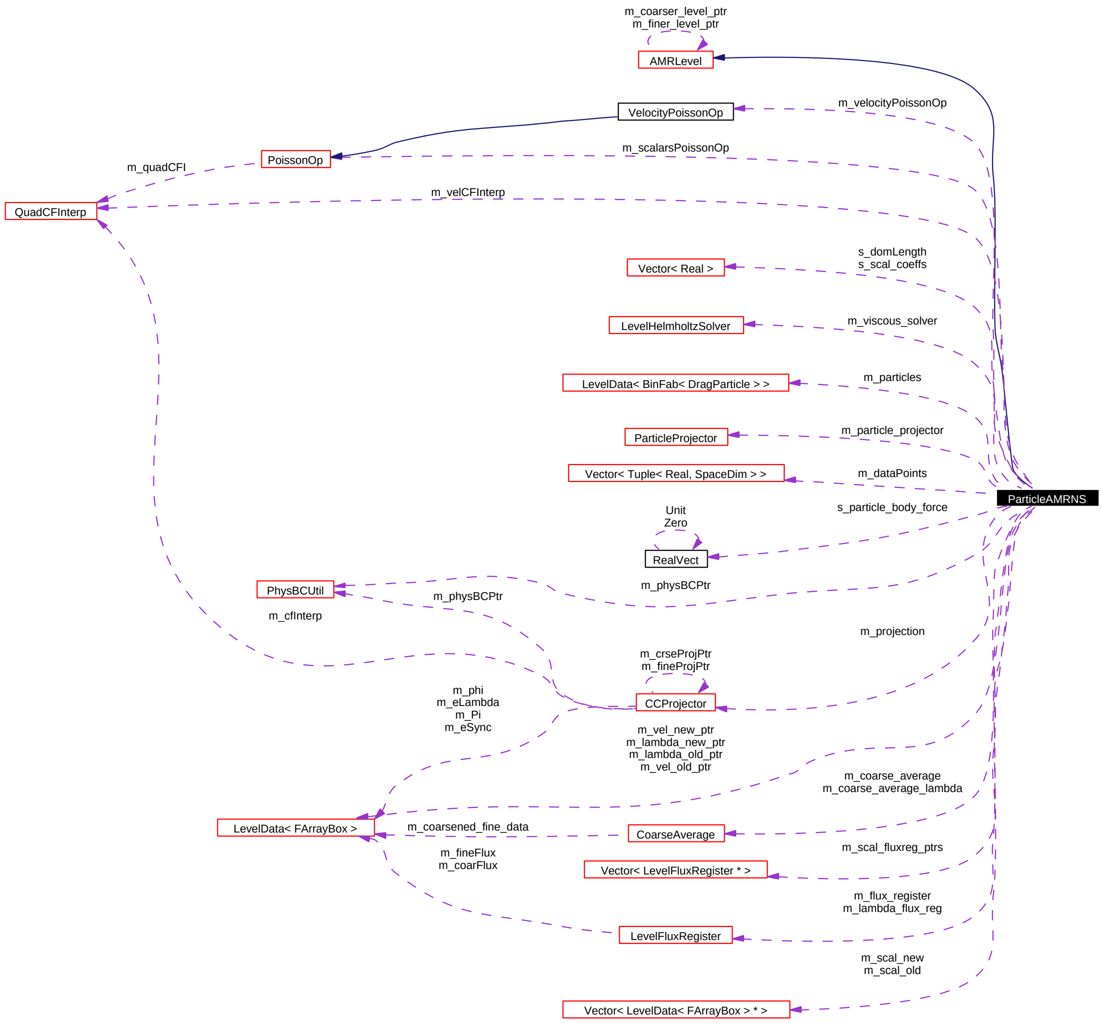
7.98 ParticleAMRNS Class Reference

`#include <ParticleAMRNS.H>`

Inheritance diagram for ParticleAMRNS:



Collaboration diagram for ParticleAMRNS:



Public Member Functions

- **ParticleAMRNS** ()

- **ParticleAMRNS** (**AMRLevel** ∗a_coarser_level_ptr, const **Box** &a_prob_domain, int a_level, int a_ref_ratio)
- **ParticleAMRNS** (**AMRLevel** ∗a_coarser_level_ptr, const **ProblemDomain** &a_prob_domain, int a_level, int a_ref_ratio)
- virtual ∼**ParticleAMRNS** ()
- virtual **AMRLevel** ∗ **makeAMRLevel** (**AMRLevel** ∗a_coarser_level_ptr, const **Box** &a_problem_domain, int a_level, int a_ref_ratio) const
- virtual **AMRLevel** ∗ **makeAMRLevel** (**AMRLevel** ∗a_coarser_level_ptr, const **ProblemDomain** &a_problem_domain, int a_level, int a_ref_ratio) const
- virtual void **define** (**AMRLevel** ∗a_coarse_level_ptr, const **Box** &a_problem_domain, int a_level, int a_ref_ratio)
- virtual void **define** (**AMRLevel** ∗a_coarse_level_ptr, const **ProblemDomain** &a_problem_domain, int a_level, int a_ref_ratio)
- virtual **Real advance** ()

  advance by one timestep (returns what?)
- virtual void **postTimeStep** ()

  things to do after a timestep
- virtual void **tagCells** (**IntVectSet** &a_tags)

  create tags
- virtual void **tagCellsInit** (**IntVectSet** &a_tags)

  create tags at initialization
- virtual void **regrid** (const **Vector**< **Box** > &a_new_grids)

  regrid
- virtual void **postRegrid** (int a_lBase)

  perform any post-regridding ops – lBase is the finest unchanged level
- virtual void **initialGrid** (const **Vector**< **Box** > &a_new_grids)

  initialize grids
- virtual void **initialData** ()

  initialize data
- virtual void **postInitialize** ()

  things do do after initialization
- virtual **Real computeDt** ()

  compute dt
- virtual **Real computeInitialDt** ()

  compute dt with initial data
- virtual void **CFL** (**Real** a_cfl)

  set CFL number
- virtual void **particleCFL** (**Real** a_particle_cfl)

set CFL number for computing timestep based on particles

- virtual void **refinementThreshold** (**Real** a_refine_threshold)

  set refinement threshold

- void **limitSolverCoarsening** (bool a_limitSolverCoarsening)

  set solver parameter

- bool **finestLevel** () const

  is this this finest level?

- bool **isEmpty** () const

  is this an empty level?

- void **computeVorticity** (**LevelData**< **FArrayBox** > &a_vorticity) const
- void **computeKineticEnergy** (**LevelData**< **FArrayBox** > &a_energy) const
- **Real particleKineticEnergy** () const
- void **dumpParticlesASCII** (std::ostream &os) const
- **Real Dx** () const

  access functions

- **Real Nu** () const

  viscous coefficient

- **ParticleAMRNS** ∗ **finerNSPtr** () const

  encapsulates the finer-level-pointer casting

- **ParticleAMRNS** ∗ **crseNSPtr** () const

  encapsulates the coarser-level pointer casting

- **LevelData**< **FArrayBox** > & **newVel** ()
- const **LevelData**< **FArrayBox** > & **newVel** () const
- **LevelData**< **FArrayBox** > & **oldVel** ()
- const **LevelData**< **FArrayBox** > & **oldVel** () const
- void **velocity** (**LevelData**< **FArrayBox** > &a_vel, **Real** a_time) const

  returns velocity at time t

- **LevelData**< **BinFab**< **DragParticle** > > & **particles** ()

  returns particles

- const **LevelData**< **BinFab**< **DragParticle** > > & **particles** () const

  returns particles (const version)

- int **numParticles** () const

  number of particles on this **AMR** *level*

- **LevelData**< **FArrayBox** > & **newLambda** ()
- const **LevelData**< **FArrayBox** > & **newLambda** () const
- **LevelData**< **FArrayBox** > & **oldLambda** ()
- const **LevelData**< **FArrayBox** > & **oldLambda** () const

- void **lambda** (**LevelData**< **FArrayBox** > &a_lambda, **Real** a_time) const

  returns lambda at time t

- **LevelData**< **FArrayBox** > & **newScal** (const int a_comp)
- const **LevelData**< **FArrayBox** > & **newScal** (const int a_comp) const
- **LevelData**< **FArrayBox** > & **oldScal** (const int a_comp)
- const **LevelData**< **FArrayBox** > & **oldScal** (const int a_comp) const
- void **scalars** (**LevelData**< **FArrayBox** > &a_scalars, const **Real** a_time, const int a_comp) const

  returns scalars at time t

- void **setPhysBC** (const **PhysBCUtil** &a_BC)

  sets physical BC object

- **PhysBCUtil** * **getPhysBCPtr** () const

  gets physical BC object

Protected Member Functions

- **DisjointBoxLayout** **loadBalance** (const **Vector**< **Box** > &a_grids)
- void **levelSetup** (const **DisjointBoxLayout** &level_domain)
- void **defineParticles** (**List**< **DragParticle** > &a_particles) const

  define particle list for entire domain

- void **readParameters** ()

  read parameters from parmParse database

- void **finestLevel** (bool a_finest_level)

  set whether this is the finest level or not

- void **swapOldAndNewStates** ()

  move new-time state into old-time state – also advances time by dt

- void **resetStates** (const **Real** a_time)

  exchange old and new states, resets time to a_time

- void **computeAdvectionVelocities** (**LevelData**< **FluxBox** > &a_adv_vel, **LevelData**< **FArrayBox** > &a_dragForce)

  compute advection velocities

- void **predictVelocities** (**LevelData**< **FArrayBox** > &a_uDelU, **LevelData**< **FluxBox** > &a_advVel, **LevelData**< **FArrayBox** > &a_dragForce)
- void **computeUStar** (**LevelData**< **FArrayBox** > &a_uStar, **LevelData**< **FArrayBox** > &a_dragForce)
- void **updateParticlePositions** (**LevelData**< **FArrayBox** > &a_newDrag, **LevelData**< **BinFab**< **DragParticle** > > &a_particles, const **LevelData**< **FArrayBox** > &a_fluidVel, bool a_completeUpdate)

  move particles and compute new-time drag forcing

- void **updateParticleForces** (**LevelData**< **BinFab**< **DragParticle** > > &a_particles)

  update particle forces

- void **interpVelToParticles** (**LevelData**< **BinFab**< **DragParticle** > > &a_particles, const **LevelData**< **FArrayBox** > &a_velocity)

  update particle velocities by interpolating a_vel to particles

- void **advectScalar** (**LevelData**< **FArrayBox** > &a_new_scal, **LevelData**< **FArrayBox** > &a_old_scal, **LevelData**< **FluxBox** > &a_adv_vel, **LevelFluxRegister** *a_crseFluxRegPtr, **LevelFluxRegister** &a_fineFluxReg, **Real** a_dt)

  do scalar advection – assumes all BC's already done

- void **advectDiffuseScalar** (**LevelData**< **FArrayBox** > &a_new_scal, **LevelData**< **FArrayBox** > &a_old_scal, **LevelData**< **FluxBox** > &a_adv_vel, const **Real** &a_diffusive_coeff, const **LevelData**< **FArrayBox** > *a_crse_scal_old, const **LevelData**< **FArrayBox** > *a_crse_scal_new, const **Real** a_old_crse_time, const **Real** a_new_crse_time, **LevelFluxRegister** *a_crseFluxRegPtr, **LevelFluxRegister** &a_fineFluxReg, **DomainGhostBC** &a_scalPhysBC, **Real** a_dt)

  advect/diffuse scalar – assumes all BC's already done?

- void **initializeGlobalPressure** ()

  manages pressure initialization after init or regrid

- void **initializeLevelPressure** (**Real** a_currentTime, **Real** a_dtInit)

  does level-based pressure initalization

- void **smoothVelocityField** (int lbase)

  smooths composite velocity field after regridding

- void **computeLapVel** (**LevelData**< **FArrayBox** > &a_lapVel, **LevelData**< **FArrayBox** > &a_vel, const **LevelData**< **FArrayBox** > *a_crseVelPtr)

  compute Laplacian(velocity)

- void **computeLapScal** (**LevelData**< **FArrayBox** > &a_lapScal, **LevelData**< **FArrayBox** > &a_scal, **DomainGhostBC** &a_physBC, const **LevelData**< **FArrayBox** > *a_crseScalPtr)

  compute Laplacian(scalars)

- void **fillVelocity** (**LevelData**< **FArrayBox** > &a_vel, **Real** a_time)

  fill grown velocity field using piecewise-constant interp

- void **fillVelocity** (**LevelData**< **FArrayBox** > &a_vel, **Real** a_time, int vel_comp, int dest_comp, int num_comp)

  fill grown velocity field using piecewise-constant interp

- void **fillLambda** (**LevelData**< **FArrayBox** > &a_lambda, **Real** a_time) const
- void **fillScalars** (**LevelData**< **FArrayBox** > &a_scal, **Real** a_time, const int a_comp) const

  fill interior and boundary cells (assume a_scal already sized)

- virtual int **numPlotComps** () const

number of components in plotfiles

- virtual void **getPlotData** (**LevelData**< **FArrayBox** > &a_plot_data) const

  stuff plotfile data

- void **defineRegridAMRSolver** (**AMRSolver** &a_solver, const **Vector**< **DisjointBox-Layout** > &a_grids, const **Vector**< **ProblemDomain** > &a_domains, const **Vector**< **Real** > &a_amrDx, const **Vector**< int > &a_refRatios, const int &a_lBase)

  defines (helmholtz) **AMRSolver** *used in smoothing during regridding*

Protected Attributes

- **LevelData**< **FArrayBox** > ∗ **m_vel_old_ptr**

  velocity vector at old time

- **LevelData**< **FArrayBox** > ∗ **m_vel_new_ptr**

  velocity vector at new time

- **LevelData**< **BinFab**< **DragParticle** > > **m_particles**

  particles

- **LevelData**< **FArrayBox** > ∗ **m_lambda_old_ptr**

  freestream preservation qty at old time

- **LevelData**< **FArrayBox** > ∗ **m_lambda_new_ptr**

  freestream preservation qty at new time

- **Vector**< **LevelData**< **FArrayBox** > ∗ > **m_scal_old**

  scalars at old time

- **Vector**< **LevelData**< **FArrayBox** > ∗ > **m_scal_new**

  scalars at new time

- **Vector**< **Tuple**< **Real**, **SpaceDim** > > **m_dataPoints**

  points where data is dumped out

- **Real m_dx**

  grid spacing

- **Real m_cfl**

  cfl number

- **Real m_particleCFL**

  cfl number for particle updates

- bool **m_limitSolverCoarsening**

  solver parameter – passed on through when defining solvers

- **Real m_dt_save**

dt from most recent timestep

- int **m_level_steps**

  number of steps on this level

- **CoarseAverage m_coarse_average**

  averaging from fine to coarse level for velocity

- **CoarseAverage m_coarse_average_lambda**

  averaging from fine to coarse level for freestream preservation qty

- **CCProjector m_projection**

  cell-centered projection object

- **ParticleProjector m_particle_projector**

  particle drag force projector object

- **LevelFluxRegister m_flux_register**

  momentum flux register

- **LevelFluxRegister m_lambda_flux_reg**

  freestream preservation qty flux register

- **Vector< LevelFluxRegister * > m_scal_fluxreg_ptrs**

  conserved scalar flux register

- **QuadCFInterp m_velCFInterp**

  does quadratic coarse-fine interpolation for velocities

- **VelocityPoissonOp m_velocityPoissonOp**

  PoissonOp *for velocities.*

- **PoissonOp m_scalarsPoissonOp**

  PoissonOp *for advected scalars (for diffusion).*

- **LevelHelmholtzSolver m_viscous_solver**

  levelsolver for viscous solves

- **Real m_refine_threshold**

  refinement threshold for regridding

- bool **m_finest_level**

  true if this is the finest extant level

- bool **m_is_empty**

  true if this level is an empty level (no grids)

- bool **m_regrid_smoothing_done**

  internal indicator whether post-regrid smoothing stuff has been done

- **PhysBCUtil** ∗ **m_physBCPtr**
  physical BC object

Static Protected Attributes

- const int **s_num_vel_comps** = **SpaceDim**
  number of components of m_state (vel comps + lambda)
- int **s_num_scal_comps**
  number of scalars
- const int **s_max_scal_comps** = **SpaceDim**
  maximum possible number of scalar components
- const char ∗ **s_vel_names** [**s_num_vel_comps**]
  names of components
- const char ∗ **s_scal_names** [**s_max_scal_comps**]
  names of scalars
- **Vector**< **Real** > **s_scal_coeffs**
  diffusion coefficients for scalar advection-diffusion
- **Vector**< **Real** > **s_domLength**
  size of physical domain
- bool **s_ppInit**
  have we initialized static variables from parmParse?
- **Real s_init_shrink**
  factor to shrink initial timestep
- **Real s_max_dt**
  maximum allowable timestep
- **Real s_prescribedDt**
- bool **s_project_initial_vel**
  should we project initial velocity field (default is yes)
- bool **s_initialize_pressures**
  should we initialize pressures after grid generation & regridding?
- bool **s_smooth_after_regrid**
  do antidiffusive smoothing after regridding?
- bool **s_dump_smoothing_plots**
  dump before-and-after plotfiles for post-regrid smoothing (for debug)
- **Real s_regrid_smoothing_coeff**

*antidiffusive smoothing coefficient (multiplies nu*dtCrse)*

- int **s_num_init_passes**

  number of times to iterate when initializing pressures

- bool **s_reflux_momentum**

  should we reflux momentum?

- bool **s_reflux_normal_momentum**

  if we're refluxing momentum, should we reflux the normal component?

- bool **s_set_bogus_values**

  debugging option – initially set arrays to s_bogus_value

- **Real s_bogus_value**

  debugging option – value to which arrays are initially set

- bool **s_tag_vorticity**

  tag on vorticity?

- **Real s_vort_factor**

  *factor for tagging – tag if vorticity greater than factor*max_vort*

- int **s_tags_grow**

  amount by which to grow tagged regions before calling meshrefine

- bool **s_implicit_reflux**

  do implicit refluxing for velocity?

- bool **s_applyFreestreamCorrection**

  apply freestream preservation correction?

- bool **s_reflux_scal**

  should we reflux scalars?

- bool **s_implicit_scal_reflux**

  do implicit refluxing for advected/diffused scalars?

- int **s_viscous_solver_type**

  viscous solver type: 0 = backwards euler, 1=Crank-Nicolson, 2=TGA

- **Real s_viscous_solver_tol**

  viscous solver tolerance (defaults to 1e-7)

- int **s_viscous_num_smooth_up**

  multigrid parameter for viscous solves

- int **s_viscous_num_smooth_down**

  multigrid parameter for viscous solves

- **Real s_nu**
  viscosity
- bool **s_do_particles**
  if true, do particle stuff, if false, don't
- bool **s_read_particles_from_file**
  read particle info from file?
- string **s_particle_file**
  file containing particle info
- **Real s_particle_drag_coeff**
  particle drag coefficient
- **RealVect s_particle_body_force**
  particle body force (gravity)
- **Real s_delta_epsilon**
  particle delta function parameter
- bool **s_use_image_particles**
  if true, use image particles in particle projector near phys boundaries
- int **s_order_vel_interp**
  order of velocity interpolation for particles
- bool **s_specifyInitialGrids**
  do we specify initial grids in a gridFile?
- string **s_initialGridFile**
  gridfile where initial grids are spec'd, if relevant
- bool **s_init_vel_from_vorticity**
  initialize velocity field from vorticity field
- **Real s_backgroundVel**
  background velocity (in the z-direction)
- bool **s_write_particles**
  – I/O options – include particles in plotfile?
- bool **s_write_divergence**
  include divergence in plotfile?
- bool **s_write_lambda**
  include lambda in plotfile?
- bool **s_write_time_derivatives**
  include du/dt and d(lambda)/dt in plotfile?

- bool **s_write_vorticity**
  include vorticity in plotfile?
- bool **s_write_scalars**
  include scalars in plotfile?
- bool **s_write_dScalar_dt**
  include d(scalars)/dt in plotfile?
- bool **s_write_strains**
  include strain-rates in plotfile? (2D only)
- bool **s_write_grad_eLambda**
  include freestream preservation correcion in plotfile?
- bool **s_write_error**
  include error (if there is an exact solution) in plotfile?
- bool **s_write_proc_ids**
  include processor distribution visualization in plotfile?
- bool **s_compute_scal_err**
  compute error and report norms after composite timesteps
- bool **s_write_initial_solve_rhs**
  dump out initial vorticity->stream function RHS's into plotfiles
- bool **s_write_grids**
  write out grids and processor assignments

7.98.1 Constructor & Destructor Documentation

7.98.1.1 ParticleAMRNS::ParticleAMRNS ()

7.98.1.2 ParticleAMRNS::ParticleAMRNS (AMRLevel * *a_coarser_level_ptr*, const Box & *a_prob_domain*, int *a_level*, int *a_ref_ratio*)

7.98.1.3 ParticleAMRNS::ParticleAMRNS (AMRLevel * *a_coarser_level_ptr*, const ProblemDomain & *a_prob_domain*, int *a_level*, int *a_ref_ratio*)

7.98.1.4 virtual ParticleAMRNS::~ParticleAMRNS () `[virtual]`

7.98.2 Member Function Documentation

7.98.2.1 virtual Real ParticleAMRNS::advance () `[virtual]`

advance by one timestep (returns what?)

Implements **AMRLevel** (p. 37).

7.98.2.2 void ParticleAMRNS::advectDiffuseScalar (LevelData< FArrayBox > & *a_new_scal*, LevelData< FArrayBox > & *a_old_scal*, LevelData< FluxBox > & *a_adv_vel*, const Real & *a_diffusive_coeff*, const LevelData< FArrayBox > * *a_crse_scal_old*, const LevelData< FArrayBox > * *a_crse_scal_new*, const Real *a_old_crse_time*, const Real *a_new_crse_time*, LevelFluxRegister * *a_crseFluxRegPtr*, LevelFluxRegister & *a_fineFluxReg*, DomainGhostBC & *a_scalPhysBC*, Real *a_dt*) [protected]

advect/diffuse scalar – assumes all BC's already done?

7.98.2.3 void ParticleAMRNS::advectScalar (LevelData< FArrayBox > & *a_new_scal*, LevelData< FArrayBox > & *a_old_scal*, LevelData< FluxBox > & *a_adv_vel*, LevelFluxRegister * *a_crseFluxRegPtr*, LevelFluxRegister & *a_fineFluxReg*, Real *a_dt*) [protected]

do scalar advection – assumes all BC's already done

7.98.2.4 virtual void ParticleAMRNS::CFL (Real *a_cfl*) [virtual]

set CFL number

7.98.2.5 void ParticleAMRNS::computeAdvectionVelocities (LevelData< FluxBox > & *a_adv_vel*, LevelData< FArrayBox > & *a_dragForce*) [protected]

compute advection velocities

7.98.2.6 virtual Real ParticleAMRNS::computeDt () [virtual]

compute dt

Implements **AMRLevel** (p. 37).

7.98.2.7 virtual Real ParticleAMRNS::computeInitialDt () [virtual]

compute dt with initial data

Implements **AMRLevel** (p. 37).

7.98.2.8 void ParticleAMRNS::computeKineticEnergy (LevelData< FArrayBox > & *a_energy*) const

7.98.2.9 void ParticleAMRNS::computeLapScal (LevelData< FArrayBox > & *a_lapScal*, LevelData< FArrayBox > & *a_scal*, DomainGhostBC & *a_physBC*, const LevelData< FArrayBox > * *a_crseScalPtr*) [protected]

compute Laplacian(scalars)

7.98.2.10 void ParticleAMRNS::computeLapVel (LevelData< FArrayBox > & *a_lapVel*, LevelData< FArrayBox > & *a_vel*, const LevelData< FArrayBox > * *a_crseVelPtr*) [protected]

compute Laplacian(velocity)

7.98.2.11 void ParticleAMRNS::computeUStar (LevelData< FArrayBox > & *a_uStar*, LevelData< FArrayBox > & *a_dragForce*) [protected]

7.98.2.12 void ParticleAMRNS::computeVorticity (LevelData< FArrayBox > & *a_vorticity*) const

7.98.2.13 ParticleAMRNS* ParticleAMRNS::crseNSPtr () const

encapsulates the coarser-level pointer casting

7.98.2.14 virtual void ParticleAMRNS::define (AMRLevel * *a_coarse_level_ptr*, const ProblemDomain & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) [virtual]

Defines this **AMRLevel**.

- a_coarser_level_ptr (not modified): pointer to next coarser level object.
- a_problem_domain (not modified): problem domain of this level.
- a_level (not modified): index of this level. The base level is zero.
- a_ref_ratio (not modified): the refinement ratio between this level and the next finer level.

Reimplemented from **AMRLevel** (p. 37).

7.98.2.15 virtual void ParticleAMRNS::define (AMRLevel * *a_coarse_level_ptr*, const Box & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) [virtual]

Defines this **AMRLevel**.

- a_coarser_level_ptr (not modified): pointer to next coarser level object.
- a_problem_domain (not modified): problem domain of this level.
- a_level (not modified): index of this level. The base level is zero.
- a_ref_ratio (not modified): the refinement ratio between this level and the next finer level.

Reimplemented from **AMRLevel** (p. 38).

7.98.2.16 void ParticleAMRNS::defineParticles (List< DragParticle > & *a_particles*) const [protected]

define particle list for entire domain

Defines all particles in domain; a_particles will then be distributed among the **AMR** levels

7.98.2.17 void ParticleAMRNS::defineRegridAMRSolver (AMRSolver & *a_solver*, const Vector< DisjointBoxLayout > & *a_grids*, const Vector< ProblemDomain > & *a_domains*, const Vector< Real > & *a_amrDx*, const Vector< int > & *a_refRatios*, const int & *a_lBase*) [protected]

defines (helmholtz) **AMRSolver** used in smoothing during regridding

7.98.2.18 void ParticleAMRNS::dumpParticlesASCII (std::ostream & *os*) const

7.98.2.19 Real ParticleAMRNS::Dx () const

access functions

7.98.2.20 void ParticleAMRNS::fillLambda (LevelData< FArrayBox > & *a_lambda*, Real *a_time*) const [protected]

resize temporary lambda for advection tracing computation and fill interior and boundary cells

7.98.2.21 void ParticleAMRNS::fillScalars (LevelData< FArrayBox > & *a_scal*, Real *a_time*, const int *a_comp*) const [protected]

fill interior and boundary cells (assume a_scal already sized)

7.98.2.22 void ParticleAMRNS::fillVelocity (LevelData< FArrayBox > & *a_vel*, Real *a_time*, int *vel_comp*, int *dest_comp*, int *num_comp*) [protected]

fill grown velocity field using piecewise-constant interp

7.98.2.23 void ParticleAMRNS::fillVelocity (LevelData< FArrayBox > & *a_vel*, Real *a_time*) [protected]

fill grown velocity field using piecewise-constant interp

7.98.2.24 ParticleAMRNS* ParticleAMRNS::finerNSPtr () const

encapsulates the finer-level-pointer casting

7.98.2.25 void ParticleAMRNS::finestLevel (bool *a_finest_level*) [protected]

set whether this is the finest level or not

7.98.2.26 bool ParticleAMRNS::finestLevel () const

is this this finest level?

7.98.2.27 PhysBCUtil* ParticleAMRNS::getPhysBCPtr () const

gets physical BC object

7.98.2.28 virtual void ParticleAMRNS::getPlotData (LevelData< FArrayBox > & *a_plot_data*) const [protected, virtual]

stuff plotfile data

7.98.2.29 virtual void ParticleAMRNS::initialData () [virtual]

initialize data

Implements **AMRLevel** (p. 38).

7.98.2.30 virtual void ParticleAMRNS::initialGrid (const Vector< Box > & *a_new_grids*) [virtual]

initialize grids

Implements **AMRLevel** (p. 39).

7.98.2.31 void ParticleAMRNS::initializeGlobalPressure () [protected]

manages pressure initialization after init or regrid

7.98.2.32 void ParticleAMRNS::initializeLevelPressure (Real *a_currentTime*, Real *a_dtInit*) [protected]

does level-based pressure initalization

7.98.2.33 void ParticleAMRNS::interpVelToParticles (LevelData< BinFab< DragParticle > > & *a_particles*, const LevelData< FArrayBox > & *a_velocity*) [protected]

update particle velocities by interpolating a_vel to particles

7.98.2.34 bool ParticleAMRNS::isEmpty () const

is this an empty level?

7.98.2.35 void ParticleAMRNS::lambda (LevelData< FArrayBox > & *a_lambda*, Real *a_time*) const

returns lambda at time t

7.98.2.36 void ParticleAMRNS::levelSetup (const DisjointBoxLayout & *level_domain*) [protected]

7.98.2.37 void ParticleAMRNS::limitSolverCoarsening (bool *a_limitSolverCoarsening*)

set solver parameter

7.98.2.38 DisjointBoxLayout ParticleAMRNS::loadBalance (const Vector< Box > & *a_grids*) [protected]

7.98.2.39 virtual AMRLevel* ParticleAMRNS::makeAMRLevel (AMRLevel * *a_coarser_level_ptr*, const ProblemDomain & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) const [virtual]

7.98.2.40 virtual AMRLevel* ParticleAMRNS::makeAMRLevel (AMRLevel * *a_coarser_level_ptr*, const Box & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) const [virtual]

7.98.2.41 const LevelData<FArrayBox>& ParticleAMRNS::newLambda () const

7.98.2.42 LevelData<FArrayBox>& ParticleAMRNS::newLambda ()

7.98.2.43 const LevelData<FArrayBox>& ParticleAMRNS::newScal (const int *a_comp*) const

7.98.2.44 LevelData<FArrayBox>& ParticleAMRNS::newScal (const int *a_comp*)

7.98.2.45 const LevelData<FArrayBox>& ParticleAMRNS::newVel () const

7.98.2.46 LevelData<FArrayBox>& ParticleAMRNS::newVel ()

7.98.2.47 Real ParticleAMRNS::Nu () const

viscous coefficient

7.98.2.48 int ParticleAMRNS::numParticles () const

number of particles on this **AMR** level

7.98.2.49 virtual int ParticleAMRNS::numPlotComps () const [protected, virtual]

number of components in plotfiles

7.98.2.50 const LevelData<FArrayBox>& ParticleAMRNS::oldLambda () const

7.98.2.51 LevelData<FArrayBox>& ParticleAMRNS::oldLambda ()

7.98.2.52 const LevelData<FArrayBox>& ParticleAMRNS::oldScal (const int *a_comp*) const

7.98.2.53 LevelData<FArrayBox>& ParticleAMRNS::oldScal (const int *a_comp*)

7.98.2.54 const LevelData<FArrayBox>& ParticleAMRNS::oldVel () const

7.98.2.55 LevelData<FArrayBox>& ParticleAMRNS::oldVel ()

7.98.2.56 virtual void ParticleAMRNS::particleCFL (Real *a_particle_cfl*) `[virtual]`

set CFL number for computing timestep based on particles

7.98.2.57 Real ParticleAMRNS::particleKineticEnergy () const

7.98.2.58 const LevelData<BinFab<DragParticle> >& ParticleAMRNS::particles () const

returns particles (const version)

7.98.2.59 LevelData<BinFab<DragParticle> >& ParticleAMRNS::particles ()

returns particles

7.98.2.60 virtual void ParticleAMRNS::postInitialize () `[virtual]`

things do do after initialization

Implements **AMRLevel** (p. 39).

7.98.2.61 virtual void ParticleAMRNS::postRegrid (int *a_lBase*) `[virtual]`

perform any post-regridding ops – lBase is the finest unchanged level

Reimplemented from **AMRLevel** (p. 39).

7.98.2.62 virtual void ParticleAMRNS::postTimeStep () `[virtual]`

things to do after a timestep

Implements **AMRLevel** (p. 39).

7.98.2.63 void ParticleAMRNS::predictVelocities (LevelData< FArrayBox > & *a_uDelU*, LevelData< FluxBox > & *a_advVel*, LevelData< FArrayBox > & *a_dragForce*) [protected]

7.98.2.64 void ParticleAMRNS::readParameters () [protected]

read parameters from parmParse database

7.98.2.65 virtual void ParticleAMRNS::refinementThreshold (Real *a_refine_threshold*) [virtual]

set refinement threshold

7.98.2.66 virtual void ParticleAMRNS::regrid (const Vector< Box > & *a_new_grids*) [virtual]

regrid

Implements **AMRLevel** (p. 39).

7.98.2.67 void ParticleAMRNS::resetStates (const Real *a_time*) [protected]

exchange old and new states, resets time to a_time

7.98.2.68 void ParticleAMRNS::scalars (LevelData< FArrayBox > & *a_scalars*, const Real *a_time*, const int *a_comp*) const

returns scalars at time t

7.98.2.69 void ParticleAMRNS::setPhysBC (const PhysBCUtil & *a_BC*)

sets physical BC object

7.98.2.70 void ParticleAMRNS::smoothVelocityField (int *lbase*) [protected]

smooths composite velocity field after regridding

7.98.2.71 void ParticleAMRNS::swapOldAndNewStates () [protected]

move new-time state into old-time state – also advances time by dt

7.98.2.72 virtual void ParticleAMRNS::tagCells (IntVectSet & *a_tags*) [virtual]

create tags

Implements **AMRLevel** (p. 40).

7.98.2.73 virtual void ParticleAMRNS::tagCellsInit (IntVectSet & *a_tags*) [virtual]

create tags at initialization

Implements **AMRLevel** (p. 40).

7.98.2.74 void ParticleAMRNS::updateParticleForces (LevelData< BinFab< DragParticle > > & *a_particles*) [protected]

update particle forces

7.98.2.75 void ParticleAMRNS::updateParticlePositions (LevelData< FArrayBox > & *a_newDrag*, LevelData< BinFab< DragParticle > > & *a_particles*, const LevelData< FArrayBox > & *a_fluidVel*, bool *a_completeUpdate*) [protected]

move particles and compute new-time drag forcing

This is a little complicated, in order to avoid creating a second particle list. We first move particles to a predicted position P^* (where P refers to the particle state). Then, we compute an estimate of the particle forcing term at the new time and place it in a_newDrag. Then, if completeUpdate is true, we go back and correct the new particle state to be formally second-order accurate.

7.98.2.76 void ParticleAMRNS::velocity (LevelData< FArrayBox > & *a_vel*, Real *a_time*) const

returns velocity at time t

7.98.3 Member Data Documentation

7.98.3.1 Real ParticleAMRNS::m_cfl [protected]

cfl number

7.98.3.2 CoarseAverage ParticleAMRNS::m_coarse_average [protected]

averaging from fine to coarse level for velocity

7.98.3.3 CoarseAverage ParticleAMRNS::m_coarse_average_lambda [protected]

averaging from fine to coarse level for freestream preservation qty

7.98.3.4 Vector<Tuple<Real, SpaceDim> > ParticleAMRNS::m_dataPoints [protected]

points where data is dumped out

7.98.3.5 Real ParticleAMRNS::m_dt_save [protected]

dt from most recent timestep

7.98.3.6 Real ParticleAMRNS::m_dx [protected]

grid spacing

7.98.3.7 bool ParticleAMRNS::m_finest_level [protected]

true if this is the finest extant level

7.98.3.8 LevelFluxRegister ParticleAMRNS::m_flux_register [protected]

momentum flux register

7.98.3.9 bool ParticleAMRNS::m_is_empty [protected]

true if this level is an empty level (no grids)

7.98.3.10 LevelFluxRegister ParticleAMRNS::m_lambda_flux_reg [protected]

freestream preservation qty flux register

7.98.3.11 LevelData<FArrayBox>* ParticleAMRNS::m_lambda_new_ptr [protected]

freestream preservation qty at new time

7.98.3.12 LevelData<FArrayBox>* ParticleAMRNS::m_lambda_old_ptr [protected]

freestream preservation qty at old time

7.98.3.13 int ParticleAMRNS::m_level_steps [protected]

number of steps on this level

7.98.3.14 bool ParticleAMRNS::m_limitSolverCoarsening [protected]

solver parameter – passed on through when defining solvers

7.98.3.15 ParticleProjector ParticleAMRNS::m_particle_projector [protected]

particle drag force projector object

7.98.3.16 Real ParticleAMRNS::m_particleCFL [protected]

cfl number for particle updates

7.98.3.17 LevelData<BinFab<DragParticle> > ParticleAMRNS::m_particles [protected]

particles

7.98.3.18 PhysBCUtil* ParticleAMRNS::m_physBCPtr [protected]

physical BC object

7.98.3.19 CCProjector ParticleAMRNS::m_projection [protected]

cell-centered projection object

7.98.3.20 Real ParticleAMRNS::m_refine_threshold [protected]

refinement threshold for regridding

7.98.3.21 bool ParticleAMRNS::m_regrid_smoothing_done [protected]

internal indicator whether post-regrid smoothing stuff has been done

7.98.3.22 Vector<LevelFluxRegister*> ParticleAMRNS::m_scal_fluxreg_ptrs [protected]

conserved scalar flux register

7.98.3.23 Vector<LevelData<FArrayBox>*> ParticleAMRNS::m_scal_new [protected]

scalars at new time

7.98.3.24 Vector<LevelData<FArrayBox>*> ParticleAMRNS::m_scal_old [protected]

scalars at old time

7.98.3.25 PoissonOp ParticleAMRNS::m_scalarsPoissonOp [protected]

PoissonOp for advected scalars (for diffusion).

7.98.3.26 LevelData<FArrayBox>* ParticleAMRNS::m_vel_new_ptr [protected]

velocity vector at new time

7.98.3.27 LevelData<FArrayBox>* ParticleAMRNS::m_vel_old_ptr [protected]

velocity vector at old time

7.98.3.28 QuadCFInterp ParticleAMRNS::m_velCFInterp [protected]

does quadratic coarse-fine interpolation for velocities

7.98.3.29 VelocityPoissonOp ParticleAMRNS::m_velocityPoissonOp [protected]

PoissonOp for velocities.

7.98.3.30 LevelHelmholtzSolver ParticleAMRNS::m_viscous_solver [protected]

levelsolver for viscous solves

7.98.3.31 bool ParticleAMRNS::s_applyFreestreamCorrection [static, protected]

apply freestream preservation correction?

7.98.3.32 Real ParticleAMRNS::s_backgroundVel [static, protected]

background velocity (in the z-direction)

7.98.3.33 Real ParticleAMRNS::s_bogus_value [static, protected]

debugging option – value to which arrays are initially set

7.98.3.34 bool ParticleAMRNS::s_compute_scal_err [static, protected]

compute error and report norms after composite timesteps

7.98.3.35 Real ParticleAMRNS::s_delta_epsilon [static, protected]

particle delta function parameter

7.98.3.36 bool ParticleAMRNS::s_do_particles [static, protected]

if true, do particle stuff, if false, don't

7.98.3.37 Vector<Real> ParticleAMRNS::s_domLength [static, protected]

size of physical domain

7.98.3.38 bool ParticleAMRNS::s_dump_smoothing_plots [static, protected]

dump before-and-after plotfiles for post-regrid smoothing (for debug)

7.98.3.39 bool ParticleAMRNS::s_implicit_reflux [static, protected]

do implicit refluxing for velocity?

7.98.3.40 bool ParticleAMRNS::s_implicit_scal_reflux [static, protected]

do implicit refluxing for advected/diffused scalars?

7.98.3.41 Real ParticleAMRNS::s_init_shrink [static, protected]

factor to shrink initial timestep

7.98.3.42 bool ParticleAMRNS::s_init_vel_from_vorticity [static, protected]

initialize velocity field from vorticity field

7.98.3.43 string ParticleAMRNS::s_initialGridFile [static, protected]

gridfile where initial grids are spec'd, if relevant

7.98.3.44 bool ParticleAMRNS::s_initialize_pressures [static, protected]

should we initialize pressures after grid generation & regridding?

7.98.3.45 Real ParticleAMRNS::s_max_dt [static, protected]

maximum allowable timestep

7.98.3.46 const int ParticleAMRNS::s_max_scal_comps = SpaceDim [static, protected]

maximum possible number of scalar components

7.98.3.47 Real ParticleAMRNS::s_nu [static, protected]

viscosity

7.98.3.48 int ParticleAMRNS::s_num_init_passes [static, protected]

number of times to iterate when initializing pressures

7.98.3.49 int ParticleAMRNS::s_num_scal_comps [static, protected]

number of scalars

7.98.3.50 const int ParticleAMRNS::s_num_vel_comps = SpaceDim [static, protected]

number of components of m_state (vel comps + lambda)

7.98.3.51 int ParticleAMRNS::s_order_vel_interp [static, protected]

order of velocity interpolation for particles

7.98.3.52 RealVect ParticleAMRNS::s_particle_body_force [static, protected]

particle body force (gravity)

7.98.3.53 Real ParticleAMRNS::s_particle_drag_coeff [static, protected]

particle drag coefficient

7.98.3.54 string ParticleAMRNS::s_particle_file [static, protected]

file containing particle info

7.98.3.55 bool ParticleAMRNS::s_ppInit [static, protected]

have we initialized static variables from parmParse?

7.98.3.56 Real ParticleAMRNS::s_prescribedDt [static, protected]

7.98.3.57 bool ParticleAMRNS::s_project_initial_vel [static, protected]

should we project initial velocity field (default is yes)

7.98.3.58 bool ParticleAMRNS::s_read_particles_from_file [static, protected]

read particle info from file?

7.98.3.59 bool ParticleAMRNS::s_reflux_momentum [static, protected]

should we reflux momentum?

7.98.3.60 bool ParticleAMRNS::s_reflux_normal_momentum [static, protected]

if we're refluxing momentum, should we reflux the normal component?

7.98.3.61 bool ParticleAMRNS::s_reflux_scal [static, protected]

should we reflux scalars?

7.98.3.62 Real ParticleAMRNS::s_regrid_smoothing_coeff [static, protected]

antidiffusive smoothing coefficient (multiplies nu*dtCrse)

7.98.3.63 Vector<Real> ParticleAMRNS::s_scal_coeffs [static, protected]

diffusion coefficients for scalar advection-diffusion

7.98.3.64 const char* ParticleAMRNS::s_scal_names[s_max_scal_comps] [static, protected]

names of scalars

7.98.3.65 bool ParticleAMRNS::s_set_bogus_values [static, protected]

debugging option – initially set arrays to s_bogus_value

7.98.3.66 bool ParticleAMRNS::s_smooth_after_regrid [static, protected]

do antidiffusive smoothing after regridding?

7.98.3.67 bool ParticleAMRNS::s_specifyInitialGrids [static, protected]

do we specify initial grids in a gridFile?

7.98.3.68 bool ParticleAMRNS::s_tag_vorticity [static, protected]

tag on vorticity?

7.98.3.69 int ParticleAMRNS::s_tags_grow [static, protected]

amount by which to grow tagged regions before calling meshrefine

7.98.3.70 bool ParticleAMRNS::s_use_image_particles [static, protected]

if true, use image particles in particle projector near phys boundaries

7.98.3.71 const char* ParticleAMRNS::s_vel_names[s_num_vel_comps] [static, protected]

names of components

7.98.3.72 int ParticleAMRNS::s_viscous_num_smooth_down [static, protected]

multigrid parameter for viscous solves

7.98.3.73 int ParticleAMRNS::s_viscous_num_smooth_up [static, protected]

multigrid parameter for viscous solves

7.98.3.74 Real ParticleAMRNS::s_viscous_solver_tol [static, protected]

viscous solver tolerance (defaults to 1e-7)

7.98.3.75 int ParticleAMRNS::s_viscous_solver_type [static, protected]

viscous solver type: 0 = backwards euler, 1=Crank-Nicolson, 2=TGA

7.98.3.76 Real ParticleAMRNS::s_vort_factor [static, protected]

factor for tagging – tag if vorticity greater than factor*max_vort

7.98.3.77 bool ParticleAMRNS::s_write_divergence [static, protected]

include divergence in plotfile?

7.98.3.78 bool ParticleAMRNS::s_write_dScalar_dt [static, protected]

include d(scalars)/dt in plotfile?

7.98.3.79 bool ParticleAMRNS::s_write_error [static, protected]

include error (if there is an exact solution) in plotfile?

7.98.3.80 bool ParticleAMRNS::s_write_grad_eLambda [static, protected]

include freestream preservation correcion in plotfile?

7.98.3.81 bool ParticleAMRNS::s_write_grids [static, protected]

write out grids and processor assignments

7.98.3.82 bool ParticleAMRNS::s_write_initial_solve_rhs [static, protected]

dump out initial vorticity->stream function RHS's into plotfiles

7.98.3.83 bool ParticleAMRNS::s_write_lambda [static, protected]

include lambda in plotfile?

7.98.3.84 bool ParticleAMRNS::s_write_particles [static, protected]

– I/O options – include particles in plotfile?

7.98.3.85 bool ParticleAMRNS::s_write_proc_ids [static, protected]

include processor distribution visualization in plotfile?

7.98.3.86 bool ParticleAMRNS::s_write_scalars [static, protected]

include scalars in plotfile?

7.98.3.87 bool ParticleAMRNS::s_write_strains [static, protected]

include strain-rates in plotfile? (2D only)

7.98.3.88 bool ParticleAMRNS::s_write_time_derivatives [static, protected]

include du/dt and d(lambda)/dt in plotfile?

7.98.3.89 bool ParticleAMRNS::s_write_vorticity [static, protected]

include vorticity in plotfile?

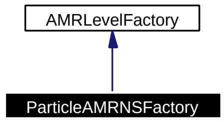
The documentation for this class was generated from the following file:

- **ParticleAMRNS.H**

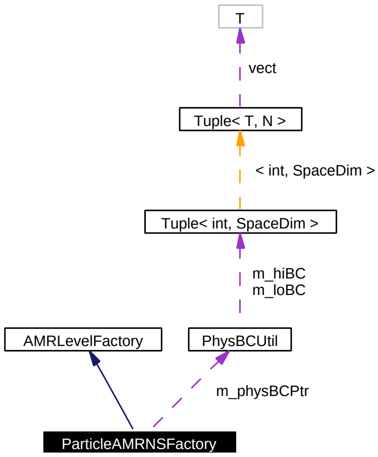
7.99 ParticleAMRNSFactory Class Reference

`#include <ParticleAMRNSFactory.H>`

Inheritance diagram for ParticleAMRNSFactory:



Collaboration diagram for ParticleAMRNSFactory:



Public Member Functions

- **ParticleAMRNSFactory** ()
- virtual **AMRLevel** ∗ **new_amrlevel** () const
- virtual ∼**ParticleAMRNSFactory** ()
- void **CFL** (const **Real** a_cfl)
- void **particleCFL** (const **Real** a_particle_cfl)

  set cfl condition based on particles

- void **refinementThreshold** (const **Real** a_refine_threshold)
- void **setPhysBC** (const **PhysBCUtil** &a_physBC)
- void **setLimitSolverCoarsening** (bool a_limitSolverCoarsening)

Protected Member Functions

- void **setDefaultValues** ()

Protected Attributes

- **Real m_cfl**
- **Real m_particleCFL**

  cfl number for computing timestep based on particles

- **Real m_refine_thresh**
- **PhysBCUtil ∗ m_physBCPtr**
- bool **m_limitSolverCoarsening**

7.99.1 Constructor & Destructor Documentation

7.99.1.1 ParticleAMRNSFactory::ParticleAMRNSFactory ()

7.99.1.2 virtual ParticleAMRNSFactory::~ParticleAMRNSFactory () `[virtual]`

7.99.2 Member Function Documentation

7.99.2.1 void ParticleAMRNSFactory::CFL (const Real *a_cfl*)

7.99.2.2 virtual AMRLevel∗ ParticleAMRNSFactory::new_amrlevel () const `[virtual]`

Factory function. Creates a new **AMRLevel**. In your instantiation you should include defining all the stuff that is not part of the **AMRLevel** API.

Implements **AMRLevelFactory** (p. 42).

7.99.2.3 void ParticleAMRNSFactory::particleCFL (const Real *a_particle_cfl*)

set cfl condition based on particles

7.99.2.4 void ParticleAMRNSFactory::refinementThreshold (const Real *a_refine_threshold*)

7.99.2.5 void ParticleAMRNSFactory::setDefaultValues () `[protected]`

7.99.2.6 void ParticleAMRNSFactory::setLimitSolverCoarsening (bool *a_limitSolverCoarsening*)

7.99.2.7 void ParticleAMRNSFactory::setPhysBC (const PhysBCUtil & *a_physBC*)

7.99.3 Member Data Documentation

7.99.3.1 Real ParticleAMRNSFactory::m_cfl `[protected]`

7.99.3.2 bool ParticleAMRNSFactory::m_limitSolverCoarsening `[protected]`

7.99.3.3 Real ParticleAMRNSFactory::m_particleCFL `[protected]`

cfl number for computing timestep based on particles

7.99.3.4 PhysBCUtil* ParticleAMRNSFactory::m_physBCPtr [protected]

7.99.3.5 Real ParticleAMRNSFactory::m_refine_thresh [protected]

The documentation for this class was generated from the following file:

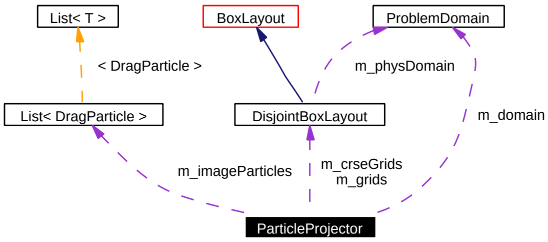
- **ParticleAMRNSFactory.H**

7.100 ParticleProjector Class Reference

class to manage projection of particle drag force onto the fluid

`#include <ParticleProjector.H>`

Collaboration diagram for ParticleProjector:



Public Member Functions

- **ParticleProjector** ()

    default constructor

- **ParticleProjector** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** &a_crseGrids, const **ProblemDomain** &a_domain, int a_nRefCrse, **Real** a_dx)

    defining constructor – calls matching define function

- **~ParticleProjector** ()

    destructor

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** &a_crseGrids, const **ProblemDomain** &a_domain, int a_nRefCrse, **Real** a_dx)

    define function

- void **projectForce** (**LevelData**< **FArrayBox** > &a_force, const **LevelData**< **BinFab**< **DragParticle** > > &a_particles)

    projects force due to particles

- void **setSpreadingRadius** (const **Real** a_rad)

    set spreading radius for MLC part of algorithm

- void **setCorrectionRadius** (const **Real** a_rad)

    set correction radius for MLC part of algorithm

- **Real getSpreadingRadius** () const

    get spreading radius for MLC part of algorithm

- **Real getCorrectionRadius** () const

    get correction radius for MLC part of algorithm

- void **doImages** (bool **m_doImages**) *set whether to do image particles at physical boundaries*

- bool **doImages** () const

  do images at physical boundaries?

- bool **isDefined** () const
- void **setVerbosity** (int a_verbosity)

  set verbosity level

- int **verbosity** () const

  get verbosity level

Protected Member Functions

- void **computeD** (**LevelData**< **FArrayBox** > &a_D, const **LevelData**< **BinFab**< **DragParticle** > > &a_particles)
- void **solveForProjForce** (**LevelData**< **FArrayBox** > &a_projectedForce, const **LevelData**< **FArrayBox** > &a_D, const **LevelData**< **BinFab**< **DragParticle** > > &a_particles)
- void **addImageEffects** (**FArrayBox** &a_rhs, int a_buffer, int a_dir) const

  add effects of image particles near boundary to D in the dir direction

- void **defineImages** (const **LevelData**< **BinFab**< **DragParticle** > > &a_particles)

  compute image particles near boundary, if necessary

- **DragParticle** * **createImageParticle** (const **DragParticle** &a_particle, **Real** bndryLoc, int dir, **Side::LoHiSide** a_side)

  helper function

- void **doInfiniteDomainSolve** (**FArrayBox** &phi, const **FArrayBox** &rhs) const
- void **doNeumannSolve** (**FArrayBox** &a_phi, const **FArrayBox** &a_rhs) const

  quick test function

- void **setDefaultValues** ()

  set default values

- int **infiniteDomainPadding** () const

  amount to pad domain for infinite domain solve

Protected Attributes

- **DisjointBoxLayout m_grids**

  grids on this level

- **DisjointBoxLayout m_crseGrids**

  coarse grids (undefined if no coarser level

- **ProblemDomain m_domain**

- int **m_nRefCrse**
- **Real m_dx**

  cell spacing

- **Real m_spreadingRadius**

  spreading radius for MLC

- **Real m_correctionRadius**

  correction radius for MLC

- int **m_s**

  parameter for infinite domain solve

- int **m_s1**

  parameter for infinite domain solve

- int **m_patchSize**

  parameter for infinite domain solve

- **List< DragParticle > m_imageParticles**

  list of image particles used to approximate solid walls

- bool **m_doImages**

  should we do the image particle computation near physical boundaries?

- bool **m_isDefined**
- int **m_verbosity**

  verbosity level

7.100.1 Detailed Description

class to manage projection of particle drag force onto the fluid

The ParticleProjector class encapsulates the functionality to take the individual drag forces due to a collection of particles and apply them to the mesh in an approximation to P(f), which may be then used as a source term for a Navier-Stokes advance step.

7.100.2 Constructor & Destructor Documentation

7.100.2.1 ParticleProjector::ParticleProjector ()

default constructor

7.100.2.2 ParticleProjector::ParticleProjector (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout & *a_crseGrids*, const ProblemDomain & *a_domain*, int *a_nRefCrse*, Real *a_dx*)

defining constructor – calls matching define function

If no coarser level, a_crseGrids is an undefined DBL.

7.100.2.3 ParticleProjector::~ParticleProjector ()

destructor

7.100.3 Member Function Documentation

7.100.3.1 void ParticleProjector::addImageEffects (FArrayBox & *a_rhs*, int *a_buffer*, int *a_dir*) const [protected]

add effects of image particles near boundary to D in the dir direction

a_buffer is the width of cells near the boundary where the effects are to be computed.

7.100.3.2 void ParticleProjector::computeD (LevelData< FArrayBox > & *a_D*, const LevelData< BinFab< DragParticle > > & *a_particles*) [protected]

7.100.3.3 DragParticle* ParticleProjector::createImageParticle (const DragParticle & *a_particle*, Real *bndryLoc*, int *dir*, Side::LoHiSide *a_side*) [protected]

helper function

7.100.3.4 void ParticleProjector::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout & *a_crseGrids*, const ProblemDomain & *a_domain*, int *a_nRefCrse*, Real *a_dx*)

define function

7.100.3.5 void ParticleProjector::defineImages (const LevelData< BinFab< DragParticle > > & *a_particles*) [protected]

compute image particles near boundary, if necessary

Note that this function is not const because the master list of image particles is incremented

7.100.3.6 bool ParticleProjector::doImages () const

do images at physical boundaries?

7.100.3.7 void ParticleProjector::doImages (bool *m_doImages*)

set whether to do image particles at physical boundaries

7.100.3.8 void ParticleProjector::doInfiniteDomainSolve (FArrayBox & *phi*, const FArrayBox & *rhs*) const [protected]

7.100.3.9 void ParticleProjector::doNeumannSolve (FArrayBox & *a_phi*, const FArrayBox & *a_rhs*) const [protected]

quick test function

7.100.3.10 Real ParticleProjector::getCorrectionRadius () const

get correction radius for MLC part of algorithm

7.100.3.11 Real ParticleProjector::getSpreadingRadius () const

get spreading radius for MLC part of algorithm

7.100.3.12 int ParticleProjector::infiniteDomainPadding () const [protected]

amount to pad domain for infinite domain solve

7.100.3.13 bool ParticleProjector::isDefined () const

7.100.3.14 void ParticleProjector::projectForce (LevelData< FArrayBox > & *a_force*, const LevelData< BinFab< DragParticle > > & *a_particles*)

projects force due to particles

given the collection of DragParticles, returns the projection of the force at cell centers in a_force, suitable for use as a source term in a fluid advance. Assumes that all particles have already computed their forces.

7.100.3.15 void ParticleProjector::setCorrectionRadius (const Real *a_rad*)

set correction radius for MLC part of algorithm

7.100.3.16 void ParticleProjector::setDefaultValues () [protected]

set default values

7.100.3.17 void ParticleProjector::setSpreadingRadius (const Real *a_rad*)

set spreading radius for MLC part of algorithm

7.100.3.18 void ParticleProjector::setVerbosity (int *a_verbosity*)

set verbosity level

7.100.3.19 void ParticleProjector::solveForProjForce (LevelData< FArrayBox > & *a_projectedForce*, const LevelData< FArrayBox > & *a_D*, const LevelData< BinFab< DragParticle > > & *a_particles*) [protected]

7.100.3.20 int ParticleProjector::verbosity () const

get verbosity level

7.100.4 Member Data Documentation

7.100.4.1 Real ParticleProjector::m_correctionRadius [protected]

correction radius for MLC

7.100.4.2 DisjointBoxLayout ParticleProjector::m_crseGrids [protected]

coarse grids (undefined if no coarser level

7.100.4.3 bool ParticleProjector::m_doImages [protected]

should we do the image particle computation near physical boundaries?

7.100.4.4 ProblemDomain ParticleProjector::m_domain [protected]

7.100.4.5 Real ParticleProjector::m_dx [protected]

cell spacing

7.100.4.6 DisjointBoxLayout ParticleProjector::m_grids [protected]

grids on this level

7.100.4.7 List<DragParticle> ParticleProjector::m_imageParticles [protected]

list of image particles used to approximate solid walls

7.100.4.8 bool ParticleProjector::m_isDefined [protected]

7.100.4.9 int ParticleProjector::m_nRefCrse [protected]

7.100.4.10 int ParticleProjector::m_patchSize [protected]

parameter for infinite domain solve

7.100.4.11 int ParticleProjector::m_s [protected]

parameter for infinite domain solve

s is the total amount domain will be grown in infinite domain solve

7.100.4.12 int ParticleProjector::m_s1 [protected]

parameter for infinite domain solve

s1 is the amount that the domain is grown for the first solve in the infinite domain algorithm. This is also the amount by which the RHS may be grown.

7.100.4.13 Real ParticleProjector::m_spreadingRadius [protected]

spreading radius for MLC

7.100.4.14 int ParticleProjector::m_verbosity [protected]

verbosity level

The documentation for this class was generated from the following file:

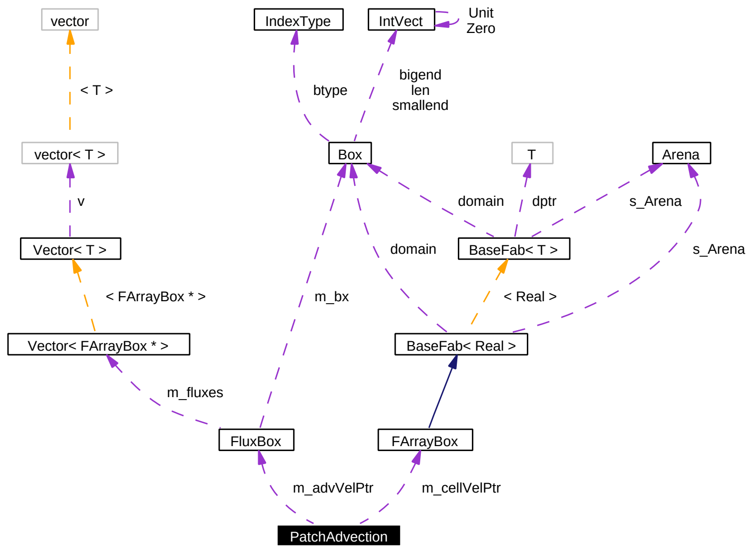
- **ParticleProjector.H**

7.101 PatchAdvection Class Reference

An PatchGodunov-derived class for simple advection-diffusion problems.

`#include <PatchAdvection.H>`

Collaboration diagram for PatchAdvection:



Public Member Functions

- **PatchAdvection** ()
- virtual PatchGodunov ∗ **new_patchGodunov** () const

    Factory method - this object is its own factory.

- virtual **Real getMaxWaveSpeed** (const **FArrayBox** &a_U, const **Box** &a_box)

    Compute the maximum wave speed.

- virtual int **numConserved** ()

    Number of conserved variables.

- virtual **Vector**< string > **stateNames** ()

    Names of the conserved variables.

- virtual int **numFluxes** ()

    Number of flux variables.

- void **setAdvVelPtr** (**FluxBox** ∗a_advVelPtr)

    set face-centered advection velocity for this patch

- const **FluxBox** ∗ **advectionVelPtr** () const

    access face-centered advection velocities

- void **setCellVelPtr** (**FArrayBox** *a_cellVelPtr)
  set cell-centered velocity field for this patch
- const **FArrayBox** * **cellVelPtr** () const
  access cell-centered velicity field pointer
- virtual void **setNumVar** (int a_numVar)
  set the number of conserved (and primitive, and flux) variables

Protected Member Functions

- virtual int **numPrimitives** ()
  Number of primitive variables.
- virtual int **numSlopes** ()
  Number of primitive variables for which slopes are computed.
- virtual void **consToPrim** (**FArrayBox** &a_W, const **FArrayBox** &a_U, const **Box** &a_box)
  Compute the primitive variables from the conserved variables within a_box.
- virtual void **primToCons** (**FArrayBox** &a_U, const **FArrayBox** &a_W, const **Box** &a_box)
  Compute the conserved variables from the primitive variables within a_box.
- virtual void **normalPred** (**FArrayBox** &a_WMinus, **FArrayBox** &a_WPlus, const **FArrayBox** &a_W, const **FArrayBox** &a_dW, const **Real** &a_scale, const int &a_dir, const **Box** &a_box)
  Extrapolate the primitive variables to the cell faces.
- virtual void **incrementWithSource** (**FArrayBox** &a_W, const **FArrayBox** &a_S, const **Real** &a_scale, const **Box** &a_box)
  increment the primitive variables by a source term
- virtual void **riemann** (**FArrayBox** &a_F, const **FArrayBox** &a_WLeft, const **FArrayBox** &a_WRight, const int &a_dir, const **Box** &a_box)
  Compute a Riemann problem and generate fluxes at the faces.
- virtual void **updatePrim** (**FArrayBox** &a_WMinus, **FArrayBox** &a_WPlus, const **FArrayBox** &a_F, const **Real** &a_scale, const int &a_dir, const **Box** &a_box)
  Update the primitive variables using fluxes (of the conserved variables).
- virtual void **updateCons** (**FArrayBox** &a_U, const **FArrayBox** &a_F, const **Real** &a_scale, const int &a_dir, const **Box** &a_box)
  Update the conserved variable using fluxes and a scaling factor.
- virtual void **finalUpdate** (**FArrayBox** &a_U, const **FArrayBox** &a_F, const **Real** &a_scale, const int &a_dir, const **Box** &a_box)
  Perform final update of conserved variable using fluxes.

- virtual **Interval velocityInterval** ()

  Interval *within the primitive variables corresponding to the velocities.*

- virtual int **pressureIndex** ()

  Component index within the primitive variables of the pressure.

- virtual int **bulkModulusIndex** ()

  Component index within the primitive variables of the bulk modulus.

Protected Attributes

- **FluxBox * m_advVelPtr**

  face-centered advection velocity, set through the advVel() intervace

- **FArrayBox * m_cellVelPtr**

  cell-centered velocity field (centered at old time)

- int **m_numCons**

  number of conserved variables

- int **m_numPrim**

  number of primitive variables

- int **m_numFlux**

  number of fluxes

- int **m_numSlope**

  number of variabls in slope computation

7.101.1 Detailed Description

An PatchGodunov-derived class for simple advection-diffusion problems.

7.101.2 Constructor & Destructor Documentation

7.101.2.1 PatchAdvection::PatchAdvection ()

7.101.3 Member Function Documentation

7.101.3.1 const FluxBox* PatchAdvection::advectionVelPtr () const

access face-centered advection velocities

7.101.3.2 virtual int PatchAdvection::bulkModulusIndex () [protected, virtual]

Component index within the primitive variables of the bulk modulus.

Return the component index withn the primitive variables for the bulk modulus. Used for slope flattening (slope computation) used as a normalization to measure shock strength.

7.101.3.3 const FArrayBox* PatchAdvection::cellVelPtr () const

access cell-centered velicity field pointer

7.101.3.4 virtual void PatchAdvection::consToPrim (FArrayBox & *a_W*, const FArrayBox & *a_U*, const Box & *a_box*) [protected, virtual]

Compute the primitive variables from the conserved variables within a_box.

7.101.3.5 virtual void PatchAdvection::finalUpdate (FArrayBox & *a_U*, const FArrayBox & *a_F*, const Real & *a_scale*, const int & *a_dir*, const Box & *a_box*) [protected, virtual]

Perform final update of conserved variable using fluxes.

Given the fluxes, a_F, in a direction, a_dir, and a scaling factor, a_scale, update the conserved variables, a_U, within a_box:

a_U = a_U - a_scale * (a_F(a_dir,HiSide) - a_F(a_dir,LoSide))

7.101.3.6 virtual Real PatchAdvection::getMaxWaveSpeed (const FArrayBox & *a_U*, const Box & *a_box*) [virtual]

Compute the maximum wave speed.

7.101.3.7 virtual void PatchAdvection::incrementWithSource (FArrayBox & *a_W*, const FArrayBox & *a_S*, const Real & *a_scale*, const Box & *a_box*) [protected, virtual]

increment the primitive variables by a source term

7.101.3.8 virtual PatchGodunov* PatchAdvection::new_patchGodunov () const [virtual]

Factory method - this object is its own factory.

Return a pointer to new PatchGodunov object with its initial and boundary condtions, slope parameters, and artificial viscosity information defined.

7.101.3.9 virtual void PatchAdvection::normalPred (FArrayBox & *a_WMinus*, FArrayBox & *a_WPlus*, const FArrayBox & *a_W*, const FArrayBox & *a_dW*, const Real & *a_scale*, const int & *a_dir*, const Box & *a_box*) [protected, virtual]

Extrapolate the primitive variables to the cell faces.

Extrapolate the primitive variables, a_W, in the minus and plus direction, a_dir, using the slopes a_dW within a_box. See document for details.

7.101.3.10 virtual int PatchAdvection::numConserved () [virtual]

Number of conserved variables.

Return the number of conserved variables.

7.101.3.11 virtual int PatchAdvection::numFluxes () [virtual]

Number of flux variables.

Return the number of flux variables. This can be greater than the number of conserved variables if addition fluxes/face-centered quantities are computed.

7.101.3.12 virtual int PatchAdvection::numPrimitives () [protected, virtual]

Number of primitive variables.

Return the number of primitive variables. This may be greater than the number of conserved variables if derived/redundant quantities are also stored for convenience.

7.101.3.13 virtual int PatchAdvection::numSlopes () [protected, virtual]

Number of primitive variables for which slopes are computed.

Return the number of primitive variables for which slopes are computed. Only slopes corresponding to primitive variables in the interval 0 to **numSlopes()** - 1 are computed and only primitive variables in that interval are updated using the slopes.

7.101.3.14 virtual int PatchAdvection::pressureIndex () [protected, virtual]

Component index within the primitive variables of the pressure.

Return the component index withn the primitive variables for the pressure. Used for slope flattening (slope computation).

7.101.3.15 virtual void PatchAdvection::primToCons (FArrayBox & *a_U*, const FArrayBox & *a_W*, const Box & *a_box*) [protected, virtual]

Compute the conserved variables from the primitive variables within a_box.

7.101.3.16 virtual void PatchAdvection::riemann (FArrayBox & *a_F*, const FArrayBox & *a_WLeft*, const FArrayBox & *a_WRight*, const int & *a_dir*, const Box & *a_box*) [protected, virtual]

Compute a Riemann problem and generate fluxes at the faces.

Given input left and right states in a direction, a_dir, compute a Riemann problem and generate fluxes at the faces within a_box.

7.101.3.17 void PatchAdvection::setAdvVelPtr (FluxBox * *a_advVelPtr*)

set face-centered advection velocity for this patch

7.101.3.18 void PatchAdvection::setCellVelPtr (FArrayBox * *a_cellVelPtr*)

set cell-centered velocity field for this patch

7.101.3.19 virtual void PatchAdvection::setNumVar (int *a_numVar*) [virtual]

set the number of conserved (and primitive, and flux) variables

Note that for this simple tracing, conservative == primitive

7.101.3.20 virtual Vector<string> PatchAdvection::stateNames () [virtual]

Names of the conserved variables.

Return the names of the conserved variables.

7.101.3.21 virtual void PatchAdvection::updateCons (FArrayBox & *a_U*, const FArrayBox & *a_F*, const Real & *a_scale*, const int & *a_dir*, const Box & *a_box*) [protected, virtual]

Update the conserved variable using fluxes and a scaling factor.

Given the fluxes, a_F, in a direction, a_dir, and a scaling factor, a_scale, update the conserved variables, a_U, within a_box:

a_U = a_U - a_scale * (a_F(a_dir,HiSide) - a_F(a_dir,LoSide))

7.101.3.22 virtual void PatchAdvection::updatePrim (FArrayBox & *a_WMinus*, FArrayBox & *a_WPlus*, const FArrayBox & *a_F*, const Real & *a_scale*, const int & *a_dir*, const Box & *a_box*) [protected, virtual]

Update the primitive variables using fluxes (of the conserved variables).

Given the fluxes, a_F, in a direction, a_dir, and a scaling factor, a_scale, update the primitive variables, a_WMinus and a_WPlus, within a_box. Note: The fluxes are fluxes of conserved variables.

7.101.3.23 virtual Interval PatchAdvection::velocityInterval () [protected, virtual]

Interval within the primitive variables corresponding to the velocities.

Return the interval of component indices within the primitive variable of the velocities. Used for slope flattening (slope computation) and computing the divergence of the velocity (artificial viscosity).

7.101.4 Member Data Documentation

7.101.4.1 FluxBox* PatchAdvection::m_advVelPtr [protected]

face-centered advection velocity, set through the advVel() intervace

7.101.4.2 FArrayBox* PatchAdvection::m_cellVelPtr [protected]

cell-centered velocity field (centered at old time)

7.101.4.3 int PatchAdvection::m_numCons [protected]

number of conserved variables

For this simple adection tracing, we assume that conservative and primitive are the same

7.101.4.4 int PatchAdvection::m_numFlux [protected]

number of fluxes

7.101.4.5 int PatchAdvection::m_numPrim [protected]

number of primitive variables

7.101.4.6 int PatchAdvection::m_numSlope [protected]

number of variabls in slope computation

The documentation for this class was generated from the following file:

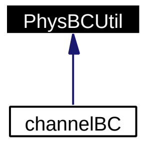
- **PatchAdvection.H**

7.102 PhysBCUtil Class Reference

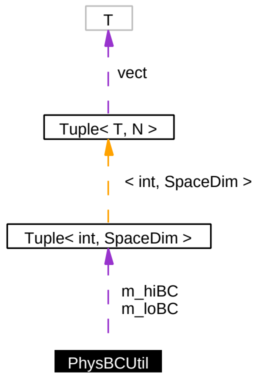
Encapsulates all physical BC's for IAMR solves.

`#include <PhysBCUtil.H>`

Inheritance diagram for PhysBCUtil:



Collaboration diagram for PhysBCUtil:



Public Types

- enum **BCType** {
  bogusBC = -1, **SolidWall**, **Inflow**, **Outflow**,
  Symmetry, **noShear**, **Custom**, **NUM_PHYS_BC_TYPES** }

Public Member Functions

- **PhysBCUtil** ()
- virtual ~**PhysBCUtil** ()
- virtual **PhysBCUtil** ∗ **newPhysBCUtil** () const

  "virtual" constructor

- virtual **DomainGhostBC** ∗ **MacPressureBC** () const

  Pressure BCs returns pressure BC for MAC projection.

- virtual **DomainGhostBC** ∗ **gradMacPressureBC** () const

  returns BC applied to MAC pressure before computing gradient

- virtual **DomainGhostBC** ∗ **LevelPressureBC** () const
  returns pressure BC for level projection
- virtual **DomainGhostBC** ∗ **gradPiBC** () const
  returns BC applied to Pi before computing gradient
- virtual **DomainGhostBC** ∗ **SyncProjBC** () const
  returns pressure BC for sync projection
- virtual **DomainGhostBC** ∗ **gradESyncBC** () const
  returns BC applied to sync correction before computing gradient
- virtual **DomainGhostBC** ∗ **FreestreamCorrBC** () const
  returns BC for freestream preservation correction
- virtual **DomainGhostBC** ∗ **gradELambdaBC** () const
  BC for grad eLambda.
- virtual **DomainGhostBC** ∗ **InitialProjBC** () const
  returns pressure BC for initial velocity projection
- virtual **DomainGhostBC** ∗ **gradInitialProjCorrBC** () const
  BC for gradient of correction for initial projection.
- virtual **basicVelBC** ∗ **viscousVelBC** () const
  returns velocity BC for viscous source
- virtual **basicVelBC** ∗ **tracingVelBC** () const
  returns velocity BC for tracing (inviscid)
- virtual **basicEdgeVelBC** ∗ **advectionVelBC** () const
  returns edge-centered vel BC for edge-ctrd vel (single component)
- virtual **basicEdgeVelBC** ∗ **edgeVelBC** () const
  returns edge-centered velocity BC for edge-centered vel (multicomponent)
- virtual **basicVelBC** ∗ **uDelUBC** (bool a_isViscous) const
  returns BC on cell-centered u-dot-del-u term
- virtual **DomainGhostBC** ∗ **viscousSolveBC** (int a_dir) const
  returns single-component BC for viscous solves
- virtual **basicVelBC** ∗ **uStarBC** (bool a_isViscous=true) const
  returns pre-projection velocity BC
- virtual **DomainGhostBC** ∗ **viscousRefluxBC** (int a_dir) const
  returns single-component BC for viscous refluxing solves
- virtual **DomainGhostBC** ∗ **diffusiveBC** (int a_scalarType) const
  Scalar BC's.

- virtual **DomainGhostBC** ∗ **scalarTraceBC** (int a_scalarType) const
- virtual **DomainGhostBC** ∗ **scalarDiffusionSolveBC** (int a_scalarType) const
- virtual **DomainGhostBC** ∗ **scalarRefluxSolveBC** (int a_scalarType) const
- virtual **DomainGhostBC** ∗ **lambdaBC** () const

  volume discrepancy BC's

- virtual **basicVelBC** ∗ **viscousSrcBC** () const

  Miscellaneous BC's BC on viscous source term itself.

- virtual **basicVelBC** ∗ **diffusiveSrcBC** (int scalarType) const

  BC on diffusive source term itself.

- virtual **DomainGhostBC** ∗ **streamFunctionBC** (const **Interval** &a_comps) const
- void **Dx** (const **Real** a_dx)

  miscellaneous utility functions sets cell spacing

- **Real Dx** () const

  returns cell spacing

- void **Time** (const **Real** a_time)

  sets time

- **Real Time** () const

  returns current time

- int **loBC** (int a_dir) const
- int **hiBC** (int a_dir) const
- int **getBC** (int a_dir, const **Side::LoHiSide** &a_side) const
- virtual void **computeBoundaryDt** (**Real** &a_dt, **Real** a_cfl, **Real** a_dx) const

  this is in case the BC's have an effect on the timestep

Protected Member Functions

- virtual **DomainGhostBC** ∗ **basicPressureBC** (bool a_isHomogeneous) const

  sets basic physical BC on pressure (called by public BC functions)

- virtual **DomainGhostBC** ∗ **basicGradPressureBC** () const

  sets basic physical BC on pressure before taking gradients

- virtual **basicVelBC** ∗ **basicCCVelBC** (bool a_isHomogeneous, bool a_isViscous) const

  basic physical BC on cell-centered velocity

- virtual void **basicCCVelBC** (**DomainGhostBC** &a_newBC, bool a_isHomogeneous, bool a_isViscous, int a_velComponent, **Interval** &a_interval) const

  returns a single-component BC for cell-centered velocity

- virtual **basicEdgeVelBC** ∗ **basicECVelBC** (bool a_isHomogeneous, bool a_isViscous, bool a_isSingleComponent) const

- virtual **DomainGhostBC** ∗ **basicScalBC** (bool a_isHomogeneous, int a_scalType) const
- virtual void **setBCs** ()

    interact with **ParmParse** *to set physical BC types*

Protected Attributes

- **Tuple**< int, **SpaceDim** > **m_loBC**
- **Tuple**< int, **SpaceDim** > **m_hiBC**

7.102.1 Detailed Description

Encapsulates all physical BC's for IAMR solves.

This class is designed to contain all the information needed to specify all physical boundary conditions needed for the IAMR code. This includes BC's for velocity, pressures, scalars. While basic solid-wall BCs are coded in to this class, the idea is that for a more complicated set of physical boundary conditions (inflow, etc) a problem-dependent class will be derived from this one which will override the generic BC's in this class

7.102.2 Member Enumeration Documentation

7.102.2.1 enum PhysBCUtil::BCType

Enumeration values:

bogusBC

SolidWall

Inflow

Outflow

Symmetry

noShear

Custom

NUM_PHYS_BC_TYPES

7.102.3 Constructor & Destructor Documentation

7.102.3.1 PhysBCUtil::PhysBCUtil ()

7.102.3.2 virtual PhysBCUtil::∼PhysBCUtil () `[virtual]`

7.102.4 Member Function Documentation

7.102.4.1 virtual basicEdgeVelBC∗ PhysBCUtil::advectionVelBC () const `[virtual]`

returns edge-centered vel BC for edge-ctrd vel (single component)

7.102.4.2 virtual void PhysBCUtil::basicCCVelBC (DomainGhostBC & *a_newBC*, bool *a_isHomogeneous*, bool *a_isViscous*, int *a_velComponent*, Interval & *a_interval*) const [protected, virtual]

returns a single-component BC for cell-centered velocity

Reimplemented in **channelBC** (p. 190).

7.102.4.3 virtual basicVelBC* PhysBCUtil::basicCCVelBC (bool *a_isHomogeneous*, bool *a_isViscous*) const [protected, virtual]

basic physical BC on cell-centered velocity

7.102.4.4 virtual basicEdgeVelBC* PhysBCUtil::basicECVelBC (bool *a_isHomogeneous*, bool *a_isViscous*, bool *a_isSingleComponent*) const [protected, virtual]

Reimplemented in **channelBC** (p. 191).

7.102.4.5 virtual DomainGhostBC* PhysBCUtil::basicGradPressureBC () const [protected, virtual]

sets basic physical BC on pressure before taking gradients

7.102.4.6 virtual DomainGhostBC* PhysBCUtil::basicPressureBC (bool *a_isHomogeneous*) const [protected, virtual]

sets basic physical BC on pressure (called by public BC functions)

7.102.4.7 virtual DomainGhostBC* PhysBCUtil::basicScalBC (bool *a_isHomogeneous*, int *a_scalType*) const [protected, virtual]

Reimplemented in **channelBC** (p. 191).

7.102.4.8 virtual void PhysBCUtil::computeBoundaryDt (Real & *a_dt*, Real *a_cfl*, Real *a_dx*) const [virtual]

this is in case the BC's have an effect on the timestep

pass in currently computed dt, along with cfl and dx. If effect of boundary conditions requires a decreased timestep, the newly reduced timestep is returned. In the default case, this just returns a_dt back; however, derived classes may actually have an effect.

Reimplemented in **channelBC** (p. 191).

7.102.4.9 virtual DomainGhostBC* PhysBCUtil::diffusiveBC (int *a_scalarType*) const [virtual]

Scalar BC's.

Reimplemented in **channelBC** (p. 191).

7.102.4.10 virtual basicVelBC* PhysBCUtil::diffusiveSrcBC (int *scalarType*) const [virtual]

BC on diffusive source term itself.

7.102.4.11 Real PhysBCUtil::Dx () const

returns cell spacing

7.102.4.12 void PhysBCUtil::Dx (const Real *a_dx*)

miscellaneous utility functions sets cell spacing

7.102.4.13 virtual basicEdgeVelBC* PhysBCUtil::edgeVelBC () const [virtual]

returns edge-centered velocity BC for edge-centered vel (multicomponent)

7.102.4.14 virtual DomainGhostBC* PhysBCUtil::FreestreamCorrBC () const [virtual]

returns BC for freestream preservation correction

7.102.4.15 int PhysBCUtil::getBC (int *a_dir*, const Side::LoHiSide & *a_side*) const

7.102.4.16 virtual DomainGhostBC* PhysBCUtil::gradELambdaBC () const [virtual]

BC for grad eLambda.

Reimplemented in **channelBC** (p. 191).

7.102.4.17 virtual DomainGhostBC* PhysBCUtil::gradESyncBC () const [virtual]

returns BC applied to sync correction before computing gradient

7.102.4.18 virtual DomainGhostBC* PhysBCUtil::gradInitialProjCorrBC () const [virtual]

BC for gradient of correction for initial projection.

7.102.4.19 virtual DomainGhostBC* PhysBCUtil::gradMacPressureBC () const [virtual]

returns BC applied to MAC pressure before computing gradient

Reimplemented in **channelBC** (p. 191).

7.102.4.20 virtual DomainGhostBC* PhysBCUtil::gradPiBC () const [virtual]

returns BC applied to Pi before computing gradient

7.102.4.21 int PhysBCUtil::hiBC (int *a_dir*) const

7.102.4.22 virtual DomainGhostBC* PhysBCUtil::InitialProjBC () const [virtual]

returns pressure BC for initial velocity projection

7.102.4.23 virtual DomainGhostBC* PhysBCUtil::lambdaBC () const [virtual]

volume discrepancy BC's

7.102.4.24 virtual DomainGhostBC* PhysBCUtil::LevelPressureBC () const [virtual]

returns pressure BC for level projection

7.102.4.25 int PhysBCUtil::loBC (int *a_dir*) const

7.102.4.26 virtual DomainGhostBC* PhysBCUtil::MacPressureBC () const [virtual]

Pressure BCs returns pressure BC for MAC projection.

7.102.4.27 virtual PhysBCUtil* PhysBCUtil::newPhysBCUtil () const [virtual]

"virtual" constructor

Reimplemented in **channelBC** (p. 191).

7.102.4.28 virtual DomainGhostBC* PhysBCUtil::scalarDiffusionSolveBC (int *a_scalarType*) const [virtual]

7.102.4.29 virtual DomainGhostBC* PhysBCUtil::scalarRefluxSolveBC (int *a_scalarType*) const [virtual]

7.102.4.30 virtual DomainGhostBC* PhysBCUtil::scalarTraceBC (int *a_scalarType*) const [virtual]

7.102.4.31 virtual void PhysBCUtil::setBCs () [protected, virtual]

interact with **ParmParse** to set physical BC types

Reimplemented in **channelBC** (p. 192).

7.102.4.32 virtual DomainGhostBC* PhysBCUtil::streamFunctionBC (const Interval & *a_comps*) const [virtual]

7.102.4.33 virtual DomainGhostBC* PhysBCUtil::SyncProjBC () const [virtual]

returns pressure BC for sync projection

7.102.4.34 Real PhysBCUtil::Time () const

returns current time

7.102.4.35 void PhysBCUtil::Time (const Real *a_time*)

sets time

7.102.4.36 virtual basicVelBC* PhysBCUtil::tracingVelBC () const [virtual]

returns velocity BC for tracing (inviscid)

7.102.4.37 virtual basicVelBC* PhysBCUtil::uDelUBC (bool *a_isViscous*) const [virtual]

returns BC on cell-centered u-dot-del-u term

7.102.4.38 virtual basicVelBC* PhysBCUtil::uStarBC (bool *a_isViscous* = true) const [virtual]

returns pre-projection velocity BC

7.102.4.39 virtual DomainGhostBC* PhysBCUtil::viscousRefluxBC (int *a_dir*) const [virtual]

returns single-component BC for viscous refluxing solves

7.102.4.40 virtual DomainGhostBC* PhysBCUtil::viscousSolveBC (int *a_dir*) const [virtual]

returns single-component BC for viscous solves

7.102.4.41 virtual basicVelBC* PhysBCUtil::viscousSrcBC () const [virtual]

Miscellaneous BC's BC on viscous source term itself.

7.102.4.42 virtual basicVelBC* PhysBCUtil::viscousVelBC () const [virtual]

returns velocity BC for viscous source

7.102.5 Member Data Documentation

7.102.5.1 Tuple<int, SpaceDim> PhysBCUtil::m_hiBC [protected]

7.102.5.2 Tuple<int, SpaceDim> PhysBCUtil::m_loBC [protected]

The documentation for this class was generated from the following file:

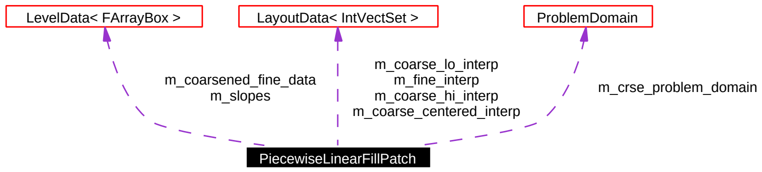
- **PhysBCUtil.H**

7.103 PiecewiseLinearFillPatch Class Reference

Fills ghost cells by linear interpolation in space and time.

`#include <PiecewiseLinearFillPatch.H>`

Collaboration diagram for PiecewiseLinearFillPatch:



Public Member Functions

- **PiecewiseLinearFillPatch** ()
- **~PiecewiseLinearFillPatch** ()
- **PiecewiseLinearFillPatch** (const **DisjointBoxLayout** &a_fine_domain, const **DisjointBoxLayout** &a_coarse_domain, int a_num_comps, const **Box** &a_crse_problem_domain, int a_ref_ratio, int a_interp_radius)
- **PiecewiseLinearFillPatch** (const **DisjointBoxLayout** &a_fine_domain, const **DisjointBoxLayout** &a_coarse_domain, int a_num_comps, const **ProblemDomain** &a_crse_problem_domain, int a_ref_ratio, int a_interp_radius)
- void **define** (const **DisjointBoxLayout** &a_fine_domain, const **DisjointBoxLayout** &a_coarse_domain, int a_num_comps, const **Box** &a_crse_problem_domain, int a_ref_ratio, int a_interp_radius)
- void **define** (const **DisjointBoxLayout** &a_fine_domain, const **DisjointBoxLayout** &a_coarse_domain, int a_num_comps, const **ProblemDomain** &a_crse_problem_domain, int a_ref_ratio, int a_interp_radius)
- bool **isDefined** () const
- void **fillInterp** (**LevelData**< **FArrayBox** > &a_fine_data, const **LevelData**< **FArrayBox** > &a_old_coarse_data, const **LevelData**< **FArrayBox** > &a_new_coarse_data, **Real** a_time_interp_coef, int a_src_comp, int a_dest_comp, int a_num_comp)
- void **printIntVectSets** () const

Protected Member Functions

- void **timeInterp** (const **LevelData**< **FArrayBox** > &a_old_coarse_data, const **LevelData**< **FArrayBox** > &a_new_coarse_data, **Real** a_time_interp_coef, int a_src_comp, int a_dest_comp, int a_num_comp)
- void **fillConstantInterp** (**LevelData**< **FArrayBox** > &a_fine_data, int a_src_comp, int a_dest_comp, int a_num_comp) const
- void **computeSlopes** (int a_src_comp, int a_num_comp)
- void **incrementLinearInterp** (**LevelData**< **FArrayBox** > &a_fine_data, int a_src_comp, int a_dest_comp, int a_num_comp) const

Protected Attributes

- bool **m_is_defined**
- int **m_ref_ratio**
- int **m_interp_radius**
- **LevelData< FArrayBox > m_coarsened_fine_data**
- **LevelData< FArrayBox > m_slopes** [3]
- **ProblemDomain m_crse_problem_domain**
- **LayoutData< IntVectSet > m_fine_interp**
- **LayoutData< IntVectSet > m_coarse_centered_interp** [**SpaceDim**]
- **LayoutData< IntVectSet > m_coarse_lo_interp** [**SpaceDim**]
- **LayoutData< IntVectSet > m_coarse_hi_interp** [**SpaceDim**]

Static Protected Attributes

- const int **s_stencil_radius**

7.103.1 Detailed Description

Fills ghost cells by linear interpolation in space and time.

Fills some fine level ghost cells by piecewise linear interpolation from the coarse level.

This class fills the first a_interp_radius layers of fine level ghost cells that are not part of other fine level grids. It uses piecewise linear interpolation from the coarse level. The slopes are computed using van leer limiting if there is enough room for the stencil. It drops order and uses first-order one-sided differences otherwise.

Below is a picture of a fine level, with a_interp_radius = 1. The ghost cells of grid 1 are shown. Cells marked X are interpolated to from the coarse level. Cells marked ^ are not interpolated to because they are in the valid domain of another fine grid. Cells marked . are not interpolated to because they are outside the problem domain. All of this changes in the presence of periodic boundary conditions – in periodic directions, all cells are considered to be within the domain.

```
+=======================================+
|                                       |
|                                       |
|                                       |
|                                       |
|         + - - - - - - - - - - - - - - | +
|          X X X X X X X X X X X X X X X|.
|         | +===========================+ |
|          X|                           |.
|         | |                           | |
|          X|        grid 1             |.
|         | |                           | |
|          X|                           |.
|         | |                           | |
|          X|                           |.
|   +=======+===+===============+=======+ |
|   |      ^ ^ ^|X X X X X X X X|^ ^ ^ ^|.
|   |     + - - | - - - - - - - | - - - | +
|   |           |               |grid 2 |
|   |  grid 0   |               +=======+
|   |           |                       |
```

```
|   |           |                       |
|   |           |                       |
|   +===========+                       |
|                                       |
|                                       |
|              problem domain           |
+=======================================+
```

the picture below shows the locations of the types of slopes. The coarse grid is shown, and the projection of the fine grid onto the coarse index space (on which slopes are computed). Coarse cells marked ^ and . do not need slopes. Van Leer slopes in both directions are used at coarse cells marked X. Coarse cells marked V and < have one-sided differencing: cells marked V have low-sided differencing in the y-direction (in this example, because the coarse grid doesn't exist on the high side of these cells), and cells marked < have low-sided differencing in the x-direction (because the problem domain doesn't exist on the high side of these cells). There can also be cells that have one-sided differences in both directions.

```
+-----------------------+===============+
|                       | coarse grid 1 |
|                       |               |
|                       |               |
+=======================+ - - - - - - - | - +
|        $\$ / $\$ / $\$ / $\$ /|$\$ / $\$ / $\$ / $\$ /|
|       | V   V   V   V | X   X   X   $<$ | . |
|        / $\$ / $\$ / $\$ / $\$|/ $\$ / $\$ / $\$ / $\$|
|       |   +---------------------------+   |
|        $\$ /|           |              |
|       | X |           |               | . |
|        / $\$|           |              |
|       |   |           |               |   |
|        $\$ /|           |              |
|       | X |           |               | . |
|        / $\$|           |              |
|   +-------+---+---------------+-------+   |
|   |           |$\$ / $\$ /|$\$ / $\$ /|      |
|   |   | ^   ^ | X   X | X   X | ^   ^ | . |
|   |           |/ $\$ / $\$|/ $\$ / $\$|      |
|   |   + - - - | - - - - - - - +-------+ - +
|   |           |       |               |
|   |           |       |               |
|   |           |       |               |
|   +-----------+       +===============+
|                       |               |
|                       |               |
|      coarse grid 0    |               |
+=======================+---------------+
```

7.103.2 Constructor & Destructor Documentation

7.103.2.1 PiecewiseLinearFillPatch::PiecewiseLinearFillPatch ()

Default constructor. User must subsequently call **define()**.

7.103.2.2 PiecewiseLinearFillPatch::~PiecewiseLinearFillPatch ()

Destructor.

7.103.2.3 PiecewiseLinearFillPatch::PiecewiseLinearFillPatch (const DisjointBoxLayout & *a_fine_domain*, const DisjointBoxLayout & *a_coarse_domain*, int *a_num_comps*, const Box & *a_crse_problem_domain*, int *a_ref_ratio*, int *a_interp_radius*)

Defining constructor.

{ Arguments:}\ a_fine_domain (not modified): domain of fine level. \ a_coarse_domain (not modified): domain of coarse level. \ a_num_comps (not modified): number of components of state vector. \ a_problem_domain (not modified): problem domain on the coarse level. \ a_ref_ratio (not modified): refinement ratio. \ a_interp_radius (not modified): number of layers of fine ghost cells to fill by interpolation. \

7.103.2.4 PiecewiseLinearFillPatch::PiecewiseLinearFillPatch (const DisjointBoxLayout & *a_fine_domain*, const DisjointBoxLayout & *a_coarse_domain*, int *a_num_comps*, const ProblemDomain & *a_crse_problem_domain*, int *a_ref_ratio*, int *a_interp_radius*)

Defining constructor.

{ Arguments:}\ a_fine_domain (not modified): domain of fine level. \ a_coarse_domain (not modified): domain of coarse level. \ a_num_comps (not modified): number of components of state vector. \ a_problem_domain (not modified): problem domain on the coarse level. \ a_ref_ratio (not modified): refinement ratio. \ a_interp_radius (not modified): number of layers of fine ghost cells to fill by interpolation. \

7.103.3 Member Function Documentation

7.103.3.1 void PiecewiseLinearFillPatch::computeSlopes (int *a_src_comp*, int *a_num_comp*) `[protected]`

7.103.3.2 void PiecewiseLinearFillPatch::define (const DisjointBoxLayout & *a_fine_domain*, const DisjointBoxLayout & *a_coarse_domain*, int *a_num_comps*, const ProblemDomain & *a_crse_problem_domain*, int *a_ref_ratio*, int *a_interp_radius*)

Defines this object. The user may call **define()** once and call **fillInterp()** multiple times with different valid data sets.

{ Arguments:}\ a_fine_domain (not modified): domain of fine level. \ a_coarse_domain (not modified): domain of coarse level. \ a_num_comps (not modified): number of components of state vector. \ a_problem_domain (not modified): problem domain on the coarse level. \ a_ref_ratio (not modified): refinement ratio. \ a_interp_radius (not modified): number of layers of fine ghost cells to fill by interpolation. \

{ This:}\ —This object is modified.—

7.103.3.3 void PiecewiseLinearFillPatch::define (const DisjointBoxLayout & *a_fine_domain*, const DisjointBoxLayout & *a_coarse_domain*, int *a_num_comps*, const Box & *a_crse_problem_domain*, int *a_ref_ratio*, int *a_interp_radius*)

Defines this object. The user may call **define()** once and call **fillInterp()** multiple times with different valid data sets.

{ Arguments:}\ a_fine_domain (not modified): domain of fine level. \ a_coarse_domain (not modified): domain of coarse level. \ a_num_comps (not modified): number of components of state vector. \ a_problem_domain (not modified): problem domain on the coarse level. \ a_ref_ratio (not modified): refinement ratio. \ a_interp_radius (not modified): number of layers of fine ghost cells to fill by interpolation. \

{ This:}\ —This object is modified.—

7.103.3.4 void PiecewiseLinearFillPatch::fillConstantInterp (LevelData< FArrayBox > & *a_fine_data*, int *a_src_comp*, int *a_dest_comp*, int *a_num_comp*) const `[protected]`

7.103.3.5 void PiecewiseLinearFillPatch::fillInterp (LevelData< FArrayBox > & *a_fine_data*, const LevelData< FArrayBox > & *a_old_coarse_data*, const LevelData< FArrayBox > & *a_new_coarse_data*, Real *a_time_interp_coef*, int *a_src_comp*, int *a_dest_comp*, int *a_num_comp*)

Fills the first m_interp_radius layers of fine ghost cells by interpolation from the coarse level. It is an error to call if not this->**isDefined()**. The range components to interpolate must be specified. The corresponding components on the coarse and fine levels may be different. It is required that a_fine_data's domain is the same as was specified in the most recent call to **define()**. It is expected that the coarse and fine level's domains are properly nested.

{ Arguments:}\ a_fine_data (modified): fine-level data being interpolated to.\ a_old_coarse_data (not modified): coarse level source data at the old time.\ a_new_coarse_data (not modified): coarse level source data at the new time.\ a_time_interp_coef (not modified): time interpolation coefficient, in the range [0:1]. 0=old time, 1=new time.\ a_src_comp (not modifed): starting coarse data component.\ a_dest_comp (not modifed): starting fine data component.\ a_num_comp (not modified): number of data components to be interpolated.

{ This:}\ Well, it's complicated. As far as the user is concerned, this object is not modified. See the design document if you care for details.

7.103.3.6 void PiecewiseLinearFillPatch::incrementLinearInterp (LevelData< FArrayBox > & *a_fine_data*, int *a_src_comp*, int *a_dest_comp*, int *a_num_comp*) const `[protected]`

7.103.3.7 bool PiecewiseLinearFillPatch::isDefined () const

Returns true if this object was created with the defining constructor or if **define()** has been called.

{ This:}\ This object is not modified.

7.103.3.8 void PiecewiseLinearFillPatch::printIntVectSets () const

7.103.3.9 void PiecewiseLinearFillPatch::timeInterp (const LevelData< FArrayBox > & *a_old_coarse_data*, const LevelData< FArrayBox > & *a_new_coarse_data*, Real *a_time_interp_coef*, int *a_src_comp*, int *a_dest_comp*, int *a_num_comp*) `[protected]`

7.103.4 Member Data Documentation

7.103.4.1 LayoutData<IntVectSet> PiecewiseLinearFillPatch::m_coarse_centered_interp[SpaceDim] `[protected]`

7.103.4.2 LayoutData<IntVectSet> PiecewiseLinearFillPatch::m_coarse_hi_interp[SpaceDim] `[protected]`

7.103.4.3 LayoutData<IntVectSet> PiecewiseLinearFillPatch::m_coarse_lo_interp[SpaceDim] `[protected]`

7.103.4.4 LevelData<FArrayBox> PiecewiseLinearFillPatch::m_coarsened_fine_data `[protected]`

7.103.4.5 ProblemDomain PiecewiseLinearFillPatch::m_crse_problem_domain `[protected]`

7.103.4.6 LayoutData<IntVectSet> PiecewiseLinearFillPatch::m_fine_interp `[protected]`

7.103.4.7 int PiecewiseLinearFillPatch::m_interp_radius `[protected]`

7.103.4.8 bool PiecewiseLinearFillPatch::m_is_defined `[protected]`

7.103.4.9 int PiecewiseLinearFillPatch::m_ref_ratio `[protected]`

7.103.4.10 LevelData<FArrayBox> PiecewiseLinearFillPatch::m_slopes[3] `[protected]`

7.103.4.11 const int PiecewiseLinearFillPatch::s_stencil_radius `[static, protected]`

The documentation for this class was generated from the following file:

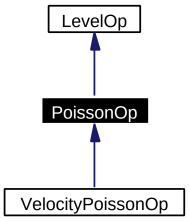
- **PiecewiseLinearFillPatch.H**

7.104 PoissonOp Class Reference

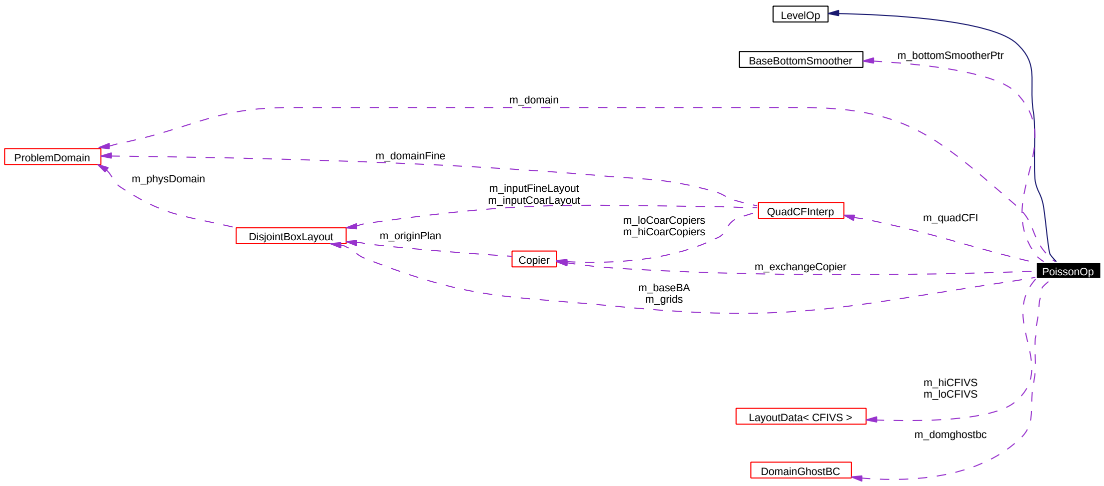
one-sided differencing version of poisson operator

`#include <PoissonOp.H>`

Inheritance diagram for PoissonOp:



Collaboration diagram for PoissonOp:



Public Member Functions

- **PoissonOp** ()

  Default constructor.

- **~PoissonOp** ()

  Destructor.

- virtual **LevelOp** $*$ **new_levelop** () const

  Virtual constructor workaround.

- virtual bool **isDefined** () const

  Has this object been defined?

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseBAPtr, **Real** a_dxLevel, int a_refRatio, const **ProblemDomain** &a_domain, bool a_homogeneousOnly=false, int a_ncomp=1)

  Full define function.

- void **define** (const **LevelOp** *a_opfine, int a_reftoFine)

  Full define function. Define from finer levelop.

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseBAPtr, **Real** a_dxLevel, int a_refRatio, const **Box** &a_domain, bool a_homogeneousOnly=false, int a_ncomp=1)

  Full define function (Deprecated – use **ProblemDomain** *version).*

- void **smooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs)

  Smoother.

- virtual void **levelPreconditioner** (**LevelData**< **FArrayBox** > &a_phihat, const **LevelData**< **FArrayBox** > &a_rhshat)

  Apply preconditioning to the solution on this level.

- void **applyOpI** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phicPtr, **LevelData**< **FArrayBox** > &a_lofPhi)

  Evaluate Operator, inhomogeneous boundary conditions.

- void **applyOpIcfHphys** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phicPtr, **LevelData**< **FArrayBox** > &a_lofPhi)

  Evaluate Operator, mixed boundary conditions.

- void **applyOpH** (**LevelData**< **FArrayBox** > &a_phi, **LevelData**< **FArrayBox** > &a_lofPhi)

  Evaluate Operator using homogeneous boundary conditions.

- void **CFInterp** (**LevelData**< **FArrayBox** > &a_phif, const **LevelData**< **FArrayBox** > &a_phic)

  Coarse / Fine interpolation operator.

- void **applyOpHcfIphys** (**LevelData**< **FArrayBox** > &a_phi, **LevelData**< **FArrayBox** > &a_lOfPhi)

  Evaluate Operator.

- void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &a_phif)

  Homogeneous coarse/fine interpolation operator.

- void **setDomainGhostBC** (const **DomainGhostBC** &a_dombcin)

  Set physical boundary condition types.

- void **setBottomSmoother** (const **BaseBottomSmoother** &a_bottomSmoother)

  Set the bottom smoother to use in solve.

- void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &a_phif, const **DataIndex** &a_datInd, int a_idir, **Side::LoHiSide** a_hiorlo)

  Does homogeneous coarse/fine interpolation.

- void **bottomSmoother** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs)

  This performs the smoothing at the bottom of a multigrid V-cycle.

- void **getFlux** (**FArrayBox** &a_fineFlux, const **FArrayBox** &a_data, const **DataIndex** &a_datInd, int a_dir)

  Get flux on this level in the direction dir on the given **Box**.

- virtual void **setConvergenceMetric** (**Real** a_metric, int a_comp)

  pass-through to bottom smoother

Protected Attributes

- **DomainGhostBC m_domghostbc**
- **ProblemDomain m_domain**
- **DisjointBoxLayout m_grids**
- **Copier m_exchangeCopier**
- **DisjointBoxLayout m_baseBA**
- int **m_ncomp**
- int **m_refRatio**
- **Real m_dxLevel**
- **Real m_dxCrse**
- **QuadCFInterp m_quadCFI**
- **BaseBottomSmoother** ∗ **m_bottomSmootherPtr**
- bool **m_isDefined**
- bool **m_isBCDefined**
- bool **m_ihcfiEnabled**
- **LayoutData**< **CFIVS** > **m_loCFIVS** [**SpaceDim**]
- **LayoutData**< **CFIVS** > **m_hiCFIVS** [**SpaceDim**]

7.104.1 Detailed Description

one-sided differencing version of poisson operator

this class contains stuff to solve div (grad phi) = rhs

7.104.2 Constructor & Destructor Documentation

7.104.2.1 PoissonOp::PoissonOp ()

Default constructor.

7.104.2.2 PoissonOp::~PoissonOp ()

Destructor.

7.104.3 Member Function Documentation

7.104.3.1 void PoissonOp::applyOpH (LevelData< FArrayBox > & *a_phi*, LevelData< FArrayBox > & *a_lofPhi*) [virtual]

Evaluate Operator using homogeneous boundary conditions.

Uses homogeneous form of both coarse-fine and physical boundary conditions.

Implements **LevelOp** (p. 378).

7.104.3.2 void PoissonOp::applyOpHcfIphys (LevelData< FArrayBox > & *a_phi*, LevelData< FArrayBox > & *a_lOfPhi*) [virtual]

Evaluate Operator.

Uses homogeneous form of C/F boundary conditions. with inhomogeneous physical boundary conditions

Implements **LevelOp** (p. 378).

7.104.3.3 void PoissonOp::applyOpI (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > * *a_phicPtr*, LevelData< FArrayBox > & *a_lofPhi*) [virtual]

Evaluate Operator, inhomogeneous boundary conditions.

Uses inhomogeneous C/F boundary conditions. and inhomogeneous phys boundary conditions

Implements **LevelOp** (p. 378).

7.104.3.4 void PoissonOp::applyOpIcfHphys (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > * *a_phicPtr*, LevelData< FArrayBox > & *a_lofPhi*) [virtual]

Evaluate Operator, mixed boundary conditions.

Uses inhomogeneous form of C/F boundary conditions. and homogeneous physical boundary conditions

Implements **LevelOp** (p. 378).

7.104.3.5 void PoissonOp::bottomSmoother (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*) [virtual]

This performs the smoothing at the bottom of a multigrid V-cycle.

Implements **LevelOp** (p. 378).

7.104.3.6 void PoissonOp::CFInterp (LevelData< FArrayBox > & *a_phif*, const LevelData< FArrayBox > & *a_phic*) [virtual]

Coarse / Fine interpolation operator.

Implements **LevelOp** (p. 379).

7.104.3.7 void PoissonOp::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseBAPtr*, Real *a_dxLevel*, int *a_refRatio*, const Box & *a_domain*, bool *a_homogeneousOnly* = false, int *a_ncomp* = 1) `[virtual]`

Full define function (Deprecated – use **ProblemDomain** version).

Full define function. makes all coarse-fine information and sets internal variables

Implements **LevelOp** (p. 379).

7.104.3.8 void PoissonOp::define (const LevelOp ∗ *a_opfine*, int *a_reftoFine*) `[virtual]`

Full define function. Define from finer levelop.

Implements **LevelOp** (p. 379).

7.104.3.9 void PoissonOp::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseBAPtr*, Real *a_dxLevel*, int *a_refRatio*, const ProblemDomain & *a_domain*, bool *a_homogeneousOnly* = false, int *a_ncomp* = 1) `[virtual]`

Full define function.

Makes all coarse-fine information and sets internal variables

Implements **LevelOp** (p. 379).

7.104.3.10 void PoissonOp::getFlux (FArrayBox & *a_fineFlux*, const FArrayBox & *a_data*, const DataIndex & *a_datInd*, int *a_dir*) `[virtual]`

Get flux on this level in the direction dir on the given **Box**.

The fluxes live on the cell faces with direction a_dir. Fluxes are computed for all interior edges of a_data. The flux fab is resized inside the routine.

Implements **LevelOp** (p. 380).

7.104.3.11 void PoissonOp::homogeneousCFInterp (LevelData< FArrayBox > & *a_phif*, const DataIndex & *a_datInd*, int *a_idir*, Side::LoHiSide *a_hiorlo*)

Does homogeneous coarse/fine interpolation.

7.104.3.12 void PoissonOp::homogeneousCFInterp (LevelData< FArrayBox > & *a_phif*) `[virtual]`

Homogeneous coarse/fine interpolation operator.

Implements **LevelOp** (p. 380).

7.104.3.13 virtual bool PoissonOp::isDefined () const `[virtual]`

Has this object been defined?

Implements **LevelOp** (p. 380).

Reimplemented in **VelocityPoissonOp** (p. 629).

7.104.3.14 virtual void PoissonOp::levelPreconditioner (LevelData< FArrayBox > & *a_phihat*, const LevelData< FArrayBox > & *a_rhshat*) [virtual]

Apply preconditioning to the solution on this level.

Given rhshat, returns phihat from M(phihat) = rhshat

Implements **LevelOp** (p. 380).

7.104.3.15 virtual LevelOp* PoissonOp::new_levelop () const [virtual]

Virtual constructor workaround.

Implements **LevelOp** (p. 380).

Reimplemented in **VelocityPoissonOp** (p. 629).

7.104.3.16 void PoissonOp::setBottomSmoother (const BaseBottomSmoother & *a_bottomSmoother*)

Set the bottom smoother to use in solve.

7.104.3.17 virtual void PoissonOp::setConvergenceMetric (Real *a_metric*, int *a_comp*) [virtual]

pass-through to bottom smoother

Implements **LevelOp** (p. 381).

7.104.3.18 void PoissonOp::setDomainGhostBC (const DomainGhostBC & *a_dombcin*)

Set physical boundary condition types.

7.104.3.19 void PoissonOp::smooth (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*) [virtual]

Smoother.

Assumes that problem has already been put in residual correction form, so that C/F boundary conditions are homogeneous.

Implements **LevelOp** (p. 381).

7.104.4 Member Data Documentation

7.104.4.1 DisjointBoxLayout PoissonOp::m_baseBA [protected]

DisjointBoxLayout for next coarser level this is == NULL if there is no coarser level

7.104.4.2 BaseBottomSmoother* PoissonOp::m_bottomSmootherPtr [protected]

Bottom smoother object

7.104.4.3 ProblemDomain PoissonOp::m_domain [protected]

Domain of fine grid

7.104.4.4 DomainGhostBC PoissonOp::m_domghostbc [protected]

Boundary conditons

7.104.4.5 Real PoissonOp::m_dxCrse [protected]

Mesh spacing at next coarser level

7.104.4.6 Real PoissonOp::m_dxLevel [protected]

Mesh spacing at this level

7.104.4.7 Copier PoissonOp::m_exchangeCopier [protected]

Copier object that knows how to perform exchange operations on **LevelData** objects defined on m_grids

7.104.4.8 DisjointBoxLayout PoissonOp::m_grids [protected]

The grids at the current level

7.104.4.9 LayoutData<CFIVS> PoissonOp::m_hiCFIVS[SpaceDim] [protected]

Coarse-fine stencils for homogeneous CFInterp

7.104.4.10 bool PoissonOp::m_ihcfiEnabled [protected]

Is inhomogeneous CF interpolation possible?

7.104.4.11 bool PoissonOp::m_isBCDefined [protected]

Has bc been defined?

7.104.4.12 bool PoissonOp::m_isDefined [protected]

Has full define function been called?

7.104.4.13 LayoutData<CFIVS> PoissonOp::m_loCFIVS[SpaceDim] [protected]

Coarse-fine stencils for homogeneous CFInterp

7.104.4.14 int PoissonOp::m_ncomp [protected]

Number of components

7.104.4.15 QuadCFInterp PoissonOp::m_quadCFI [protected]

7.104.4.16 int PoissonOp::m_refRatio [protected]

Refinement ratio between this and the next coarser level

The documentation for this class was generated from the following file:

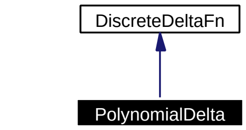
- **PoissonOp.H**

7.105 PolynomialDelta Class Reference

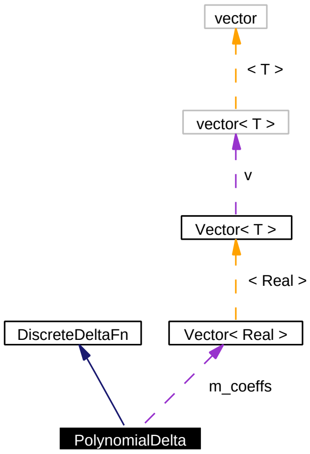
Instantiation of discrete delta function using a polynomial.

`#include <PolynomialDelta.H>`

Inheritance diagram for PolynomialDelta:



Collaboration diagram for PolynomialDelta:



Public Member Functions

- **PolynomialDelta** ()

  constructor

- **PolynomialDelta** (**Real** a_epsilon)

  constructor with radius – calls corresponding define fn.

- virtual ~**PolynomialDelta** ()

  destructor

- virtual void **define** (**Real** a_epsilon)

  define function

- virtual **Real epsilon** () const

  returns epsilon

- virtual **DiscreteDeltaFn** ∗ **clone** () const *make a copy of this delta function*

- virtual **Real evaluateDelta** (**Real** a_radius) const

  evaluate the delta function at the given radius

- virtual **Real integralDelta** (**Real** a_radius) const

  returns the integral from 0 to radius of delta

- virtual **Real computeK** (**RealVect** &a_radius, int a_idir, int a_jdir) const

  returns the kernel_{ij} at radius r

- virtual void **sumForce** (**FArrayBox** &a_sum, const **RealVect** &a_force, const **Box** &a_box, const **RealVect** &a_position, const **RealVect** &a_origin, **Real** a_dx)

  returns sum_j force_j kernel_ij over a box

- virtual void **sumForce** (**FArrayBox** &a_sum, const **RealVect** &a_force, const **Box** &a_box, int a_dir, int a_destComp, const **RealVect** &a_position, const **RealVect** &a_origin, **Real** a_dx)

  returns sum_j force_j kernel_ij over a box

- virtual **Real computeLapDelta** (**Real** a_radius) const

  returns laplacian(delta) at radius r

- void **setCoeffs** (const **Vector**< **Real** > &a_coeffs)

  set coefficients for polynomial

- void **setCoeff** (**Real** a_coeff, int n)
- **Vector**< **Real** > **getCoeffs** () const

Protected Member Functions

- void **setDefaultValues** ()
- void **defineFactor** ()

Protected Attributes

- **Vector**< **Real** > **m_coeffs**

  polynomial coefficients

- **Real m_epsilon**

  radius of numerical delta function

- **Real m_factor**

  what we multiply the polynomial by for normalization

7.105.1 Detailed Description

Instantiation of discrete delta function using a polynomial.

The parameter epsilon defines the spreading radius of the discrete delta fn.

7.105.2 Constructor & Destructor Documentation

7.105.2.1 PolynomialDelta::PolynomialDelta ()

constructor

7.105.2.2 PolynomialDelta::PolynomialDelta (Real *a_epsilon*)

constructor with radius – calls corresponding define fn.

7.105.2.3 virtual PolynomialDelta::~PolynomialDelta () `[virtual]`

destructor

7.105.3 Member Function Documentation

7.105.3.1 virtual DiscreteDeltaFn* PolynomialDelta::clone () const `[virtual]`

make a copy of this delta function

Implements **DiscreteDeltaFn** (p. 226).

7.105.3.2 virtual Real PolynomialDelta::computeK (RealVect & *a_radius*, int *a_idir*, int *a_jdir*) const `[virtual]`

returns the kernel_{ij} at radius r

Implements **DiscreteDeltaFn** (p. 226).

7.105.3.3 virtual Real PolynomialDelta::computeLapDelta (Real *a_radius*) const `[virtual]`

returns laplacian(delta) at radius r

Implements **DiscreteDeltaFn** (p. 226).

7.105.3.4 virtual void PolynomialDelta::define (Real *a_epsilon*) `[virtual]`

define function

Implements **DiscreteDeltaFn** (p. 227).

7.105.3.5 void PolynomialDelta::defineFactor () `[protected]`

7.105.3.6 virtual Real PolynomialDelta::epsilon () const `[inline, virtual]`

returns epsilon

Implements **DiscreteDeltaFn** (p. 227).

7.105.3.7 virtual Real PolynomialDelta::evaluateDelta (Real *a_radius*) const [virtual]

evaluate the delta function at the given radius

Implements **DiscreteDeltaFn** (p. 227).

7.105.3.8 Vector<Real> PolynomialDelta::getCoeffs () const

7.105.3.9 virtual Real PolynomialDelta::integralDelta (Real *a_radius*) const [virtual]

returns the integral from 0 to radius of delta

Implements **DiscreteDeltaFn** (p. 227).

7.105.3.10 void PolynomialDelta::setCoeff (Real *a_coeff*, int *n*)

7.105.3.11 void PolynomialDelta::setCoeffs (const Vector< Real > & *a_coeffs*)

set coefficients for polynomial

7.105.3.12 void PolynomialDelta::setDefaultValues () [protected]

7.105.3.13 virtual void PolynomialDelta::sumForce (FArrayBox & *a_sum*, const RealVect & *a_force*, const Box & *a_box*, int *a_dir*, int *a_destComp*, const RealVect & *a_position*, const RealVect & *a_origin*, Real *a_dx*) [virtual]

returns sum_j force_j kernel_ij over a box

this version computes a single (idir) component and places it in the destComp component of a_sum. this is done over a box for effeciency.

Implements **DiscreteDeltaFn** (p. 227).

7.105.3.14 virtual void PolynomialDelta::sumForce (FArrayBox & *a_sum*, const RealVect & *a_force*, const Box & *a_box*, const RealVect & *a_position*, const RealVect & *a_origin*, Real *a_dx*) [virtual]

returns sum_j force_j kernel_ij over a box

sum is a SpaceDim-component fab which will contain the sum this is done over a box for effeciency.

Implements **DiscreteDeltaFn** (p. 227).

7.105.4 Member Data Documentation

7.105.4.1 Vector<Real> PolynomialDelta::m_coeffs [protected]

polynomial coefficients

7.105.4.2 Real PolynomialDelta::m_epsilon [protected]

radius of numerical delta function

7.105.4.3 Real PolynomialDelta::m_factor [protected]

what we multiply the polynomial by for normalization

The documentation for this class was generated from the following file:

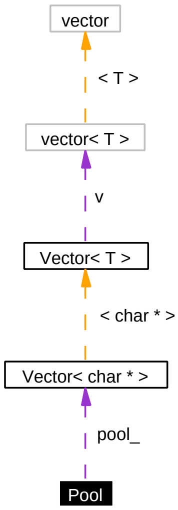
- **PolynomialDelta.H**

7.106 Pool Class Reference

Pool is a class to optimize memory allocation.

`#include <Pool.H>`

Collaboration diagram for Pool:



Public Member Functions

- **Pool** (int ptrSize, const char *name="unnamed", int poolSize=100, int alignment=sizeof(int))
- **~Pool** ()
- void * **getPtr** ()

    request a section of memory of ptrSize_ contiguous bytes.

- void **returnPtr** (void *ptr)

    return memory previous acquired with the **getPtr()** *function.*

- long **memUsage** () const

    report how much memory this Pool is currently using.

- void **clear** ()

Public Attributes

- char **name_** [64]

Static Public Attributes

- **PoolList** * **poolList_**

7.106.1 Detailed Description

Pool is a class to optimize memory allocation.

Pool is a class to optimize memory allocation. It is specialized to allocate fixed size chunks of memory specified by ptrSize in the constructor. Its operation is analogous to malloc, not new. It does not initialize the memory in any way. The constructor can optionally specify an initial pool size, and memory alignment. The pool size will grow as needed by calling ::new. The pool size never shrinks. Memory will be reclaimed at **~Pool()**. The units of poolSize are number-of-ptr-Size-chunks. The units of alignment are bytes.

Pool can only be used for objects of fixed size. Typically used in situations where a class or struct is being constantly constructed and deleted. Objects returned by value from functions, elements in a database, etc. Pool's tend to be allocated statically.

from Copier.cpp

```
// static data member s_motionIemPool getting constructed
Pool Copier::s_motionItemPool(sizeof(MotionItem), "Copier::MotionItem");



// note the use of a 'placement new'
MotionItem* item = new (s_motionItemPool.getPtr()) MotionItem(fromdi, todi, box);


// if your object does requires it's destructor to be called.
item->~MotionItem();

// then return the memory chunk to your pool
s_motionItemPool.returnPtr(item)
```

7.106.2 Constructor & Destructor Documentation

7.106.2.1 Pool::Pool (int *ptrSize*, const char * *name* = "unnamed", int *poolSize* = 100, int *alignment* = sizeof(int))

Parameters:

ptrSize Size of fixed memory allocations needed

poolSize Size of allocations (ptrSize*poolSize) made of the operating system. This controls the granularity of the Pool allocations. Smaller values result in less wasted memory, at the cost of more calls to the operating system for heap memory.

name optional name for this Pool. Used in reporting memory by the memory tracking system in Chombo.

7.106.2.2 Pool::~Pool ()

7.106.3 Member Function Documentation

7.106.3.1 void Pool::clear ()

undocumented function, call this at own risk. You must be absolutely positive that you have returned all the ptr's you've asked for, or you can have major seg faults.

7.106.3.2 void* Pool::getPtr ()

request a section of memory of ptrSize_ contiguous bytes.

7.106.3.3 long Pool::memUsage () const

report how much memory this Pool is currently using.

memUsage for a Pool only grows until Pool destruction. The Pool object has no knowledge of what pieces of memory it has parcelled out to a user, so it keeps it all available. The user is responsible for not leaking Pool memory.

7.106.3.4 void Pool::returnPtr (void * *ptr*)

return memory previous acquired with the **getPtr()** function.

7.106.4 Member Data Documentation

7.106.4.1 char Pool::name_[64]

7.106.4.2 PoolList* Pool::poolList_ `[static]`

The documentation for this class was generated from the following file:

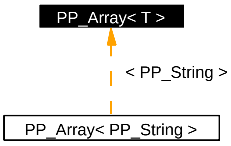
- **Pool.H**

7.107 PP_Array< T > Class Template Reference

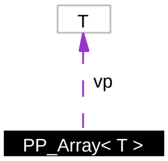
An Array of Objects of Type T for **ParmParse**.

`#include <ParmParse.H>`

Inheritance diagram for PP_Array< T >:



Collaboration diagram for PP_Array< T >:



Public Member Functions

- **PP_Array** ()

  : Constructs an 'PP_Array' with no elements

- **PP_Array** (long len)
- **PP_Array** (long len, const T &initialvalue)
- **PP_Array** (const T ∗vec, long len)
- **PP_Array** (const **PP_Array**< T > &rhs)

  : The copy constructor.

- **PP_Array**< T > & **operator=** (const **PP_Array**< T > &rhs)
- **~PP_Array** ()

  : Deletes the 'PP_Array' and all objects it contains.

- void **clear** ()
- bool **ready** () const

  : Returns true if this 'PP_Array' is not empty.

- void **reserve** (long _truesize)
- void **shrinkWrap** ()
- void **resize** (long newlen)
- void **resize** (long newlen, const T &initialvalue)
- long **length** () const

  : Return number of elements in the array.

- long **trueSize** () const
- T & **operator**[ ] (long K)
- const T & **operator**[ ] (long K) const

  : Same as above, except acts on const PP_Array's.

- T & **get** (long i)

  : Different syntax for operator[] (long i).

- const T & **get** (long i) const

  : Different syntax for const operator[] (long i).

- T ∗ **dataPtr** ()
- const T ∗ **dataPtr** () const

  : Same as above for constant PP_Arrays.

- void **set** (long i, const T &elem)

  : Changes the i'th element of this 'PP_Array' to elem.

- void **swap** (long i, long j)

  : This function swaps the i'th and j'th element of the PP_Array.

- bool **operator==** (const **PP_Array**< T > &rhs) const

  : Test for equality.

- bool **operator!=** (const **PP_Array**< T > &rhs) const

  : Test for inequality.

Protected Attributes

- long **truesize**
- long **nelem**
- T ∗ **vp**

7.107.1 Detailed Description

template<class T> class PP_Array< T >

An Array of Objects of Type T for **ParmParse**.

This class implements an array of objects of the parameterized type T. In contrast with the predefined C++ array type, an 'PP_Array' object knows its length, can be dynamically resized, and provides automatic bounds checking. The bounds checking can be turned off by specifying the -DNDEBUG flag on the command line when compiling the BOXLIB library. The main reason for defining the PP_Array class is that it is used, either by composition or inheritance, to implement many of the other classes in the BOXLIB library.

The 'PP_Array' class works by storing copies of the objects it contains. If the objects are large, such as 'FARRAYBOX's it is probably better to use the 'PArray' class which is an array class that stores pointers to the objects (avoiding expensive copies). The 'PP_Array' class destructs all objects in the array when it is itself destructed. If this is not the desired action, the 'PArray' class should be used.

In the PP_Array<T> class, there are two different concepts of size: the amount of space allocated, and the amount of space actually used. Obviously, the allocated space must be larger than the used space. We separate these two concepts to allow the user to optionally avoid memory allocation costs. Rather than construct and destroy a temporary PP_Array<T> many times, it may be less expensive to allocate it once with enough memory to handle all uses, and resize the PP_Array<T> for each particular use. See the member functions 'reserve', 'shrinkWrap', and 'resize'.

Note that care must be taken when deriving classes from 'PP_Array'. It is a concrete class, not a polymorphic one.

This class does NOT provide an assignment operator for assigning an integer to an PP_Array<T>.

This is a convenience class for **ParmParse** and will not be in any way supported by anyone at ANAG.

7.107.2 Constructor & Destructor Documentation

7.107.2.1 template<class T> PP_Array< T >::PP_Array () `[inline]`

: Constructs an 'PP_Array' with no elements

7.107.2.2 template<class T> PP_Array< T >::PP_Array (long *len*) `[inline]`

: Constructs an 'PP_Array' of length len with the value of each element defined by the default constructor for 'T'.

7.107.2.3 template<class T> PP_Array< T >::PP_Array (long *len*, const T & *initialvalue*)

: Constructs an 'PP_Array' of length len with the value of each elements given by initialvalue.

7.107.2.4 template<class T> PP_Array< T >::PP_Array (const T * *vec*, long *len*)

: Constructs an 'PP_Array' of length len in which the K'th value is a copy of vec[K].

7.107.2.5 template<class T> PP_Array< T >::PP_Array (const PP_Array< T > & *rhs*)

: The copy constructor.

7.107.2.6 template<class T> PP_Array< T >::~PP_Array () `[inline]`

: Deletes the 'PP_Array' and all objects it contains.

7.107.3 Member Function Documentation

7.107.3.1 template<class T> void PP_Array< T >::clear () [inline]

: Destructs each element in this 'PP_Array'. The resulting array has length zero.

7.107.3.2 template<class T> const T * PP_Array< T >::dataPtr () const [inline]

: Same as above for constant PP_Arrays.

7.107.3.3 template<class T> T * PP_Array< T >::dataPtr () [inline]

: Returns pointer to vector of data. This function breaks object encapsulation and should only be used for interfacing to Fortran subroutines.

7.107.3.4 template<class T> const T & PP_Array< T >::get (long *i*) const [inline]

: Different syntax for const operator[] (long i).

7.107.3.5 template<class T> T & PP_Array< T >::get (long *i*) [inline]

: Different syntax for operator[] (long i).

7.107.3.6 template<class T> long PP_Array< T >::length () const [inline]

: Return number of elements in the array.

7.107.3.7 template<class T> bool PP_Array< T >::operator!= (const PP_Array< T > & *rhs*) const [inline]

: Test for inequality.

7.107.3.8 template<class T> PP_Array< T > & PP_Array< T >::operator= (const PP_Array< T > & *rhs*)

: This operator deletes the current 'PP_Array' and replaces it with a copy of rhs.

7.107.3.9 template<class T> bool PP_Array< T >::operator== (const PP_Array< T > & *rhs*) const

: Test for equality.

7.107.3.10 template<class T> const T & PP_Array< T >::operator[] (long *K*) const [inline]

: Same as above, except acts on const PP_Array's.

7.107.3.11 template<class T> T & PP_Array< T >::operator[] (long *K*) [inline]

: Returns a reference to the K'th element in this 'PP_Array'. The element can be modified through this reference. The result can be used as an L-value.

7.107.3.12 template<class T> bool PP_Array< T >::ready () const [inline]

: Returns true if this 'PP_Array' is not empty.

7.107.3.13 template<class T> void PP_Array< T >::reserve (long *_truesize*)

: Reserve space for future expansion of memory. You still must 'resize' to address the memory.

7.107.3.14 template<class T> void PP_Array< T >::resize (long *newlen*, const T & *initialvalue*) [inline]

: This function changes the size of this 'PP_Array' to newlen while preserving the value of as many array elements as possible. If newlen is greater than the current length, the array is grown and the new elements have the value given by initialvalue. If newlen is less than the current length the array is cropped with the remaining elements retaining their original values.

7.107.3.15 template<class T> void PP_Array< T >::resize (long *newlen*) [inline]

: This function changes the size of this 'PP_Array' to newlen while preserving the value of as many array elements as possible. If newlen is greater than the current length, the array is grown and the new elements have the value given by the default constructor for 'T'. If newlen is less than the current length the array is cropped with the remaining elements retaining their original values.

7.107.3.16 template<class T> void PP_Array< T >::set (long *i*, const T & *elem*) [inline]

: Changes the i'th element of this 'PP_Array' to elem.

7.107.3.17 template<class T> void PP_Array< T >::shrinkWrap ()

: Shrink allocated memory to be just enough for elements in PP_Array. This is useful if you allocated a lot of memory to avoid memory allocation delays as you add elements. Once all the elements are added, you can reduce memory usage to the minimum by calling this function.

7.107.3.18 template<class T> void PP_Array< T >::swap (long *i*, long *j*) [inline]

: This function swaps the i'th and j'th element of the PP_Array.

7.107.3.19 template<class T> long PP_Array< T >::trueSize () const [inline]

: Return the maximum number of elements the array can hold without doing a 'resize'.

7.107.4 Member Data Documentation

7.107.4.1 template<class T> long PP_Array< T >::nelem [protected]

7.107.4.2 template<class T> long PP_Array< T >::truesize [protected]

7.107.4.3 template<class T> T* PP_Array< T >::vp [protected]

The documentation for this class was generated from the following file:

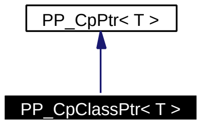
- **ParmParse.H**

7.108 PP_CpClassPtr< T > Class Template Reference

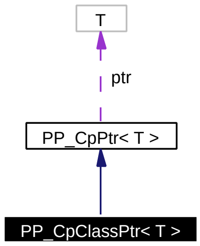
A Smart Pointer for User-Defined Types for **ParmParse**.

`#include <ParmParse.H>`

Inheritance diagram for PP_CpClassPtr< T >:



Collaboration diagram for PP_CpClassPtr< T >:



Public Member Functions

- **PP_CpClassPtr** ()

  : The default constructor. The wrapped pointer is null.

- **PP_CpClassPtr** (T ∗rhs)

  : Construct a PP_CpPtr<T> setting the wrapped pointer to rhs.

- **PP_CpClassPtr** (const **PP_CpClassPtr**< T > &rhs)
- **PP_CpClassPtr**< T > & **operator=** (T ∗rhs)
- **PP_CpClassPtr**< T > & **operator=** (const **PP_CpClassPtr**< T > &rhs)
- T ∗ **operator** → () const

  : Applies operator-> to the wrapped pointer.

7.108.1 Detailed Description

template<class T> class PP_CpClassPtr< T >

A Smart Pointer for User-Defined Types for **ParmParse**.

The template class PP_CpClassPtr<T> is derived from PP_CpPtr<T>. It provides a simple wrapper around a pointer to type T (a T∗) that "does the right thing" when copied from one PP_CpPtr<T> to another. The type T MUST be a user-defined type, not an intrinsic type. Given this restriction, we can supply an **operator → ()**.

This is a convenience class for **ParmParse** and will not be in any way supported by anyone at ANAG.

7.108.2 Constructor & Destructor Documentation

7.108.2.1 template<class T> PP_CpClassPtr< T >::PP_CpClassPtr () [inline]

: The default constructor. The wrapped pointer is null.

7.108.2.2 template<class T> PP_CpClassPtr< T >::PP_CpClassPtr (T * *rhs*) [inline]

: Construct a PP_CpPtr<T> setting the wrapped pointer to rhs.

7.108.2.3 template<class T> PP_CpClassPtr< T >::PP_CpClassPtr (const PP_CpClassPtr< T > & *rhs*) [inline]

: The copy constructor. If the pointer wrapped by rhs is null, the wrapped pointer is null here as well. Otherwise, the contained pointer here is set to a new'd copy of that wrapped by rhs, with the two pointed-to values being identical. This assumes that type T has a well-defined and accessible copy constructor. T must also be a concrete type, not a abstract type.

7.108.3 Member Function Documentation

7.108.3.1 template<class T> T * PP_CpClassPtr< T >::operator → () const [inline]

: Applies operator-> to the wrapped pointer.

7.108.3.2 template<class T> PP_CpClassPtr< T > & PP_CpClassPtr< T >::operator= (const PP_CpClassPtr< T > & *rhs*) [inline]

: The copy assignment operator. If the pointer wrapped by rhs is null, the wrapped pointer is null here as well. Otherwise, the contained pointer here is set to a new'd copy of that wrapped by rhs, with the two pointed-to values being identical. This assumes that type T has a well-defined and accessible copy constructor. T must also be a concrete type, not a abstract type.

7.108.3.3 template<class T> PP_CpClassPtr< T > & PP_CpClassPtr< T >::operator= (T * *rhs*) [inline]

: Sets the wrapped pointer to rhs. Deletes the previously wrapped pointer.

Reimplemented from **PP_CpPtr< T >** (p. 527).

The documentation for this class was generated from the following file:

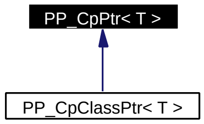
- **ParmParse.H**

7.109 PP_CpPtr< T > Class Template Reference

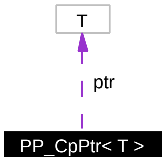
: A Smart Pointer for Intrinsic or User-Defined Types for **ParmParse**

`#include <ParmParse.H>`

Inheritance diagram for PP_CpPtr< T >:



Collaboration diagram for PP_CpPtr< T >:



Public Member Functions

- **PP_CpPtr** ()

  : The default constructor. The wrapped pointer is null.

- **PP_CpPtr** (T ∗rhs)

  : Construct a PP_CpPtr<T> setting the wrapped pointer to rhs.

- ~**PP_CpPtr** ()

  : The destructor. Deletes the wrapped pointer.

- **PP_CpPtr** (const **PP_CpPtr**< T > &rhs)
- **PP_CpPtr**< T > & **operator=** (T ∗rhs)
- **PP_CpPtr**< T > & **operator=** (const **PP_CpPtr**< T > &rhs)
- T & **operator** ∗ () const
- bool **isNull** () const

  : Returns true if the wrapped pointer null.

- T ∗ **release** ()

  : Sets the wrapped pointer to null and returns the previous value.

- bool **operator==** (const **PP_CpPtr**< T > &rhs) const

  : Are the two pointers (not the values to which they point) equal?

- bool **operator!=** (const **PP_CpPtr**< T > &rhs) const

  : Are the two pointers not equal?

Protected Attributes

- T * **ptr**

7.109.1 Detailed Description

template<class T> class PP_CpPtr< T >

: A Smart Pointer for Intrinsic or User-Defined Types for **ParmParse**

The template class PP_CpPtr<T> provides a simple wrapper around a pointer to type T (T*) that builds a copy of the pointed-to object when copied from one PP_CpPtr<T> to another. This is in contrast to a reference-counted pointer class that would maintain one pointed-to object with a reference count indicating the number of references. Hence we call this a "copied" smart pointer class. It is intended for use with any type type T, including the intrinsic types. This class does not supply an operator $\rightarrow$ (), as such an operator on intrinsic types has only recently become a part of the C++ language, and many compilers do not yet implement it.

This is a convenience class for **ParmParse** and will not be in any way supported by anyone at ANAG.

7.109.2 Constructor & Destructor Documentation

7.109.2.1 template<class T> PP_CpPtr< T >::PP_CpPtr () `[inline]`

: The default constructor. The wrapped pointer is null.

7.109.2.2 template<class T> PP_CpPtr< T >::PP_CpPtr (T * *rhs*) `[inline]`

: Construct a PP_CpPtr<T> setting the wrapped pointer to rhs.

7.109.2.3 template<class T> PP_CpPtr< T >::~PP_CpPtr () `[inline]`

: The destructor. Deletes the wrapped pointer.

7.109.2.4 template<class T> PP_CpPtr< T >::PP_CpPtr (const PP_CpPtr< T > & *rhs*) `[inline]`

: The copy constructor. If the pointer wrapped by rhs is null, the wrapped pointer is null here as well. Otherwise, the contained pointer here is set to a new'd copy of that wrapped by rhs, with the two pointed-to values being identical. This assumes that type T has a well-defined and accessible copy constructor. T must also be a concrete type, not a abstract type.

7.109.3 Member Function Documentation

7.109.3.1 template<class T> bool PP_CpPtr< T >::isNull () const `[inline]`

: Returns true if the wrapped pointer null.

7.109.3.2 template<class T> T & PP_CpPtr< T >::operator * () const [inline]

: Returns a reference to the value pointed to by the wrapped pointer; i.e. dereferencing this PP_CpPtr<T>, returns the dereferenced wrapped pointer. It is an error if the wrapped pointer is null.

7.109.3.3 template<class T> bool PP_CpPtr< T >::operator!= (const PP_CpPtr< T > & *rhs*) const [inline]

: Are the two pointers not equal?

7.109.3.4 template<class T> PP_CpPtr< T > & PP_CpPtr< T >::operator= (const PP_CpPtr< T > & *rhs*) [inline]

: The copy assignment operator. If the pointer wrapped by rhs is null, the wrapped pointer is null here as well. Otherwise, the contained pointer here is set to a new'd copy of that wrapped by rhs, with the two pointed-to values being identical. This assumes that type T has a well-defined and accessible copy constructor. T must also be a concrete type, not a abstract type.

7.109.3.5 template<class T> PP_CpPtr< T > & PP_CpPtr< T >::operator= (T * *rhs*) [inline]

: Sets the wrapped pointer to rhs. Deletes the previously wrapped pointer.

Reimplemented in **PP_CpClassPtr< T >** (p. 524).

7.109.3.6 template<class T> bool PP_CpPtr< T >::operator== (const PP_CpPtr< T > & *rhs*) const [inline]

: Are the two pointers (not the values to which they point) equal?

7.109.3.7 template<class T> T * PP_CpPtr< T >::release () [inline]

: Sets the wrapped pointer to null and returns the previous value.

7.109.4 Member Data Documentation

7.109.4.1 template<class T> T* PP_CpPtr< T >::ptr [protected]

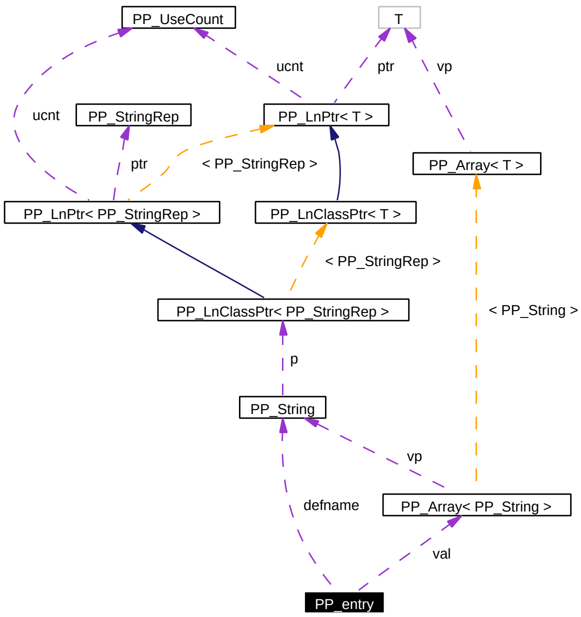
The documentation for this class was generated from the following file:

- **ParmParse.H**

7.110 PP_entry Class Reference

`#include <ParmParse.H>`

Collaboration diagram for PP_entry:



Friends

- class **ParmParse**

7.110.1 Friends And Related Function Documentation

7.110.1.1 friend class ParmParse [friend]

The documentation for this class was generated from the following file:

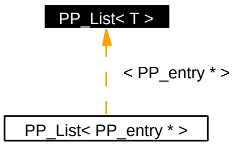
- **ParmParse.H**

7.111 PP_List< T > Class Template Reference

A Doubly-Linked **List** for **ParmParse**.

`#include <ParmParse.H>`

Inheritance diagram for PP_List< T >:



Public Member Functions

- **PP_List** ()
  : Construct an empty PP_List<T>.
- **PP_List** (const **PP_List**< T > &rhs)
  : The copy constructor.
- **PP_List**< T > & **operator=** (const **PP_List**< T > &rhs)
  : The assignment operator.
- **~PP_List** ()
  : The destructor.
- void **prepend** (const T &value)
  : Adds a copy of the value to the beginning of the PP_List<T>.
- void **append** (const T &value)
  : Adds a copy of the value to the end of the PP_List<T>.
- void **add** (const T &value)
  : Adds a copy of the value to the end of the PP_List<T>.
- void **join** (const **PP_List**< T > &src)
  : Appends a copy of all items in PP_List<T> src to this PP_List<T>.
- void **catenate** (**PP_List**< T > &src)
- void **clear** ()
  : Removes all objects from the PP_List<T>.
- T & **firstElement** () const
  : Returns a reference to the first element in the PP_List<T>.
- T & **lastElement** () const
  : Returns a reference to the last element in the PP_List<T>.

- bool **includes** (const T &value) const
- bool **operator==** (const **PP_List**< T > &rhs) const
- bool **operator!=** (const **PP_List**< T > &rhs) const

  : Returns true if the this and rhs are not equal.

- bool **isEmpty** () const

  : Returns true if the PP_List<T> is empty.

- bool **isNotEmpty** () const

  : Returns true if the PP_List<T> is not empty.

- int **length** () const

  : Returns the number of objects in the PP_List<T>.

- void **removeFirst** ()

  : Removes the first element in the PP_List<T>.

- void **removeLast** ()

  : Removes the last element in the PP_List<T>.

- const T & **operator**[ ] (const **PP_ListIterator**< T > &li) const

  : Returns reference to object pointed to by the PP_ListIterator<T>.

- T & **operator**[ ] (const **PP_ListIterator**< T > &li)

  : Returns reference to object pointed to by the PP_ListIterator<T>.

- void **remove** (const T &value)

  : Removes all objects in the PP_List<T> equal to value.

- void **remove** (const **PP_List**< T > &lst)
- void **remove** (**PP_ListIterator**< T > &lit)

  : Removes the object pointed to by the PP_ListIterator<T>.

- void **replace** (**PP_ListIterator**< T > &li, const T &val)

  : Replace the value pointed to by the PP_ListIterator<T> by val.

- void **addAfter** (**PP_ListIterator**< T > &lit, const T &val)
- void **addBefore** (**PP_ListIterator**< T > &lit, const T &val)
- **PP_ListIterator**< T > **first** () const

  : Returns a PP_ListIterator<T> to the first object in this PP_List<T>.

- **PP_ListIterator**< T > **last** () const

  : Returns a PP_ListIterator<T> to the last object in this PP_List<T>.

Protected Member Functions

- void **remove** (**PP_ListLink**< T > *ln)
- **PP_ListLink**< T > * **addBefore** (**PP_ListLink**< T > *ln, const T &val)
- **PP_ListLink**< T > * **addAfter** (**PP_ListLink**< T > *ln, const T &val)

Protected Attributes

- **PP_ListLink**< T > * **head**
- **PP_ListLink**< T > * **tail**

Friends

- class **PP_ListIterator**< **T** >

7.111.1 Detailed Description

template<class T> class PP_List< T >

A Doubly-Linked **List** for **ParmParse**.

The PP_List<T> class is a template class that implements a doubly-linked list of objects. A PP_List<T> is a useful container class when the number of objects in the collection is not known ahead of time. A PP_List<T> can contain an arbitrary number of elements; operations such as insertion, deletion, and catenation are easily implemented and inexpensive.

The only difficulty when defining a list class is devising a mechanism to access the elements. In an array, an element is accessed using an integer index. Since the elements in a PP_List<T> are ordered by position, we could define an integer indexing operation that walks along the PP_-List<T> links from the beginning until the numbered element is found. Unfortunately, this would be very inefficient when accessing elements near the end of a long list. Another solution is to allow user access to the individual link objects that contain the element as well as the forward and backward pointers. This is not a satisfactory solution since it allows user access to the internal representation of the class. The solution chosen is to define a PP_ListIterator<T> template class.

Think of a PP_ListIterator<T> as a pointer to an object in the PP_List<T>. You can access the element currently pointed to by the iterator, move the iterator forward and backward through the PP_List<T>, and use it as a mechanism to define where elements should be inserted and deleted. If the iterator is moved off the end of the list it behaves as a null pointer.

This is a concrete class, not a polymorphic one.

This is a convenience class for **ParmParse** and will not be in any way supported by anyone at ANAG.

7.111.2 Constructor & Destructor Documentation

7.111.2.1 template<class T> PP_List< T >::PP_List () `[inline]`

: Construct an empty PP_List<T>.

7.111.2.2 template<class T> PP_List< T >::PP_List (const PP_List< T > & *rhs*) `[inline]`

: The copy constructor.

7.111.2.3 template<class T> PP_List< T >::~PP_List () `[inline]`

: The destructor.

7.111.3 Member Function Documentation

7.111.3.1 template<class T> void PP_List< T >::add (const T & *value*) [inline]

: Adds a copy of the value to the end of the PP_List<T>.

7.111.3.2 template<class T> PP_ListLink< T > * PP_List< T >::addAfter (PP_ListLink< T > * *ln*, const T & *val*) [inline, protected]

7.111.3.3 template<class T> void PP_List< T >::addAfter (PP_ListIterator< T > & *lit*, const T & *val*) [inline]

: Insert val into PP_List<T> after the object pointed to by PP_ListIterator<T>.

7.111.3.4 template<class T> PP_ListLink< T > * PP_List< T >::addBefore (PP_ListLink< T > * *ln*, const T & *val*) [inline, protected]

7.111.3.5 template<class T> void PP_List< T >::addBefore (PP_ListIterator< T > & *lit*, const T & *val*) [inline]

: Insert val into PP_List<T> before the object pointed to by PP_ListIterator<T>.

7.111.3.6 template<class T> void PP_List< T >::append (const T & *value*) [inline]

: Adds a copy of the value to the end of the PP_List<T>.

7.111.3.7 template<class T> void PP_List< T >::catenate (PP_List< T > & *src*) [inline]

: Appends a copy of all items in PP_List<T> src to this PP_List<T>. This differs from **join()** in that it unlinks the objects from the PP_List<T> src and glues them to the end of this PP_List<T>, leaving PP_List<T> src empty. This is more efficient that **join()** if src is no longer needed.

7.111.3.8 template<class T> void PP_List< T >::clear () [inline]

: Removes all objects from the PP_List<T>.

7.111.3.9 template<class T> PP_ListIterator< T > PP_List< T >::first () const [inline]

: Returns a PP_ListIterator<T> to the first object in this PP_List<T>.

7.111.3.10 template<class T> T & PP_List< T >::firstElement () const [inline]

: Returns a reference to the first element in the PP_List<T>.

7.111.3.11 template<class T> bool PP_List< T >::includes (const T & *value*) const [inline]

: Returns true if the PP_List<T> contains an object identical to value. Type T must have an **operator==()** defined, or be an intrinsic type.

7.111.3.12 template<class T> bool PP_List< T >::isEmpty () const [inline]

: Returns true if the PP_List<T> is empty.

7.111.3.13 template<class T> bool PP_List< T >::isNotEmpty () const [inline]

: Returns true if the PP_List<T> is not empty.

7.111.3.14 template<class T> void PP_List< T >::join (const PP_List< T > & *src*) [inline]

: Appends a copy of all items in PP_List<T> src to this PP_List<T>.

7.111.3.15 template<class T> PP_ListIterator< T > PP_List< T >::last () const [inline]

: Returns a PP_ListIterator<T> to the last object in this PP_List<T>.

7.111.3.16 template<class T> T & PP_List< T >::lastElement () const [inline]

: Returns a reference to the last element in the PP_List<T>.

7.111.3.17 template<class T> int PP_List< T >::length () const [inline]

: Returns the number of objects in the PP_List<T>.

7.111.3.18 template<class T> bool PP_List< T >::operator!= (const PP_List< T > & *rhs*) const [inline]

: Returns true if the this and rhs are not equal.

7.111.3.19 template<class T> PP_List< T > & PP_List< T >::operator= (const PP_List< T > & *rhs*) [inline]

: The assignment operator.

7.111.3.20 template<class T> bool PP_List< T >::operator== (const PP_List< T > & *rhs*) const [inline]

: Returns true if the this and rhs are memberwise equal; i.e. the two lists are the same size and each of the elements in the list compare equal. Type T must have an **operator==()** defined, or be an intrinsic type.

7.111.3.21 template<class T> T & PP_List< T >::operator[] (const PP_ListIterator< T > & *li*) [inline]

: Returns reference to object pointed to by the PP_ListIterator<T>.

7.111.3.22 template<class T> const T & PP_List< T >::operator[] (const PP_ListIterator< T > & *li*) const [inline]

: Returns reference to object pointed to by the PP_ListIterator<T>.

7.111.3.23 template<class T> void PP_List< T >::prepend (const T & *value*) [inline]

: Adds a copy of the value to the beginning of the PP_List<T>.

7.111.3.24 template<class T> void PP_List< T >::remove (PP_ListLink< T > * *ln*) [inline, protected]

7.111.3.25 template<class T> void PP_List< T >::remove (PP_ListIterator< T > & *lit*) [inline]

: Removes the object pointed to by the PP_ListIterator<T>.

7.111.3.26 template<class T> void PP_List< T >::remove (const PP_List< T > & *lst*) [inline]

: Removes all objects in the PP_List<T> equal to any of the values in lst.

7.111.3.27 template<class T> void PP_List< T >::remove (const T & *value*) [inline]

: Removes all objects in the PP_List<T> equal to value.

7.111.3.28 template<class T> void PP_List< T >::removeFirst () [inline]

: Removes the first element in the PP_List<T>.

7.111.3.29 template<class T> void PP_List< T >::removeLast () [inline]

: Removes the last element in the PP_List<T>.

7.111.3.30 template<class T> void PP_List< T >::replace (PP_ListIterator< T > & *li*, const T & *val*) [inline]

: Replace the value pointed to by the PP_ListIterator<T> by val.

7.111.4 Friends And Related Function Documentation

7.111.4.1 template<class T> friend class PP_ListIterator< T > [friend]

7.111.5 Member Data Documentation

7.111.5.1 template<class T> PP_ListLink<T>* PP_List< T >::head [protected]

7.111.5.2 template<class T> PP_ListLink<T>* PP_List< T >::tail [protected]

The documentation for this class was generated from the following file:

- **ParmParse.H**

7.112 PP_ListIterator< T > Class Template Reference

Iterate over a **List** for **ParmParse**.

`#include <ParmParse.H>`

Public Member Functions

- **PP_ListIterator** (const **PP_List**< T > &aList)

  : Construct a PP_ListIterator<T> to first element of aList.

- **PP_ListIterator** (const **PP_ListIterator**< T > &rhs)

  : The copy constructor.

- void **rewind** ()
- const T & **operator()** () const
- const T & **operator** * () const
- **operator void** * ()
- bool **operator!** () const
- const T & **value** () const
- **PP_ListIterator**< T > & **operator++** ()
- **PP_ListIterator**< T > & **operator--** ()
- **PP_ListIterator**< T > **operator--** (int)
- **PP_ListIterator**< T > **operator++** (int)
- bool **operator==** (const **PP_ListIterator**< T > &) const
- bool **operator!=** (const **PP_ListIterator**< T > &) const

  : Are the PP_ListIterator<T>s not equal?

Protected Member Functions

- **PP_ListIterator** (const **PP_List**< T > &_list, **PP_ListLink**< T > *_p)

Protected Attributes

- const **PP_List**< T > & **list**
- **PP_ListLink**< T > * **p**

Friends

- class **PP_List**< **T** >

7.112.1 Detailed Description

template<class T> class PP_ListIterator< T >

Iterate over a **List** for **ParmParse**.

The class PP_ListIterator<T> is an iterator over class PP_List<T>.

This class does NOT provide a default constructor or an assignment operator.

This is a convenience class for **ParmParse** and will not be in any way supported by anyone at ANAG.

7.112.2 Constructor & Destructor Documentation

7.112.2.1 template<class T> PP_ListIterator< T >::PP_ListIterator (const PP_List< T > & *aList*) [inline]

: Construct a PP_ListIterator<T> to first element of aList.

7.112.2.2 template<class T> PP_ListIterator< T >::PP_ListIterator (const PP_ListIterator< T > & *rhs*) [inline]

: The copy constructor.

7.112.2.3 template<class T> PP_ListIterator< T >::PP_ListIterator (const PP_List< T > & *_list*, PP_ListLink< T > * *_p*) [inline, protected]

7.112.3 Member Function Documentation

7.112.3.1 template<class T> const T & PP_ListIterator< T >::operator * () const [inline]

: Return a constant reference to the object in the PP_List<T> currently pointed to by this PP_ListIterator<T>.

7.112.3.2 template<class T> PP_ListIterator< T >::operator void * () [inline]

: This is a conversion operator that makes the iterator look like a pointer. This operator makes it easy to check if the iterator is pointing to an element on the PP_List<T>. If the iterator has been moved off the PP_List<T> or if the PP_List<T> is empty, this conversion returns the NULL pointer.

7.112.3.3 template<class T> bool PP_ListIterator< T >::operator! () const [inline]

: Returns true if PP_ListIterator<T> doesn't point to any element on the PP_List<T>.

7.112.3.4 template<class T> bool PP_ListIterator< T >::operator!= (const PP_ListIterator< T > &) const [inline]

: Are the PP_ListIterator<T>s not equal?

7.112.3.5 template<class T> const T & PP_ListIterator< T >::operator() () const [inline]

: Return a constant reference to the object in the PP_List<T> currently pointed to by this PP_ListIterator<T>.

7.112.3.6 template<class T> PP_ListIterator< T > PP_ListIterator< T >::operator++ (int) [inline]

: This is the postfix auto-increment operator. It advances the PP_ListIterator<T> to point to the next element on the PP_List<T>. It then returns a PP_ListIterator<T> that points to the old element to allow for chaining with other operators.

7.112.3.7 template<class T> PP_ListIterator< T > & PP_ListIterator< T >::operator++ () [inline]

: This is the prefix auto-increment operator. It advances the PP_ListIterator<T> to point to the next element on the PP_List<T>. It then returns a reference to itself to allow for chaining with other operators.

7.112.3.8 template<class T> PP_ListIterator< T > PP_ListIterator< T >::operator– (int) [inline]

: This is the postfix auto-decrement operator. It moves the PP_ListIterator<T> to point to the previous element on the PP_List<T>. It then returns a PP_ListIterator<T> that points to the old element to allow for chaining with other operators.

7.112.3.9 template<class T> PP_ListIterator< T > & PP_ListIterator< T >::operator– () [inline]

: This is the prefix auto-decrement operator. It moves the PP_ListIterator<T> to point to the previous element on the PP_List<T>. It then returns a reference to itself to allow for chaining with other operators.

7.112.3.10 template<class T> bool PP_ListIterator< T >::operator== (const PP_ListIterator< T > &) const [inline]

: Do the two PP_ListIterator<T>s point to the same PP_List<T> and the same element within the PP_List<T>?

7.112.3.11 template<class T> void PP_ListIterator< T >::rewind () [inline]

: Reset this PP_ListIterator<T> to point to the first element in the PP_List<T>.

7.112.3.12 template<class T> const T & PP_ListIterator< T >::value () const [inline]

: Return a constant reference to the object in the PP_List<T> currently pointed to by the iterator.

7.112.4 Friends And Related Function Documentation

7.112.4.1 template<class T> friend class PP_List< T > [friend]

7.112.5 Member Data Documentation

7.112.5.1 template<class T> const PP_List<T>& PP_ListIterator< T >::list [protected]

7.112.5.2 template<class T> PP_ListLink<T>* PP_ListIterator< T >::p [protected]

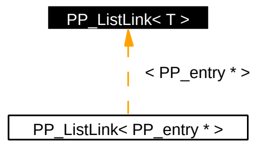
The documentation for this class was generated from the following file:

- **ParmParse.H**

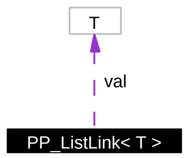
7.113 PP_ListLink< T > Class Template Reference

`#include <ParmParse.H>`

Inheritance diagram for PP_ListLink< T >:



Collaboration diagram for PP_ListLink< T >:



Friends

- class **PP_List< T >**
- class **PP_ListIterator< T >**

template<class T> class PP_ListLink< T >

7.113.1 Friends And Related Function Documentation

7.113.1.1 template<class T> friend class PP_List< T > `[friend]`

7.113.1.2 template<class T> friend class PP_ListIterator< T > `[friend]`

The documentation for this class was generated from the following file:

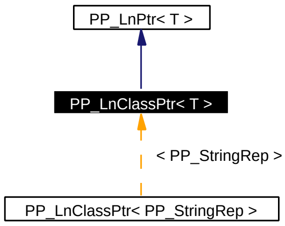
- **ParmParse.H**

7.114 PP_LnClassPtr< T > Class Template Reference

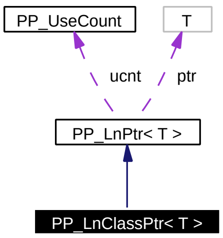
A Smart Reference Counted Pointer for User-Defined Types for **ParmParse**.

`#include <ParmParse.H>`

Inheritance diagram for PP_LnClassPtr< T >:



Collaboration diagram for PP_LnClassPtr< T >:



Public Member Functions

- **PP_LnClassPtr** ()

  : The default constructor. The wrapped pointer is null.

- **PP_LnClassPtr** (T ∗rhs)

  : Construct a PP_LnPtr<T> setting the wrapped pointer to rhs.

- **PP_LnClassPtr**< T > & **operator=** (const **PP_LnClassPtr**< T > &rhs)
- **PP_LnClassPtr**< T > & **operator=** (T ∗rhs)
- T ∗ **operator** → () const

  : Applies operator-> to the wrapped pointer.

7.114.1 Detailed Description

template<class T> class PP_LnClassPtr< T >

A Smart Reference Counted Pointer for User-Defined Types for **ParmParse**.

The template class PP_LnClassPtr<T> is derived from PP_LnPtr<T>. It provides a reference counted wrapper around a pointer to type T (a T∗). The type T MUST be a user-defined type, not an intrinsic type. Given this restriction, we can supply an **operator** → **()**.

This is a convenience class for **ParmParse** and will not be in any way supported by anyone at ANAG.

7.114.2 Constructor & Destructor Documentation

7.114.2.1 template<class T> PP_LnClassPtr< T >::PP_LnClassPtr () [inline]

: The default constructor. The wrapped pointer is null.

7.114.2.2 template<class T> PP_LnClassPtr< T >::PP_LnClassPtr (T * *rhs*) [inline]

: Construct a PP_LnPtr<T> setting the wrapped pointer to rhs.

7.114.3 Member Function Documentation

7.114.3.1 template<class T> T * PP_LnClassPtr< T >::operator → () const [inline]

: Applies operator-> to the wrapped pointer.

7.114.3.2 template<class T> PP_LnClassPtr< T > & PP_LnClassPtr< T >::operator= (T * *rhs*) [inline]

: Sets the wrapped pointer to rhs. Decrements the count on the previously wrapped pointer and deletes it if there was only one reference.

Reimplemented from **PP_LnPtr< T >** (p. 545).

7.114.3.3 template<class T> PP_LnClassPtr< T > & PP_LnClassPtr< T >::operator= (const PP_LnClassPtr< T > & *rhs*) [inline]

: The copy assignment operator. The contained pointer is set to the one wrapped by rhs. The reference count is decremented on this object and the reference count is incremented for the newly wrapped pointer.

The documentation for this class was generated from the following file:

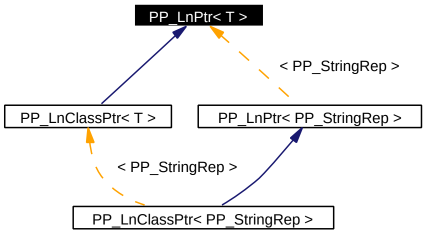
- **ParmParse.H**

7.115 PP_LnPtr< T > Class Template Reference

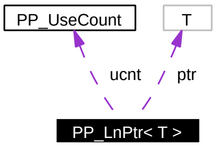
A Reference Counted Smart Pointer for Intrinsic or User-Defined Types for **ParmParse**.

`#include <ParmParse.H>`

Inheritance diagram for PP_LnPtr< T >:



Collaboration diagram for PP_LnPtr< T >:



Public Member Functions

- **PP_LnPtr** ()

  : The default constructor. The wrapped pointer is null.

- **PP_LnPtr** (T ∗rhs)

  : Construct a PP_LnPtr<T> setting the wrapped pointer to rhs.

- **PP_LnPtr**< T > & **operator=** (const **PP_LnPtr**< T > &rhs)
- **PP_LnPtr**< T > & **operator=** (T ∗rhs)
- ∼**PP_LnPtr** ()
- bool **unique** () const

  : Returns true if only one reference to the wrapped pointer.

- int **linkCount** () const

  : Returns the number of references to the wrapped pointer.

- T & **operator** ∗ () const
- bool **isNull** () const

  : Returns true if the wrapped pointer is null.

- bool **operator==** (const **PP_LnPtr**< T > &rhs) const

  : Are the two pointers (not the values to which they point) equal?

- bool **operator!=** (const **PP_LnPtr**< T > &rhs) const

  : Are the two pointers not equal?

Protected Attributes

- T * **ptr**

7.115.1 Detailed Description

template<class T> class PP_LnPtr< T >

A Reference Counted Smart Pointer for Intrinsic or User-Defined Types for **ParmParse**.

The template class PP_LnPtr<T> provides a reference counted wrapper around a pointer to type T (a T*). This "smart" pointer is intended for use with any type type T, including the intrinsic types. For this reason, we do not supply an operator → (), as such an operator on intrinsic types has only recently become a part of the C++ language and many compilers do not yet implement it.

This is a convenience class for **ParmParse** and will not be in any way supported by anyone at ANAG.

7.115.2 Constructor & Destructor Documentation

7.115.2.1 template<class T> PP_LnPtr< T >::PP_LnPtr () `[inline]`

: The default constructor. The wrapped pointer is null.

7.115.2.2 template<class T> PP_LnPtr< T >::PP_LnPtr (T * *rhs*) `[inline]`

: Construct a PP_LnPtr<T> setting the wrapped pointer to rhs.

7.115.2.3 template<class T> PP_LnPtr< T >::~PP_LnPtr () `[inline]`

: The destructor – decrements the reference count and deletes the wrapped pointer if there is only one reference.

7.115.3 Member Function Documentation

7.115.3.1 template<class T> bool PP_LnPtr< T >::isNull () const `[inline]`

: Returns true if the wrapped pointer is null.

7.115.3.2 template<class T> int PP_LnPtr< T >::linkCount () const `[inline]`

: Returns the number of references to the wrapped pointer.

7.115.3.3 template<class T> T & PP_LnPtr< T >::operator * () const [inline]

: Returns a reference to the value pointed to by the wrapped pointer; i.e. dereferencing this PP_LnPtr<T>, returns the dereferenced wrapped pointer. It is an error if the wrapped pointer is null.

7.115.3.4 template<class T> bool PP_LnPtr< T >::operator!= (const PP_LnPtr< T > & *rhs*) const [inline]

: Are the two pointers not equal?

7.115.3.5 template<class T> PP_LnPtr<T>& PP_LnPtr< T >::operator= (T * *rhs*) [inline]

: Sets the wrapped pointer to rhs. Decrements the count on the previously wrapped pointer and deletes it if there was only one reference.

Reimplemented in **PP_LnClassPtr< T >** (p. 542), and **PP_LnClassPtr< PP_StringRep >** (p. 542).

7.115.3.6 template<class T> PP_LnPtr< T > & PP_LnPtr< T >::operator= (const PP_LnPtr< T > & *rhs*) [inline]

: The copy assignment operator. The contained pointer is set to the one wrapped by rhs. The reference count is decremented on this object and the reference count is incremented for the newly wrapped pointer.

7.115.3.7 template<class T> bool PP_LnPtr< T >::operator== (const PP_LnPtr< T > & *rhs*) const [inline]

: Are the two pointers (not the values to which they point) equal?

7.115.3.8 template<class T> bool PP_LnPtr< T >::unique () const [inline]

: Returns true if only one reference to the wrapped pointer.

7.115.4 Member Data Documentation

7.115.4.1 template<class T> T* PP_LnPtr< T >::ptr [protected]

The documentation for this class was generated from the following file:

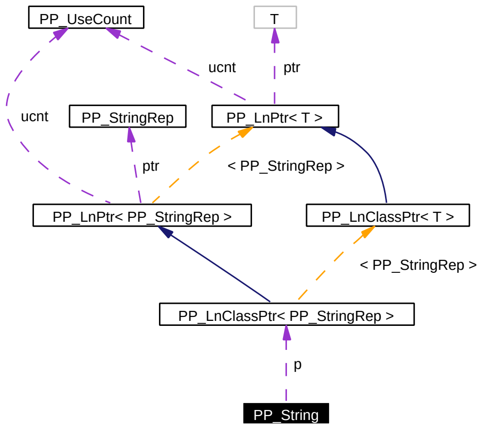
- **ParmParse.H**

7.116 PP_String Class Reference

A String Class for **ParmParse**.

`#include <ParmParse.H>`

Collaboration diagram for PP_String:



Public Member Functions

- **PP_String** ()

    : Constructs an empty string.

- **PP_String** (char c)
- **PP_String** (int len)
- **PP_String** (const char ∗s)
- **PP_String** (const **PP_String** &rhs)

    : The copy constructor.

- **PP_String** & **operator=** (const **PP_String** &rhs)

    : The assignment operator

- **PP_String** & **operator+=** (const **PP_String** &right)
- **PP_String** & **operator+=** (const char ∗right)
- **PP_String** & **operator+=** (char c)
- **PP_String** & **toUpper** ()
- **PP_String** & **toLower** ()
- std::istream & **getline** (std::istream &strm)
- int **length** () const
- bool **isNull** () const

    : Returns true if this is the null string.

- char & **operator[]** (int k)
- char **operator[]** (int k) const
- const char ∗ **c_str** () const
- double **toDouble** () const

- int **toInteger** () const
- long **toLong** () const

Protected Member Functions

- void **copyModify** ()

Friends

- std::ostream & **operator<<** (std::ostream &os, const **PP_String** &str)

  : Write to an ostream in ASCII format.

- std::istream & **operator>>** (std::istream &is, **PP_String** &str)
- bool **operator<** (const **PP_String** &left, const **PP_String** &right)
- bool **operator<=** (const **PP_String** &left, const **PP_String** &right)
- bool **operator!=** (const **PP_String** &left, const **PP_String** &right)
- bool **operator==** (const **PP_String** &left, const **PP_String** &right)
- bool **operator>=** (const **PP_String** &left, const **PP_String** &right)
- bool **operator>** (const **PP_String** &left, const **PP_String** &right)
- bool **operator<** (const **PP_String** &left, const char *right)
- bool **operator<=** (const **PP_String** &left, const char *right)
- bool **operator!=** (const **PP_String** &left, const char *right)
- bool **operator==** (const **PP_String** &left, const char *right)
- bool **operator>=** (const **PP_String** &left, const char *right)
- bool **operator>** (const **PP_String** &left, const char *right)
- bool **operator<** (const char *left, const **PP_String** &right)
- bool **operator<=** (const char *left, const **PP_String** &right)
- bool **operator!=** (const char *left, const **PP_String** &right)
- bool **operator==** (const char *left, const **PP_String** &right)
- bool **operator>=** (const char *left, const **PP_String** &right)
- bool **operator>** (const char *left, const **PP_String** &right)

7.116.1 Detailed Description

A String Class for **ParmParse**.

The class PP_String is used to store and manipulate character strings. It has an efficient underlying storage mechanism and some useful string manipulation operations.

The PP_String class is implemented using a character array and reference count. Two PP_Strings may reference the same underlying character array with a reference count of two. When an PP_String copy constructor or copy operator is applied the reference count on the underlying character array is incremented but the actual string is not copied. That is, copying an PP_String is an inexpensive operation. When an PP_String is destructed, the reference count is decremented. The underlying character array is deleted only when the reference count goes to zero. Any operator that modifies an PP_String will make its own copy of the character array before the modification, unless it's the sole owner of the character array in the PP_String.

This is a convenience class for **ParmParse** and will not be in any way supported by anyone at ANAG.

7.116.2 Constructor & Destructor Documentation

7.116.2.1 PP_String::PP_String ()

: Constructs an empty string.

7.116.2.2 PP_String::PP_String (char *c*)

: Constructs an PP_String containing the single character c. If c is the null character then this will be the empty string.

7.116.2.3 PP_String::PP_String (int *len*)

: Constructs an empty PP_String but allocates enough space to hold a character string of length len. This may be useful when reading in very long lines with the 'getline' function; i.e. it may be slightly more efficient to allocate a very large PP_String once and then use 'getline' than to use 'getline' on an empty string. In general though, you won't notice any difference and you really shouldn't use this constructor.

7.116.2.4 PP_String::PP_String (const char * *s*)

: Constructs an PP_String initialized to the character string s. It is an error is 's' is the null string.

7.116.2.5 PP_String::PP_String (const PP_String & *rhs*)

: The copy constructor.

7.116.3 Member Function Documentation

7.116.3.1 const char * PP_String::c_str () const `[inline]`

: Convert an PP_String to a const char *. This allows an PP_String to be used in any context where a const char* type is needed.

7.116.3.2 void PP_String::copyModify () `[protected]`

7.116.3.3 std::istream& PP_String::getline (std::istream & *strm*)

: Read the next line from the input stream strm and store it as the value of the string. The delimiter for a line of text is the newline character. The newline is extracted from the istream, but it is NOT stored in the PP_String. There is no limit to the length of string that can be extracted.

7.116.3.4 bool PP_String::isNull () const `[inline]`

: Returns true if this is the null string.

7.116.3.5 int PP_String::length () const [inline]

: Returns the number of characters stored in this PP_String. This does not include the terminating null character.

7.116.3.6 PP_String& PP_String::operator+= (char *c*)

: Catenate character c onto end of this PP_String. Returns a reference to self to allow for operator chaining. This does nothing if c is the null character.

7.116.3.7 PP_String& PP_String::operator+= (const char * *right*)

: Catenate character string right onto end of this PP_String. Returns a reference to self to allow for operator chaining. It is an error is 'right' is the null string.

7.116.3.8 PP_String& PP_String::operator+= (const PP_String & *right*)

: Catenate PP_String right onto end of this PP_String. Return a reference to self to allow for operator chaining.

7.116.3.9 PP_String& PP_String::operator= (const PP_String & *rhs*)

: The assignment operator

7.116.3.10 char PP_String::operator[] (int *k*) const [inline]

: Returns kth character in the string. An error occurs if k < 0 or k >= **length()**.

7.116.3.11 char& PP_String::operator[] (int *k*)

: Returns a reference to the kth character in the string. An error occurs if k < 0 or k >= **length()**.

7.116.3.12 double PP_String::toDouble () const [inline]

: Returns the value of the string as a double. In the case when the string is empty or when the initial characters in the string are strictly alphabetic characters, this returns zero. It's equivalent to 'atof(**c_str()**)'.

7.116.3.13 int PP_String::toInteger () const [inline]

: Returns the value of the string as a integer. In the case when the string is empty or when the initial characters in the string are strictly alphabetic characters, this returns zero. It's equivalent to 'atoi(**c_str()**)'.

7.116.3.14 long PP_String::toLong () const [inline]

: Returns the value of the string as a long. In the case when the string is empty or when the initial characters in the string are strictly alphabetic characters, this returns zero. It's equivalent to 'atol(**c_str()**)'.

7.116.3.15 PP_String& PP_String::toLower ()

: Converts all characters in this PP_String to lower case. Returns a reference to self to allow for operator chaining.

7.116.3.16 PP_String& PP_String::toUpper ()

: Converts all characters in this PP_String to upper case. Returns a reference to self to allow for operator chaining.

7.116.4 Friends And Related Function Documentation

7.116.4.1 bool operator!= (const char ∗ *left*, const PP_String & *right*) [friend]

7.116.4.2 bool operator!= (const PP_String & *left*, const char ∗ *right*) [friend]

7.116.4.3 bool operator!= (const PP_String & *left*, const PP_String & *right*) [friend]

7.116.4.4 bool operator< (const char ∗ *left*, const PP_String & *right*) [friend]

7.116.4.5 bool operator< (const PP_String & *left*, const char ∗ *right*) [friend]

7.116.4.6 bool operator< (const PP_String & *left*, const PP_String & *right*) [friend]

7.116.4.7 std::ostream& operator<< (std::ostream & *os*, const PP_String & *str*) [friend]

: Write to an ostream in ASCII format.

7.116.4.8 bool operator<= (const char * *left*, const PP_String & *right*) [friend]

7.116.4.9 bool operator<= (const PP_String & *left*, const char * *right*) [friend]

7.116.4.10 bool operator<= (const PP_String & *left*, const PP_String & *right*) [friend]

7.116.4.11 bool operator== (const char * *left*, const PP_String & *right*) [friend]

7.116.4.12 bool operator== (const PP_String & *left*, const char * *right*) [friend]

7.116.4.13 bool operator== (const PP_String & *left*, const PP_String & *right*) [friend]

7.116.4.14 bool operator> (const char * *left*, const PP_String & *right*) [friend]

7.116.4.15 bool operator> (const PP_String & *left*, const char * *right*) [friend]

7.116.4.16 bool operator> (const PP_String & *left*, const PP_String & *right*) [friend]

7.116.4.17 bool operator>= (const char * *left*, const PP_String & *right*) [friend]

7.116.4.18 bool operator>= (const PP_String & *left*, const char * *right*) [friend]

7.116.4.19 bool operator>= (const PP_String & *left*, const PP_String & *right*) [friend]

7.116.4.20 std::istream& operator>> (std::istream & *is*, PP_String & *str*) [friend]

: Read a whitespace delimited string from an istream. This function discards leading whitespace and then reads in non-whitespace character until the next whitespace character or end-of-file. Note that there is no limit, on the length of the character that can be read in, except that dictated by the resources of the machine. Note also that operator>> and operator<< are not completely symmetrical in the case where operator<< writes out a string that contains whitespace. If you're trying to read in a string that contains whitespace, you might want to use **getline()** instead.

The documentation for this class was generated from the following file:

- **ParmParse.H**

7.117 PP_StringRep Class Reference

`#include <ParmParse.H>`

Public Member Functions

- **PP_StringRep** (int _len=0)
- **~PP_StringRep** ()
- void **resize** (int n)

Friends

- class **PP_String**

7.117.1 Constructor & Destructor Documentation

7.117.1.1 PP_StringRep::PP_StringRep (int *_len* = 0) [inline]

7.117.1.2 PP_StringRep::~PP_StringRep () [inline]

7.117.2 Member Function Documentation

7.117.2.1 void PP_StringRep::resize (int *n*)

7.117.3 Friends And Related Function Documentation

7.117.3.1 friend class PP_String [friend]

The documentation for this class was generated from the following file:

- **ParmParse.H**

7.118 PP_UseCount Class Reference

`#include <ParmParse.H>`

Public Member Functions

- **PP_UseCount** ()

  : Construct a PP_UseCount initialized to one.

- **PP_UseCount** (const **PP_UseCount** &rhs)

  : The copy constructor – bumps reference count.

- **PP_UseCount** & **operator=** (const **PP_UseCount** &rhs)
- **~PP_UseCount** ()

  : The destructor – decrements the reference count.

- bool **unique** () const

  : The PP_UseCount is unique if the reference count == 1.

- int **linkCount** () const

  : Returns the reference count.

7.118.1 Constructor & Destructor Documentation

7.118.1.1 PP_UseCount::PP_UseCount () `[inline]`

: Construct a PP_UseCount initialized to one.

7.118.1.2 PP_UseCount::PP_UseCount (const PP_UseCount & *rhs*) `[inline]`

: The copy constructor – bumps reference count.

7.118.1.3 PP_UseCount::~PP_UseCount () `[inline]`

: The destructor – decrements the reference count.

7.118.2 Member Function Documentation

7.118.2.1 int PP_UseCount::linkCount () const `[inline]`

: Returns the reference count.

7.118.2.2 PP_UseCount & PP_UseCount::operator= (const PP_UseCount & *rhs*) `[inline]`

: The assignment operator. Increments the reference count on rhs, decrements the reference count on this, and then makes this and rhs point to the same reference count.

7.118.2.3 bool PP_UseCount::unique () const [inline]

: The PP_UseCount is unique if the reference count == 1.

The documentation for this class was generated from the following file:

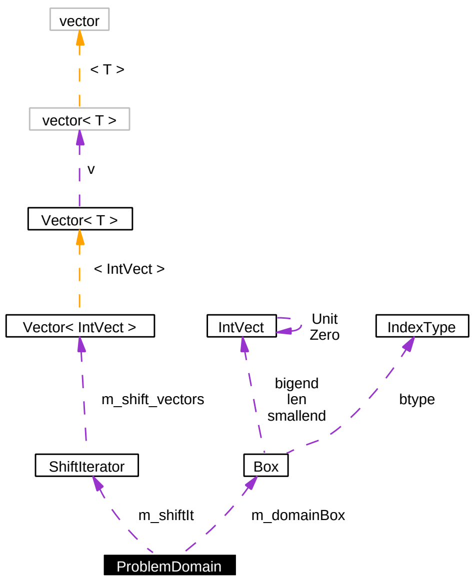
- **ParmParse.H**

7.119 ProblemDomain Class Reference

A class to facilitate interaction with physical boundary conditions.

`#include <ProblemDomain.H>`

Collaboration diagram for ProblemDomain:



Public Member Functions

- **ProblemDomain** ()

  The default constructor. The constructed domain box is empty.

- **ProblemDomain** (const **Box** &a_domBox)

  Construct ProblemDomain with a_domBox *as computational domain.*

- **ProblemDomain** (const **Box** &a_domBox, const bool *a_isPeriodic)

  Construct ProblemDomain with a_domBox as computational domain.

- **ProblemDomain** (const **IntVect** &small, const **IntVect** &big)

  Construct a ProblemDomain.

- **ProblemDomain** (const **IntVect** &small, const **IntVect** &big, const bool *a_isPeriodic)

  Construct a ProblemDomain.

- **ProblemDomain** (const **IntVect** &small, const int *vec_len)

  Construct ProblemDomain with specified lengths.

- **ProblemDomain** (const **IntVect** &small, const int *vec_len, const bool *a_isPeriodic)

  Construct ProblemDomain with specified lengths.

- **ProblemDomain** (const **ProblemDomain** &a_src)
  The copy constructor.
- const **Box** & **domainBox** () const
  Returns the logical computational domain.
- bool **isPeriodic** (int a_dir) const
  Returns true if BC is periodic in direction a_dir.
- bool **isPeriodic** () const
  Returns true is BC is periodic in _any_ direction.
- **ShiftIterator shiftIterator** () const
  Returns the shiftIterator for this ProblemDomain.
- bool **isEmpty** () const
  Returns true if this ProblemDomain is empty or undefined.
- bool **contains** (const **IntVect** &p) const
  Returns true if argument is contained within this ProblemDomain.
- bool **image** (**IntVect** &p) const
  Returns the periodic image of this **IntVect** *inside of the ProblemDomain.*
- bool **contains** (const **Box** &b) const
  Returns true if argument is contained within this ProblemDomain.
- bool **contains_box** (const **Box** &b) const
  Equivalent to **contains()** *function.*
- bool **intersects** (const **Box** &a_box) const
  Returns true if this ProblemDomain and the argument intersect.
- bool **intersectsNotEmpty** (const **Box** &a_box) const
  Returns true if this ProblemDomain and the argument intersect.
- bool **intersects** (const **Box** &box1, const **Box** &box2) const
  Returns true if box1 and box2 or any of their periodic images intersect.
- **ProblemDomain** & **operator=** (const **ProblemDomain** &b)
  The assignment operator.
- void **setPeriodic** (int a_dir, bool a_isPeriodic)
  Sets whether BC is periodic in direction a_dir (true == periodic).
- **ProblemDomain** & **grow** (int i)
  Grows (or shrinks) the domain **Box** *by i in all directions.*
- **ProblemDomain** & **grow** (const **IntVect** &v)
  Grow this ProblemDomain.

- **ProblemDomain** & **grow** (int idir, int n_cell)

  Grow this ProblemDomain.

- **ProblemDomain** & **growLo** (int idir, int n_cell=1)

  Grow this ProblemDomain on the low side.

- **ProblemDomain** & **growHi** (int idir, int n_cell=1)

  Grow this ProblemDomain on the high side.

- **Box operator &** (const **Box** &a_b) const

  Returns the **Box** *intersection of this ProblemDomain and* a_b.

- **ProblemDomain** & **refine** (int a_refinement_ratio)

  Refine this problem domain.

- **ProblemDomain** & **refine** (const **IntVect** &a_refinement_ratio)

  Refine this ProblemDomain.

- **ProblemDomain** & **coarsen** (int a_refinement_ratio)

  Coarsen this ProblemDomain.

- **ProblemDomain** & **coarsen** (const **IntVect** &refinement_ratio)

  Coarsen this ProblemDomain.

- void **dumpOn** (std::ostream &strm) const

  Gives more detail than printOn.

Protected Attributes

- bool **m_isPeriodic** [**SpaceDim**]
- **Box m_domainBox**
- **ShiftIterator m_shiftIt**

Friends

- class **HDF5Handle**
- void **operator &=** (**Box** &a_box, const **ProblemDomain** &a_probdomain)

  Modifies a_box to be the intersection of a_box and a_probdomain.

- **Box operator &** (const **Box** &a_box, const **ProblemDomain** &a_probdomain)

  Returns a **Box** *which is the interesection of a_box and* a_probdomain.

- **ProblemDomain grow** (const **ProblemDomain** &pd, int i)

  Returns a ProblemDomain with a domainBox grown by the given amount.

- **ProblemDomain grow** (const **ProblemDomain** &pd, const **IntVect** &v)

  Returns a grown version of pd.

- **Box bdryLo** (const **ProblemDomain** &a_pd, int a_dir, int a_len=1)
  Returns a face-centered **Box** *at the low side of* a_pd.
- **Box bdryHi** (const **ProblemDomain** &a_pd, int a_dir, int a_len=1)
  Returns a face-centered **Box** *at the high side of* a_pd.
- **Box adjCellLo** (const **ProblemDomain** &a_pd, int a_dir, int a_len=1)
  Returns the cell-centered **Box** *adjacent to the low side of* a_pd.
- **Box adjCellHi** (const **ProblemDomain** &a_pd, int a_dir, int a_len=1)
  Returns the cell-centered **Box** *adjacent to the low side of* a_pd.
- **ProblemDomain refine** (const **ProblemDomain** &a_probdomain, int a_refinement_ratio)
  Return a ProblemDomain which is a refinement of a_probdomain.
- **ProblemDomain refine** (const **ProblemDomain** &a_probdomain, const **IntVect** &a_refinement_ratio)
  Refinement function.
- **ProblemDomain coarsen** (const **ProblemDomain** &a_probdomain, int a_refinement_ratio)
  Coarsening function.
- **ProblemDomain coarsen** (const **ProblemDomain** &a_probdomain, const **IntVect** &a_refinement_ratio)
  Coarsening function.
- std::ostream & **operator<<** (std::ostream &os, const **ProblemDomain** &bx)
  Write an ASCII representation to the ostream.
- std::istream & **operator>>** (std::istream &is, **ProblemDomain** &bx)
  Read from istream.

7.119.1 Detailed Description

A class to facilitate interaction with physical boundary conditions.

ProblemDomain is a class which facilitates the application of physical boundary conditions, both periodic and non-periodic. This class contains much of the functionality of the **Box** class, since logically the computational domain is generally a **Box**.

Intersection with a ProblemDomain object will result in only removing regions which are outside the physical domain in non-periodic directions. Regions outside the logical computational domain in periodic directions will be treated as ghost cells which can be filled with an exchange() function or through suitable interpolation from a coarser domain.

Since ProblemDomain will contain a **Box**, it is a dimension dependent class, so SpaceDim must be defined as either 1, 2, or 3 when compiling.

Note that this implementation of ProblemDomain is inherently cell-centered.

7.119.2 Constructor & Destructor Documentation

7.119.2.1 ProblemDomain::ProblemDomain () `[inline]`

The default constructor. The constructed domain box is empty.

7.119.2.2 ProblemDomain::ProblemDomain (const Box & *a_domBox*)

Construct ProblemDomain with *a_domBox* as computational domain.

This constructor defaults to non-periodic domain

7.119.2.3 ProblemDomain::ProblemDomain (const Box & *a_domBox*, const bool ∗ *a_isPeriodic*)

Construct ProblemDomain with a_domBox as computational domain.

a_isPeriodic is a SpaceDim array of bools; if true, the physical boundary condition is periodic in the coordinate direction. False means a non-periodic BC.

7.119.2.4 ProblemDomain::ProblemDomain (const IntVect & *small*, const IntVect & *big*)

Construct a ProblemDomain.

It is an error if small is greater than big. Defaults to non-periodic domain.

7.119.2.5 ProblemDomain::ProblemDomain (const IntVect & *small*, const IntVect & *big*, const bool ∗ *a_isPeriodic*)

Construct a ProblemDomain.

It is an error if small is greater than big. *a_isPeriodic* is a SpaceDim array of bools; if true, the physical boundary condition is periodic in the coordinate direction. False means a non-periodic BC.

7.119.2.6 ProblemDomain::ProblemDomain (const IntVect & *small*, const int ∗ *vec_len*)

Construct ProblemDomain with specified lengths.

It is an error if the lengths are negative. Defaults to non-periodic domain.

7.119.2.7 ProblemDomain::ProblemDomain (const IntVect & *small*, const int ∗ *vec_len*, const bool ∗ *a_isPeriodic*)

Construct ProblemDomain with specified lengths.

It is an error if the lengths are negative. *a_isPeriodic* is a SpaceDim array of bools; if true, the physical boundary condition is periodic in the coordinate direction. False means a non-periodic BC.

7.119.2.8 ProblemDomain::ProblemDomain (const ProblemDomain & *a_src*) [inline]

The copy constructor.

7.119.3 Member Function Documentation

7.119.3.1 ProblemDomain& ProblemDomain::coarsen (const IntVect & *refinement_ratio*)

Coarsen this ProblemDomain.

Modifies this ProblemDomain by coarsening by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function.

7.119.3.2 ProblemDomain& ProblemDomain::coarsen (int *a_refinement_ratio*)

Coarsen this ProblemDomain.

Modifies this ProblemDomain by coarsening it by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function.

7.119.3.3 bool ProblemDomain::contains (const Box & *b*) const [inline]

Returns true if argument is contained within this ProblemDomain.

It is an error if the **Box** is not cell-centered. An empty ProblemDomain does not contain any **Box**. An entirely periodic domain contains any **Box**.

7.119.3.4 bool ProblemDomain::contains (const IntVect & *p*) const [inline]

Returns true if argument is contained within this ProblemDomain.

An empty ProblemDomain does not contain and is not contained by any ProblemDomain, including itself. Note also that in a periodic Direction, any index is valid, since the domain is infinite in that direction.

7.119.3.5 bool ProblemDomain::contains_box (const Box & *b*) const [inline]

Equivalent to **contains()** function.

7.119.3.6 const Box & ProblemDomain::domainBox () const [inline]

Returns the logical computational domain.

7.119.3.7 void ProblemDomain::dumpOn (std::ostream & *strm*) const

Gives more detail than printOn.

Useful for exiting due to an error.

7.119.3.8 ProblemDomain & ProblemDomain::grow (int *idir*, int *n_cell*) [inline]

Grow this ProblemDomain.

Modifies this ProblemDomain by growing it on the low and high end by n_cell cells in direction idir.

7.119.3.9 ProblemDomain & ProblemDomain::grow (const IntVect & *v*) [inline]

Grow this ProblemDomain.

Modifies this ProblemDomain by growing the domainBox in each direction by the specified amount

7.119.3.10 ProblemDomain & ProblemDomain::grow (int *i*) [inline]

Grows (or shrinks) the domain **Box** by i in all directions.

7.119.3.11 ProblemDomain & ProblemDomain::growHi (int *idir*, int *n_cell* = 1) [inline]

Grow this ProblemDomain on the high side.

Modifies this ProblemDomain by growing it on the high end by n_Cell cells in direction idir

7.119.3.12 ProblemDomain & ProblemDomain::growLo (int *idir*, int *n_cell* = 1) [inline]

Grow this ProblemDomain on the low side.

Modifies this ProblemDomain by growing it on the low end by n_cell cells in direction idir.

7.119.3.13 bool ProblemDomain::image (IntVect & *p*) const [inline]

Returns the periodic image of this **IntVect** inside of the ProblemDomain.

Return true if the domain contains this **IntVect**, returning the image in 'p', otherwise returns false

7.119.3.14 bool ProblemDomain::intersects (const Box & *box1*, const Box & *box2*) const

Returns true if box1 and box2 or any of their periodic images intersect.

(useful for checking disjointness)

7.119.3.15 bool ProblemDomain::intersects (const Box & *a_box*) const

Returns true if this ProblemDomain and the argument intersect.

It is an error if a_box is not cell-centered. An empty ProblemDomain does not intersect any **Box**. This will do nothing in periodic directions (since a periodic domain is infinite in the periodic direction. If periodic in all dimensions, this will always return true.

7.119.3.16 bool ProblemDomain::intersectsNotEmpty (const Box & *a_box*) const

Returns true if this ProblemDomain and the argument intersect.

It is an error if a_box is not cell-centered. This routine does not perform the check to see if *this or b are empty Boxes. It is the callers responsibility to ensure that this never happens. If you are unsure, the use the .intersects(..) routine. In periodic directions, will always return true.

7.119.3.17 bool ProblemDomain::isEmpty () const `[inline]`

Returns true if this ProblemDomain is empty or undefined.

7.119.3.18 bool ProblemDomain::isPeriodic () const `[inline]`

Returns true is BC is periodic in _any_ direction.

7.119.3.19 bool ProblemDomain::isPeriodic (int *a_dir*) const `[inline]`

Returns true if BC is periodic in direction a_dir.

7.119.3.20 Box ProblemDomain::operator & (const Box & *a_b*) const

Returns the **Box** intersection of this ProblemDomain and *a_b*.

The **Box** MUST be cell-centered. The intersection of the Empty ProblemDomain and any **Box** is the Empty **Box**. This operator does nothing in periodic directions (since a periodic domain is an infinite domain).

7.119.3.21 ProblemDomain & ProblemDomain::operator= (const ProblemDomain & *b*) `[inline]`

The assignment operator.

7.119.3.22 ProblemDomain& ProblemDomain::refine (const IntVect & *a_refinement_ratio*)

Refine this ProblemDomain.

Modifies this ProblemDomain by refining it by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function.

7.119.3.23 ProblemDomain& ProblemDomain::refine (int *a_refinement_ratio*)

Refine this problem domain.

Modifies this ProblemDomain by refining it by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function.

7.119.3.24 void ProblemDomain::setPeriodic (int *a_dir*, bool *a_isPeriodic*)

Sets whether BC is periodic in direction a_dir (true == periodic).

7.119.3.25 ShiftIterator ProblemDomain::shiftIterator () const [inline]

Returns the shiftIterator for this ProblemDomain.

The shiftIterator is defined based on the periodicity of this ProblemDomain.

7.119.4 Friends And Related Function Documentation

7.119.4.1 Box adjCellHi (const ProblemDomain & *a_pd*, int *a_dir*, int *a_len* = 1) [friend]

Returns the cell-centered **Box** adjacent to the low side of *a_pd*.

Returns the cell centered **Box** of the given length adjacent to the argument ProblemDomain on the high end along the given coordinate direction. The return **Box** is identical to the argument ProblemDomain in the other directions. The return **Box** and the argument ProblemDomain have an empty intersection.

NOTES:

- len >= 1.
- **Box** retval = adjCellHi(b,dir,len) is equivalent to the following set of operations:

```
Box retval(b);

retval.convert(dir,ProblemDomain::CELL);

retval.setrange(dir,retval.bigEnd(dir)+1,len);
```

Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty ProblemDomain is an Empty **Box** of the appropriate type. If *a_dir* is a periodic direction, will return an empty **Box**.

7.119.4.2 Box adjCellLo (const ProblemDomain & *a_pd*, int *a_dir*, int *a_len* = 1) [friend]

Returns the cell-centered **Box** adjacent to the low side of *a_pd*.

Returns the cell centered **Box** of the given length adjacent to the argument ProblemDomain on the low end along the given coordinate direction. The return **Box** is identical to the argument ProblemDomain in the other directions. The return ProblemDomain and the argument ProblemDomain have an empty intersection.

NOTES:

- len >= 1.
- **Box** retval = adjCellLo(b,dir,len) is equivalent to the following set of operations:

```
Box retval(b);

retval.convert(dir,ProblemDomain::CELL);

retval.setrange(dir,retval.smallEnd(dir)-len,len);
```

Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty ProblemDomain is an Empty **Box** of the appropriate type. If *a_dir* is a periodic direction, will return an empty **Box** as well.

7.119.4.3 Box bdryHi (const ProblemDomain & *a_pd*, int *a_dir*, int *a_len* = 1) [friend]

Returns a face-centered **Box** at the high side of *a_pd*.

Returns the edge-centered **Box** (in direction *a_dir*) defining the high side of the argument ProblemDomain. The return **Box** will have the given length in the given direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty ProblemDomain is an Empty **Box** of the appropriate type. If *a_dir* is a periodic direction, will return an empty **Box**.

7.119.4.4 Box bdryLo (const ProblemDomain & *a_pd*, int *a_dir*, int *a_len* = 1) [friend]

Returns a face-centered **Box** at the low side of *a_pd*.

Returns the edge-centered **Box** (in direction *a_dir*) defining the low side of the argument ProblemDomain. The output **Box** will have the given length in the given direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty ProblemDomain is an Empty **Box** of the appropriate type. If *a_dir* is a periodic direction, will return an empty **Box**.

7.119.4.5 ProblemDomain coarsen (const ProblemDomain & *a_probdomain*, const IntVect & *a_refinement_ratio*) [friend]

Coarsening function.

Returns a ProblemDomain that is the argument ProblemDomain coarsened by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function.

7.119.4.6 ProblemDomain coarsen (const ProblemDomain & *a_probdomain*, int *a_refinement_ratio*) [friend]

Coarsening function.

Returns a ProblemDomain that is the argument ProblemDomain coarsened by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function.

7.119.4.7 ProblemDomain grow (const ProblemDomain & *pd*, const IntVect & *v*) [friend]

Returns a grown version of *pd*.

Returns a ProblemDomain that is the argument ProblemDomain with a DomainBox grown by the given amount.

7.119.4.8 ProblemDomain grow (const ProblemDomain & *pd*, int *i*) [friend]

Returns a ProblemDomain with a domainBox grown by the given amount.

7.119.4.9 friend class HDF5Handle [friend]

7.119.4.10 Box operator & (const Box & *a_box*, const ProblemDomain & *a_probdomain*) [friend]

Returns a **Box** which is the interesection of a_box and *a_probdomain*.

7.119.4.11 void operator &= (Box & *a_box*, const ProblemDomain & *a_probdomain*) [friend]

Modifies a_box to be the intersection of a_box and *a_probdomain*.

7.119.4.12 std::ostream& operator<< (std::ostream & *os*, const ProblemDomain & *bx*) [friend]

Write an ASCII representation to the ostream.

7.119.4.13 std::istream& operator>> (std::istream & *is*, ProblemDomain & *bx*) [friend]

Read from istream.

7.119.4.14 ProblemDomain refine (const ProblemDomain & *a_probdomain*, const IntVect & *a_refinement_ratio*) [friend]

Refinement function.

Returns a ProblemDomain that is the argument ProblemDomain refined by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function.

7.119.4.15 ProblemDomain refine (const ProblemDomain & *a_probdomain*, int *a_refinement_ratio*) [friend]

Return a ProblemDomain which is a refinement of *a_probdomain*.

Returns a ProblemDomain that is the argument ProblemDomain refined by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function.

7.119.5 Member Data Documentation

7.119.5.1 Box ProblemDomain::m_domainBox [protected]

Domain index extents

7.119.5.2 bool ProblemDomain::m_isPeriodic[SpaceDim] [protected]

Periodicity info

7.119.5.3 ShiftIterator ProblemDomain::m_shiftIt [protected]

Shift iterator for this ProblemDomain

The documentation for this class was generated from the following file:

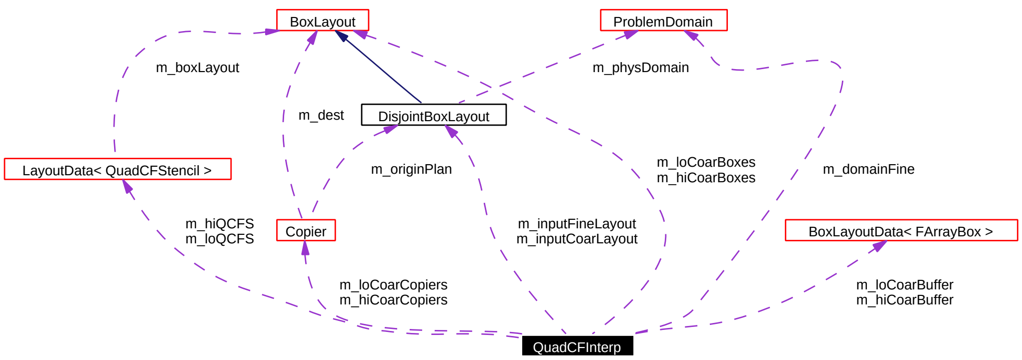
- **ProblemDomain.H**

7.120 QuadCFInterp Class Reference

Quadratic coarse-fine interpolation utility.

`#include <QuadCFInterp.H>`

Collaboration diagram for QuadCFInterp:



Public Member Functions

- **QuadCFInterp** (const **DisjointBoxLayout** &a_fineBoxes, const **DisjointBoxLayout** ∗a_coarBoxes, **Real** a_dxFine, int a_refRatio, int a_nComp, const **Box** &a_domf)

    constructors, destructors, defines

- **QuadCFInterp** (const **DisjointBoxLayout** &a_fineBoxes, const **DisjointBoxLayout** ∗a_coarBoxes, **Real** a_dxFine, int a_refRatio, int a_nComp, const **ProblemDomain** &a_domf)
- void **define** (const **DisjointBoxLayout** &a_fineBoxes, const **DisjointBoxLayout** ∗a_coarBoxes, **Real** a_dxFine, int a_refRatio, int a_nComp, const **ProblemDomain** &a_domf)
- **QuadCFInterp** ()
- **∼QuadCFInterp** ()
- void **clear** ()
- void **coarseFineInterp** (**LevelData**< **FArrayBox** > &a_phif, const **LevelData**< **FArrayBox** > &a_phic)
- bool **isDefined** () const

Protected Member Functions

- void **coarseFineInterp** (**BaseFab**< **Real** > &a_phif, const **BaseFab**< **Real** > &a_phic, const **QuadCFStencil** &a_qcfs, const **Side::LoHiSide** a_hiorlo, const int a_idir, const **Interval** &a_variables) const
- void **getPhiStar** (**BaseFab**< **Real** > &a_phistar, const **BaseFab**< **Real** > &a_phic, const **QuadCFStencil** &a_qcfs, const **Side::LoHiSide** a_hiorlo, const int a_idir, const **Interval** &a_variables) const
- void **interpOnIVS** (**BaseFab**< **Real** > &a_phif, const **BaseFab**< **Real** > &a_phiStar, const **QuadCFStencil** &a_qcfs, const **Side::LoHiSide** a_hiorlo, const int a_idir, const **Interval** &a_variables) const

Protected Attributes

- **BoxLayout m_loCoarBoxes** [SpaceDim]
- **BoxLayout m_hiCoarBoxes** [SpaceDim]
- int **m_level**
- int **m_nComp**
- int **m_refRatio**
- **Real m_dxFine**
- **LayoutData< QuadCFStencil > m_loQCFS** [SpaceDim]
- **LayoutData< QuadCFStencil > m_hiQCFS** [SpaceDim]
- **BoxLayoutData< FArrayBox > m_hiCoarBuffer** [SpaceDim]
- **Copier m_hiCoarCopiers** [SpaceDim]
- **BoxLayoutData< FArrayBox > m_loCoarBuffer** [SpaceDim]
- **Copier m_loCoarCopiers** [SpaceDim]
- **DisjointBoxLayout m_inputFineLayout**
- **DisjointBoxLayout m_inputCoarLayout**
- bool **m_isDefined**

  has full define function been called?

- **ProblemDomain m_domainFine**

7.120.1 Detailed Description

Quadratic coarse-fine interpolation utility.

Class to fill ghost cells on coarse-fine interface using quadratic interpolation

7.120.2 Constructor & Destructor Documentation

7.120.2.1 QuadCFInterp::QuadCFInterp (const DisjointBoxLayout & *a_fineBoxes*, const DisjointBoxLayout * *a_coarBoxes*, Real *a_dxFine*, int *a_refRatio*, int *a_nComp*, const Box & *a_domf*)

constructors, destructors, defines

Full Constructor. makes all coarse-fine information and sets internal variables calls full define

7.120.2.2 QuadCFInterp::QuadCFInterp (const DisjointBoxLayout & *a_fineBoxes*, const DisjointBoxLayout * *a_coarBoxes*, Real *a_dxFine*, int *a_refRatio*, int *a_nComp*, const ProblemDomain & *a_domf*)

Full Constructor. makes all coarse-fine information and sets internal variables calls full define

7.120.2.3 QuadCFInterp::QuadCFInterp ()

Default constructor leaves QCFI undefined.

7.120.2.4 QuadCFInterp::~QuadCFInterp ()

7.120.3 Member Function Documentation

7.120.3.1 void QuadCFInterp::clear ()

return QCFI to undefined state

7.120.3.2 void QuadCFInterp::coarseFineInterp (BaseFab< Real > & *a_phif*, const BaseFab< Real > & *a_phic*, const QuadCFStencil & *a_qcfs*, const Side::LoHiSide *a_hiorlo*, const int *a_idir*, const Interval & *a_variables*) const [protected]

7.120.3.3 void QuadCFInterp::coarseFineInterp (LevelData< FArrayBox > & *a_phif*, const LevelData< FArrayBox > & *a_phic*)

Coarse / Fine interpolation operator.

7.120.3.4 void QuadCFInterp::define (const DisjointBoxLayout & *a_fineBoxes*, const DisjointBoxLayout * *a_coarBoxes*, Real *a_dxFine*, int *a_refRatio*, int *a_nComp*, const ProblemDomain & *a_domf*)

Full define function. makes all coarse-fine information and sets internal variables

7.120.3.5 void QuadCFInterp::getPhiStar (BaseFab< Real > & *a_phistar*, const BaseFab< Real > & *a_phic*, const QuadCFStencil & *a_qcfs*, const Side::LoHiSide *a_hiorlo*, const int *a_idir*, const Interval & *a_variables*) const [protected]

7.120.3.6 void QuadCFInterp::interpOnIVS (BaseFab< Real > & *a_phif*, const BaseFab< Real > & *a_phiStar*, const QuadCFStencil & *a_qcfs*, const Side::LoHiSide *a_hiorlo*, const int *a_idir*, const Interval & *a_variables*) const [protected]

7.120.3.7 bool QuadCFInterp::isDefined () const

has full define function been called? return true if so

7.120.4 Member Data Documentation

7.120.4.1 ProblemDomain QuadCFInterp::m_domainFine [protected]

7.120.4.2 Real QuadCFInterp::m_dxFine [protected]

7.120.4.3 BoxLayout QuadCFInterp::m_hiCoarBoxes[SpaceDim] [protected]

7.120.4.4 BoxLayoutData<FArrayBox> QuadCFInterp::m_hiCoarBuffer[SpaceDim] [protected]

7.120.4.5 Copier QuadCFInterp::m_hiCoarCopiers[SpaceDim] [protected]

7.120.4.6 LayoutData<QuadCFStencil> QuadCFInterp::m_hiQCFS[SpaceDim] [protected]

7.120.4.7 DisjointBoxLayout QuadCFInterp::m_inputCoarLayout [protected]

7.120.4.8 DisjointBoxLayout QuadCFInterp::m_inputFineLayout [protected]

7.120.4.9 bool QuadCFInterp::m_isDefined [protected]

has full define function been called?

7.120.4.10 int QuadCFInterp::m_level [protected]

7.120.4.11 BoxLayout QuadCFInterp::m_loCoarBoxes[SpaceDim] [protected]

7.120.4.12 BoxLayoutData<FArrayBox> QuadCFInterp::m_loCoarBuffer[SpaceDim] [protected]

7.120.4.13 Copier QuadCFInterp::m_loCoarCopiers[SpaceDim] [protected]

7.120.4.14 LayoutData<QuadCFStencil> QuadCFInterp::m_loQCFS[SpaceDim] [protected]

7.120.4.15 int QuadCFInterp::m_nComp [protected]

7.120.4.16 int QuadCFInterp::m_refRatio [protected]

The documentation for this class was generated from the following file:

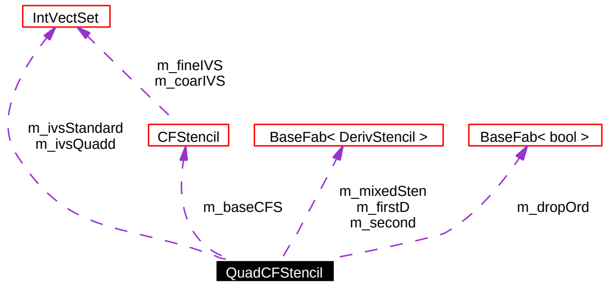
- **QuadCFInterp.H**

7.121 QuadCFStencil Class Reference

class to encapsulate CF info for quadratic interpolation

`#include <QuadCFStencil.H>`

Collaboration diagram for QuadCFStencil:



Public Member Functions

- **Real computeSecondDerivative** (const **BaseFab**< **Real** > &a_phic, int a_derivdir, int a_ivar, const **IntVect** &a_ivin, **Real** a_dx) const

  { *Derivative functions*}

- **Real computeFirstDerivative** (const **BaseFab**< **Real** > &a_phic, int a_derivdir, int a_ivar, const **IntVect** &a_ivin, **Real** a_dx) const
- **Real computeMixedDerivative** (const **BaseFab**< **Real** > &a_phic, int a_ivar, const **IntVect** &a_ivin, **Real** a_dx) const
- **QuadCFStencil** ()
- ~**QuadCFStencil** ()
- **QuadCFStencil** (const **Box** &a_fine_domain, const **Box** &a_grid, const **DisjointBoxLayout** &a_fineBoxes, const **DisjointBoxLayout** &a_coarBoxes, int a_refRatio, int a_direction, **Side::LoHiSide** a_hiorlo)
- **QuadCFStencil** (const **ProblemDomain** &a_fine_domain, const **Box** &a_grid, const **DisjointBoxLayout** &a_fineBoxes, const **DisjointBoxLayout** &a_coarBoxes, int a_refRatio, int a_direction, **Side::LoHiSide** a_hiorlo)
- void **define** (const **Box** &a_fine_domain, const **Box** &a_grid, const **DisjointBoxLayout** &a_fineBoxes, const **DisjointBoxLayout** &a_coarBoxes, int a_refRatio, int a_direction, **Side::LoHiSide** a_hiorlo)
- void **define** (const **ProblemDomain** &a_fine_domain, const **Box** &a_grid, const **DisjointBoxLayout** &a_fineBoxes, const **DisjointBoxLayout** &a_coarBoxes, int a_refRatio, int a_direction, **Side::LoHiSide** a_hiorlo)
- bool **isDefined** () const

  { *Access functions*}

- bool **isEmpty** () const
- const **IntVectSet** & **getFineIVS** () const
- const **IntVectSet** & **getCoarIVS** () const

Protected Attributes

- bool **m_isDefined**
- int **m_direction**
- **IntVectSet m_ivsStandard**
- **IntVectSet m_ivsQuadd**
- **BaseFab< DerivStencil > m_mixedSten**
- **BaseFab< DerivStencil > m_second** [**SpaceDim**]
- **BaseFab< DerivStencil > m_firstD** [**SpaceDim**]
- **BaseFab<** bool **> m_dropOrd**
- **CFStencil m_baseCFS**

7.121.1 Detailed Description

class to encapsulate CF info for quadratic interpolation

QuadCFStencil is a wrapper around a bunch of things that allow **LevelOp** to do coarse-fine interpolation on a particular face of a grid.

7.121.2 Constructor & Destructor Documentation

7.121.2.1 QuadCFStencil::QuadCFStencil ()

7.121.2.2 QuadCFStencil::~QuadCFStencil ()

7.121.2.3 QuadCFStencil::QuadCFStencil (const Box & *a_fine_domain*, const Box & *a_grid*, const DisjointBoxLayout & *a_fineBoxes*, const DisjointBoxLayout & *a_coarBoxes*, int *a_refRatio*, int *a_direction*, Side::LoHiSide *a_hiorlo*)

7.121.2.4 QuadCFStencil::QuadCFStencil (const ProblemDomain & *a_fine_domain*, const Box & *a_grid*, const DisjointBoxLayout & *a_fineBoxes*, const DisjointBoxLayout & *a_coarBoxes*, int *a_refRatio*, int *a_direction*, Side::LoHiSide *a_hiorlo*)

7.121.3 Member Function Documentation

7.121.3.1 Real QuadCFStencil::computeFirstDerivative (const BaseFab< Real > & *a_phic*, int *a_derivdir*, int *a_ivar*, const IntVect & *a_ivin*, Real *a_dx*) const

compute first derivative in devdir_a direction at coarse point a_ivin Uses centered finite diff approximation if ivs_standard_m.contains(a_ivin). Otherwise, it uses the stencil from firstd_m Asserts that a_derivdir != direction of face

7.121.3.2 Real QuadCFStencil::computeMixedDerivative (const BaseFab< Real > & *a_phic*, int *a_ivar*, const IntVect & *a_ivin*, Real *a_dx*) const

compute mixed derivative (direction is unambiguous. x and y are the two directions tangential to face) at coarse point a_ivin. \ In two dimensions, always returns 0. \ Uses centered finite diff approximation if ivs_standard_m.contains(a_ivin). It uses the stencil from mixed_sten_m.\ Returns 0 if SpaceDim != 3

7.121.3.3 Real QuadCFStencil::computeSecondDerivative (const BaseFab< Real > & *a_phic*, int *a_derivdir*, int *a_ivar*, const IntVect & *a_ivin*, Real *a_dx*) const

{ Derivative functions}

compute second derivative in devdir_a direction at coarse point a_ivin Uses centered finite diff approximation if ivs_standard_m.contains(a_ivin). Otherwise, it uses the stencil from second_m Asserts that a_derivdir != direction of face

7.121.3.4 void QuadCFStencil::define (const ProblemDomain & *a_fine_domain*, const Box & *a_grid*, const DisjointBoxLayout & *a_fineBoxes*, const DisjointBoxLayout & *a_coarBoxes*, int *a_refRatio*, int *a_direction*, Side::LoHiSide *a_hiorlo*)

7.121.3.5 void QuadCFStencil::define (const Box & *a_fine_domain*, const Box & *a_grid*, const DisjointBoxLayout & *a_fineBoxes*, const DisjointBoxLayout & *a_coarBoxes*, int *a_refRatio*, int *a_direction*, Side::LoHiSide *a_hiorlo*)

7.121.3.6 const IntVectSet& QuadCFStencil::getCoarIVS () const `[inline]`

get coarse intvects that underly fiinterpivs. This will be empty if **isEmpty()** returns true

7.121.3.7 const IntVectSet& QuadCFStencil::getFineIVS () const `[inline]`

get fine intvects which need to be interpolated \ This will be empty if **isEmpty()** returns true

7.121.3.8 bool QuadCFStencil::isDefined () const

{ Access functions}

has full define function been called? return true if so

7.121.3.9 bool QuadCFStencil::isEmpty () const `[inline]`

are there any interpolation points? \ returns false if so.

7.121.4 Member Data Documentation

7.121.4.1 CFStencil QuadCFStencil::m_baseCFS `[protected]`

7.121.4.2 int QuadCFStencil::m_direction `[protected]`

7.121.4.3 BaseFab<bool> QuadCFStencil::m_dropOrd `[protected]`

whether to drop order at any one-sided point

7.121.4.4 BaseFab<DerivStencil> QuadCFStencil::m_firstD[SpaceDim] `[protected]`

stencil information for unmixed derivatives (first derivatives)

7.121.4.5 bool QuadCFStencil::m_isDefined [protected]

7.121.4.6 IntVectSet QuadCFStencil::m_ivsQuadd [protected]

set of all points which have at least one derivative shifted

7.121.4.7 IntVectSet QuadCFStencil::m_ivsStandard [protected]

set of all points which have ALL their derivatives taken normally. This means that no stencil either leaves the domain or intersects the next finer level.

7.121.4.8 BaseFab<DerivStencil> QuadCFStencil::m_mixedSten [protected]

stencil information for mixed derivatives undefined if SpaceDim != 3

7.121.4.9 BaseFab<DerivStencil> QuadCFStencil::m_second[SpaceDim] [protected]

stencil information for unmixed derivatives (second derivatives)

The documentation for this class was generated from the following file:

- **QuadCFStencil.H**

7.122 RealVect Class Reference

A Real vector in SpaceDim-dimensional space.

`#include <RealVect.H>`

Collaboration diagram for RealVect:

Constants

- const **RealVect Zero**
- const **RealVect Unit**
- **RealVect BASISREALV** (int dir)

Public Member Functions

Constructors and Accessors

- **RealVect** ()
- **RealVect** (D_DECL(**Real** i, **Real** j, **Real** k))
- **RealVect** (const **RealVect** &rhs)
- **RealVect** (const **IntVect** &iv)
- **RealVect** & **operator=** (const **RealVect** &rhs)
- **Real** & **operator**[ ] (int i)
- const **Real** & **operator**[ ] (int i) const

Comparison Operators

- bool **operator==** (const **RealVect** &p) const
- bool **operator!=** (const **RealVect** &p) const

Arithmetic Operators

- **RealVect** & **operator+=** (**Real** s)
- **RealVect** & **operator+=** (const **RealVect** &p)
- **RealVect** & **operator-=** (**Real** s)
- **RealVect** & **operator-=** (const **RealVect** &p)
- **RealVect operator-** (**Real** s) const
- **RealVect** & **operator** ∗= (**Real** s)
- **RealVect** & **operator** ∗= (const **RealVect** &p)
- **RealVect operator** ∗ (const **RealVect** &p) const
- **RealVect operator** ∗ (**Real** s) const
- **RealVect** & **operator/=** (**Real** s)
- **RealVect** & **operator/=** (const **RealVect** &p)
- **RealVect operator/** (**Real** s) const

Data pointer functions

- const **Real** ∗ **dataPtr** () const
- **Real** ∗ **dataPtr** ()

Static Public Attributes

- size_t **io_offset**

Friends

- class **HDF5Handle**
- std::ostream & **operator<<** (std::ostream &ostr, const **RealVect** &p)

Arithmetic friend functions

- **RealVect operator+** (**Real** s, const **RealVect** &p)
- **RealVect operator-** (**Real** s, const **RealVect** &p)
- **RealVect operator** * (**Real** s, const **RealVect** &p)
- **RealVect operator/** (**Real** s, const **RealVect** &p)
- **RealVect operator+** (const **RealVect** &s, const **RealVect** &p)
- **RealVect operator-** (const **RealVect** &s, const **RealVect** &p)
- **RealVect operator** * (const **RealVect** &s, const **RealVect** &p)
- **RealVect operator/** (const **RealVect** &s, const **RealVect** &p)

7.122.1 Detailed Description

A Real vector in SpaceDim-dimensional space.

The class RealVect is an implementation of a Real vector in a SpaceDim-dimensional space. RealVect values are accessed using the operator[] function, as for a normal C++ array. In addition, the basic arithmetic operators have been overloaded to implement scaling and translation operations.

7.122.2 Constructor & Destructor Documentation

7.122.2.1 RealVect::RealVect ()

Construct a RealVect whose components are zero.

7.122.2.2 RealVect::RealVect (D_DECL(Real i, Real j, Real k))

Construct a RealVect given the specific values for its coordinates. D_DECL is a macro that sets the constructor to take CH_SPACEDIM arguments.

7.122.2.3 RealVect::RealVect (const RealVect & *rhs*)

The copy constructor.

7.122.2.4 RealVect::RealVect (const IntVect & *iv*) `[inline]`

Construct a RealVect from an **IntVect** by coercing each component from `int` to Real.

7.122.3 Member Function Documentation

7.122.3.1 Real* RealVect::dataPtr ()

Only for sending stuff to Fortran

7.122.3.2 const Real* RealVect::dataPtr () const

Only for sending stuff to Fortran

7.122.3.3 RealVect RealVect::operator * (Real *s*) const

Returns a RealVect that is this RealVect with each component multiplied by a scalar.

7.122.3.4 RealVect RealVect::operator * (const RealVect & *p*) const

Returns component-wise product of this RealVect and argument.

7.122.3.5 RealVect& RealVect::operator *= (const RealVect & *p*)

Modifies this RealVect by component-wise multiplication by argument.

7.122.3.6 RealVect& RealVect::operator *= (Real *s*)

Modifies this RealVect by multiplying each component by a scalar.

7.122.3.7 bool RealVect::operator!= (const RealVect & *p*) const

Returns true if this RealVect is different from argument RealVect. All comparisons between analogous components must be satisfied.

7.122.3.8 RealVect& RealVect::operator+= (const RealVect & *p*)

Modifies this RealVect by component-wise addition by argument.

7.122.3.9 RealVect& RealVect::operator+= (Real *s*)

Modifies this RealVect by addition of a scalar to each component.

7.122.3.10 RealVect RealVect::operator- (Real *s*) const

Returns a RealVect that is this RealVect with a scalar s subtracted from each component.

7.122.3.11 RealVect& RealVect::operator-= (const RealVect & *p*)

Modifies this RealVect by component-wise subtraction by argument.

7.122.3.12 RealVect& RealVect::operator-= (Real *s*)

Modifies this RealVect by subtraction of a scalar from each component.

7.122.3.13 RealVect RealVect::operator/ (Real *s*) const

Returns a RealVect that is this RealVect with each component divided by a scalar.

7.122.3.14 RealVect& RealVect::operator/= (const RealVect & *p*)

Modifies this RealVect by component-wise division by argument.

7.122.3.15 RealVect& RealVect::operator/= (Real *s*)

Modifies this RealVect by dividing each component by a scalar.

7.122.3.16 RealVect& RealVect::operator= (const RealVect & *rhs*)

The assignment operator.

7.122.3.17 bool RealVect::operator== (const RealVect & *p*) const

Returns true if this RealVect is equivalent to argument RealVect. All comparisons between analogous components must be satisfied.

7.122.3.18 const Real& RealVect::operator[] (int *i*) const

Returns the i'th coordinate of the RealVect.

7.122.3.19 Real& RealVect::operator[] (int *i*)

Returns a modifiable lvalue reference to the i'th coordinate of the RealVect.

7.122.4 Friends And Related Function Documentation

7.122.4.1 RealVect BASISREALV (int *dir*) `[friend]`

Returns a basis vector in the given coordinate direction.

In 2-D:

BASISREALV(0) == (1.,0.); BASISREALV(1) == (0.,1.).

In 3-D:

BASISREALV(0) == (1.,0.,0.); BASISREALV(1) == (0.,1.,0.); BASISREALV(2) == (0.,0.,1.).

Note that the coordinate directions are based at zero.

7.122.4.2 friend class HDF5Handle `[friend]`

7.122.4.3 RealVect operator * (const RealVect & *s*, const RealVect & *p*) `[friend]`

Returns component-wise product of *s* and *p*.

7.122.4.4 RealVect operator * (Real *s*, const RealVect & *p*) `[friend]`

Returns a RealVect that is a RealVect *p* with each component multiplied by a scalar *s*.

7.122.4.5 RealVect operator+ (const RealVect & *s*, const RealVect & *p*) `[friend]`

Returns component-wise sum of RealVects *s* and *p*.

7.122.4.6 RealVect operator+ (Real *s*, const RealVect & *p*) `[friend]`

Returns a RealVect that is a RealVect *p* with a scalar *s* added to each component.

7.122.4.7 RealVect operator- (const RealVect & *s*, const RealVect & *p*) `[friend]`

Returns *s* - *p*.

7.122.4.8 RealVect operator- (Real *s*, const RealVect & *p*) `[friend]`

Returns *s* - *p*.

7.122.4.9 RealVect operator/ (const RealVect & *s*, const RealVect & *p*) `[friend]`

Returns component-wise quotient *p* / *s*.

7.122.4.10 RealVect operator/ (Real *s*, const RealVect & *p*) `[friend]`

Returns a RealVect that is a RealVect *p* with each component divided by a scalar *s*.

7.122.4.11 std::ostream& operator<< (std::ostream & *ostr*, const RealVect & *p*) `[friend]`

Print to the given output stream in ASCII.

7.122.5 Member Data Documentation

7.122.5.1 size_t RealVect::io_offset `[static]`

7.122.5.2 const RealVect RealVect::Unit `[static]`

This is a RealVect all of whose components are equal to one.

7.122.5.3 const RealVect RealVect::Zero [static]

This is a RealVect all of whose components are equal to zero.

The documentation for this class was generated from the following file:

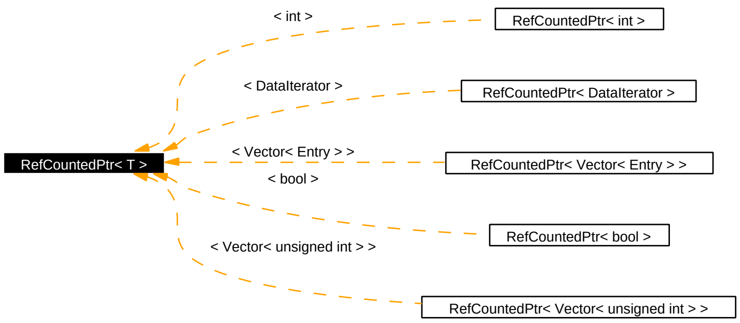
- **RealVect.H**

7.123 RefCountedPtr< T > Class Template Reference

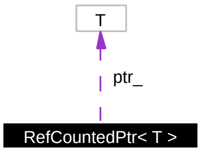
A reference-counting handle class.

`#include <RefCountedPtr.H>`

Inheritance diagram for RefCountedPtr< T >:



Collaboration diagram for RefCountedPtr< T >:



Public Member Functions

- **RefCountedPtr** (T *ptr=0)
- **RefCountedPtr** (const **RefCountedPtr**< T > &other)
- ~**RefCountedPtr** ()
- const **RefCountedPtr**< T > & **operator=** (const **RefCountedPtr**< T > &rhs)

    assignement operator. copies pointer member

- T * **operator** → ()

    dereference access operator. use like a pointer derefence access function.

- const T * **operator** → () const
- bool **isNull** () const
- T & **operator** * ()

    pointer dereference.

- const T & **operator** * () const
- **operator const T** * () const

    auto conversion to regular const pointer where required.

- bool **isNonUnique** () const

    true when refcount is one.

- int **refCount** () const

Protected Attributes

- T ∗ **ptr_**
- int ∗ **refCount_**

7.123.1 Detailed Description

template<class T> class RefCountedPtr< T >

A reference-counting handle class.

This is to be used as a pointer to class T. This will feel and smell just like a built-in pointer except:

1. There is no need to call delete on the pointer.
2. The default copy constructor and assignment implement ref-counting.
3. The user may call isNonUnique to determine if this pointer is the only pointer to the data. This can be used to hide the ref-counting behavior of a class.
4. Checks for dereference of a null pointer.

This class is completely inlined.

typical usage:

```
{
  Box b;
  IntVect a;
  //refCount() == 1
  RefCounterPtr<IntVectSet> set(new IntVectSet());

  // still just one IntVectSet, but refCount()==2
  RefCountedPtr<IntVectSet>  otherSet(set);

  // Like a pointer, modifying one modifies the other
  otherSet->define(b);
  (*set)|=a;
  {
        RefCountedPtr<IntVectSet> anotherSet; // null
        anotherSet = otherSet ;  //now all three have refCount()==3
  }//anotherSet out of scope, so deleted.  IntVectSet NOT deleted.

  // set.refCount() == 2
  // otherSet.refCount() == 2;
  // otherset == set;

}
// when all RefCountedPtr's for a given object are deleted, the last
// one calls 'delete' on the member pointer.
```

7.123.2 Constructor & Destructor Documentation

7.123.2.1 template<class T> RefCountedPtr< T >::RefCountedPtr (T * *ptr* = 0) [inline]

7.123.2.2 template<class T> RefCountedPtr< T >::RefCountedPtr (const RefCountedPtr< T > & *other*) [inline]

7.123.2.3 template<class T> RefCountedPtr< T >::~RefCountedPtr () [inline]

7.123.3 Member Function Documentation

7.123.3.1 template<class T> bool RefCountedPtr< T >::isNonUnique () const [inline]

true when refcount is one.

7.123.3.2 template<class T> bool RefCountedPtr< T >::isNull () const [inline]

7.123.3.3 template<class T> const T & RefCountedPtr< T >::operator * () const [inline]

7.123.3.4 template<class T> T & RefCountedPtr< T >::operator * () [inline]

pointer dereference.

7.123.3.5 template<class T> RefCountedPtr< T >::operator const T * () const [inline]

auto conversion to regular const pointer where required.

7.123.3.6 template<class T> const T * RefCountedPtr< T >::operator → () const [inline]

7.123.3.7 template<class T> T * RefCountedPtr< T >::operator → () [inline]

dereference access operator. use like a pointer derefence access function.

7.123.3.8 template<class T> const RefCountedPtr< T > & RefCountedPtr< T >::operator= (const RefCountedPtr< T > & *rhs*) [inline]

assignement operator. copies pointer member

copies pointer member and integer pointer, decreases refcount of rhs member before assignment. this refcount increased my one.

7.123.3.9 template<class T> int RefCountedPtr< T >::refCount () const [inline]

7.123.4 Member Data Documentation

7.123.4.1 template<class T> T* RefCountedPtr< T >::ptr_ [protected]

7.123.4.2 template<class T> int* RefCountedPtr< T >::refCount_ [protected]

The documentation for this class was generated from the following file:

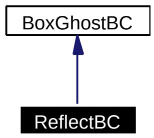
- **RefCountedPtr.H**

7.124 ReflectBC Class Reference

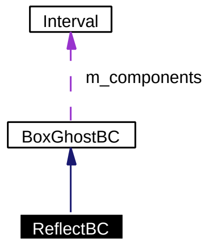
class to do extrapolation BC's

`#include <ReflectBC.H>`

Inheritance diagram for ReflectBC:



Collaboration diagram for ReflectBC:



Public Member Functions

- **ReflectBC** ()
- ~**ReflectBC** ()
- **ReflectBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- **ReflectBC** (int dir, **Side::LoHiSide** sd)

Protected Member Functions

- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const
- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyReflectBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

  this doesn't do anything since we don't use the factors

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const *this doesn't do anything since we don't use the factors*

- **BoxGhostBC** ∗ **new_boxghostbc** () const

Friends

- class **DomainGhostBC**

7.124.1 Detailed Description

class to do extrapolation BC's

7.124.2 Constructor & Destructor Documentation

7.124.2.1 ReflectBC::ReflectBC ()

7.124.2.2 ReflectBC::~ReflectBC ()

7.124.2.3 ReflectBC::ReflectBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.124.2.4 ReflectBC::ReflectBC (int *dir*, Side::LoHiSide *sd*)

7.124.3 Member Function Documentation

7.124.3.1 virtual void ReflectBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.124.3.2 virtual void ReflectBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_domain. (Deprecated function – use **ProblemDomain** instead of a **Box** to represent the domain)

Reimplemented from **BoxGhostBC** (p. 140).

7.124.3.3 virtual void ReflectBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.124.3.4 virtual void ReflectBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const [protected, virtual]

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_-domain. (Deprecated function – use **ProblemDomain** instead of a **Box** to represent the domain)

Reimplemented from **BoxGhostBC** (p. 141).

7.124.3.5 virtual void ReflectBC::applyReflectBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

7.124.3.6 virtual void ReflectBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const [protected, virtual]

this doesn't do anything since we don't use the factors

Implements **BoxGhostBC** (p. 141).

7.124.3.7 virtual void ReflectBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const [protected, virtual]

this doesn't do anything since we don't use the factors

Implements **BoxGhostBC** (p. 141).

7.124.3.8 BoxGhostBC* ReflectBC::new_boxghostbc () const [protected, virtual]

Virtual constructor workaround.

Implements **BoxGhostBC** (p. 141).

7.124.4 Friends And Related Function Documentation

7.124.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 142).

The documentation for this class was generated from the following file:

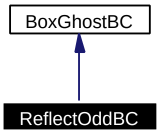
- **ReflectBC.H**

7.125 ReflectOddBC Class Reference

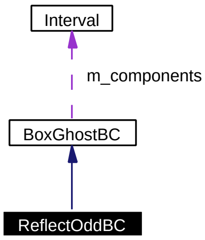
class to do extrapolation BC's

`#include <ReflectOddBC.H>`

Inheritance diagram for ReflectOddBC:



Collaboration diagram for ReflectOddBC:



Public Member Functions

- **ReflectOddBC** ()
- **~ReflectOddBC** ()
- **ReflectOddBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- **ReflectOddBC** (int dir, **Side::LoHiSide** sd)

Protected Member Functions

- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const
- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **Box** &a_domain, **Real** a_dx) const
- virtual void **applyInhomogeneousBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **applyReflectOddBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, **Real** a_dx) const
- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **Box** &a_domain) const

  this doesn't do anything since we don't use the factors

- virtual void **fillBCValues** (**FArrayBox** &a_neumfac, **FArrayBox** &a_dircfac, **FArrayBox** &a_inhmval, **Real** a_dx, const **ProblemDomain** &a_domain) const *this doesn't do anything since we don't use the factors*

- **BoxGhostBC** * **new_boxghostbc** () const

Friends

- class **DomainGhostBC**

7.125.1 Detailed Description

class to do extrapolation BC's

7.125.2 Constructor & Destructor Documentation

7.125.2.1 ReflectOddBC::ReflectOddBC ()

7.125.2.2 ReflectOddBC::~ReflectOddBC ()

7.125.2.3 ReflectOddBC::ReflectOddBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.125.2.4 ReflectOddBC::ReflectOddBC (int *dir*, Side::LoHiSide *sd*)

7.125.3 Member Function Documentation

7.125.3.1 virtual void ReflectOddBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_-domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.125.3.2 virtual void ReflectOddBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the homogeneous form of the boundary conditions to a_state along the boundary of a_-domain. (Deprecated function – use **ProblemDomain** instead of a **Box** to represent the domain)

Reimplemented from **BoxGhostBC** (p. 140).

7.125.3.3 virtual void ReflectOddBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_-domain.

Reimplemented from **BoxGhostBC** (p. 140).

7.125.3.4 virtual void ReflectOddBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const [protected, virtual]

Apply the inhomogeneous form of the boundary conditions to a_state along the boundary of a_domain. (Deprecated function – use **ProblemDomain** instead of a **Box** to represent the domain)

Reimplemented from **BoxGhostBC** (p. 141).

7.125.3.5 virtual void ReflectOddBC::applyReflectOddBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

7.125.3.6 virtual void ReflectOddBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const [protected, virtual]

this doesn't do anything since we don't use the factors

Implements **BoxGhostBC** (p. 141).

7.125.3.7 virtual void ReflectOddBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const [protected, virtual]

this doesn't do anything since we don't use the factors

Implements **BoxGhostBC** (p. 141).

7.125.3.8 BoxGhostBC* ReflectOddBC::new_boxghostbc () const [protected, virtual]

Virtual constructor workaround.

Implements **BoxGhostBC** (p. 141).

7.125.4 Friends And Related Function Documentation

7.125.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 142).

The documentation for this class was generated from the following file:

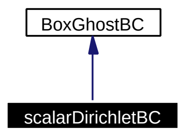
- **ReflectOddBC.H**

7.126 scalarDirichletBC Class Reference

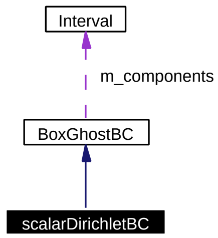
implements dirichlet BCs for scalars

`#include <scalarDirichletBC.H>`

Inheritance diagram for scalarDirichletBC:



Collaboration diagram for scalarDirichletBC:



Public Member Functions

- **scalarDirichletBC** (int dir, **Side::LoHiSide** sd)
- **scalarDirichletBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- virtual ~**scalarDirichletBC** ()
- **scalarDirichletBC** ()
- void **setVal** (const **Real** a_bcVal)

  sets Dirichlet Boundary value

Friends

- class **DomainGhostBC**

7.126.1 Detailed Description

implements dirichlet BCs for scalars

7.126.2 Constructor & Destructor Documentation

7.126.2.1 scalarDirichletBC::scalarDirichletBC (int *dir*, Side::LoHiSide *sd*)

7.126.2.2 scalarDirichletBC::scalarDirichletBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.126.2.3 virtual scalarDirichletBC::~scalarDirichletBC () `[virtual]`

7.126.2.4 scalarDirichletBC::scalarDirichletBC ()

7.126.3 Member Function Documentation

7.126.3.1 void scalarDirichletBC::setVal (const Real *a_bcVal*)

sets Dirichlet Boundary value

7.126.4 Friends And Related Function Documentation

7.126.4.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 142).

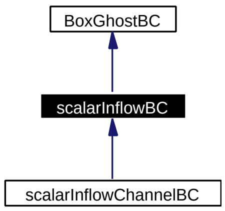
The documentation for this class was generated from the following file:

- **scalarDirichletBC.H**

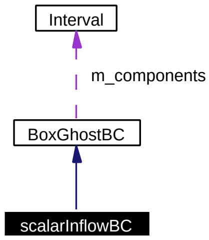
7.127 scalarInflowBC Class Reference

`#include <scalarDirichletBC.H>`

Inheritance diagram for scalarInflowBC:



Collaboration diagram for scalarInflowBC:



Public Member Functions

- **scalarInflowBC** ()
- **scalarInflowBC** (int dir, **Side::LoHiSide** sd)
- **scalarInflowBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- virtual ~**scalarInflowBC** ()
- void **setScalarType** (int a_scalType)

  set which scalar this is

- int **scalarType** () const
- void **isHomogeneous** (bool a_isHomogeneous)
- bool **isHomogeneous** () const

Friends

- class **DomainGhostBC**

7.127.1 Constructor & Destructor Documentation

7.127.1.1 scalarInflowBC::scalarInflowBC ()

7.127.1.2 scalarInflowBC::scalarInflowBC (int *dir*, Side::LoHiSide *sd*)

7.127.1.3 scalarInflowBC::scalarInflowBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.127.1.4 virtual scalarInflowBC::~scalarInflowBC () `[virtual]`

7.127.2 Member Function Documentation

7.127.2.1 bool scalarInflowBC::isHomogeneous () const

7.127.2.2 void scalarInflowBC::isHomogeneous (bool *a_isHomogeneous*)

7.127.2.3 int scalarInflowBC::scalarType () const

7.127.2.4 void scalarInflowBC::setScalarType (int *a_scalType*)

set which scalar this is

7.127.3 Friends And Related Function Documentation

7.127.3.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 142).

Reimplemented in **scalarInflowChannelBC** (p. 596).

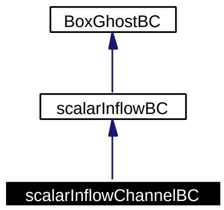
The documentation for this class was generated from the following file:

- **scalarDirichletBC.H**

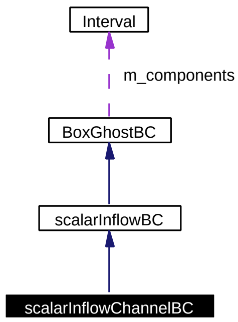
7.128 scalarInflowChannelBC Class Reference

`#include <channelBC.H>`

Inheritance diagram for scalarInflowChannelBC:



Collaboration diagram for scalarInflowChannelBC:



Public Member Functions

- **scalarInflowChannelBC** ()
- **scalarInflowChannelBC** (int dir, **Side::LoHiSide** sd)
- **scalarInflowChannelBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- virtual ~**scalarInflowChannelBC** ()

Friends

- class **DomainGhostBC**

7.128.1 Constructor & Destructor Documentation

7.128.1.1 scalarInflowChannelBC::scalarInflowChannelBC ()

7.128.1.2 scalarInflowChannelBC::scalarInflowChannelBC (int *dir*, Side::LoHiSide *sd*)

7.128.1.3 scalarInflowChannelBC::scalarInflowChannelBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

7.128.1.4 virtual scalarInflowChannelBC::~scalarInflowChannelBC () `[virtual]`

7.128.2 Friends And Related Function Documentation

7.128.2.1 friend class DomainGhostBC `[friend]`

Reimplemented from **scalarInflowBC** (p. 594).

The documentation for this class was generated from the following file:

- **channelBC.H**

7.129 SerialTask Class Reference

`#include <SPMD.H>`

Public Types

- enum **task** { **compute** = 0 }

7.129.1 Member Enumeration Documentation

7.129.1.1 enum SerialTask::task

Enumeration values:
  compute

The documentation for this class was generated from the following file:

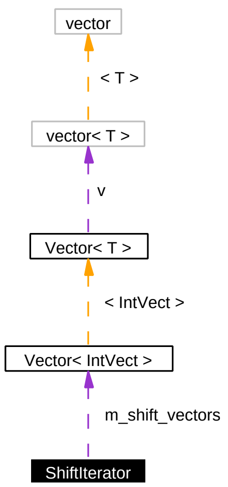
- **SPMD.H**

7.130 ShiftIterator Class Reference

Class to manage box-shifting used to enforce periodic BC's.

`#include <ProblemDomain.H>`

Collaboration diagram for ShiftIterator:



Public Member Functions

- **ShiftIterator** ()

  Basic constructor.

- **ShiftIterator** (const bool ∗a_isPeriodic)

  Defining Constructor.

- **ShiftIterator** (const **ShiftIterator** &a_shiftIt)

  Copy constructor.

- ∼**ShiftIterator** ()

  Destructor.

- **ShiftIterator** & **operator=** (const **ShiftIterator** &a_src)

  Assignment operator.

- void **computeShifts** (const bool ∗a_isPeriodic)

  Recompute shift vectors based on periodic directions.

- **IntVect operator()** () const

  Returns the current shift unit vector.

- **IntVect i** () const

  Equivalent to the () operator.

- void **operator++** ()

  Increment to the next shift unit vector.

- void **incr** ()

  Equivalent to the ++ operator.

- bool **ok** () const

  Is the iterator still within its range of shift vectors?

- void **reset** ()

  Reset to first shift unit vector.

- void **begin** ()

  Equivalent to **reset()**.

- void **end** ()

  Skip the iterator to the end.

7.130.1 Detailed Description

Class to manage box-shifting used to enforce periodic BC's.

The ShiftIterator class contains a list of shift vectors necessary to enforce periodic boundary conditions (enforced by shifting the **Box** of each Fab, then copying any overlapping valid cells with ghost cells of the shifted Fab. Depending on the number of periodic directions, the list of shift vectors over which to iterate differs.

7.130.2 Constructor & Destructor Documentation

7.130.2.1 ShiftIterator::ShiftIterator () `[inline]`

Basic constructor.

7.130.2.2 ShiftIterator::ShiftIterator (const bool * *a_isPeriodic*)

Defining Constructor.

Builds a ShiftIterator based on the periodicity in *a_isPeriodic*

7.130.2.3 ShiftIterator::ShiftIterator (const ShiftIterator & *a_shiftIt*) `[inline]`

Copy constructor.

7.130.2.4 ShiftIterator::~ShiftIterator ()

Destructor.

7.130.3 Member Function Documentation

7.130.3.1 void ShiftIterator::begin () [inline]

Equivalent to **reset()**.

7.130.3.2 void ShiftIterator::computeShifts (const bool * *a_isPeriodic*)

Recompute shift vectors based on periodic directions.

7.130.3.3 void ShiftIterator::end () [inline]

Skip the iterator to the end.

ok() will return false after this method is called.

7.130.3.4 IntVect ShiftIterator::i () const [inline]

Equivalent to the () operator.

7.130.3.5 void ShiftIterator::incr () [inline]

Equivalent to the ++ operator.

7.130.3.6 bool ShiftIterator::ok () const [inline]

Is the iterator still within its range of shift vectors?

7.130.3.7 IntVect ShiftIterator::operator() () const [inline]

Returns the current shift unit vector.

7.130.3.8 void ShiftIterator::operator++ () [inline]

Increment to the next shift unit vector.

7.130.3.9 ShiftIterator & ShiftIterator::operator= (const ShiftIterator & *a_src*) [inline]

Assignment operator.

7.130.3.10 void ShiftIterator::reset () [inline]

Reset to first shift unit vector.

The documentation for this class was generated from the following file:

- **ProblemDomain.H**

7.131 Side Class Reference

Encapsulation of high or low side.

`#include <LoHiSide.H>`

Public Types

- enum **LoHiSide** { **Invalid** = -1, **Lo** = 0, **Hi**, **NUMSIDES** }

Public Member Functions

- **Side** ()
- ~**Side** ()

Static Public Member Functions

- **Side::LoHiSide flip** (**Side::LoHiSide** side)

Friends

- **Side::LoHiSide flip** (**Side::LoHiSide** side)
- int **sign** (**Side::LoHiSide** side)

7.131.1 Detailed Description

Encapsulation of high or low side.

Encapsulation of high or low side.

7.131.2 Member Enumeration Documentation

7.131.2.1 enum Side::LoHiSide

Lo is the low side.

Hi is the high side.

Enumeration values:

Invalid

Lo

Hi

NUMSIDES

7.131.3 Constructor & Destructor Documentation

7.131.3.1 Side::Side () `[inline]`

Constructor.

7.131.3.2 Side::∼Side () [inline]

Destructor.

7.131.4 Member Function Documentation

7.131.4.1 Side::LoHiSide Side::flip (Side::LoHiSide *side*) [static]

7.131.5 Friends And Related Function Documentation

7.131.5.1 Side::LoHiSide flip (Side::LoHiSide *side*) [friend]

Returns the other side. That is, swap Hi for Lo and vice versa.

7.131.5.2 int sign (Side::LoHiSide *side*) [friend]

Returns -1 for Lo Side, +1 for Hi Side.

The documentation for this class was generated from the following file:

- **LoHiSide.H**

7.132 SideIterator Class Reference

Iterator for low and high side.

`#include <LoHiSide.H>`

Access functions

- **Side::LoHiSide operator()** () const
- bool **ok** () const

Public Member Functions

- **SideIterator** ()
- **~SideIterator** ()

Modification functions

- void **begin** ()
- void **reset** ()
- void **next** ()
- void **operator++** ()

7.132.1 Detailed Description

Iterator for low and high side.

SideIterator iterates over the values **Side::Lo** and **Side::Hi**. There are no adjustable parameters whatsoever.

7.132.2 Constructor & Destructor Documentation

7.132.2.1 SideIterator::SideIterator ()

Default, and only, constructor.

7.132.2.2 SideIterator::~SideIterator () `[inline]`

Default destructor.

7.132.3 Member Function Documentation

7.132.3.1 void SideIterator::begin ()

Set the iterator to the first side.

7.132.3.2 void SideIterator::next ()

Advance the iterator to the next face.

7.132.3.3 bool SideIterator::ok () const

Returns `true` if there is a next side, `false` if not.

7.132.3.4 Side::LoHiSide SideIterator::operator() () const

Returns the value of the side for the current iteration.

7.132.3.5 void SideIterator::operator++ ()

Advance the iterator to the next face.

7.132.3.6 void SideIterator::reset () `[inline]`

Set the iterator to the first side.

The documentation for this class was generated from the following file:

- **LoHiSide.H**

7.133 Timer Class Reference

`#include <Timer.H>`

Public Member Functions

- **Timer** ()
- **Timer** (const string &)
- **Timer** (const string &, **Timer** &)
- **Timer** (const string &, **Timer** &, const int)
- **~Timer** ()
- void **setup** ()
- void **start** ()
- void **stop** ()
- void **clear** ()
- double **getTimeStampWC** ()
- double **wc_time** ()
- string **Name** ()
- long int **Count** ()

Static Public Member Functions

- void **TimerSummary** (const int bert, const int ernie)
- void **TimerSummary** (void)

7.133.1 Constructor & Destructor Documentation

7.133.1.1 Timer::Timer () [inline]

7.133.1.2 Timer::Timer (const string &) [inline]

7.133.1.3 Timer::Timer (const string &, Timer &) [inline]

7.133.1.4 Timer::Timer (const string &, Timer &, const *int*) [inline]

7.133.1.5 Timer::~Timer () [inline]

7.133.2 Member Function Documentation

7.133.2.1 void Timer::clear () [inline]

7.133.2.2 long int Timer::Count () [inline]

7.133.2.3 double Timer::getTimeStampWC () [inline]

7.133.2.4 string Timer::Name () [inline]

7.133.2.5 void Timer::setup () [inline]

7.133.2.6 void Timer::start () [inline]

7.133.2.7 void Timer::stop () [inline]

7.133.2.8 void Timer::TimerSummary (void) [inline, static]

7.133.2.9 void Timer::TimerSummary (const int *bert*, const int *ernie*) [inline, static]

7.133.2.10 double Timer::wc_time () [inline]

The documentation for this class was generated from the following file:

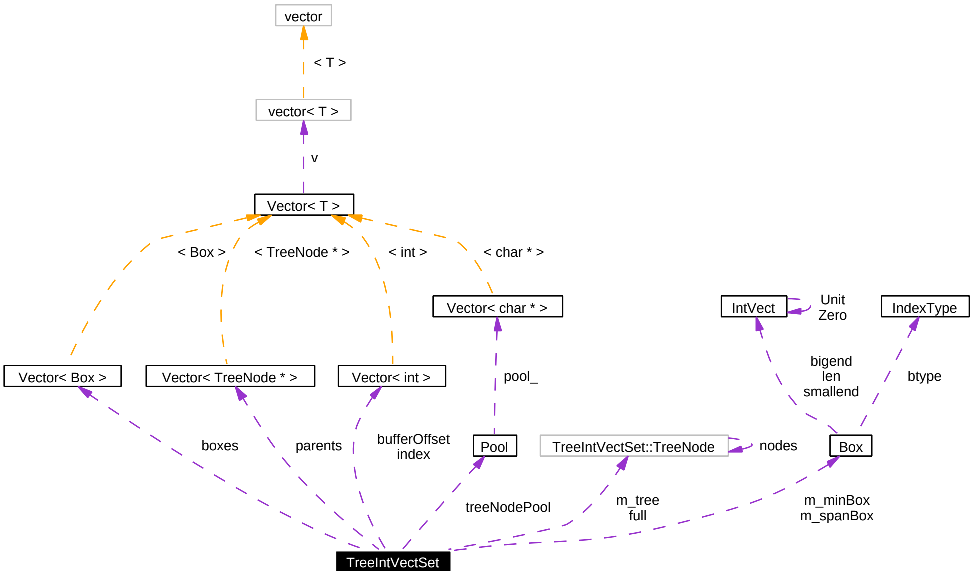
- **Timer.H**

7.134 TreeIntVectSet Class Reference

IntVectSet implementation based on tree representation.

`#include <TreeIntVectSet.H>`

Collaboration diagram for TreeIntVectSet:



Public Member Functions

- **TreeIntVectSet** ()
- **TreeIntVectSet** (const **Box** &)
- **TreeIntVectSet** (const **TreeIntVectSet** &a_sivs)
- ~**TreeIntVectSet** ()
- void **define** (const **Box** &)
- void **define** (const **TreeIntVectSet** &a_sivs)
- **TreeIntVectSet** & **operator=** (const **TreeIntVectSet** &a_sivs)
- **TreeIntVectSet** & **operator\|=** (const **TreeIntVectSet** &a_sivs)

  or

- **TreeIntVectSet** & **operator\|=** (const **IntVect** &a_iv)
- **TreeIntVectSet** & **operator\|=** (const **Box** &a_box)
- **TreeIntVectSet** & **operator &=** (const **TreeIntVectSet** &s_sivs)

  and

- **TreeIntVectSet** & **operator &=** (const **Box** &a_box)

  and

- **TreeIntVectSet** & **operator &=** (const **ProblemDomain** &a_domain)

  and

- **TreeIntVectSet** & **operator-=** (const **TreeIntVectSet** &a_sivs)

not

- **TreeIntVectSet** & **operator-=** (const **IntVect** &a_iv)

  not

- **TreeIntVectSet** & **operator-=** (const **Box** &a_box)

  not

- bool **operator==** (const **TreeIntVectSet** &lhs) const

  returns true if

- **Vector**< **Box** > **createBoxes** () const

  Returns **Vector<Box>** *representation of this* **IntVectSet**.

- void **createBoxes** (**Vector**< **Box** > &boxes, int &size) const

  Returns **Vector<Box>** *representation of this* **IntVectSet**.

- bool **contains** (const **IntVect** &iv) const
- bool **contains** (const **Box** &box) const
- **TreeIntVectSet chop** (int idir, int chop_pnt)
- void **grow** (int igrow)

  See also:
  IntVectSet::grow

- void **grow** (int idir, int igrow)
- void **refine** (int iref=2)

  fast if iref is power of 2

- void **coarsen** (int iref=2)

  fast if iref is power of 2

- void **shift** (const **IntVect** &iv)

  slow operation

- void **clear** ()
- void **nestingRegion** (int a_radius, const **Box** &a_domain, int granularity)
- void **nestingRegion** (int a_radius, const **ProblemDomain** &a_domain, int granularity)
- const **Box** & **minBox** () const
- bool **isEmpty** () const
- int **numPts** () const
- void **compact** () const
- void **recalcMinBox** () const

Linearization routines

- int **linearSize** () const
- void **linearIn** (const void *const inBuf)
- void **linearOut** (void *const a_outBuf) const

Friends

- class **TreeIntVectSetIterator**
- struct **Flag**
- class **IntVectSet**
- void **dumpTree** (const **TreeIntVectSet** ∗set)

7.134.1 Detailed Description

IntVectSet implementation based on tree representation.

For explanations of these functions please look at **IntVectSet** class when the documentation doesn't appear here.

See also:
IntVectSet

```
 Further details of how non-recursive TreeNode design works:

 (for a 2D tree)

   (m_tree)
          + -- 0

            (a)+ - 0
                   1
               1   +
                   1  $<$------you are here

               + - + - 0
                   0   1
                   0   1
                   0   0

 for the node indicated, the 'index' vector  would contain

 index=[ 0      1    3  ..............]
 parents=[&m_tree &a   .................]

or directly refered to as m_tree.nodes[1].nodes[3]

 the tree indicates a covering of an index space in either
 1 or 0.   1 is stored in the tree by pointing at the static
 'full' data member, 0 is stored as a 0.

 every 'nodes' member of a TreeNode object can be either

  0, &full, or a pointer .

  The interpretation of the tree depends on what m_spanBox is.
    nodes[i] indicates whether the i'th quadrant of the parent
     Box is full, empty, or needs to be parsed deeper.
```

7.134.2 Constructor & Destructor Documentation

7.134.2.1 TreeIntVectSet::TreeIntVectSet () `[inline]`

7.134.2.2 TreeIntVectSet::TreeIntVectSet (const Box &) `[inline]`

7.134.2.3 TreeIntVectSet::TreeIntVectSet (const TreeIntVectSet & *a_sivs*) `[inline]`

7.134.2.4 TreeIntVectSet::~TreeIntVectSet () `[inline]`

7.134.3 Member Function Documentation

7.134.3.1 TreeIntVectSet TreeIntVectSet::chop (int *idir*, int *chop_pnt*)

somewhat expensive, but shouldn't be ;-)

See also:
IntVectSet::chop

7.134.3.2 void TreeIntVectSet::clear ()

7.134.3.3 void TreeIntVectSet::coarsen (int *iref* = 2)

fast if iref is power of 2

fast if iref is power of 2

7.134.3.4 void TreeIntVectSet::compact () const

7.134.3.5 bool TreeIntVectSet::contains (const Box & *box*) const

7.134.3.6 bool TreeIntVectSet::contains (const IntVect & *iv*) const

7.134.3.7 void TreeIntVectSet::createBoxes (Vector< Box > & *boxes*, int & *size*) const

Returns **Vector<Box>** representation of this **IntVectSet**.

Returns **Vector<Box>** representation of this **IntVectSet**.

7.134.3.8 Vector<Box> TreeIntVectSet::createBoxes () const

Returns **Vector<Box>** representation of this **IntVectSet**.

Returns **Vector<Box>** representation of this **IntVectSet**.

7.134.3.9 void TreeIntVectSet::define (const TreeIntVectSet & *a_sivs*)

7.134.3.10 void TreeIntVectSet::define (const Box &)

7.134.3.11 void TreeIntVectSet::grow (int *idir*, int *igrow*)

expensive

7.134.3.12 void TreeIntVectSet::grow (int *igrow*)

See also:
IntVectSet::grow

expensive

7.134.3.13 bool TreeIntVectSet::isEmpty () const

7.134.3.14 void TreeIntVectSet::linearIn (const void *const *inBuf*)

7.134.3.15 void TreeIntVectSet::linearOut (void *const *a_outBuf*) const

7.134.3.16 int TreeIntVectSet::linearSize () const

7.134.3.17 const Box & TreeIntVectSet::minBox () const [inline]

7.134.3.18 void TreeIntVectSet::nestingRegion (int *a_radius*, const ProblemDomain & *a_domain*, int *granularity*)

7.134.3.19 void TreeIntVectSet::nestingRegion (int *a_radius*, const Box & *a_domain*, int *granularity*)

7.134.3.20 int TreeIntVectSet::numPts () const

7.134.3.21 TreeIntVectSet& TreeIntVectSet::operator &= (const ProblemDomain & *a_domain*)

and

7.134.3.22 TreeIntVectSet& TreeIntVectSet::operator &= (const Box & *a_box*)

and

7.134.3.23 TreeIntVectSet& TreeIntVectSet::operator &= (const TreeIntVectSet & *s_sivs*)

and

7.134.3.24 TreeIntVectSet& TreeIntVectSet::operator-= (const Box & *a_box*)

not

7.134.3.25 TreeIntVectSet & TreeIntVectSet::operator-= (const IntVect & *a_iv*) [inline]

not

7.134.3.26 TreeIntVectSet& TreeIntVectSet::operator-= (const TreeIntVectSet & *a_sivs*)

not

7.134.3.27 TreeIntVectSet& TreeIntVectSet::operator= (const TreeIntVectSet & *a_sivs*)

7.134.3.28 bool TreeIntVectSet::operator== (const TreeIntVectSet & *lhs*) const

returns true if

7.134.3.29 TreeIntVectSet& TreeIntVectSet::operator\|= (const Box & *a_box*)

7.134.3.30 TreeIntVectSet & TreeIntVectSet::operator\|= (const IntVect & *a_iv*) [inline]

7.134.3.31 TreeIntVectSet& TreeIntVectSet::operator\|= (const TreeIntVectSet & *a_sivs*)

or

7.134.3.32 void TreeIntVectSet::recalcMinBox () const

7.134.3.33 void TreeIntVectSet::refine (int *iref* = 2)

fast if iref is power of 2

fast if iref is power of 2

7.134.3.34 void TreeIntVectSet::shift (const IntVect & *iv*)

slow operation

7.134.4 Friends And Related Function Documentation

7.134.4.1 void dumpTree (const TreeIntVectSet * *set*) [friend]

7.134.4.2 friend struct Flag [friend]

7.134.4.3 friend class IntVectSet [friend]

7.134.4.4 friend class TreeIntVectSetIterator [friend]

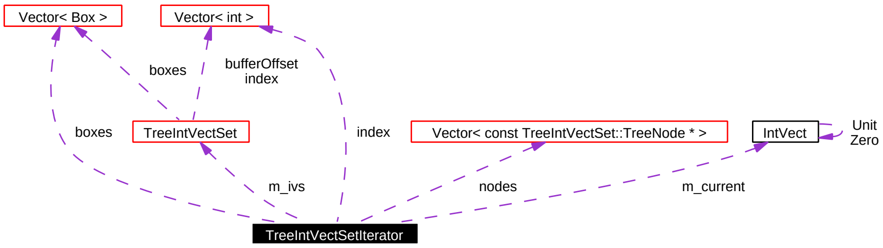
The documentation for this class was generated from the following file:

- **TreeIntVectSet.H**

7.135 TreeIntVectSetIterator Class Reference

`#include <TreeIntVectSet.H>`

Collaboration diagram for TreeIntVectSetIterator:



Public Member Functions

- **TreeIntVectSetIterator** ()
- **TreeIntVectSetIterator** (const **TreeIntVectSet** &ivs)
- void **define** (const **TreeIntVectSet** &ivs)
- const **IntVect** & **operator()** () const
- bool **ok** () const
- void **operator++** ()
- void **begin** ()
- void **end** ()
- void **clear** ()

7.135.1 Constructor & Destructor Documentation

7.135.1.1 TreeIntVectSetIterator::TreeIntVectSetIterator () `[inline]`

7.135.1.2 TreeIntVectSetIterator::TreeIntVectSetIterator (const TreeIntVectSet & *ivs*) `[inline]`

7.135.2 Member Function Documentation

7.135.2.1 void TreeIntVectSetIterator::begin ()

7.135.2.2 void TreeIntVectSetIterator::clear () `[inline]`

7.135.2.3 void TreeIntVectSetIterator::define (const TreeIntVectSet & *ivs*) `[inline]`

7.135.2.4 void TreeIntVectSetIterator::end () `[inline]`

7.135.2.5 bool TreeIntVectSetIterator::ok () const `[inline]`

7.135.2.6 const IntVect & TreeIntVectSetIterator::operator() () const `[inline]`

7.135.2.7 void TreeIntVectSetIterator::operator++ () `[inline]`

The documentation for this class was generated from the following file:

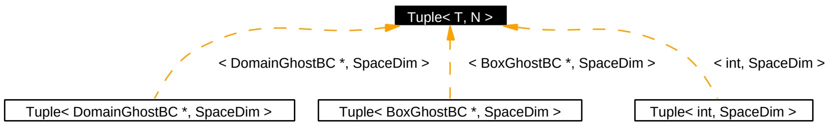
- **TreeIntVectSet.H**

7.136 Tuple< T, N > Class Template Reference

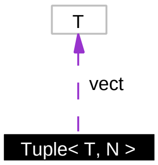
Ordered Tuples for Types T.

`#include <Tuple.H>`

Inheritance diagram for Tuple< T, N >:



Collaboration diagram for Tuple< T, N >:



Public Member Functions

- **Tuple** ()
- **Tuple** (const **Tuple** &rhs)
- **Tuple** & **operator=** (const **Tuple** &rhs)
- T & **operator**[ ] (int i)
- const T & **operator**[ ] (int i) const
- **operator const T** ∗ () const

Protected Attributes

- T **vect** [N]

7.136.1 Detailed Description

template<class T, size_t N> class Tuple< T, N >

Ordered Tuples for Types T.

This class represents ordered tuples of some user-specified concrete type T for N > 0. The type T must have a default constructor. If the non-default constructor, copy constructor, or copy assignment operator are used, T must also have a copy constructor.

7.136.2 Constructor & Destructor Documentation

7.136.2.1 template<class T, size_t N> Tuple< T, N >::Tuple () [inline]

: The default constructor. For user-defined types T, the default constructor for T will be run on each of the N objects in the Tuple. For builtin (intrinsic) types, the values in the Tuple will be garbage.

7.136.2.2 template<class T, size_t N> Tuple< T, N >::Tuple (const Tuple< T, N > & *rhs*)

: Constructs a Tuple, initializing the elements in the Tuple with the corresponding elements in the vector v. This assumes that v contains at least N elements of type T – an assumption that is NOT checked. For user-defined types, T must have a well-defined and accessible copy constructor.

7.136.3 Member Function Documentation

7.136.3.1 template<class T, size_t N> Tuple< T, N >::operator const T * () const [inline]

: Returns the address of the underlying vector of T representation. This should ONLY be used when interfacing to Fortran as it breaks the encapsulation of the class.

7.136.3.2 template<class T, size_t N> Tuple< T, N > & Tuple< T, N >::operator= (const Tuple< T, N > & *rhs*)

7.136.3.3 template<class T, size_t N> const T & Tuple< T, N >::operator[] (int *i*) const [inline]

: Returns a constant reference to the i'th element in the Tuple, counting from zero. Performs range checking when the library is compiled in debug mode.

7.136.3.4 template<class T, size_t N> T & Tuple< T, N >::operator[] (int *i*) [inline]

: Returns a modifiable lvalue reference to the i'th element in the Tuple, counting from zero. Performs range checking when the library is compiled in debug mode.

7.136.4 Member Data Documentation

7.136.4.1 template<class T, size_t N> T Tuple< T, N >::vect[N] [protected]

The documentation for this class was generated from the following file:

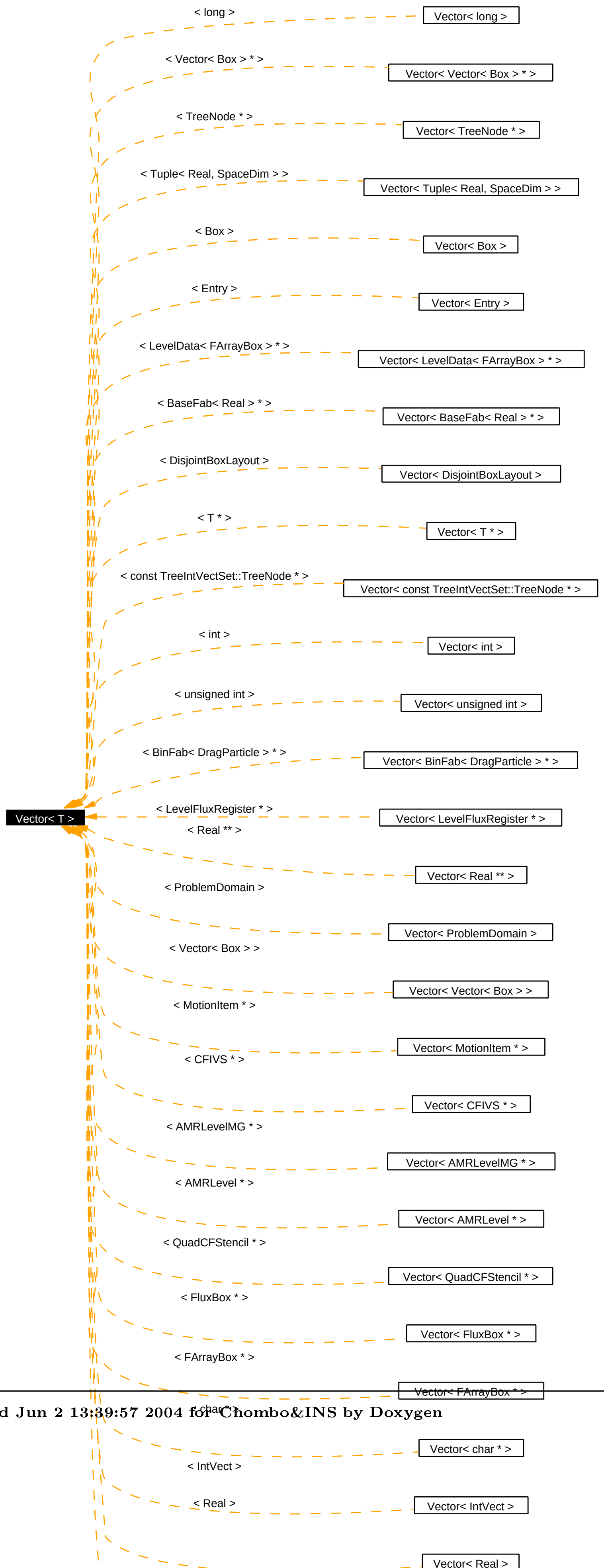
- **Tuple.H**

7.137 Vector< T > Class Template Reference

one dimensional dynamic array

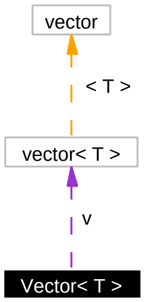
`#include <Vector.H>`

Inheritance diagram for Vector< T >:



Vector< T >
< long >
Vector< long >
< Vector< Box > * >
Vector< Vector< Box > * >
< TreeNode * >
Vector< TreeNode * >
< Tuple< Real, SpaceDim > >
Vector< Tuple< Real, SpaceDim > >
< Box >
Vector< Box >
< Entry >
Vector< Entry >
< LevelData< FArrayBox > * >
Vector< LevelData< FArrayBox > * >
< BaseFab< Real > * >
Vector< BaseFab< Real > * >
< DisjointBoxLayout >
Vector< DisjointBoxLayout >
< T * >
Vector< T * >
< const TreeIntVectSet::TreeNode * >
Vector< const TreeIntVectSet::TreeNode * >
< int >
Vector< int >
< unsigned int >
Vector< unsigned int >
< BinFab< DragParticle > * >
Vector< BinFab< DragParticle > * >
< LevelFluxRegister * >
Vector< LevelFluxRegister * >
< Real ** >
Vector< Real ** >
< ProblemDomain >
Vector< ProblemDomain >
< Vector< Box > >
Vector< Vector< Box > >
< MotionItem * >
Vector< MotionItem * >
< CFIVS * >
Vector< CFIVS * >
< AMRLevelMG * >
Vector< AMRLevelMG * >
< AMRLevel * >
Vector< AMRLevel * >
< QuadCFStencil * >
Vector< QuadCFStencil * >
< FluxBox * >
Vector< FluxBox * >
< FArrayBox * >
Vector< FArrayBox * >
< char * >
Vector< char * >
< IntVect >
Vector< IntVect >
< Real >
Vector< Real >

Collaboration diagram for Vector< T >:



Public Member Functions

- **Vector** ()
- virtual ~**Vector** ()
- **Vector** (const **Vector**< T > &invec)
- **Vector** (const std::vector< T > &invec)
- **Vector**< T > & **operator=** (const std::vector< T > &invec)
- **Vector**< T > & **operator=** (const **Vector**< T > &invec)
- **Vector**< T > & **assign** (const T &inval)

    assign a scalar to every element of the vector

- **Vector** (unsigned int isize)
- void **clear** ()
- size_t **size** () const
- **Vector** (unsigned int isize, const T &value)
- T & **operator**[ ] (unsigned int n)
- const T & **operator**[ ] (unsigned int n) const
- void **swap** (**Vector**< T > &other)
- void **push_back** (const T &in)
- void **append** (const **Vector**< T > &invec)
- void **resize** (unsigned int isize)
- void **reserve** (size_t isize)
- size_t **capacity** () const
- void **resize** (unsigned int isize, const T &value)
- void **sort** ()

7.137.1 Detailed Description

template<class T> class Vector< T >

one dimensional dynamic array

Vector is a resizable one-dimensional array with constant-time random access and range checking. The template type T must have a default constructor, a copy constructor, and an assignment operator.

Vector<T> is publically derived from std::vector<T>. Read the friendly STL manual.

7.137.2 Constructor & Destructor Documentation

7.137.2.1 template<class T> Vector< T >::Vector () `[inline]`

Default constructor. Creates a Vector of zero length with null data.

7.137.2.2 template<class T> virtual Vector< T >::~Vector () `[inline, virtual]`

Destructor.

7.137.2.3 template<class T> Vector< T >::Vector (const Vector< T > & *invec*) `[inline]`

Copy constructor.

7.137.2.4 template<class T> Vector< T >::Vector (const std::vector< T > & *invec*) `[inline]`

conversion constructor

7.137.2.5 template<class T> Vector< T >::Vector (unsigned int *isize*) `[inline]`

Constructs a Vector with given number of elements.\

{ Arguments:} \ size (not modified): number of elements of Vector to construct.\ { This:} \ ——-The object is modified———-

7.137.2.6 template<class T> Vector< T >::Vector (unsigned int *isize*, const T & *value*) `[inline]`

Constructs a Vector with given number of elements and constant value.\

{ Arguments:} \ size (not modified): number of elements of Vector to construct.\ value (not modified): value to set every element to.\ { This:} \ ——-The object is modified———-

7.137.3 Member Function Documentation

7.137.3.1 template<class T> void Vector< T >::append (const Vector< T > & *invec*) `[inline]`

Modifies this Vector by appending the elements of the argument Vector. The new Vector will have a size of this->**size()** + invec.size() (where this Vector is considered before the append is performed). The first element of invec will have index this->**size()**, the second element will have index this->**size()**+1, etc.\

{ Arguments:} \ invec (not modified): Vector whose elements to append to this Vector.\ { This:} \ ——-The object is modified———-

7.137.3.2 template<class T> Vector<T>& Vector< T >::assign (const T & *inval*) [inline]

assign a scalar to every element of the vector

[NOTE: cant use **operator=()** for this because it would be ambiguous with the (int) constructor for number T types.]

7.137.3.3 template<class T> size_t Vector< T >::capacity () const [inline]

7.137.3.4 template<class T> void Vector< T >::clear () [inline]

7.137.3.5 template<class T> Vector<T>& Vector< T >::operator= (const Vector< T > & *invec*) [inline]

7.137.3.6 template<class T> Vector<T>& Vector< T >::operator= (const std::vector< T > & *invec*) [inline]

7.137.3.7 template<class T> const T& Vector< T >::operator[] (unsigned int *n*) const [inline]

Returns a constant reference to the given element in this Vector.\

{ Arguments: } \ n (not modified) index of desired element.\ { Returns:} \ constant reference to value in Vector at index n.\ { This:} \ This object is not modified.

7.137.3.8 template<class T> T& Vector< T >::operator[] (unsigned int *n*) [inline]

Returns a modifiable lvalue reference to the value of the given element in this Vector. It is an error if n < 0 or n >= this->**size()**. \

{ Arguments: } \ n (not modified) index of desired element.\ { Returns:} \ modifiable reference to value in Vector at index n.\ { This:} \ —– This object is modified if the returned reference is assigned a new value —–

7.137.3.9 template<class T> void Vector< T >::push_back (const T & *in*) [inline]

7.137.3.10 template<class T> void Vector< T >::reserve (size_t *isize*) [inline]

7.137.3.11 template<class T> void Vector< T >::resize (unsigned int *isize*, const T & *value*) [inline]

7.137.3.12 template<class T> void Vector< T >::resize (unsigned int *isize*) [inline]

7.137.3.13 template<class T> size_t Vector< T >::size () const [inline]

size function. returns current size of Vector

7.137.3.14 template<class T> void Vector< T >::sort () [inline]

7.137.3.15 template<class T> void Vector< T >::swap (Vector< T > & *other*) [inline]

The documentation for this class was generated from the following file:

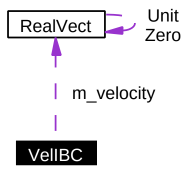
- **Vector.H**

7.138 VelIBC Class Reference

IBC for simple Velocity advection (solid wall) boundary conditions.

`#include <VelIBC.H>`

Collaboration diagram for VelIBC:



Public Member Functions

- **VelIBC** ()

  Null constructor.

- **VelIBC** (const **Real** &a_gamma, const **Real** &a_ambientDensity, const **Real** &a_deltaDensity, const int &a_pressure, const **RealVect** &a_center, const **Real** &a_size, const **RealVect** &a_velocity)

  Constructor which defines parameters used by Fortran routines.

- ∼**VelIBC** ()

  Destructor.

- PhysIBC ∗ **new_physIBC** ()

  Factory method - this object is its own factory.

- void **setDefaultValues** ()

  set default boundary values

- void **fluxBC** (**FArrayBox** &a_F, const **FArrayBox** &a_W, const **FArrayBox** &a_Wextrap, const int &a_dir, const **Side::LoHiSide** &a_side, const **Real** &a_time)

  Set boundary fluxes.

- void **setBdrySlopes** (**FArrayBox** &a_dW, const **FArrayBox** &a_W, const int &a_dir, const **Real** &a_time)

  Set boundary slopes.

- void **initialize** (**LevelData**< **FArrayBox** > &a_U)

  Set up initial conditions.

- void **advectionVel** (const **RealVect** &a_advVel)

  set velocity

- const **RealVect** & **advectionVel** () const
- void **probType** (const int a_probtype)

set problem type

- void **setNormalWallVel** (**Real** a_bcVal, int a_dir, **Side::LoHiSide** a_hiLo)
  set boundary value (default is 0)
- **Real getNormalWallVel** (int a_dir, **Side::LoHiSide** a_hiLo) const
  access boundary value
- void **setSlopeValue** (**Real** a_slopeVal, int a_dir, **Side::LoHiSide** a_hiLo)
  set slope value (default is 0)
- **Real getSlopeValue** (int a_dir, **Side::LoHiSide** a_hiLo) const
  access slope value
- int **probType** () const
  accessor

Protected Attributes

- **RealVect m_velocity**
  advection velocity field
- **Real m_bcVal** [**SpaceDim**][2]
- **Real m_slopeVal** [**SpaceDim**][2]
- bool **m_isBCvalSet**
- bool **m_isSlopeValSet**
- int **m_probtype**
  initial condition type

7.138.1 Detailed Description

IBC for simple Velocity advection (solid wall) boundary conditions.

This IBC sets normal component of velocities to the prescribed value (normally 0) at physical boundaries, and uses extrapolation to set tangential velocities.

7.138.2 Constructor & Destructor Documentation

7.138.2.1 VelIBC::VelIBC ()

Null constructor.

7.138.2.2 VelIBC::VelIBC (const Real & *a_gamma*, const Real & *a_ambientDensity*, const Real & *a_deltaDensity*, const int & *a_pressure*, const RealVect & *a_center*, const Real & *a_size*, const RealVect & *a_velocity*)

Constructor which defines parameters used by Fortran routines.

7.138.2.3 VelIBC::~VelIBC () [inline]

Destructor.

7.138.3 Member Function Documentation

7.138.3.1 const RealVect& VelIBC::advectionVel () const

7.138.3.2 void VelIBC::advectionVel (const RealVect & *a_advVel*)

set velocity

7.138.3.3 void VelIBC::fluxBC (FArrayBox & *a_F*, const FArrayBox & *a_W*, const FArrayBox & *a_Wextrap*, const int & *a_dir*, const Side::LoHiSide & *a_side*, const Real & *a_time*)

Set boundary fluxes.

7.138.3.4 Real VelIBC::getNormalWallVel (int *a_dir*, Side::LoHiSide *a_hiLo*) const

access boundary value

7.138.3.5 Real VelIBC::getSlopeValue (int *a_dir*, Side::LoHiSide *a_hiLo*) const

access slope value

7.138.3.6 void VelIBC::initialize (LevelData< FArrayBox > & *a_U*)

Set up initial conditions.

7.138.3.7 PhysIBC* VelIBC::new_physIBC ()

Factory method - this object is its own factory.

Return a pointer to a new PhysIBC object with m_isDefined = false (i.e., its define() must be called before it is used) and m_isFortranCommonSet set to value of m_isFortranCommonset in the current (factory) object.

7.138.3.8 int VelIBC::probType () const

accessor

7.138.3.9 void VelIBC::probType (const int *a_probtype*)

set problem type

7.138.3.10 void VelIBC::setBdrySlopes (FArrayBox & *a_dW*, const FArrayBox & *a_W*, const int & *a_dir*, const Real & *a_time*)

Set boundary slopes.

The boundary slopes in a_dW are already set to one sided difference approximations. If this function doesn't change them they will be used for the slopes at the boundaries.

7.138.3.11 void VelIBC::setDefaultValues ()

set default boundary values

7.138.3.12 void VelIBC::setNormalWallVel (Real *a_bcVal*, int *a_dir*, Side::LoHiSide *a_hiLo*)

set boundary value (default is 0)

7.138.3.13 void VelIBC::setSlopeValue (Real *a_slopeVal*, int *a_dir*, Side::LoHiSide *a_hiLo*)

set slope value (default is 0)

7.138.4 Member Data Documentation

7.138.4.1 Real VelIBC::m_bcVal[SpaceDim][2] `[protected]`

7.138.4.2 bool VelIBC::m_isBCvalSet `[protected]`

7.138.4.3 bool VelIBC::m_isSlopeValSet `[protected]`

7.138.4.4 int VelIBC::m_probtype `[protected]`

initial condition type

7.138.4.5 Real VelIBC::m_slopeVal[SpaceDim][2] `[protected]`

7.138.4.6 RealVect VelIBC::m_velocity `[protected]`

advection velocity field

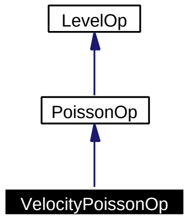
The documentation for this class was generated from the following file:

- **VelIBC.H**

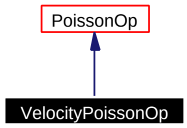
7.139 VelocityPoissonOp Class Reference

`#include <VelocityPoissonOp.H>`

Inheritance diagram for VelocityPoissonOp:



Collaboration diagram for VelocityPoissonOp:



Public Member Functions

- **LevelOp** ∗ **new_levelop** () const
- **VelocityPoissonOp** ()
- ∼**VelocityPoissonOp** ()
- void **applyOpI** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiCoarse, **LevelData**< **FArrayBox** > &a_LOfPhi, **Vector**< **DomainGhostBC** > &a_bcVect)
- virtual bool **isDefined** () const

  Has this object been defined?

7.139.1 Constructor & Destructor Documentation

7.139.1.1 VelocityPoissonOp::VelocityPoissonOp ()

7.139.1.2 VelocityPoissonOp::∼VelocityPoissonOp ()

7.139.2 Member Function Documentation

7.139.2.1 void VelocityPoissonOp::applyOpI (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCoarse*, LevelData< FArrayBox > & *a_LOfPhi*, Vector< DomainGhostBC > & *a_bcVect*)

The raison d'etre for this class – want to be able to apply operator with different BCs for each component in a_phi.

7.139.2.2 virtual bool VelocityPoissonOp::isDefined () const [virtual]

Has this object been defined?

Reimplemented from **PoissonOp** (p. 505).

7.139.2.3 LevelOp* VelocityPoissonOp::new_levelop () const [virtual]

virtual constructor workaround

Reimplemented from **PoissonOp** (p. 506).

The documentation for this class was generated from the following file:

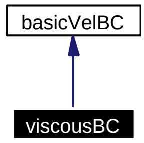
- **VelocityPoissonOp.H**

7.140 viscousBC Class Reference

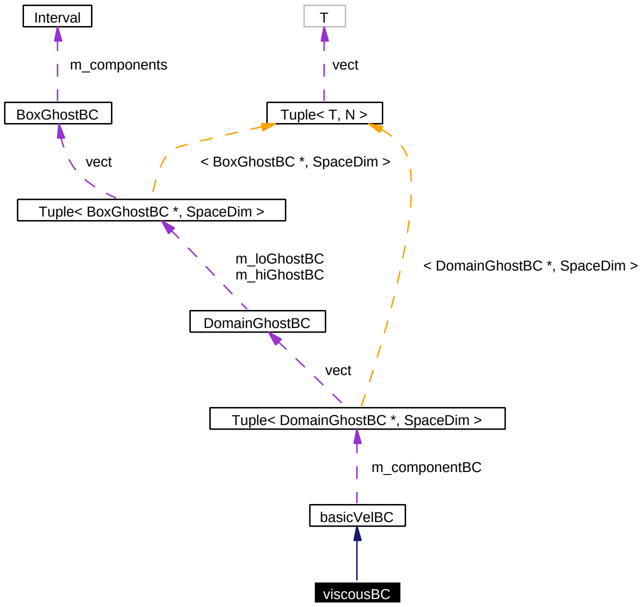
implements boundary conditions on viscous terms themselves

`#include <viscousBC.H>`

Inheritance diagram for viscousBC:



Collaboration diagram for viscousBC:



Public Member Functions

- **viscousBC** ()

    constructor

- **~viscousBC** ()
- virtual void **applyBCs** (**LevelData< FArrayBox >** &a_viscousTerm, const **ProblemDomain** &a_domain, const **Real** a_dx) const

    apply BC's

- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx) const
- virtual void **applyBCs** (**FArrayBox** &a_state, const **ProblemDomain** &a_domain, const **Real** a_dx, int a_comp) const

THIS HAS NOT BEEN IMPLEMENTED!

- virtual void **applyBCs** (**LevelData**< **FArrayBox** > &a_viscousTerm, const **Box** &a_domain, const **Real** a_dx) const

  apply BC's

- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx) const

  old interface using **Box** *instead of* **ProblemDomain**

- virtual void **applyBCs** (**FArrayBox** &a_state, const **Box** &a_domain, const **Real** a_dx, int a_comp) const

  THIS HAS NOT BEEN IMPLEMENTED!

7.140.1 Detailed Description

implements boundary conditions on viscous terms themselves

since this class just does 0th-order extrapolation for all kinds of boundaries at the moment, it's a pretty simple class

7.140.2 Constructor & Destructor Documentation

7.140.2.1 viscousBC::viscousBC ()

constructor

7.140.2.2 viscousBC::~viscousBC ()

7.140.3 Member Function Documentation

7.140.3.1 virtual void viscousBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*, int *a_comp*) const `[virtual]`

THIS HAS NOT BEEN IMPLEMENTED!

Reimplemented from **basicVelBC** (p. 89).

7.140.3.2 virtual void viscousBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*) const `[virtual]`

old interface using **Box** instead of **ProblemDomain**

Reimplemented from **basicVelBC** (p. 90).

7.140.3.3 virtual void viscousBC::applyBCs (LevelData< FArrayBox > & *a_viscousTerm*, const Box & *a_domain*, const Real *a_dx*) const `[virtual]`

apply BC's

Reimplemented from **basicVelBC** (p. 90).

7.140.3.4 virtual void viscousBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*, int *a_comp*) const [virtual]

THIS HAS NOT BEEN IMPLEMENTED!

Reimplemented from **basicVelBC** (p. 89).

7.140.3.5 virtual void viscousBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

Reimplemented from **basicVelBC** (p. 90).

7.140.3.6 virtual void viscousBC::applyBCs (LevelData< FArrayBox > & *a_viscousTerm*, const ProblemDomain & *a_domain*, const Real *a_dx*) const [virtual]

apply BC's

Reimplemented from **basicVelBC** (p. 90).

The documentation for this class was generated from the following file:

- **viscousBC.H**

Chapter 8

Chombo&INS File Documentation

8.1 AdvectIBC.H File Reference

`#include <iostream>`

`#include "LevelData.H"`

`#include "FArrayBox.H"`

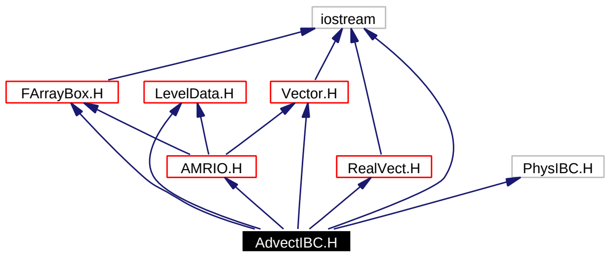
`#include "Vector.H"`

`#include "RealVect.H"`

`#include "AMRIO.H"`

`#include "PhysIBC.H"`

Include dependency graph for AdvectIBC.H:



Compounds

- class **AdvectIBC**

  IBC for simple advection.

8.2 AMR.H File Reference

`#include <iostream>`

`#include <string>`

`#include <ctime>`

`#include "REAL.H"`

`#include "Vector.H"`

`#include "AMRLevel.H"`

`#include "AMRLevelFactory.H"`

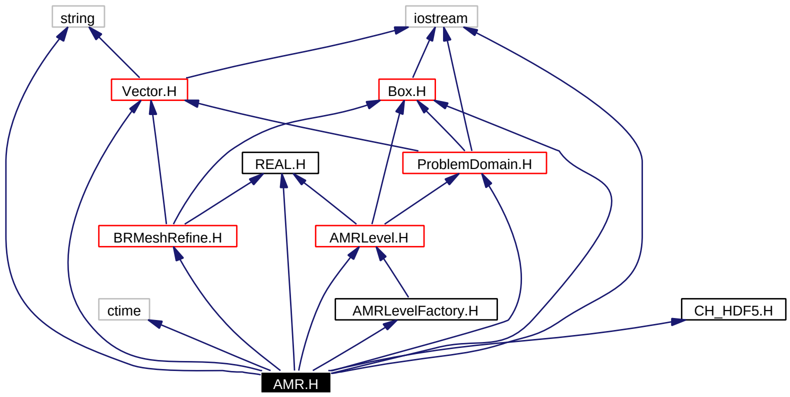
`#include "BRMeshRefine.H"`

`#include "ProblemDomain.H"`

`#include "Box.H"`

`#include "CH_HDF5.H"`

Include dependency graph for AMR.H:



Compounds

- class **AMR**

    Framework for Berger-Oliger timestepping for AMR.

8.3 AMRIO.H File Reference

`#include "REAL.H"`

`#include "LevelData.H"`

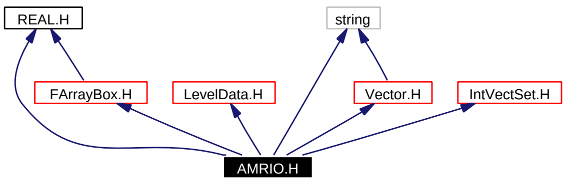
`#include "Vector.H"`

`#include "FArrayBox.H"`

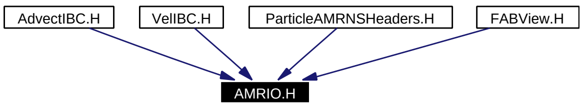
`#include "IntVectSet.H"`

`#include <string>`

Include dependency graph for AMRIO.H:



This graph shows which files directly or indirectly include this file:



Functions

- void **writeAFabASCII** (std::ostream &os, const **FArrayBox** &fab)
- void **readAFabASCII** (istream &is, **FArrayBox** &fab)
- void **WriteAMRHierarchyASCII** (const string &filename, const **Vector**< **DisjointBoxLayout** > &a_vectGrids, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_vectData, const **Vector**< string > &a_vectNames, const **Box** &a_domain, const **Real** &a_dx, const **Real** &a_dt, const **Real** &a_time, const **Vector**< int > &a_vectRatio, const int &a_numLevels)
- int **ReadAMRHierarchyASCII** (const string &filename, **Vector**< **DisjointBoxLayout** > &a_vectGrids, **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_vectData, **Vector**< string > &a_vectNames, **Box** &a_domain, **Real** &a_dx, **Real** &a_dt, **Real** &a_time, **Vector**< int > &a_vectRatio, int &a_numLevels, const **IntVect** &ghostVector=IntVect::TheZeroVector())

8.3.1 Function Documentation

8.3.1.1 void readAFabASCII (istream & *is*, FArrayBox & *fab*)

8.3.1.2 int ReadAMRHierarchyASCII (const string & *filename*, Vector< DisjointBoxLayout > & *a_vectGrids*, Vector< LevelData< FArrayBox > * > & *a_vectData*, Vector< string > & *a_vectNames*, Box & *a_domain*, Real & *a_dx*, Real & *a_dt*, Real & *a_time*, Vector< int > & *a_vectRatio*, int & *a_numLevels*, const IntVect & *ghostVector* = IntVect::TheZeroVector())

Reads hierarchy of levels in ASCII format. Returns 0 on success.

{ Arguments:}\ filename : file to output to.\ a_vectGrids : grids at each level.\ a_vectData : data at each level.\ a_vectNames: names of variables.\ a_domain : domain at coarsest level.\ a_dx : grid spacing at coarsest level.\ a_dt : time step at coarsest level.\ a_time : time.\ a_vectRatio : refinement ratio at all levels (ith entry is refinement ratio between levels i and i + 1).\ a_numLevels : number of levels to output.\ a_ghostVector : number of ghost cells on output .\

{ Returns:}\ status code with values:\ 0: success\ -1: failure\

8.3.1.3 void writeAFabASCII (std::ostream & *os*, const FArrayBox & *fab*)

8.3.1.4 void WriteAMRHierarchyASCII (const string & *filename*, const Vector< DisjointBoxLayout > & *a_vectGrids*, const Vector< LevelData< FArrayBox > * > & *a_vectData*, const Vector< string > & *a_vectNames*, const Box & *a_domain*, const Real & *a_dx*, const Real & *a_dt*, const Real & *a_time*, const Vector< int > & *a_vectRatio*, const int & *a_numLevels*)

Writes hierarchy of levels in ASCII format.

{ Arguments:}\ filename : file to output to.\ a_vectGrids : grids at each level.\ a_vectData : data at each level.\ a_vectNames: names of variables.\ a_domain : domain at coarsest level.\ a_dx : grid spacing at coarsest level.\ a_dt : time step at coarsest level.\ a_time : time.\ a_vectRatio : refinement ratio at all levels (ith entry is refinement ratio between levels i and i + 1).\ a_numLevels : number of levels to output.\

8.4 AMRLevel.H File Reference

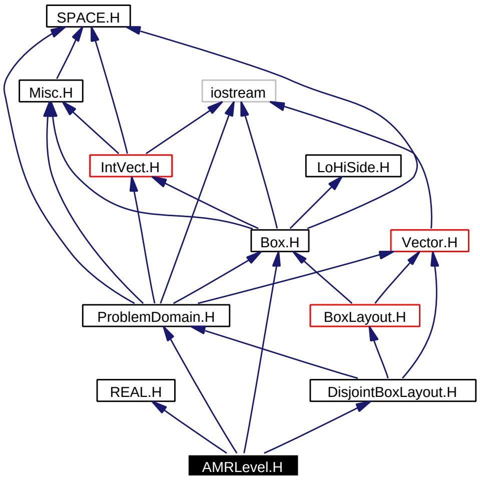
`#include "REAL.H"`

`#include "Box.H"`

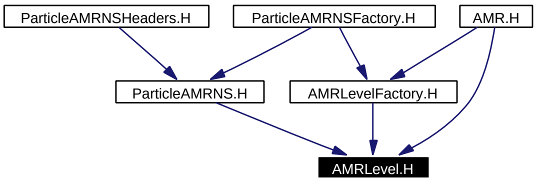
`#include "DisjointBoxLayout.H"`

`#include "ProblemDomain.H"`

Include dependency graph for AMRLevel.H:



This graph shows which files directly or indirectly include this file:



Compounds

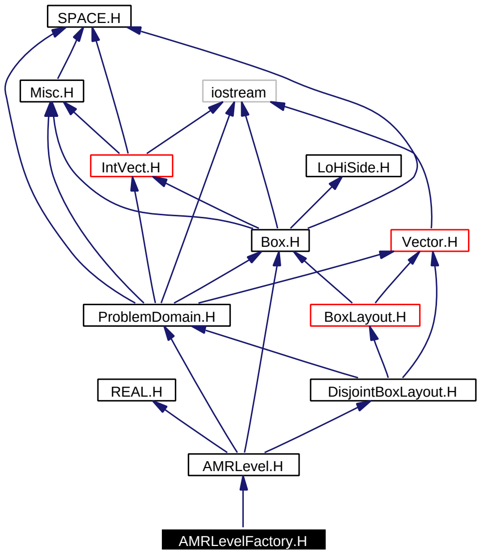
- class **AMRLevel**

  Abstract base class for time-dependent data at a level of refinement.

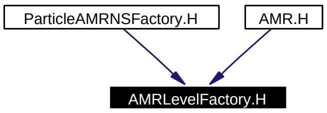
8.5 AMRLevelFactory.H File Reference

`#include "AMRLevel.H"`

Include dependency graph for AMRLevelFactory.H:



This graph shows which files directly or indirectly include this file:



Compounds

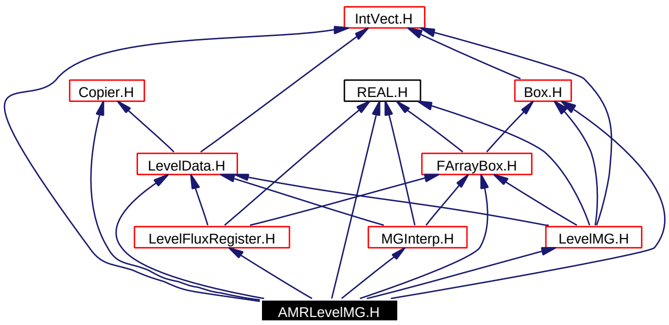
- class **AMRLevelFactory**

  Factory class to produce an **AMRLevel**.

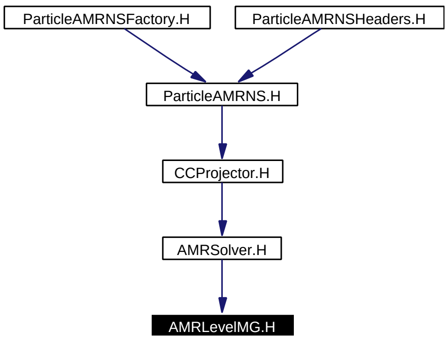
8.6 AMRLevelMG.H File Reference

```
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "FArrayBox.H"
#include "LevelData.H"
#include "LevelFluxRegister.H"
#include "LevelMG.H"
#include "MGInterp.H"
#include "Copier.H"
```

Include dependency graph for AMRLevelMG.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **AMRLevelMG**

  Class which manages data and operations on a level for **AMRSolver**.

8.7 AMRSolver.H File Reference

#include <cstdlib>

#include <cmath>

#include <iostream>

#include <cassert>

#include "REAL.H"

#include "IntVect.H"

#include "Box.H"

#include "FArrayBox.H"

#include "Vector.H"

#include "IntVectSet.H"

#include "DisjointBoxLayout.H"

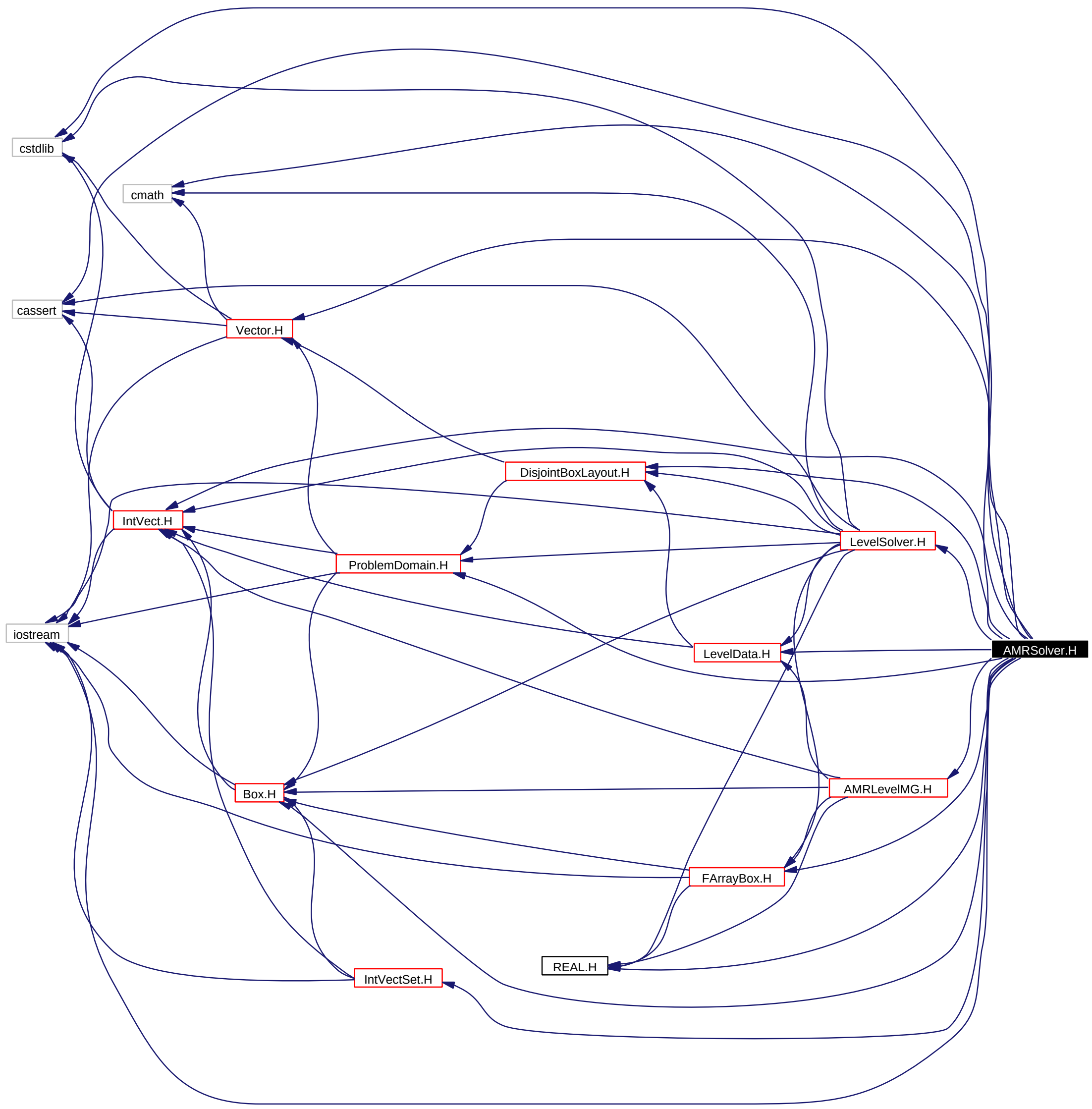
#include "LevelData.H"

#include "AMRLevelMG.H"

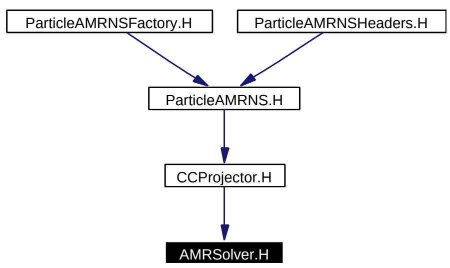
#include "LevelSolver.H"

#include "ProblemDomain.H"

Include dependency graph for AMRSolver.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **AMRSolver**

  Class which manages grid hierarchy and composite elliptic solution.

8.8 Arena.H File Reference

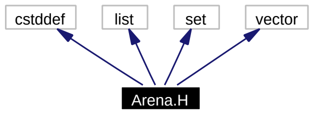
`#include <cstddef>`

`#include <list>`

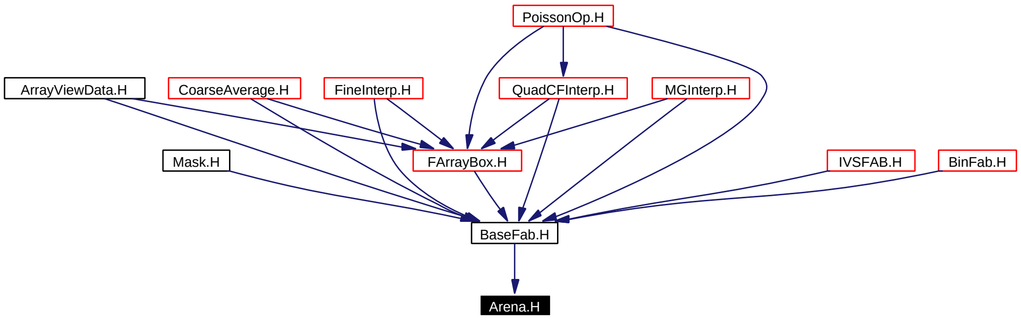
`#include <set>`

`#include <vector>`

Include dependency graph for Arena.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Arena**

  A Virtual Base Class for Dynamic Memory Managemen.

- union **Arena::Word**
- class **BArena**

  A Concrete Class for Dynamic Memory Management.

- class **CArena**

  A Concrete Class for Dynamic Memory Management.

- class **CArena::Node**

Typedefs

- typedef std::list< **Arena** * > **ArenaList**

Variables

- **Arena** * **The_FAB_Arena**

8.8.1 Typedef Documentation

8.8.1.1 typedef std::list<Arena*> ArenaList

8.8.2 Variable Documentation

8.8.2.1 Arena* The_FAB_Arena

8.9 ArrayViewData.H File Reference

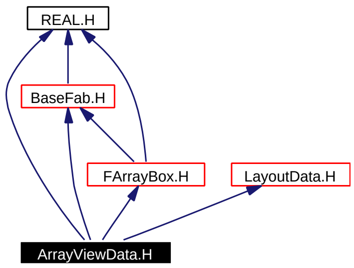
`#include "REAL.H"`

`#include "BaseFab.H"`

`#include "FArrayBox.H"`

`#include "LayoutData.H"`

Include dependency graph for ArrayViewData.H:



Compounds

- class **ArrayViewData**

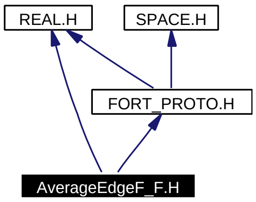
    Special **LayoutData** *type for use with ArrayView.*

8.10 AverageEdgeF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for AverageEdgeF_F.H:



Defines

- #define **FORT_AVERAGEEDGE** FORTRAN_NAME(AVERAGEEDGE ,averageedge)

Functions

- void **FORT_AVERAGEEDGE** (CHFp_FRA(coarse),CHFp_CONST_FRA(fine),CHFp_BOX(crseBox),CHFp_CONST_INT(dir),CHFp_CONST_INT(nRef),CHFp_BOX(refBox))

8.10.1 Define Documentation

8.10.1.1 #define FORT_AVERAGEEDGE FORTRAN_NAME(AVERAGEEDGE ,averageedge)

8.10.2 Function Documentation

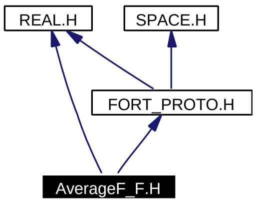
8.10.2.1 void FORT_AVERAGEEDGE (CHFp_FRA(coarse), CHFp_CONST_FRA(fine), CHFp_BOX(crseBox), CHFp_CONST_INT(dir), CHFp_CONST_INT(nRef), CHFp_BOX(refBox))

8.11 AverageF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for AverageF_F.H:



Defines

- #define **FORT_AVERAGE** FORTRAN_NAME(AVERAGE ,average)

Functions

- void **FORT_AVERAGE** (CHFp_FRA(coarse),CHFp_CONST_FRA(fine),CHFp_BOX(b),CHFp_CONST_INT(ref_ratio),CHFp_BOX(bref))

8.11.1 Define Documentation

8.11.1.1 #define FORT_AVERAGE FORTRAN_NAME(AVERAGE ,average)

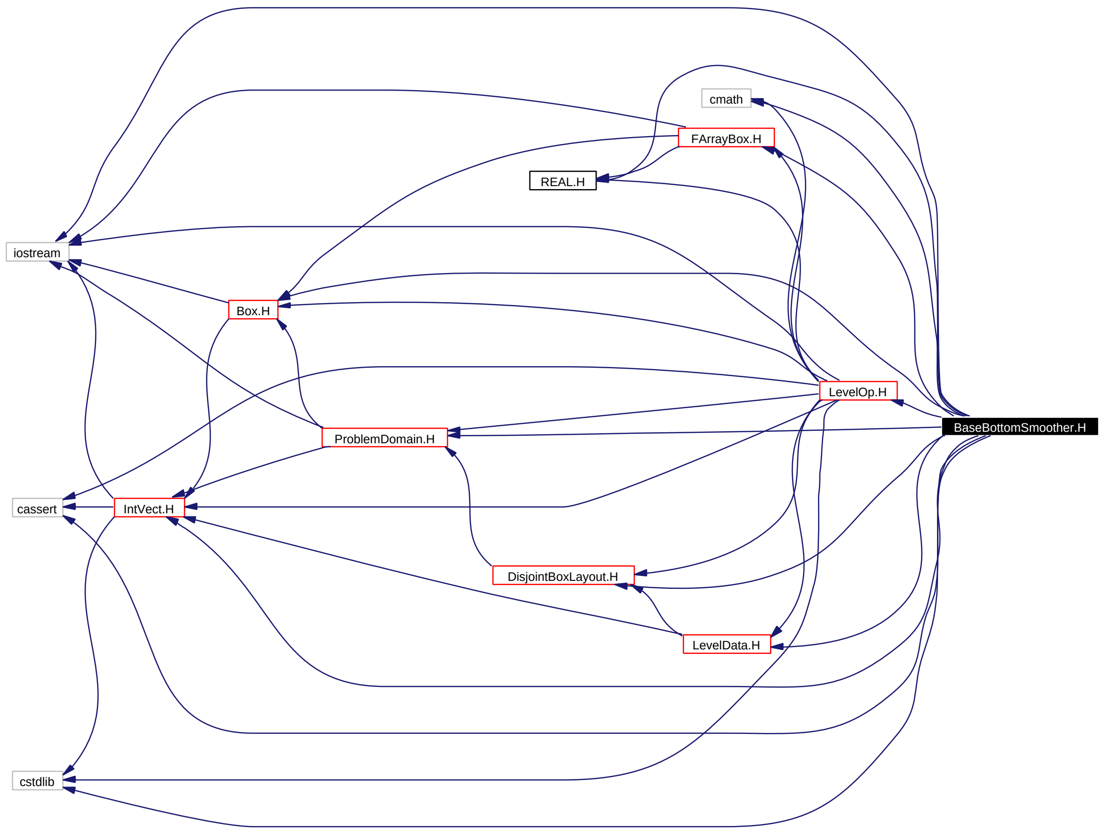
8.11.2 Function Documentation

8.11.2.1 void FORT_AVERAGE (CHFp_FRA(coarse), CHFp_CONST_FRA(fine), CHFp_BOX(b), CHFp_CONST_INT(ref_ratio), CHFp_BOX(bref))

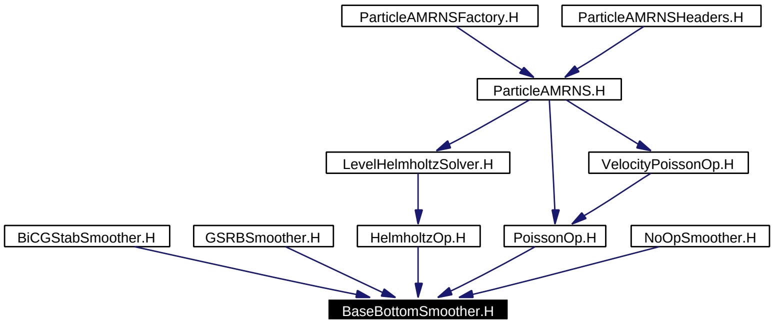
8.12 BaseBottomSmoother.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "FArrayBox.H"
#include "ProblemDomain.H"
#include "LevelOp.H"
```

Include dependency graph for BaseBottomSmoother.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BaseBottomSmoother**

    Pure base class to encapsulate bottom smooth for LevelOp-derived objects.

8.13 BaseFab.H File Reference

`#include <cstdlib>`

`#include <assert.h>`

`#include <string>`

`#include <typeinfo>`

`#include "Box.H"`

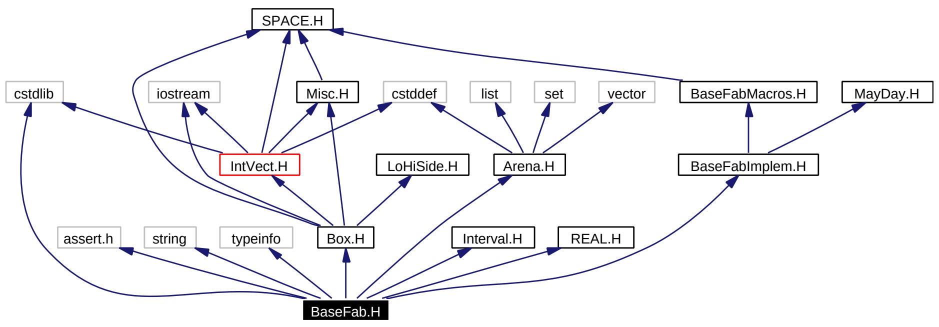
`#include "Arena.H"`

`#include "Interval.H"`

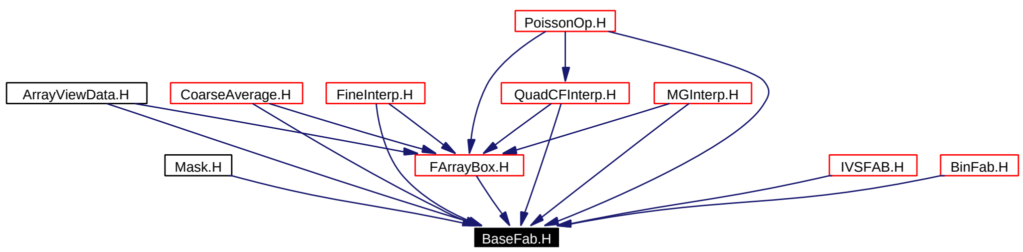
`#include "REAL.H"`

`#include "BaseFabImplem.H"`

Include dependency graph for BaseFab.H:



This graph shows which files directly or indirectly include this file:



Compounds

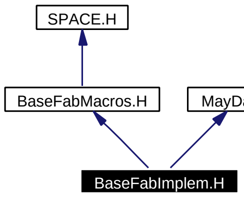
- class **BaseFab**

8.14 BaseFabImplem.H File Reference

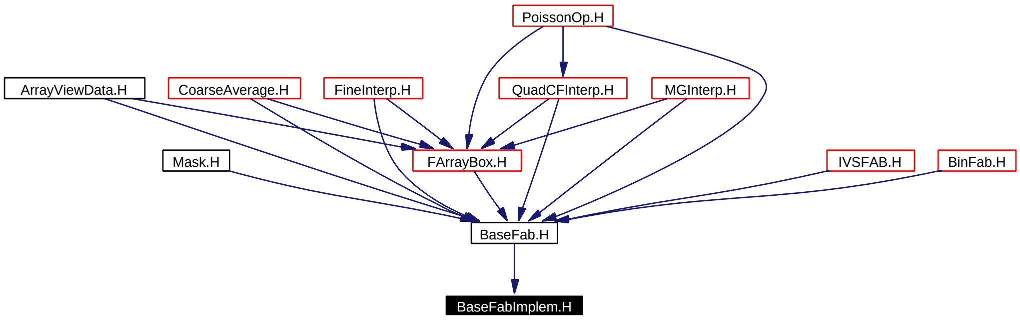
`#include "BaseFabMacros.H"`

`#include "MayDay.H"`

Include dependency graph for BaseFabImplem.H:



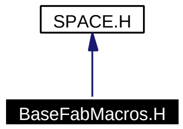
This graph shows which files directly or indirectly include this file:



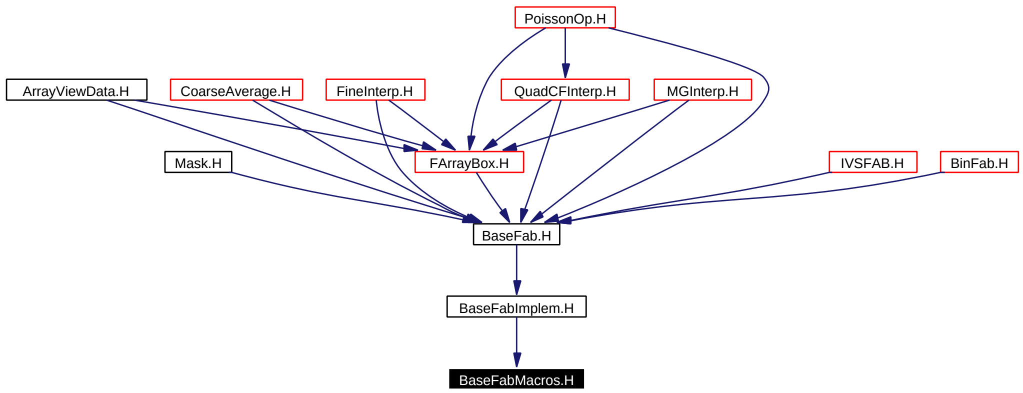
8.15 BaseFabMacros.H File Reference

`#include "SPACE.H"`

Include dependency graph for BaseFabMacros.H:



This graph shows which files directly or indirectly include this file:



Defines

- #define **ForAllXPencil**(T, x) ForAllXBPencil(T,x,((x).box()),0,((x).nComp()))
- #define **ForAllX**(T, x) ForAllXBNN(T,x,((x).box()),0,((x).nComp()))
- #define **ForAllXC**(T, x) ForAllXCBNN(T,x,((x).box()),0,((x).nComp()))
- #define **ForAllXB**(T, x, b) ForAllXBNN(T,x,(b),0,(x).nComp())
- #define **ForAllXBC**(T, x, b) ForAllXCBNN(T,x,(b),0,(x).nComp())
- #define **ForAllThis**(T) ForAllThisBNN(T,domain,0,nComp())
- #define **ForAllThisC**(T) ForAllThisCBNN(T,domain,0,nComp())
- #define **ForAllThisB**(T, b) ForAllThisBNN(T,(b),0,nComp())
- #define **ForAllThisCB**(T, b) ForAllThisCBNN(T,(b),0,nComp())
- #define **ForAllThisNN**(T, ns, nc) ForAllThisBNN(T,domain,ns,nc)
- #define **ForAllThisXC**(T, x) ForAllThisBNNXC(T,domain,0,nComp(),x,0)

8.15.1 Define Documentation

8.15.1.1 #define ForAllThis(T) ForAllThisBNN(T,domain,0,nComp())

The macro **ForAllThis(T)** is a shortened form of 'ForAllThisBNN' where the **Box** defaults to the domain of x and the components run over all the components of x.

8.15.1.2 #define ForAllThisB(T, b) ForAllThisBNN(T,(b),0,nComp())

The macro **ForAllThisB(T,b)** is a shortened form of 'ForAllThisBNN' where the components run over all the components of x.

8.15.1.3 #define ForAllThisC(T) ForAllThisCBNN(T,domain,0,nComp())

The macro **ForAllThisC(T)** is the constant form of **ForAllThis(T)**.

8.15.1.4 #define ForAllThisCB(T, b) ForAllThisCBNN(T,(b),0,nComp())

The macro **ForAllThisCB(T,b)** is the constant form of **ForAllThisB(T,b)**.

8.15.1.5 #define ForAllThisNN(T, ns, nc) ForAllThisBNN(T,domain,ns,nc)

The macro **ForAllThisNN(T,ns,nc)** is a shortened form of 'ForAllThisBNN' where the **Box** defaults to the domain of *this.

8.15.1.6 #define ForAllThisXC(T, x) ForAllThisBNNXC(T,domain,0,nComp(),x,0)

The macro **ForAllThisXC(T,x)** is a shortened form of 'ForAllThisBNNXC' where the **Box** defaults to the domain of *this and the components run over all the components of *this.

8.15.1.7 #define ForAllX(T, x) ForAllXBNN(T,x,((x).box()),0,((x).nComp()))

The macro **ForAllX(T,x)** is a shortened form of 'ForAllXBNN' where the **Box** defaults to the domain of x and the components run over all the components of x.

8.15.1.8 #define ForAllXB(T, x, b) ForAllXBNN(T,x,(b),0,(x).nComp())

The macro **ForAllXB(T,x,b)** is a shortened form of 'ForAllXBNN' where the components run over all the components of x.

8.15.1.9 #define ForAllXBC(T, x, b) ForAllXCBNN(T,x,(b),0,(x).nComp())

The macro **ForAllXBC(T,x,b)** is the constant form of **ForAllXB(T,x,b)**.

8.15.1.10 #define ForAllXC(T, x) ForAllXCBNN(T,x,((x).box()),0,((x).nComp()))

The macro **ForAllXC(T,x)** is the constant form of **ForAllX(T,x)**.

8.15.1.11 #define ForAllXPencil(T, x) ForAllXBPencil(T,x,((x).box()),0,((x).nComp()))

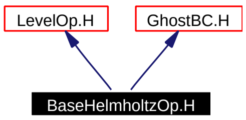
The Macro ForAllXPencil is a shortened version of ForAllXBPencil where the box is defaulted to the entire domain the the components run over all components of x

8.16 BaseHelmholtzOp.H File Reference

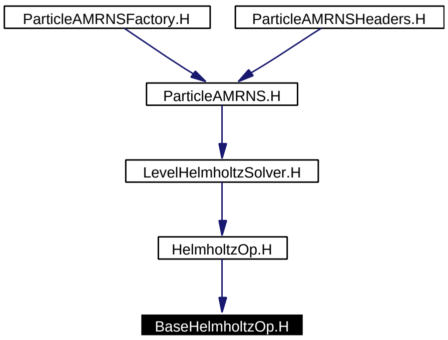
`#include "LevelOp.H"`

`#include "GhostBC.H"`

Include dependency graph for BaseHelmholtzOp.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BaseHelmholtzOp**

  *Pure-virtual base Helmholtz operator class (alpha*I + div*beta*grad).*

8.17 basicScalBC.H File Reference

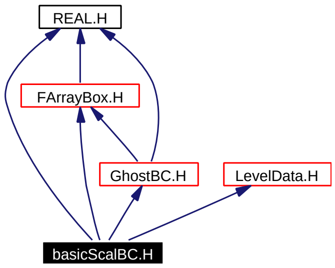
`#include "FArrayBox.H"`

`#include "REAL.H"`

`#include "LevelData.H"`

`#include "GhostBC.H"`

Include dependency graph for basicScalBC.H:



Compounds

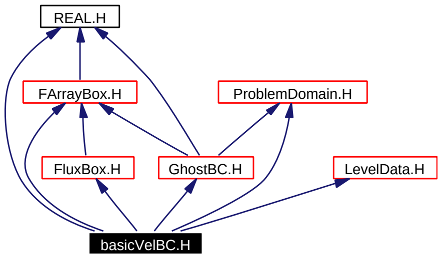
- class **basicScalBC**

    this is a basic physical BC class designed to handle scalars

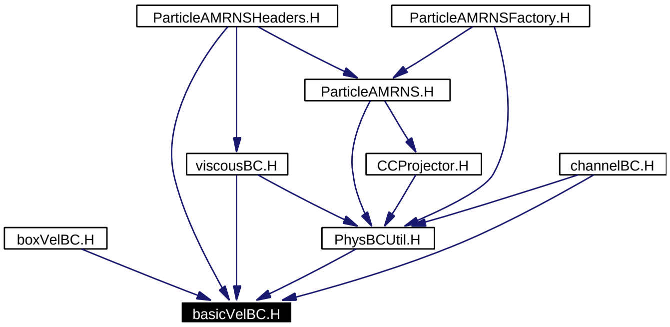
8.18 basicVelBC.H File Reference

```
#include "FArrayBox.H"
#include "FluxBox.H"
#include "REAL.H"
#include "LevelData.H"
#include "GhostBC.H"
#include "ProblemDomain.H"
```

Include dependency graph for basicVelBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **basicEdgeVelBC**

  this is a basic physical BC class designed to handle velocities

- class **basicVelBC**

  this is a basic physical BC class designed to handle velocities

- class **InflowVelBC**

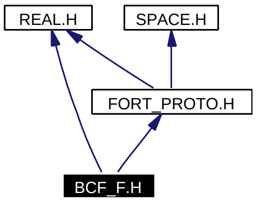
  does inflow

8.19 BCF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for BCF_F.H:



Defines

- #define **FORT_EXTRAPGHOSTBC** FORTRAN_NAME(EXTRAPGHOSTBC ,extrapghostbc)
- #define **FORT_HOEXTRAPGHOSTBC** FORTRAN_NAME(HOEXTRAPGHOSTBC ,hoextrapghostbc)
- #define **FORT_REFLECTGHOSTBC** FORTRAN_NAME(REFLECTGHOSTBC ,reflectghostbc)

Functions

- void **FORT_EXTRAPGHOSTBC** (CHFp_FRA(state),CHFp_BOX(bcBox),CHFp_CONST_INT(idir),CHFp_CONST_INT(side),CHFp_CONST_REAL(dx),CHFp_CONST_INT(startcomp),CHFp_CONST_INT(endcomp))
- void **FORT_HOEXTRAPGHOSTBC** (CHFp_FRA(state),CHFp_BOX(bcBox),CHFp_CONST_INT(idir),CHFp_CONST_INT(side),CHFp_CONST_REAL(dx),CHFp_CONST_INT(startcomp),CHFp_CONST_INT(endcomp))
- void **FORT_REFLECTGHOSTBC** (CHFp_FRA(state),CHFp_BOX(bcBox),CHFp_CONST_INT(idir),CHFp_CONST_INT(side),CHFp_CONST_REAL(dx),CHFp_CONST_INT(startcomp),CHFp_CONST_INT(endcomp),CHFp_CONST_REAL(scale))

8.19.1 Define Documentation

8.19.1.1 #define FORT_EXTRAPGHOSTBC FORTRAN_NAME(EXTRAPGHOSTBC ,extrapghostbc)

8.19.1.2 #define FORT_HOEXTRAPGHOSTBC FORTRAN_NAME(HOEXTRAPGHOSTBC ,hoextrapghostbc)

8.19.1.3 #define FORT_REFLECTGHOSTBC FORTRAN_NAME(REFLECTGHOSTBC ,reflectghostbc)

8.19.2 Function Documentation

8.19.2.1 void FORT_EXTRAPGHOSTBC (CHFp_FRA(state), CHFp_BOX(bcBox), CHFp_CONST_INT(idir), CHFp_CONST_INT(side), CHFp_CONST_REAL(dx), CHFp_CONST_INT(startcomp), CHFp_CONST_INT(endcomp))

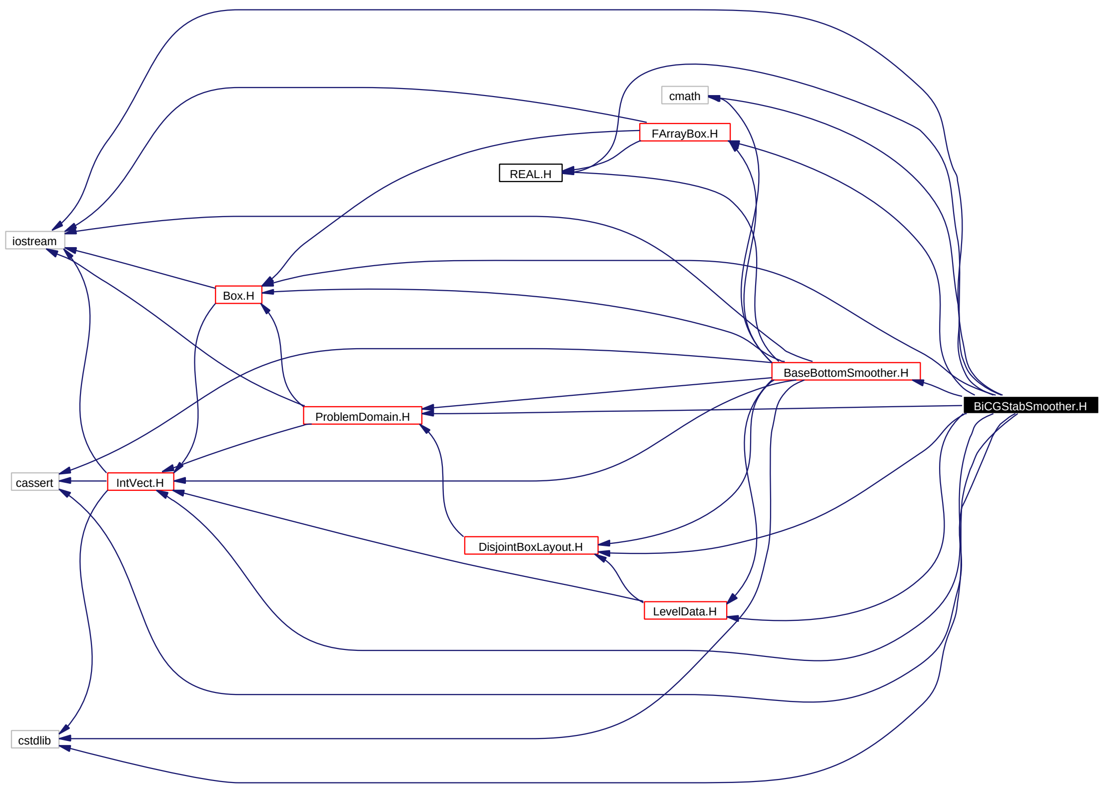
8.19.2.2 void FORT_HOEXTRAPGHOSTBC (CHFp_FRA(state), CHFp_BOX(bcBox), CHFp_CONST_INT(idir), CHFp_CONST_INT(side), CHFp_CONST_REAL(dx), CHFp_CONST_INT(startcomp), CHFp_CONST_INT(endcomp))

8.19.2.3 void FORT_REFLECTGHOSTBC (CHFp_FRA(state), CHFp_BOX(bcBox), CHFp_CONST_INT(idir), CHFp_CONST_INT(side), CHFp_CONST_REAL(dx), CHFp_CONST_INT(startcomp), CHFp_CONST_INT(endcomp), CHFp_CONST_REAL(scale))

8.20 BiCGStabSmoother.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "FArrayBox.H"
#include "ProblemDomain.H"
#include "BaseBottomSmoother.H"
```

Include dependency graph for BiCGStabSmoother.H:



Compounds

- class **BiCGStabSmoother**

  Implements BiCGStab as a bottom smoother for LevelOp-derived classes.

8.21 BinFab.H File Reference

`#include "BaseFab.H"`

`#include "RealVect.H"`

`#include "FluxBox.H"`

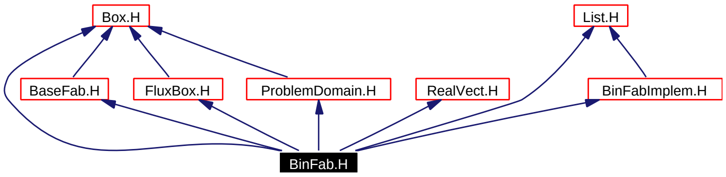
`#include "List.H"`

`#include "Box.H"`

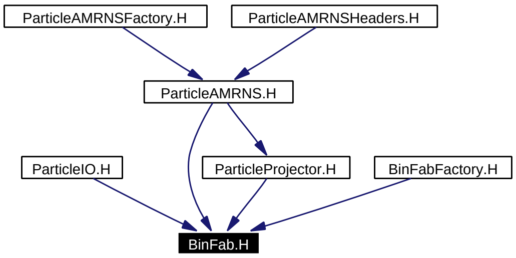
`#include "ProblemDomain.H"`

`#include "BinFabImplem.H"`

Include dependency graph for BinFab.H:



This graph shows which files directly or indirectly include this file:



Compounds

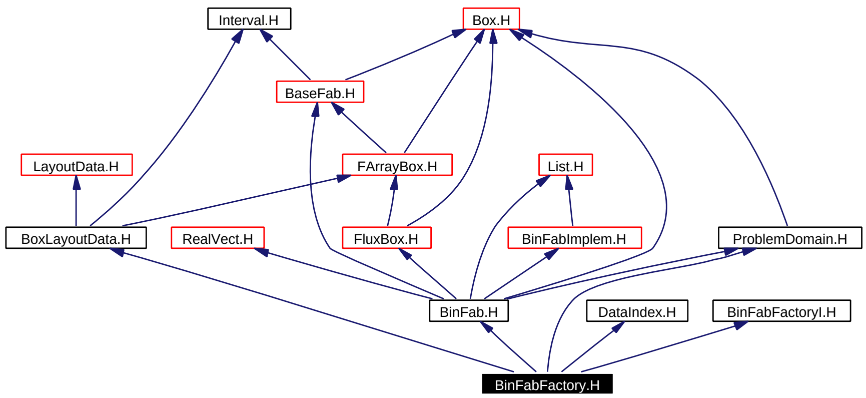
- class **BinFab**

  Base class for particle data on a **Box***.*

8.22 BinFabFactory.H File Reference

```
#include "BoxLayoutData.H"
#include "BinFab.H"
#include "DataIndex.H"
#include "ProblemDomain.H"
#include "BinFabFactoryI.H"
```

Include dependency graph for BinFabFactory.H:



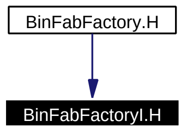
Compounds

- class **BinFabFactory**

  Factory class to produce one or more instances of **BinFab** *on the same level.*

8.23 BinFabFactoryI.H File Reference

This graph shows which files directly or indirectly include this file:

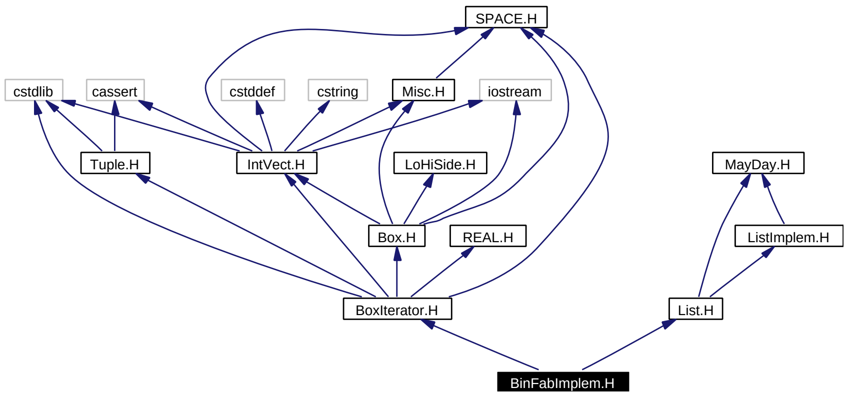


8.24 BinFabImplem.H File Reference

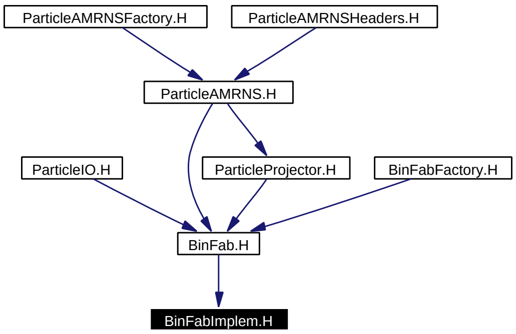
`#include "BoxIterator.H"`

`#include "List.H"`

Include dependency graph for BinFabImplem.H:



This graph shows which files directly or indirectly include this file:

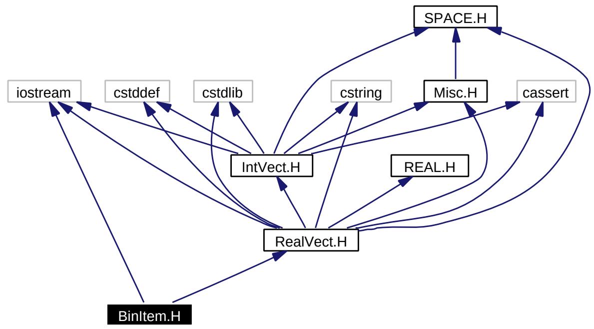


8.25 BinItem.H File Reference

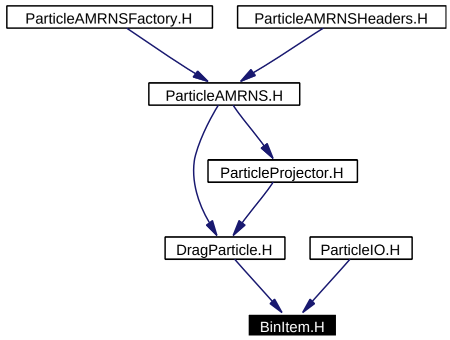
`#include <iostream>`

`#include "RealVect.H"`

Include dependency graph for BinItem.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BinItem**

  Virtual base class for particle data.

Functions

- std::ostream & **operator<<** (std::ostream &ostr, const **BinItem** &a_item)

8.25.1 Function Documentation

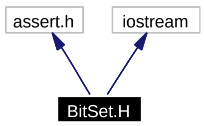
8.25.1.1 std::ostream& operator<< (std::ostream & *ostr*, const BinItem & *a_item*)

8.26 BitSet.H File Reference

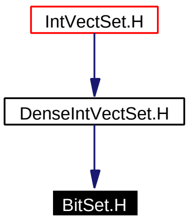
`#include <assert.h>`

`#include <iostream>`

Include dependency graph for BitSet.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BitSet**
- class **BitSetIterator**

Defines

- #define **BITSETWORD** unsigned int

Variables

- const int **BITSETWORDSIZE** = 8∗sizeof(unsigned int)

8.26.1 Define Documentation

8.26.1.1 #define BITSETWORD unsigned int

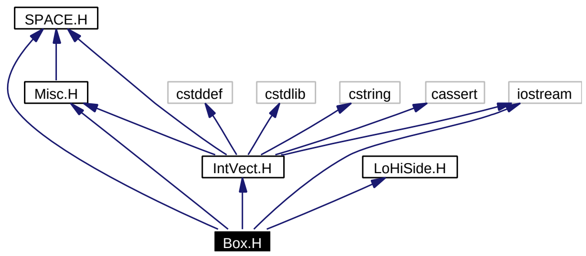
8.26.2 Variable Documentation

8.26.2.1 const int BITSETWORDSIZE = 8∗sizeof(unsigned int)

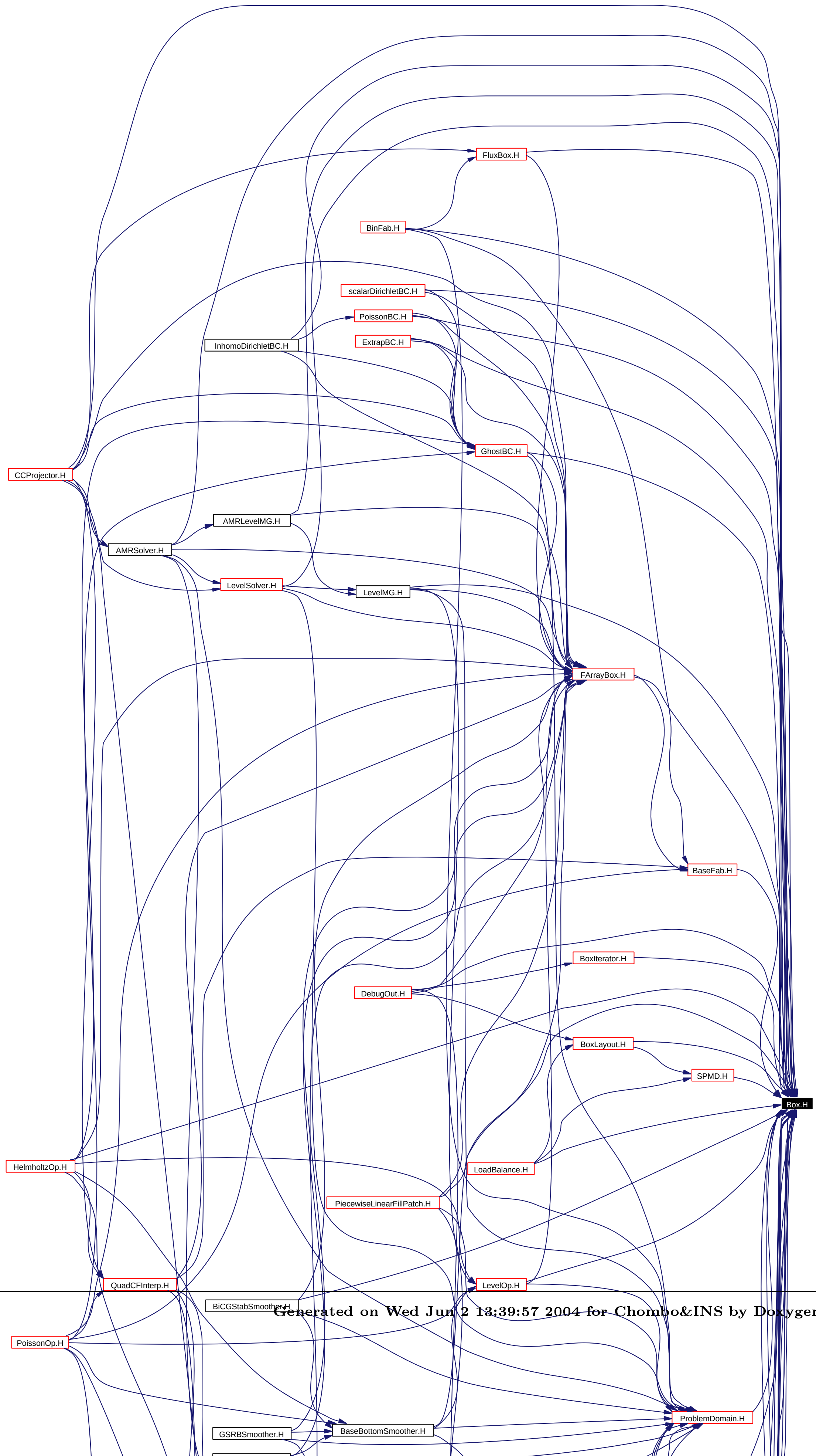
8.27 Box.H File Reference

```
#include "SPACE.H"
#include <iostream>
#include "IntVect.H"
#include "Misc.H"
#include "LoHiSide.H"
```

Include dependency graph for Box.H:



This graph shows which files directly or indirectly include this file:


FluxBox.H
BinFab.H
scalarDirichletBC.H
PoissonBC.H
ExtrapBC.H
InhomoDirichletBC.H
GhostBC.H
CCProjector.H
AMRLevelMG.H
AMRSolver.H
LevelSolver.H
LevelMG.H
FArrayBox.H
BaseFab.H
BoxIterator.H
DebugOut.H
BoxLayout.H
SPMD.H
Box.H
HelmholtzOp.H
LoadBalance.H
PiecewiseLinearFillPatch.H
LevelOp.H
QuadCFInterp.H
BiCGStabSmoother.H
PoissonOp.H
ProblemDomain.H
GSRBSmoother.H
BaseBottomSmoother.H

Compounds

- class **Box**

  A Rectangular Domain on an Integer Lattice.

- class **IndexType**

  Cell-Based or Node-Based Indices.

Functions

- **Box surroundingNodes** (const **Box** &b, int dir)
- **Box surroundingNodes** (const **Box** &b)
- **Box enclosedCells** (const **Box** &b, int dir)
- **Box enclosedCells** (const **Box** &b)
- **Box bdryBox** (const **Box** &b, int dir, **Side::LoHiSide** a_sd, int len=1)
- **Box bdryLo** (const **Box** &b, int dir, int len=1)
- **Box bdryHi** (const **Box** &b, int dir, int len=1)
- **Box adjCellLo** (const **Box** &b, int dir, int len=1)
- **Box adjCellHi** (const **Box** &b, int dir, int len=1)
- **Box minBox** (const **Box** &b1, const **Box** &b2)
- **Box grow** (const **Box** &b, int i)
- **Box grow** (const **Box** &b, const **IntVect** &v)

8.27.1 Function Documentation

8.27.1.1 Box adjCellHi (const Box & *b*, int *dir*, int *len* = 1)

Returns the cell centered **Box** of the given length adjacent to the argument **Box** on the high end along the given coordinate direction. The return **Box** is identical to the argument **Box** in the other directions. The return **Box** and the argument **Box** have an empty intersection. \

NOTE: len >= 1. \

NOTE: **Box** retval = adjCellHi(b,dir,len) is equivalent to the following set of operations: \

Box retval(b); \

retval.convert(dir,Box::CELL); \

retval.setrange(dir,retval.bigEnd(dir)+1,len);\

Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty **Box** is an Empty **Box** of the appropriate type.

8.27.1.2 Box adjCellLo (const Box & *b*, int *dir*, int *len* = 1)

Returns the cell centered **Box** of the given length adjacent to the argument **Box** on the low end along the given coordinate direction. The return **Box** is identical to the argument **Box** in the other directions. The return **Box** and the argument **Box** have an empty intersection. \

NOTE: len >= 1. \

NOTE: **Box** retval = adjCellLo(b,dir,len) is equivalent to the following set of operations: \

Box retval(b); \

retval.convert(dir,Box::CELL); \

retval.setrange(dir,retval.smallEnd(dir)-len,len); \

Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty **Box** is an Empty **Box** of the appropriate type.

8.27.1.3 Box bdryBox (const Box & *b*, int *dir*, Side::LoHiSide *a_sd*, int *len* = 1)

8.27.1.4 Box bdryHi (const Box & *b*, int *dir*, int *len* = 1)

Returns the edge-centered **Box** (in direction dir) defining the high side of the argument **Box**. The return **Box** will have the given length in the given direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty **Box** is an Empty **Box** of the appropriate type.

8.27.1.5 Box bdryLo (const Box & *b*, int *dir*, int *len* = 1)

Returns the edge-centered **Box** (in direction dir) defining the low side of the argument **Box**. The output **Box** will have the given length in the given direction. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The neighbor of an Empty **Box** is an Empty **Box** of the appropriate type.

8.27.1.6 Box enclosedCells (const Box & *b*) `[inline]`

Returns a **Box** with CELL based coordinates in all directions that is enclosed by argument **Box**. The upper corner of the return **Box** will be one less in all components than the upper corner of the argument **Box**. The Empty **Box** is not modified by this function except for changing of type.

8.27.1.7 Box enclosedCells (const Box & *b*, int *dir*) `[inline]`

Returns a **Box** with CELL based coordinates in direction dir that is enclosed by argument **Box**. NOTE: equivalent to b.convert(dir,CELL). NOTE: error if b.type(dir) == CELL. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The Empty **Box** is not modified by this function except for changing of type.

8.27.1.8 Box grow (const Box & *b*, const IntVect & *v*) `[inline]`

Returns a **Box** that is the argument **Box** grown in each direction by specified amount. The Empty **Box** is not modified by this function. NOTE: components of iv may be negative, which would return a shrunken **Box**. If shrinking produces an empty **Box**, the **Box** is transformed into the canonical Empty **Box**.

8.27.1.9 Box grow (const Box & *b*, int *i*) `[inline]`

Returns a **Box** that is the argument **Box** grown in all directions by given amount. The Empty **Box** is not modified by this function. NOTE: n negative shrinks the **Box** by that number of cells. If shrinking produces an empty **Box**, the **Box** is transformed into the canonical Empty **Box**.

8.27.1.10 Box minBox (const Box & *b1*, const Box & *b2*)

Returns a **Box** that is the minimum **Box** containing both the argument Boxes. Both Boxes must have identical type.

8.27.1.11 Box surroundingNodes (const Box & *b*) `[inline]`

Returns a **Box** with NODE based coordinates in all directions that encloses argument **Box**. The upper corner of the return **Box** will be one more in all components than the upper corner of the argument **Box**. The Empty **Box** is not modified by this function except for changing of type.

8.27.1.12 Box surroundingNodes (const Box & *b*, int *dir*) `[inline]`

Returns a **Box** with NODE based coordinates in direction dir that encloses the argument **Box**. NOTE: equivalent to b.convert(dir,NODE). NOTE: error if b.type(dir) == NODE. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim. The Empty **Box** is not modified by this function except for changing of type.

8.28 BoxIterator.H File Reference

`#include <cstdlib>`

`#include "Box.H"`

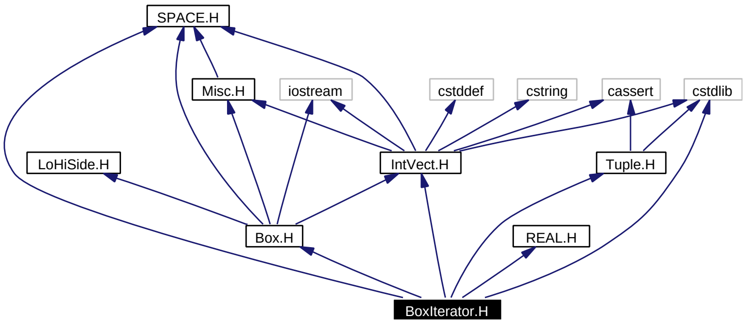
`#include "REAL.H"`

`#include "SPACE.H"`

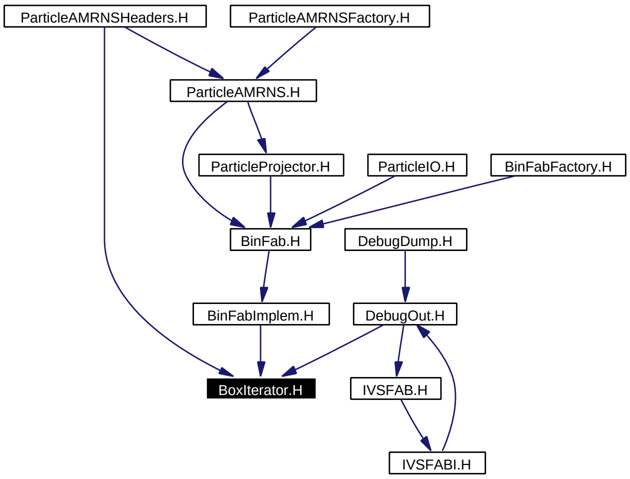
`#include "Tuple.H"`

`#include "IntVect.H"`

Include dependency graph for BoxIterator.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BoxIterator**

    iterates through the IntVects of a **Box**

8.29 BoxLayout.H File Reference

`#include "Box.H"`

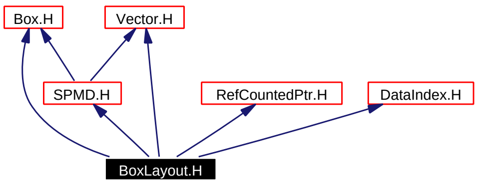
`#include "Vector.H"`

`#include "RefCountedPtr.H"`

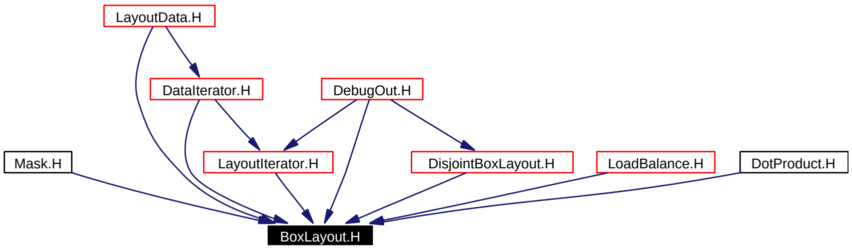
`#include "DataIndex.H"`

`#include "SPMD.H"`

Include dependency graph for BoxLayout.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BoxLayout**

  A not-necessarily-disjoint collective of boxes.

- struct **Entry**

Functions

- void **coarsen_bl** (**BoxLayout** &output, const **BoxLayout** &input, int refinement)
- void **refine_bl** (**BoxLayout** &output, const **BoxLayout** &input, int refinement)
- std::ostream & **operator<<** (std::ostream &os, const **BoxLayout** &a_layout)

8.29.1 Function Documentation

8.29.1.1 void coarsen_bl (BoxLayout & *output*, const BoxLayout & *input*, int *refinement*) [inline]

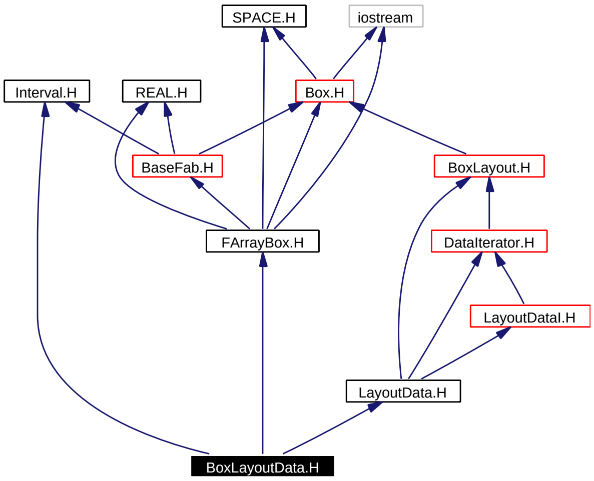
8.29.1.2 std::ostream& operator<< (std::ostream & *os*, const BoxLayout & *a_layout*)

8.29.1.3 void refine_bl (BoxLayout & *output*, const BoxLayout & *input*, int *refinement*) [inline]

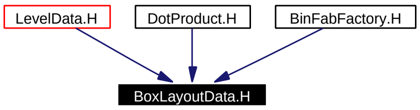
8.30 BoxLayoutData.H File Reference

```
#include "LayoutData.H"
#include "Interval.H"
#include "FArrayBox.H"
```

Include dependency graph for BoxLayoutData.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **AliasDataFactory**
- class **BoxLayoutData**

  Data on a **BoxLayout**.
- class **DataFactory**

  Factory object to data members of a **BoxLayoutData** *container.*
- class **DefaultDataFactory**

  Factory object to data members of a **BoxLayoutData** *container.*
- class **FABAliasDataFactory**

Functions

- **Real norm** (const **BoxLayoutData**< **FArrayBox** > &A, const **Interval** &interval, const int &p=2)

8.30.1 Function Documentation

8.30.1.1 Real norm (const BoxLayoutData< FArrayBox > & *A*, const Interval & *interval*, const int & *p* = 2)

not actually L-p norms, since it doesn't take into account the dx of the system. A user can take that into account or not.

For p != 0, returns pth root of sum of pth powers over all points in all fabs and all components in the interval:

(sum [|A[i][pt,var]|^p : **FArrayBox** A[i], point pt in A[i].box(), var in interval])^(1/p)

To turn into an L-p norm, one needs to multiply this by dx^(SpaceDim/p).

For p == 0, returns global max over all points in all fabs and all components in the interval:

max [|A[i][pt,var]| : **FArrayBox** A[i], point pt in A[i].box(), var in interval]

Some people don't like that this norm is not normalized based on number of points in A. Normalization is your problem.

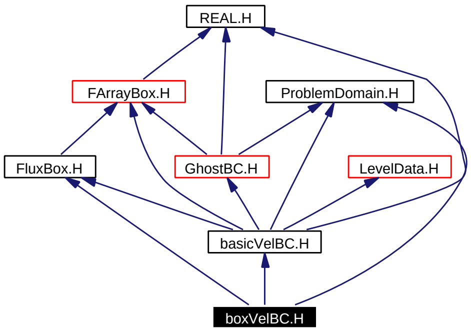
8.31 boxVelBC.H File Reference

`#include "basicVelBC.H"`

`#include "FluxBox.H"`

`#include "ProblemDomain.H"`

Include dependency graph for boxVelBC.H:



Compounds

- class **boxEdgeVelBC**

  implements edge-centered velocity BC's for flow in a box!

- class **boxVelBC**

  implements cell-centered velocity BC's for flow in a box!

8.32 BRMeshRefine.H File Reference

`#include <climits>`

`#include "Vector.H"`

`#include "Box.H"`

`#include "IntVectSet.H"`

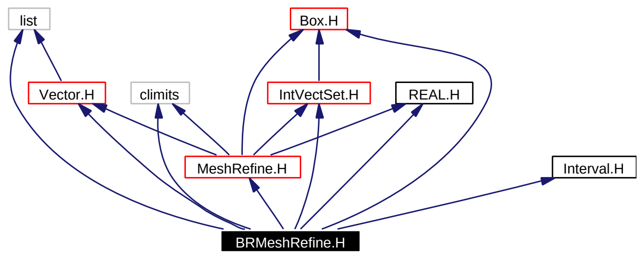
`#include "REAL.H"`

`#include "MeshRefine.H"`

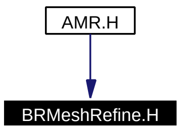
`#include "Interval.H"`

`#include <list>`

Include dependency graph for BRMeshRefine.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BRMeshRefine**

  Class which manages Berger-Rigoutsos grid generation.

Defines

- #define **_BR_MIN_BOX_FILL_RATIO_** (0.75)

Functions

- void **domainSplit** (const **ProblemDomain** &a_domain, **Vector**< **Box** > &a_vbox, int a_maxSize, int a_blockfactor=1)

  Splits domain into vector of disjoint boxes with max size maxsize.

- void **domainSplit** (const **Box** &a_domain, **Vector**< **Box** > &a_vbox, int a_maxSize, int a_blockfactor=1)
- void **breakBoxes** (**Vector**< **Box** > &a_vboxin, const int &a_maxSize, const int &a_idir)

8.32.1 Define Documentation

8.32.1.1 #define _BR_MIN_BOX_FILL_RATIO_ (0.75)

8.32.2 Function Documentation

8.32.2.1 void breakBoxes (Vector< Box > & *a_vboxin*, const int & *a_maxSize*, const int & *a_idir*)

Recursive function to enforce max size of boxes in a given direction. Does not call Meshrefine.

8.32.2.2 void domainSplit (const Box & *a_domain*, Vector< Box > & *a_vbox*, int *a_maxSize*, int *a_blockfactor* = 1)

Splits domain into a vector of disjoint boxes with maximum size maxsize. blocking factor is default to one. If you make minimum size > 1, then domain must be coarsenable and refineable by blockfactor (refine(coarsen(domain,blockfactor), minsize) == domain) or an error is thrown. This would be defined in **MeshRefine.H**, except that it needs to use a **BRMeshRefine** object.

8.32.2.3 void domainSplit (const ProblemDomain & *a_domain*, Vector< Box > & *a_vbox*, int *a_maxSize*, int *a_blockfactor* = 1)

Splits domain into vector of disjoint boxes with max size maxsize.

Blocking factor is default to one. If you make minimum size > 1, then domain must be coarsenable and refineable by blockfactor (refine(coarsen(domain,blockfactor), minsize) == domain) or an error is thrown. This would be defined in **MeshRefine.H**, except that it needs to use a **BRMeshRefine** object.

8.33 CCProjector.H File Reference

#include <cstdlib>

#include <cmath>

#include <cassert>

#include <iostream>

#include "REAL.H"

#include "IntVect.H"

#include "Box.H"

#include "FArrayBox.H"

#include "Vector.H"

#include "IntVectSet.H"

#include "DisjointBoxLayout.H"

#include "LevelData.H"

#include "LevelSolver.H"

#include "AMRSolver.H"

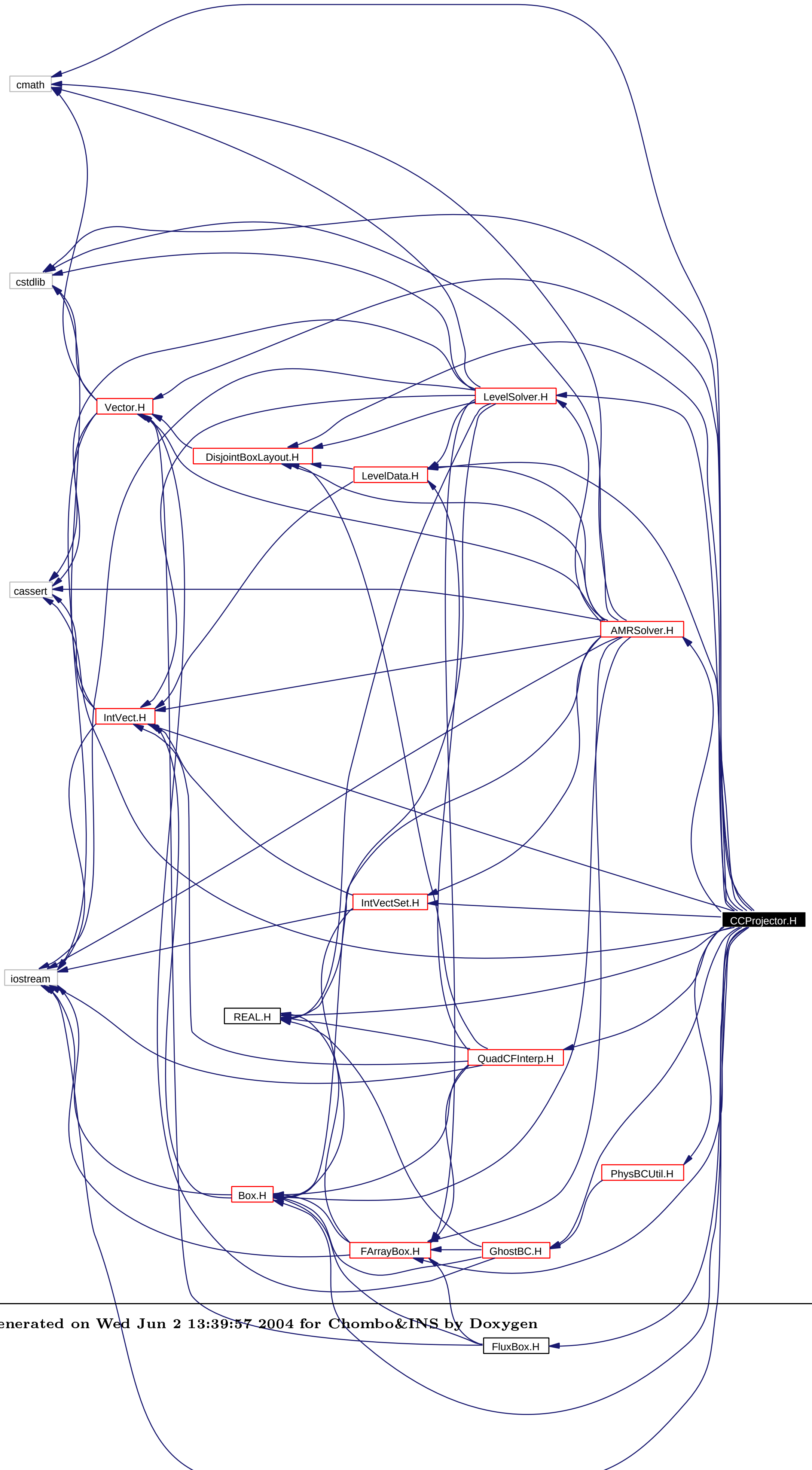
#include "QuadCFInterp.H"

#include "GhostBC.H"

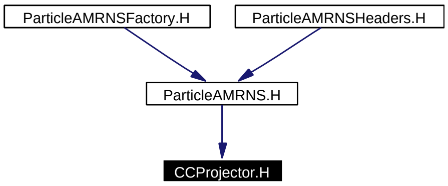
#include "PhysBCUtil.H"

#include "FluxBox.H"

Include dependency graph for CCProjector.H:


cmath
cstdlib
Vector.H
DisjointBoxLayout.H
LevelSolver.H
LevelData.H
cassert
AMRSolver.H
IntVect.H
IntVectSet.H
CCProjector.H
iostream
REAL.H
QuadCFInterp.H
PhysBCUtil.H
Box.H
FArrayBox.H
GhostBC.H
FluxBox.H

This graph shows which files directly or indirectly include this file:



Compounds

- class **CCProjector**

  this class manages the various forms of the CC projection

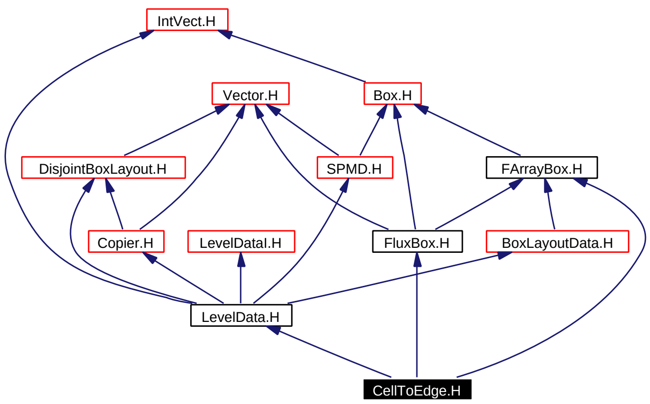
8.34 CellToEdge.H File Reference

`#include "LevelData.H"`

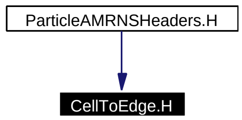
`#include "FArrayBox.H"`

`#include "FluxBox.H"`

Include dependency graph for CellToEdge.H:



This graph shows which files directly or indirectly include this file:



Functions

- void **CellToEdge** (const **LevelData**< **FArrayBox** > &a_cellData, **LevelData**< **FluxBox** > &a_edgeData)
- void **CellToEdge** (const **FArrayBox** &a_cellData, **FluxBox** &a_edgeData)
- void **CellToEdge** (const **FArrayBox** &a_cellData, **FArrayBox** &a_edgeData, const int a_dir)

  this one does a single direction

- void **CellToEdge** (const **FArrayBox** &a_cellData, const int a_cellComp, **FArrayBox** &a_edgeData, const int a_edgeComp, const int a_dir)

  this one does a single direction and component

8.34.1 Function Documentation

8.34.1.1 void CellToEdge (const FArrayBox & *a_cellData*, const int *a_cellComp*, FArrayBox & *a_edgeData*, const int *a_edgeComp*, const int *a_dir*)

this one does a single direction and component

8.34.1.2 void CellToEdge (const FArrayBox & *a_cellData*, FArrayBox & *a_edgeData*, const int *a_dir*)

this one does a single direction

8.34.1.3 void CellToEdge (const FArrayBox & *a_cellData*, FluxBox & *a_edgeData*)

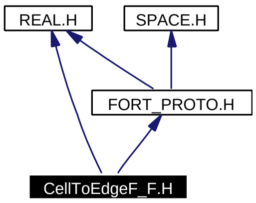
8.34.1.4 void CellToEdge (const LevelData< FArrayBox > & *a_cellData*, LevelData< FluxBox > & *a_edgeData*)

8.35 CellToEdgeF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for CellToEdgeF_F.H:



Defines

- #define **FORT_CELLTOEDGE** FORTRAN_NAME(CELLTOEDGE ,celltoedge)

Functions

- void **FORT_CELLTOEDGE** (CHFp_CONST_FRA1(cellData),CHFp_FRA1(edgeData),CHFp_BOX(edgeBox),CHFp_CONST_INT(dir))

8.35.1 Define Documentation

8.35.1.1 #define FORT_CELLTOEDGE FORTRAN_NAME(CELLTOEDGE ,celltoedge)

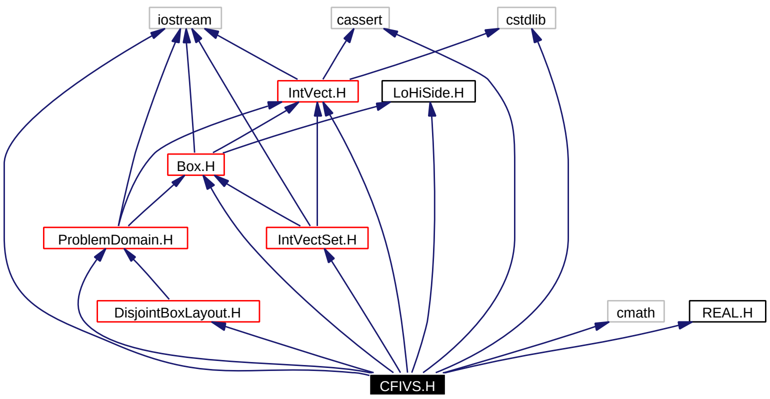
8.35.2 Function Documentation

8.35.2.1 void FORT_CELLTOEDGE (CHFp_CONST_FRA1(cellData), CHFp_FRA1(edgeData), CHFp_BOX(edgeBox), CHFp_CONST_INT(dir))

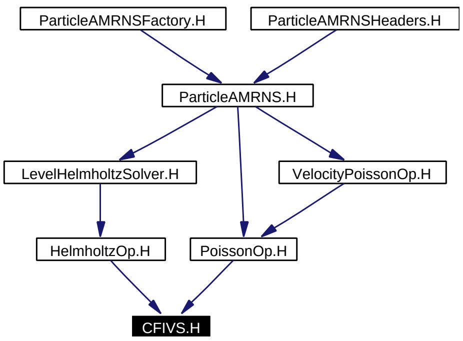
8.36 CFIVS.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "LoHiSide.H"
#include "DisjointBoxLayout.H"
#include "IntVectSet.H"
#include "ProblemDomain.H"
```

Include dependency graph for CFIVS.H:



This graph shows which files directly or indirectly include this file:



Compounds

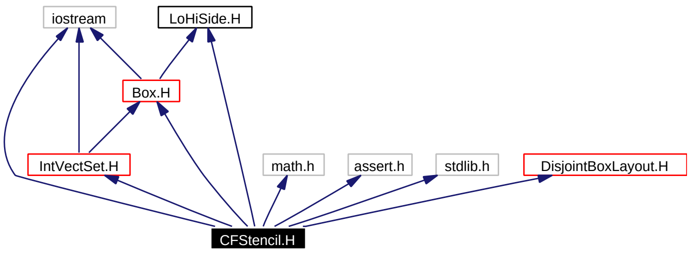
- class **CFIVS**

  Internal Chombo class to get fine intvectset at CF interface.

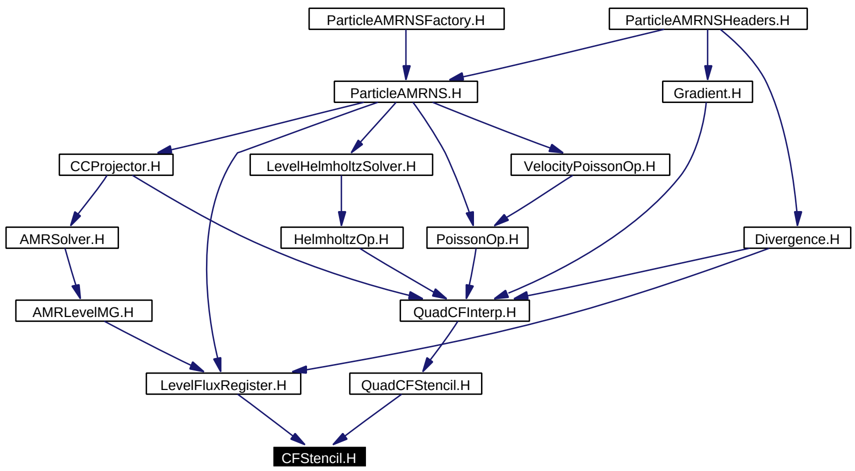
8.37 CFStencil.H File Reference

```
#include <iostream>
#include <math.h>
#include <assert.h>
#include <stdlib.h>
#include "DisjointBoxLayout.H"
#include "Box.H"
#include "LoHiSide.H"
#include "IntVectSet.H"
```

Include dependency graph for CFStencil.H:



This graph shows which files directly or indirectly include this file:



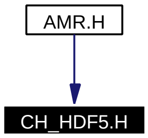
Compounds

- class **CFStencil**

  Class to encapsulate coarse-fine information for **AMR** *operations.*

8.38 CH_HDF5.H File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define **HOFFSET**(S, M) (offsetof(S,M))

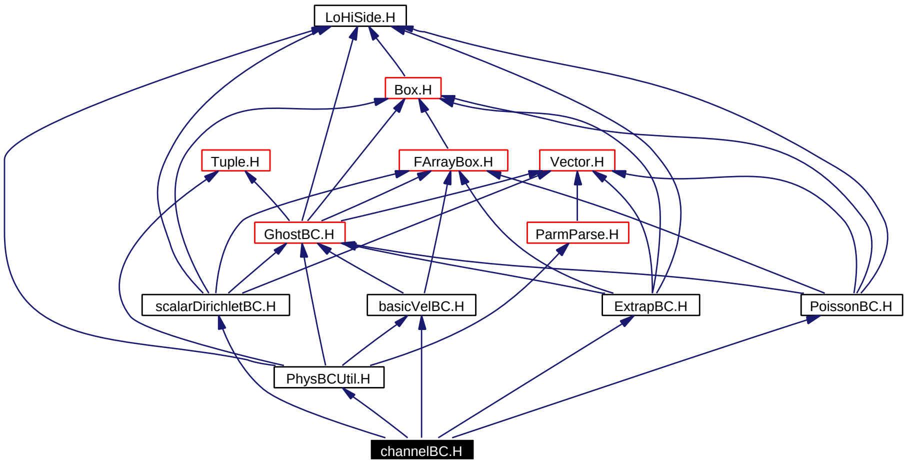
8.38.1 Define Documentation

8.38.1.1 #define HOFFSET(S, M) (offsetof(S,M))

8.39 channelBC.H File Reference

```
#include "PhysBCUtil.H"
#include "scalarDirichletBC.H"
#include "basicVelBC.H"
#include "ExtrapBC.H"
#include "PoissonBC.H"
```

Include dependency graph for channelBC.H:



Compounds

- class **channelBC**

  specific BC's for the convection problem (particularly for scalars)

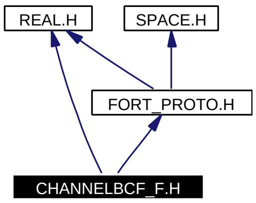
- class **NormalInflowVelBC**
- class **scalarInflowChannelBC**

8.40 CHANNELBCF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for CHANNELBCF_F.H:



Defines

- #define **FORT_CHANNELNORMINFLOWBC** FORTRAN_NAME(CHANNEL-NORMINFLOWBC ,channelnorminflowbc)

Functions

- void **FORT_CHANNELNORMINFLOWBC** (CHFp_FRA1(inflowVel),CHFp_-REAL(dx),CHFp_CONST_INT(dir),CHFp_BOX(domain))

8.40.1 Define Documentation

8.40.1.1 #define FORT_CHANNELNORMINFLOWBC FORTRAN_NAME(CHANNELNORMINFLOWBC ,channelnorminflowbc)

8.40.2 Function Documentation

8.40.2.1 void FORT_CHANNELNORMINFLOWBC (CHFp_FRA1(inflowVel), CHFp_REAL(dx), CHFp_CONST_INT(dir), CHFp_BOX(domain))

8.41 CHOMBO_VERSION.H File Reference

Defines

- #define **CHOMBO_VERSION** 1.5
- #define **CHOMBO_VERSION_MAJOR** 1
- #define **CHOMBO_VERSION_MINOR** 5

8.41.1 Define Documentation

8.41.1.1 #define CHOMBO_VERSION 1.5

8.41.1.2 #define CHOMBO_VERSION_MAJOR 1

The major version number of Chombo – an integer.

The Chombo version number has the form:

CHOMBO_VERSION_MAJOR::CHOMBO_VERSION_MINOR

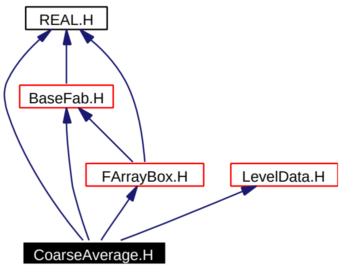
8.41.1.3 #define CHOMBO_VERSION_MINOR 5

The minor version number of Chombo – an integer.

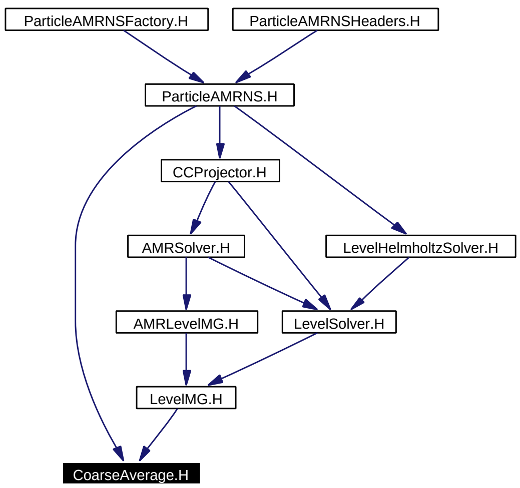
8.42 CoarseAverage.H File Reference

```
#include "REAL.H"
#include "BaseFab.H"
#include "FArrayBox.H"
#include "LevelData.H"
```

Include dependency graph for CoarseAverage.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **CoarseAverage**

  replaces coarse level data with an average of fine level data.

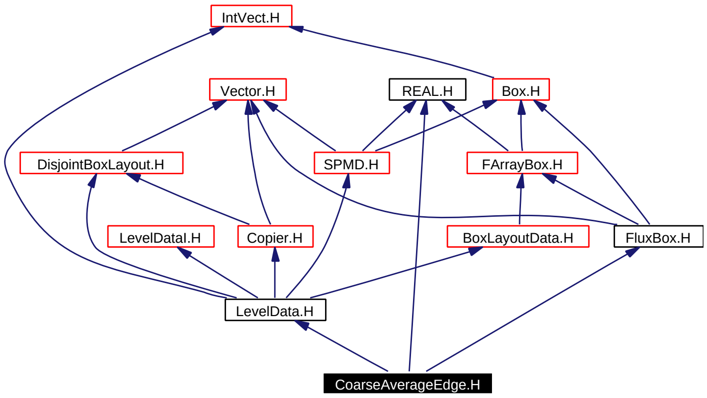
8.43 CoarseAverageEdge.H File Reference

`#include "REAL.H"`

`#include "FluxBox.H"`

`#include "LevelData.H"`

Include dependency graph for CoarseAverageEdge.H:



Compounds

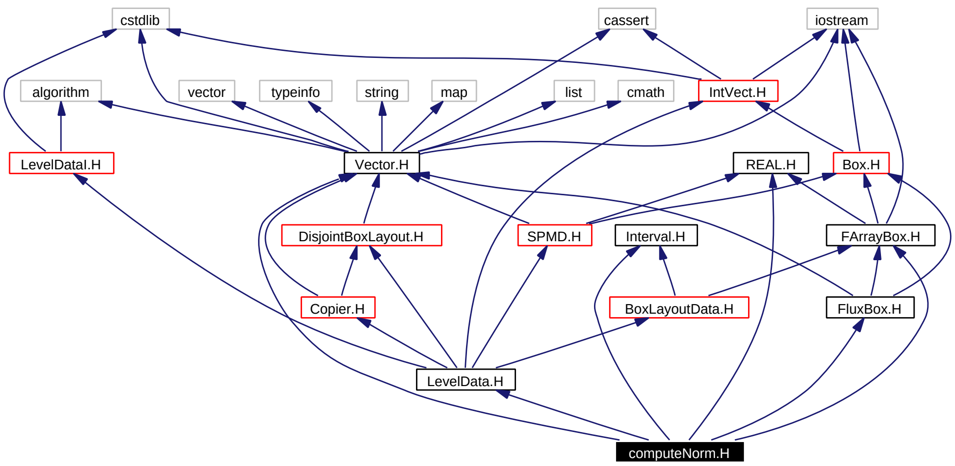
- class **CoarseAverageEdge**

  replaces edge-centered coarse-level data w/ averaged fine-level data

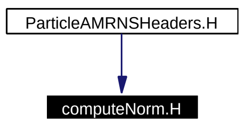
8.44 computeNorm.H File Reference

```
#include "REAL.H"
#include "Vector.H"
#include "LevelData.H"
#include "FArrayBox.H"
#include "Interval.H"
#include "FluxBox.H"
```

Include dependency graph for computeNorm.H:



This graph shows which files directly or indirectly include this file:



Functions

- **Real computeNorm** (const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phi, const **Vector**< int > &a_nRefFine, const **Real** a_dxCrse, const **Interval** a_comps=**Interval**(0, 0), const int p=2, const int lBase=0)

  Returns the volume-weighted sum (integral) of phi over all valid regions.

- **Real computeNorm** (const **LevelData**< **FArrayBox** > &a_phi, const **Disjoint-BoxLayout** ∗a_finerGrids, const int a_nRefFine, const **Real** a_dx, const **Interval** a_comps=**Interval**(0, 0), const int p=2)

  Returns the volume-weighted sum (integral) of phi over the valid region.

- **Real computeMax** (const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phi, const **Vector**< int > &a_nRefFine, const **Interval** a_comps=**Interval**(0, 0), const int lBase=0) *returns the max value of phi over all valid regions*

- **Real computeMax** (const **LevelData**< **FArrayBox** > &a_phi, const **DisjointBoxLayout** ∗a_finerGrids, const int a_nRefFine, const **Interval** a_comps=**Interval**(0, 0))

  Returns the volume-weighted sum (integral) of phi over the valid region.

- **Real computeMin** (const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phi, const **Vector**< int > &a_nRefFine, const **Interval** a_comps=**Interval**(0, 0), const int lBase=0)

  returns the min value of phil over all valid regions

- **Real computeMin** (const **LevelData**< **FArrayBox** > &a_phi, const **DisjointBoxLayout** ∗a_finerGrids, const int a_nRefFine, const **Interval** a_comps=**Interval**(0, 0))

  Returns the volume-weighted sum (integral) of phi over the valid region.

- **Real norm** (const **BoxLayoutData**< **FluxBox** > &a_layout, const **Interval** &interval, const int &p)

  extension of the norm(...) function for FluxBoxes

8.44.1 Function Documentation

8.44.1.1 Real computeMax (const LevelData< FArrayBox > & *a_phi*, const DisjointBoxLayout ∗ *a_finerGrids*, const int *a_nRefFine*, const Interval *a_comps* = Interval(0, 0))

Returns the volume-weighted sum (integral) of phi over the valid region.

8.44.1.2 Real computeMax (const Vector< LevelData< FArrayBox > ∗ > & *a_phi*, const Vector< int > & *a_nRefFine*, const Interval *a_comps* = Interval(0, 0), const int *lBase* = 0)

returns the max value of phi over all valid regions

8.44.1.3 Real computeMin (const LevelData< FArrayBox > & *a_phi*, const DisjointBoxLayout ∗ *a_finerGrids*, const int *a_nRefFine*, const Interval *a_comps* = Interval(0, 0))

Returns the volume-weighted sum (integral) of phi over the valid region.

8.44.1.4 Real computeMin (const Vector< LevelData< FArrayBox > ∗ > & *a_phi*, const Vector< int > & *a_nRefFine*, const Interval *a_comps* = Interval(0, 0), const int *lBase* = 0)

returns the min value of phil over all valid regions

8.44.1.5 Real computeNorm (const LevelData< FArrayBox > & *a_phi*, const DisjointBoxLayout ∗ *a_finerGrids*, const int *a_nRefFine*, const Real *a_dx*, const Interval *a_comps* = Interval(0, 0), const int *p* = 2)

Returns the volume-weighted sum (integral) of phi over the valid region.

8.44.1.6 Real computeNorm (const Vector< LevelData< FArrayBox > * > & *a_phi*, const Vector< int > & *a_nRefFine*, const Real *a_dxCrse*, const Interval *a_comps* = Interval(0, 0), const int *p* = 2, const int *lBase* = 0)

Returns the volume-weighted sum (integral) of phi over all valid regions.

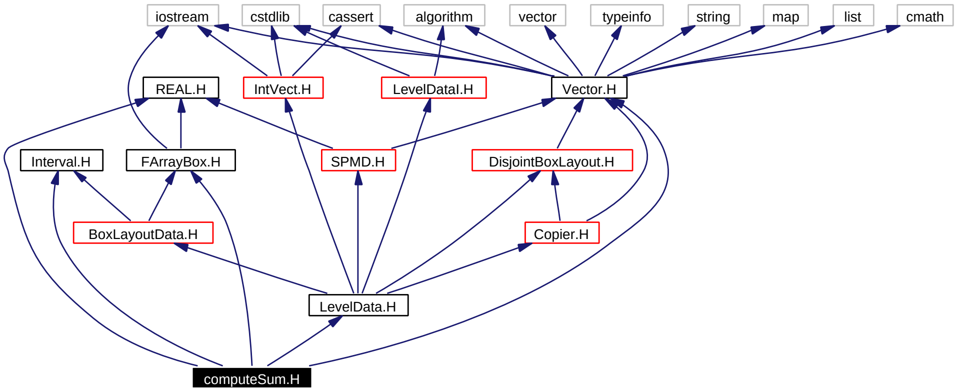
8.44.1.7 Real norm (const BoxLayoutData< FluxBox > & *a_layout*, const Interval & *interval*, const int & *p*)

extension of the norm(...) function for FluxBoxes

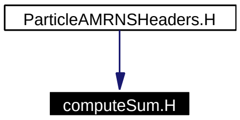
8.45 computeSum.H File Reference

```
#include "REAL.H"
#include "Vector.H"
#include "LevelData.H"
#include "FArrayBox.H"
#include "Interval.H"
```

Include dependency graph for computeSum.H:



This graph shows which files directly or indirectly include this file:



Functions

- **Real computeSum** (const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phi, const **Vector**< int > &a_nRefFine, const **Real** a_dxCrse, const **Interval** a_comps=**Interval**(0, 0), const int lBase=0)

  Returns the volume-weighted sum (integral) of phi over all valid regions.

- **Real computeSum** (const **LevelData**< **FArrayBox** > &a_phi, const **DisjointBoxLayout** ∗a_finerGrids, const int a_nRefFine, const **Real** a_dx, const **Interval** a_comps=**Interval**(0, 0))

  Returns the volume-weighted sum (integral) of phi over the valid region.

8.45.1 Function Documentation

8.45.1.1 Real computeSum (const LevelData< FArrayBox > & *a_phi*, const DisjointBoxLayout ∗ *a_finerGrids*, const int *a_nRefFine*, const Real *a_dx*, const Interval *a_comps* = Interval(0, 0))

Returns the volume-weighted sum (integral) of phi over the valid region.

8.45.1.2 Real computeSum (const Vector< LevelData< FArrayBox > ∗ > & *a_phi*, const Vector< int > & *a_nRefFine*, const Real *a_dxCrse*, const Interval *a_comps* = Interval(0, 0), const int *lBase* = 0)

Returns the volume-weighted sum (integral) of phi over all valid regions.

8.46 CONSTANTS.H File Reference

8.47 Copier.H File Reference

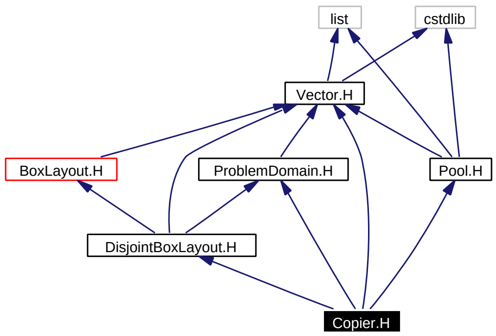
`#include "DisjointBoxLayout.H"`

`#include "Pool.H"`

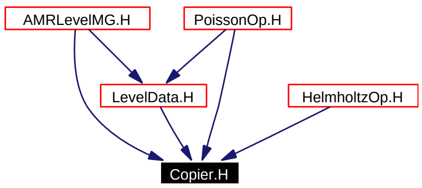
`#include "Vector.H"`

`#include "ProblemDomain.H"`

Include dependency graph for Copier.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Copier**

  A strange but true thing to make copying from one boxlayoutdata to another fast.

- class **CopyIterator**
- class **MotionItem**

Functions

- std::ostream & **operator**<< (std::ostream &os, const **Copier** &copier)

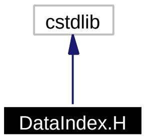
8.47.1 Function Documentation

8.47.1.1 std::ostream& operator<< (std::ostream & *os*, const Copier & *copier*)

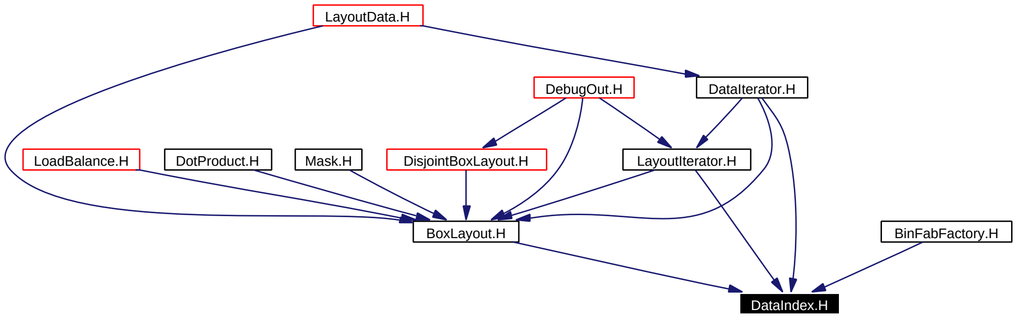
8.48 DataIndex.H File Reference

`#include <cstdlib>`

Include dependency graph for DataIndex.H:



This graph shows which files directly or indirectly include this file:



Compounds

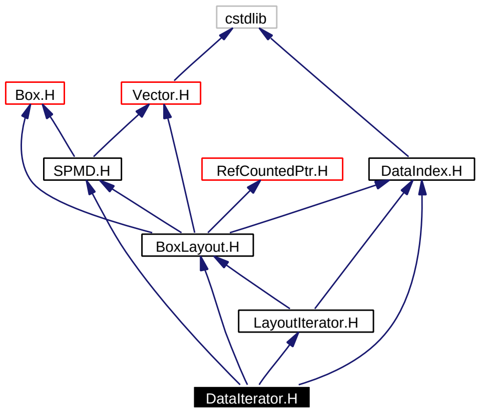
- class **DataIndex**
- class **LayoutIndex**

  An index for **LayoutIterator**.

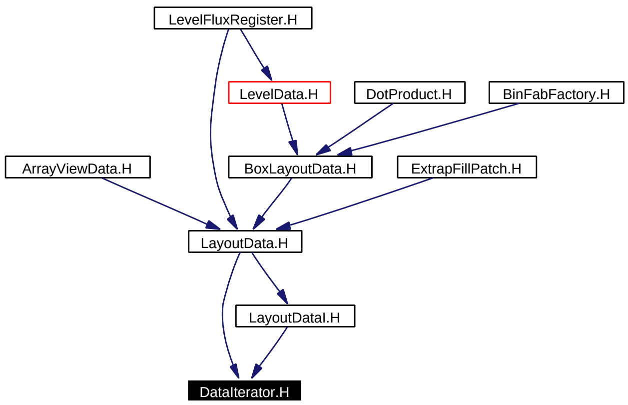
8.49 DataIterator.H File Reference

```
#include "DataIndex.H"
#include "BoxLayout.H"
#include "SPMD.H"
#include "LayoutIterator.H"
```

Include dependency graph for DataIterator.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DataIterator**

Defines

- #define **DATAITERATOR**(CLASS, BOXLAYOUT)
- #define **ENDITERATOR**(CLASS)
- #define **DATAITERATOR1**(CLASS, BOXLAYOUT, TYPE1, VAL1)

- #define **ENDITERATOR1**(CLASS, TYPE1, VAL1)
- #define **DATAITERATOR2**(CLASS, BOXLAYOUT, TYPE1, VAL1, TYPE2, VAL2)
- #define **ENDITERATOR2**(CLASS, TYPE1, VAL1, TYPE2, VAL2)

8.49.1 Define Documentation

8.49.1.1 #define DATAITERATOR(CLASS, BOXLAYOUT)

Value:

```
DataIterator dit = BOXLAYOUT .dataIterator(); \
       for(dit.begin(); dit.ok(); ++dit) {     \
       DataIndex di = dit();                   \
       MT_BEGIN1(CLASS, DataIndex, di)
```

8.49.1.2 #define DATAITERATOR1(CLASS, BOXLAYOUT, TYPE1, VAL1)

Value:

```
DataIterator dit = BOXLAYOUT .dataIterator(); \
       for(dit.begin(); dit.ok(); ++dit) {     \
       DataIndex di = dit();                   \
       MT_BEGIN2(CLASS, TYPE1, VAL1, DataIndex, di)
```

8.49.1.3 #define DATAITERATOR2(CLASS, BOXLAYOUT, TYPE1, VAL1, TYPE2, VAL2)

Value:

```
DataIterator dit = BOXLAYOUT .dataIterator(); \
       for(dit.begin(); dit.ok(); ++dit) {     \
       DataIndex di = dit();                   \
       MT_BEGIN3(CLASS, TYPE1, VAL1, TYPE2, VAL2, DataIndex, di)
```

8.49.1.4 #define ENDITERATOR(CLASS)

Value:

```
MT_END1(CLASS, DataIndex, di) \
       }
```

8.49.1.5 #define ENDITERATOR1(CLASS, TYPE1, VAL1)

Value:

```
MT_END2(CLASS, TYPE1, VAL1, DataIndex, di) \
       }
```

8.49.1.6 #define ENDITERATOR2(CLASS, TYPE1, VAL1, TYPE2, VAL2)

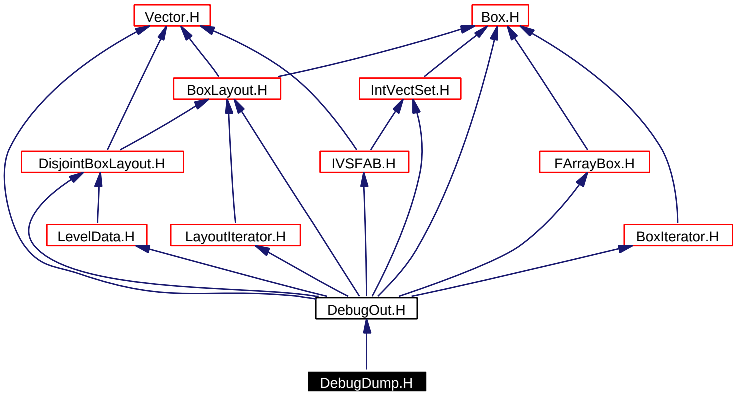
Value:

```
MT_END3(CLASS, TYPE1, VAL1, TYPE2, VAL2, DataIndex, di) \
        }
```

8.50 DebugDump.H File Reference

`#include "DebugOut.H"`

Include dependency graph for DebugDump.H:



Functions

- void **DontYouDareCallMe** ()

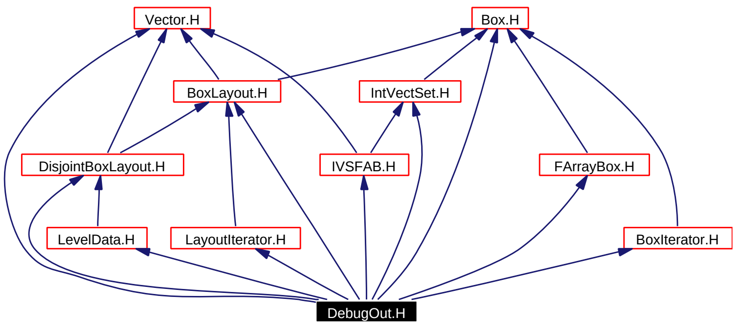
8.50.1 Function Documentation

8.50.1.1 void DontYouDareCallMe ()

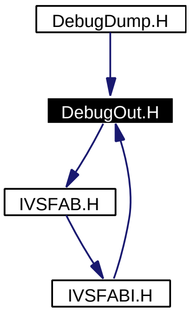
8.51 DebugOut.H File Reference

```
#include "DisjointBoxLayout.H"
#include "BoxLayout.H"
#include "Vector.H"
#include "Box.H"
#include "IVSFAB.H"
#include "IntVectSet.H"
#include "FArrayBox.H"
#include "LevelData.H"
#include "LayoutIterator.H"
#include "BoxIterator.H"
```

Include dependency graph for DebugOut.H:



This graph shows which files directly or indirectly include this file:



Functions

- void **dumpLDF** (const **LevelData**< **FArrayBox** > *memLDF)
- void **dumpIVSFAB** (const **IVSFAB**< **Real** > *memLDF)
- void **dumpDBL** (const **DisjointBoxLayout** *a_dblInPtr)
- void **dumpBL** (const **BoxLayout** *a_dblInPtr)

- void **dumpIVS** (const **IntVectSet** *a_ivsInPtr)
- void **dumpBox** (const **Box** *a_boxPtr)
- void **dumpVBox** (const **Vector**< **Box** > *a_vectPtr)
- void **dumpVVBox** (const **Vector**< **Vector**< **Box** > > *a_vectPtr)

8.51.1 Function Documentation

8.51.1.1 void dumpBL (const BoxLayout * *a_dblInPtr*)

Dump inputs to standard out. For use inside debuggers.

8.51.1.2 void dumpBox (const Box * *a_boxPtr*)

Dump inputs to standard out. For use inside debuggers.

8.51.1.3 void dumpDBL (const DisjointBoxLayout * *a_dblInPtr*)

Dump inputs to standard out. For use inside debuggers.

8.51.1.4 void dumpIVS (const IntVectSet * *a_ivsInPtr*)

Dump inputs to standard out. For use inside debuggers.

8.51.1.5 void dumpIVSFAB (const IVSFAB< Real > * *memLDF*)

Dump boxlayout of inputs to standard out. For use inside debuggers.

8.51.1.6 void dumpLDF (const LevelData< FArrayBox > * *memLDF*)

Dump boxlayout of inputs to standard out. For use inside debuggers.

8.51.1.7 void dumpVBox (const Vector< Box > * *a_vectPtr*)

Dump inputs to standard out. For use inside debuggers.

8.51.1.8 void dumpVVBox (const Vector< Vector< Box > > * *a_vectPtr*)

Dump inputs to standard out. For use inside debuggers.

8.52 DenseIntVectSet.H File Reference

`#include "BitSet.H"`

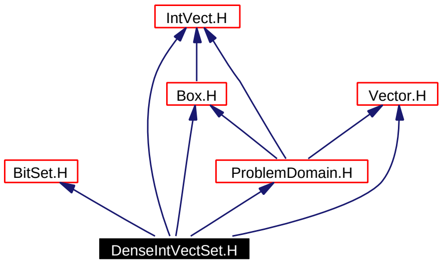
`#include "IntVect.H"`

`#include "Box.H"`

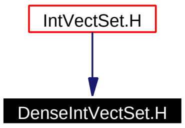
`#include "Vector.H"`

`#include "ProblemDomain.H"`

Include dependency graph for DenseIntVectSet.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DenseIntVectSet**

    Dense representation implementation of **IntVectSet** *class.*

- class **DenseIntVectSetIterator**

    DenseIntVectSetIterator iterates over all the 'true' members of a **DenseIntVectSet** *set.*

8.53 DerivStencil.H File Reference

`#include <iostream>`

`#include <math.h>`

`#include <assert.h>`

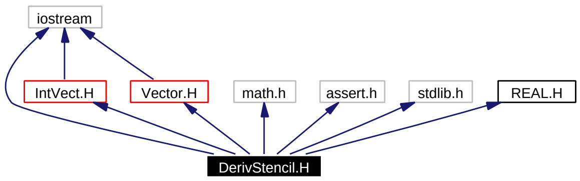
`#include <stdlib.h>`

`#include "REAL.H"`

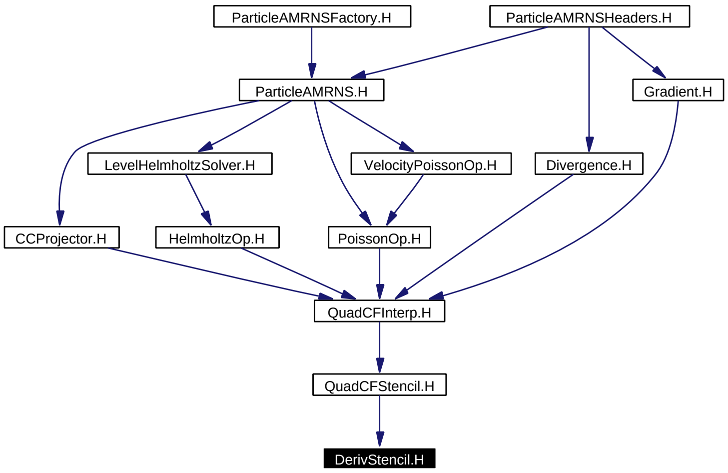
`#include "IntVect.H"`

`#include "Vector.H"`

Include dependency graph for DerivStencil.H:



This graph shows which files directly or indirectly include this file:



Compounds

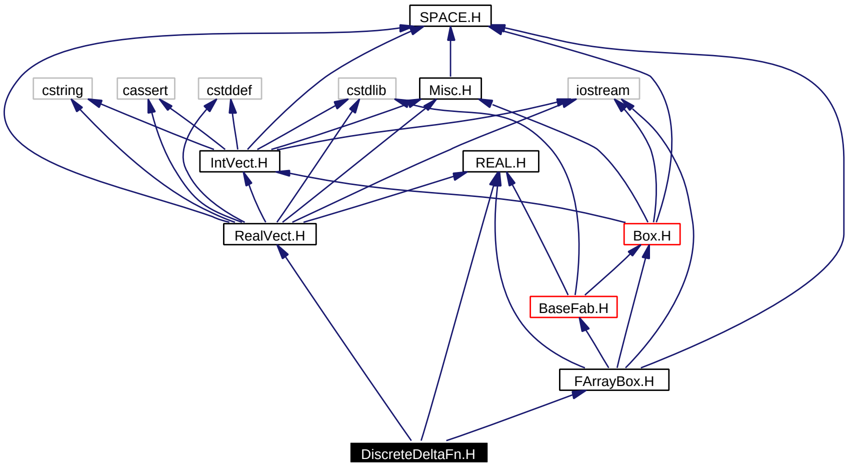
- class **DerivStencil**

  class to encapsulate the operations to create derivs on irreg stencils

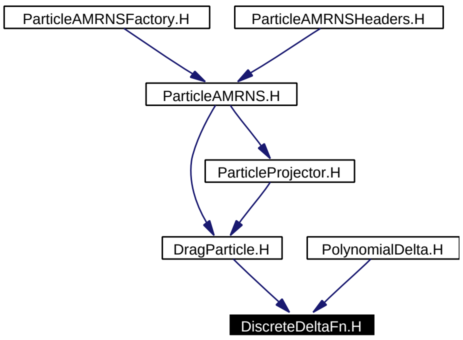
8.54 DiscreteDeltaFn.H File Reference

```
#include "REAL.H"
#include "RealVect.H"
#include "FArrayBox.H"
```

Include dependency graph for DiscreteDeltaFn.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DiscreteDeltaFn**

    Pure virtual base class to encapsulate discrete delta function operator.

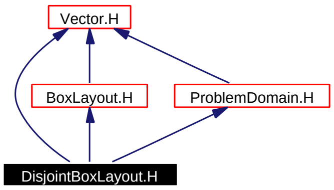
8.55 DisjointBoxLayout.H File Reference

`#include "Vector.H"`

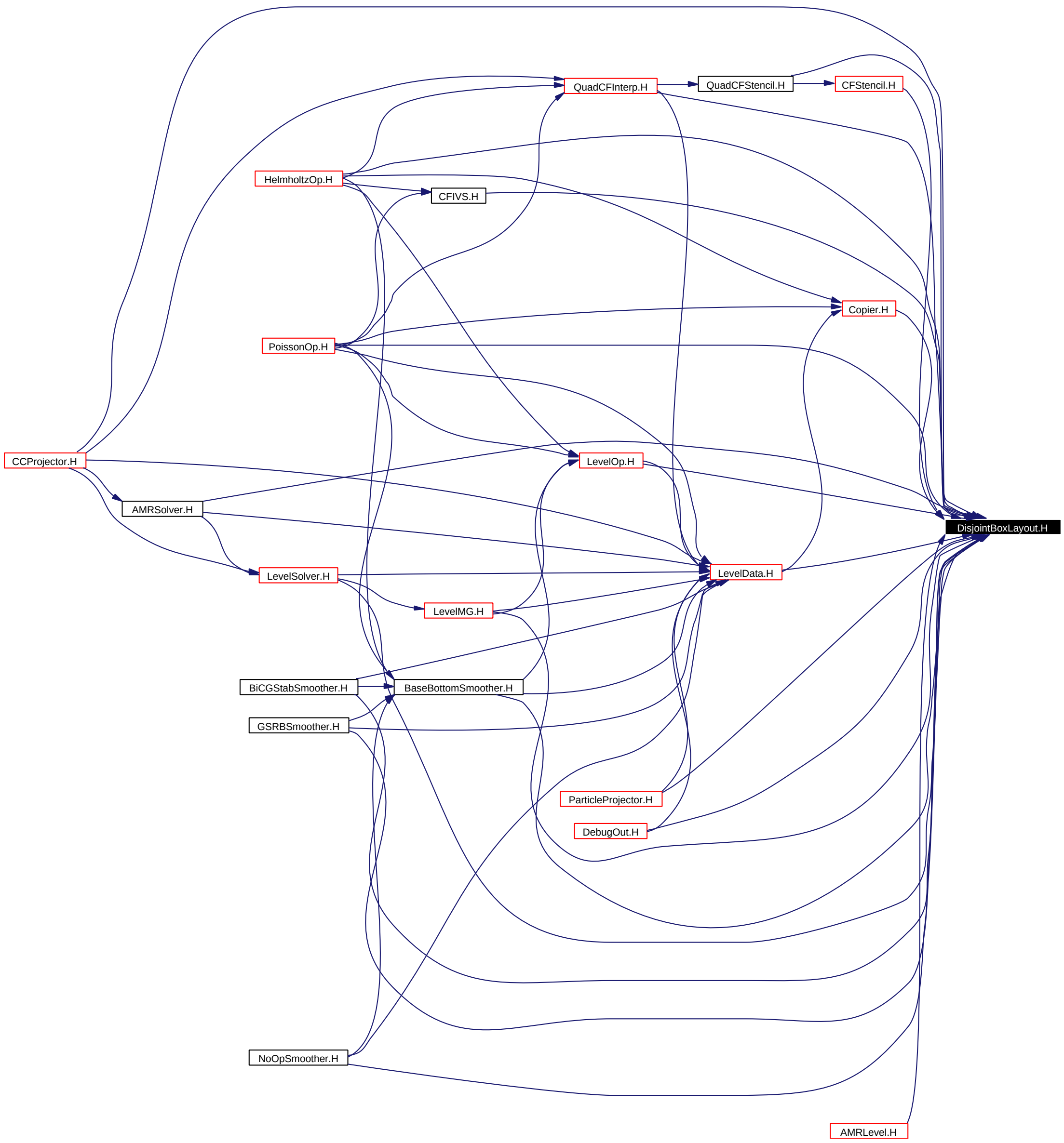
`#include "BoxLayout.H"`

`#include "ProblemDomain.H"`

Include dependency graph for DisjointBoxLayout.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DisjointBoxLayout**

  A **BoxLayout** *that has a concept of disjointedness.*

Functions

- void **adjCellLo_dbl** (**DisjointBoxLayout** &a_output, const **DisjointBoxLayout** &a_input, int a_dir, int a_len)
- void **adjCellHi_dbl** (**DisjointBoxLayout** &a_output, const **DisjointBoxLayout** &a_input, int a_dir, int a_len)

- void **coarsen_dbl** (**DisjointBoxLayout** &output, const **DisjointBoxLayout** &input, int refinement)
- void **refine_dbl** (**DisjointBoxLayout** &output, const **DisjointBoxLayout** &input, int refinement)

8.55.1 Function Documentation

8.55.1.1 void adjCellHi_dbl (DisjointBoxLayout & *a_output*, const DisjointBoxLayout & *a_input*, int *a_dir*, int *a_len*) `[inline]`

8.55.1.2 void adjCellLo_dbl (DisjointBoxLayout & *a_output*, const DisjointBoxLayout & *a_input*, int *a_dir*, int *a_len*) `[inline]`

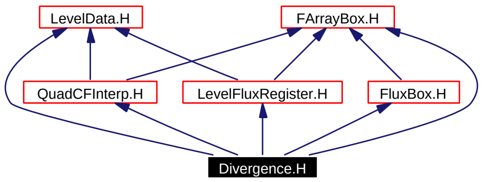
8.55.1.3 void coarsen_dbl (DisjointBoxLayout & *output*, const DisjointBoxLayout & *input*, int *refinement*) `[inline]`

8.55.1.4 void refine_dbl (DisjointBoxLayout & *output*, const DisjointBoxLayout & *input*, int *refinement*) `[inline]`

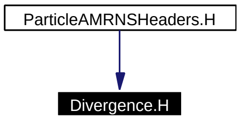
8.56 Divergence.H File Reference

```
#include "LevelData.H"
#include "FArrayBox.H"
#include "FluxBox.H"
#include "LevelFluxRegister.H"
#include "QuadCFInterp.H"
```

Include dependency graph for Divergence.H:



This graph shows which files directly or indirectly include this file:



Functions

- void **levelDivergenceCC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FArrayBox** > &a_u, **LevelData**< **FArrayBox** > *a_uCrsePtr, const **Real** a_dx, const int a_nRefCrse, const **ProblemDomain** &a_dProblem, const bool a_quadInterp)
- void **levelDivergenceCC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FArrayBox** > &a_u, **LevelData**< **FArrayBox** > *a_uCrsePtr, const **Real** a_dx, const int a_nRefCrse, const **Box** &a_dProblem, const bool a_quadInterp)
- void **levelDivergenceCC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FArrayBox** > &a_u, **LevelData**< **FArrayBox** > *a_uCrsePtr, const **Real** a_dx, const bool a_quadInterp, **QuadCFInterp** &a_cfInterp)
- void **compDivergenceCC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FArrayBox** > &a_u, **LevelData**< **FArrayBox** > *a_uCrsePtr, **LevelData**< **FArrayBox** > *a_uFinePtr, const **Real** a_dx, const int a_nRefCrse, const int a_nRefFine, const **ProblemDomain** &a_dProblem, const bool a_quadInterp)
- void **compDivergenceCC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FArrayBox** > &a_u, **LevelData**< **FArrayBox** > *a_uCrsePtr, **LevelData**< **FArrayBox** > *a_uFinePtr, const **Real** a_dx, const int a_nRefCrse, const int a_nRefFine, const **Box** &a_dProblem, const bool a_quadInterp)
- void **compDivergenceCC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FArrayBox** > &a_u, **LevelData**< **FArrayBox** > *a_uCrsePtr, **LevelData**< **FArrayBox** > *a_uFinePtr, const **Real** a_dx, const int a_nRefFine, const int a_nRefCrse, const **ProblemDomain** &a_dProblem, const bool a_quadInterp, **LevelFluxRegister** *a_fluxRegFinePtr, **QuadCFInterp** &a_cfInterpCrse, **QuadCFInterp** &a_cfInterpFine)

- void **compDivergenceCC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FArrayBox** > &a_u, **LevelData**< **FArrayBox** > *a_uCrsePtr, **LevelData**< **FArrayBox** > *a_uFinePtr, const **Real** a_dx, const int a_nRefFine, const int a_nRefCrse, const **Box** &a_dProblem, const bool a_quadInterp, **LevelFluxRegister** *a_fluxRegFinePtr, **QuadCFInterp** &a_cfInterpCrse, **QuadCFInterp** &a_cfInterpFine)

- void **levelDivergenceMAC** (**LevelData**< **FArrayBox** > &a_div, const **LevelData**< **FluxBox** > &a_uEdge, const **Real** a_dx)

- void **simpleDivergenceMAC** (**FArrayBox** &a_div, const **FluxBox** &a_uEdge, const **Real** a_dx)

  really basic single-fab MAC divergence

- void **compDivergenceMAC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FluxBox** > &a_uEdge, **LevelData**< **FluxBox** > *a_uEdgeFinePtr, const **Real** a_dx, const **Real** *a_dxFine, const int a_nRefFine, const **ProblemDomain** &a_dProblem)

- void **compDivergenceMAC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FluxBox** > &a_uEdge, **LevelData**< **FluxBox** > *a_uEdgeFinePtr, const **Real** a_dx, const **Real** *a_dxFine, const int a_nRefFine, const **Box** &a_dProblem)

- void **compDivergenceMAC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FluxBox** > &a_uEdge, **LevelData**< **FluxBox** > *a_uEdgeFinePtr, **LevelFluxRegister** *a_fluxRegPtr, const **Real** a_dx, const **Real** *a_dxFine, const int a_nRefine, const **ProblemDomain** &a_dProblem)

- void **compDivergenceMAC** (**LevelData**< **FArrayBox** > &a_div, **LevelData**< **FluxBox** > &a_uEdge, **LevelData**< **FluxBox** > *a_uEdgeFinePtr, **LevelFluxRegister** *a_fluxRegPtr, const **Real** a_dx, const **Real** *a_dxFine, const int a_nRefine, const **Box** &a_dProblem)

8.56.1 Function Documentation

8.56.1.1 void compDivergenceCC (LevelData< FArrayBox > & *a_div*, LevelData< FArrayBox > & *a_u*, LevelData< FArrayBox > ∗ *a_uCrsePtr*, LevelData< FArrayBox > ∗ *a_uFinePtr*, const Real *a_dx*, const int *a_nRefFine*, const int *a_nRefCrse*, const Box & *a_dProblem*, const bool *a_quadInterp*, LevelFluxRegister ∗ *a_fluxRegFinePtr*, QuadCFInterp & *a_cfInterpCrse*, QuadCFInterp & *a_cfInterpFine*)

8.56.1.2 void compDivergenceCC (LevelData< FArrayBox > & *a_div*, LevelData< FArrayBox > & *a_u*, LevelData< FArrayBox > ∗ *a_uCrsePtr*, LevelData< FArrayBox > ∗ *a_uFinePtr*, const Real *a_dx*, const int *a_nRefFine*, const int *a_nRefCrse*, const ProblemDomain & *a_dProblem*, const bool *a_quadInterp*, LevelFluxRegister ∗ *a_fluxRegFinePtr*, QuadCFInterp & *a_cfInterpCrse*, QuadCFInterp & *a_cfInterpFine*)

8.56.1.3 void compDivergenceCC (LevelData< FArrayBox > & *a_div*, LevelData< FArrayBox > & *a_u*, LevelData< FArrayBox > ∗ *a_uCrsePtr*, LevelData< FArrayBox > ∗ *a_uFinePtr*, const Real *a_dx*, const int *a_nRefCrse*, const int *a_nRefFine*, const Box & *a_dProblem*, const bool *a_quadInterp*)

8.56.1.4 void compDivergenceCC (LevelData< FArrayBox > & *a_div*, LevelData< FArrayBox > & *a_u*, LevelData< FArrayBox > ∗ *a_uCrsePtr*, LevelData< FArrayBox > ∗ *a_uFinePtr*, const Real *a_dx*, const int *a_nRefCrse*, const int *a_nRefFine*, const ProblemDomain & *a_dProblem*, const bool *a_quadInterp*)

8.56.1.5 void compDivergenceMAC (LevelData< FArrayBox > & *a_div*, LevelData< FluxBox > & *a_uEdge*, LevelData< FluxBox > ∗ *a_uEdgeFinePtr*, LevelFluxRegister ∗ *a_fluxRegPtr*, const Real *a_dx*, const Real ∗ *a_dxFine*, const int *a_nRefine*, const Box & *a_dProblem*)

8.56.1.6 void compDivergenceMAC (LevelData< FArrayBox > & *a_div*, LevelData< FluxBox > & *a_uEdge*, LevelData< FluxBox > ∗ *a_uEdgeFinePtr*, LevelFluxRegister ∗ *a_fluxRegPtr*, const Real *a_dx*, const Real ∗ *a_dxFine*, const int *a_nRefine*, const ProblemDomain & *a_dProblem*)

8.56.1.7 void compDivergenceMAC (LevelData< FArrayBox > & *a_div*, LevelData< FluxBox > & *a_uEdge*, LevelData< FluxBox > ∗ *a_uEdgeFinePtr*, const Real *a_dx*, const Real ∗ *a_dxFine*, const int *a_nRefFine*, const Box & *a_dProblem*)

8.56.1.8 void compDivergenceMAC (LevelData< FArrayBox > & *a_div*, LevelData< FluxBox > & *a_uEdge*, LevelData< FluxBox > ∗ *a_uEdgeFinePtr*, const Real *a_dx*, const Real ∗ *a_dxFine*, const int *a_nRefFine*, const ProblemDomain & *a_dProblem*)

8.56.1.9 void levelDivergenceCC (LevelData< FArrayBox > & *a_div*, LevelData< FArrayBox > & *a_u*, LevelData< FArrayBox > ∗ *a_uCrsePtr*, const Real *a_dx*, const bool *a_quadInterp*, QuadCFInterp & *a_cfInterp*)

8.56.1.10 void levelDivergenceCC (LevelData< FArrayBox > & *a_div*, LevelData< FArrayBox > & *a_u*, LevelData< FArrayBox > ∗ *a_uCrsePtr*, const Real *a_dx*, const int *a_nRefCrse*, const Box & *a_dProblem*, const bool *a_quadInterp*)

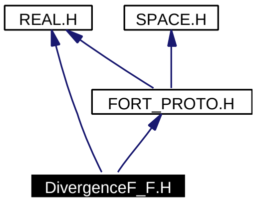
8.56.1.11 void levelDivergenceCC (LevelData< FArrayBox > & *a_div*, LevelData< FArrayBox > & *a_u*, LevelData< FArrayBox > ∗ *a_uCrsePtr*, const Real *a_dx*, const int *a_nRefCrse*, const ProblemDomain & *a_dProblem*, const bool *a_quadInterp*)

8.57 DivergenceF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for DivergenceF_F.H:



Defines

- #define **FORT_DIVERGENCE** FORTRAN_NAME(DIVERGENCE ,divergence)

Functions

- void **FORT_DIVERGENCE** (CHFp_CONST_FRA(uEdge),CHFp_FRA(div),CHFp_BOX(gridInt),CHFp_CONST_REAL(dx),CHFp_INT(dir))

8.57.1 Define Documentation

8.57.1.1 #define FORT_DIVERGENCE FORTRAN_NAME(DIVERGENCE ,divergence)

8.57.2 Function Documentation

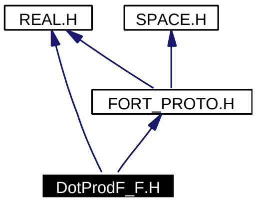
8.57.2.1 void FORT_DIVERGENCE (CHFp_CONST_FRA(uEdge), CHFp_FRA(div), CHFp_BOX(gridInt), CHFp_CONST_REAL(dx), CHFp_INT(dir))

8.58 DotProdF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for DotProdF_F.H:



Defines

- #define **FORT_DOTPRODUCT** FORTRAN_NAME(DOTPRODUCT ,dotproduct)

Functions

- void **FORT_DOTPRODUCT** (CHFp_REAL(dotprodout),CHFp_CONST_-FRA(afab),CHFp_CONST_FRA(bfab),CHFp_BOX(region),CHFp_CONST_-INT(startcomp),CHFp_CONST_INT(endcomp))

8.58.1 Define Documentation

8.58.1.1 #define FORT_DOTPRODUCT FORTRAN_NAME(DOTPRODUCT ,dotproduct)

8.58.2 Function Documentation

8.58.2.1 void FORT_DOTPRODUCT (CHFp_REAL(dotprodout), CHFp_-CONST_FRA(afab), CHFp_CONST_FRA(bfab), CHFp_BOX(region), CHFp_CONST_INT(startcomp), CHFp_CONST_INT(endcomp))

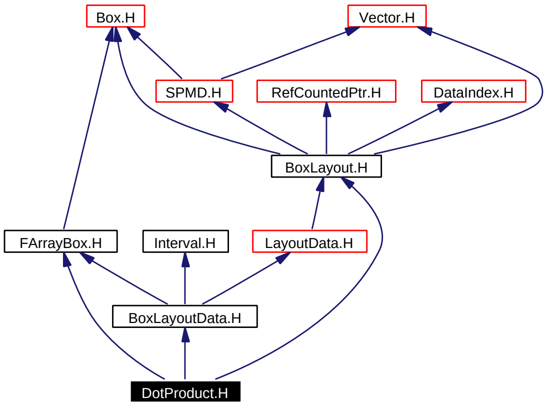
8.59 DotProduct.H File Reference

`#include "BoxLayout.H"`

`#include "BoxLayoutData.H"`

`#include "FArrayBox.H"`

Include dependency graph for DotProduct.H:



Functions

- **Real DotProduct** (const **BoxLayoutData**< **FArrayBox** > &a_dataOne, const **BoxLayoutData**< **FArrayBox** > &a_dataTwo, const **BoxLayout** &a_dblIn)
- **Real DotProduct** (const **BoxLayoutData**< **FArrayBox** > &a_dataOne, const **BoxLayoutData**< **FArrayBox** > &a_dataTwo, const **BoxLayout** &a_dblIn, const **Interval** &a_comps)

8.59.1 Function Documentation

8.59.1.1 Real DotProduct (const BoxLayoutData< FArrayBox > & *a_dataOne*, const BoxLayoutData< FArrayBox > & *a_dataTwo*, const BoxLayout & *a_dblIn*, const Interval & *a_comps*)

8.59.1.2 Real DotProduct (const BoxLayoutData< FArrayBox > & *a_dataOne*, const BoxLayoutData< FArrayBox > & *a_dataTwo*, const BoxLayout & *a_dblIn*)

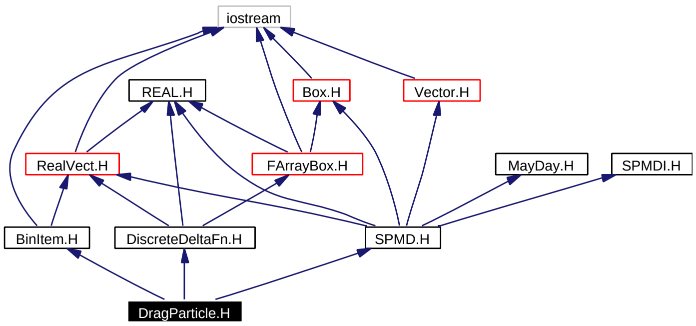
8.60 DragParticle.H File Reference

`#include "BinItem.H"`

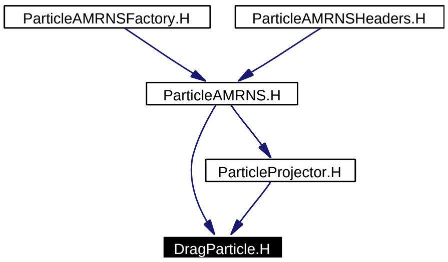
`#include "DiscreteDeltaFn.H"`

`#include "SPMD.H"`

Include dependency graph for DragParticle.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DragParticle**

  BinItem-derived object to encapsulate definition of particle in fluid.

Functions

- template<> int **linearSize** (const **Vector**< **DragParticle** > &a_input)
- template<> void **linearIn** (**Vector**< **DragParticle** > &a_outputT, const void *const inBuf)
- template<> void **linearOut** (void *const a_outBuf, const **Vector**< **DragParticle** > &a_inputT)

8.60.1 Function Documentation

8.60.1.1 template<> void linearIn (Vector< DragParticle > & *a_outputT*, const void *const *inBuf*)

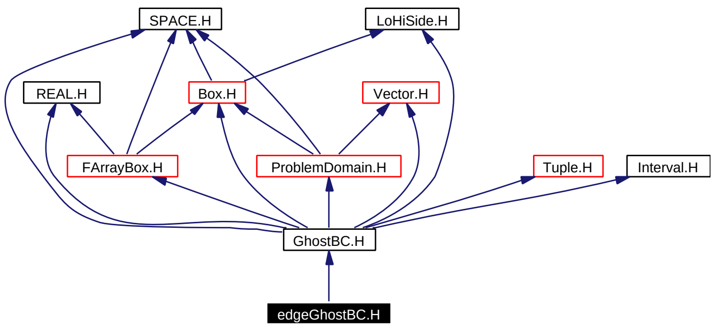
8.60.1.2 template<> void linearOut (void *const *a_outBuf*, const Vector< DragParticle > & *a_inputT*)

8.60.1.3 template<> int linearSize (const Vector< DragParticle > & *a_input*)

8.61 edgeGhostBC.H File Reference

`#include "GhostBC.H"`

Include dependency graph for edgeGhostBC.H:



Compounds

- class **BoxEdgeBC**

  class to do edge-centered physical boundary conditions

- class **edgeDirichletBC**

  class to do Dirichlet edge-centered BC's

- class **NoOpBC**

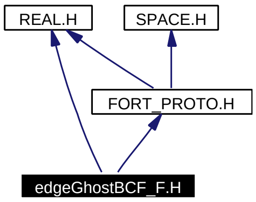
  class to do nothing on an edge

8.62 edgeGhostBCF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for edgeGhostBCF_F.H:



Defines

- #define **FORT_BOXEDGEBC** FORTRAN_NAME(BOXEDGEBC ,boxedgebc)

Functions

- void **FORT_BOXEDGEBC** (CHFp_FRA(state),CHFp_CONST_FRA(neumfac),CHFp_-CONST_FRA(dircfac),CHFp_CONST_FRA(inhmval),CHFp_BOX(bcBox),CHFp_-CONST_INT(dir),CHFp_CONST_INT(side),CHFp_CONST_REAL(dx),CHFp_CONST_-INT(startcomp),CHFp_CONST_INT(endcomp))

8.62.1 Define Documentation

8.62.1.1 #define FORT_BOXEDGEBC FORTRAN_NAME(BOXEDGEBC ,boxedgebc)

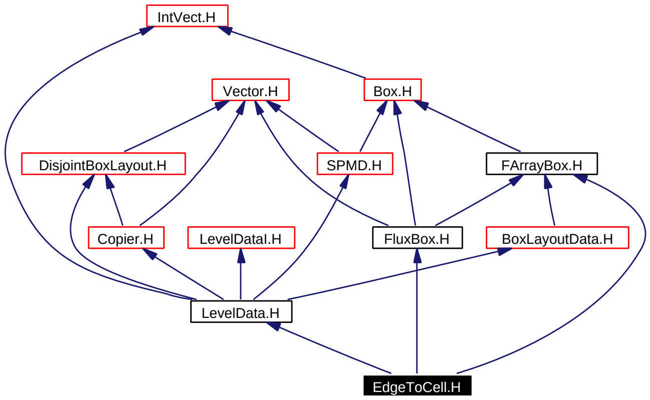
8.62.2 Function Documentation

8.62.2.1 void FORT_BOXEDGEBC (CHFp_FRA(state), CHFp_-CONST_FRA(neumfac), CHFp_CONST_FRA(dircfac), CHFp_CONST_FRA(inhmval), CHFp_BOX(bcBox), CHFp_CONST_-INT(dir), CHFp_CONST_INT(side), CHFp_CONST_REAL(dx), CHFp_CONST_INT(startcomp), CHFp_CONST_INT(endcomp))

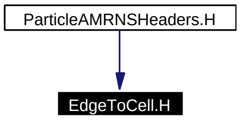
8.63 EdgeToCell.H File Reference

```
#include "LevelData.H"
#include "FArrayBox.H"
#include "FluxBox.H"
```

Include dependency graph for EdgeToCell.H:



This graph shows which files directly or indirectly include this file:



Functions

- void **EdgeToCell** (const **LevelData**< **FluxBox** > &a_edgeData, **LevelData**< **FArray-Box** > &a_cellData)

  averages cell-centered cellData to edge-centered edgeData

- void **EdgeToCell** (const **FluxBox** &a_edgeData, **FArrayBox** &a_cellData)

  averages cellData->edgeData (all directions) for one FAB

- void **EdgeToCell** (const **FluxBox** &a_edgeData, const int a_edgeComp, **FArrayBox** &a_cellData, const int a_cellComp, const int a_dir)

  averages cellData->edgeData in 1 direction, for one component in a FAB

8.63.1 Function Documentation

8.63.1.1 void EdgeToCell (const FluxBox & *a_edgeData*, const int *a_edgeComp*, FArrayBox & *a_cellData*, const int *a_cellComp*, const int *a_dir*)

averages cellData->edgeData in 1 direction, for one component in a FAB

8.63.1.2 void EdgeToCell (const FluxBox & *a_edgeData*, FArrayBox & *a_cellData*)

averages cellData->edgeData (all directions) for one FAB

averages cell-centered cellData to SpaceDim-dimensioned EdgeData – assumes all boundary conditions already set. Does this for all grids.

8.63.1.3 void EdgeToCell (const LevelData< FluxBox > & *a_edgeData*, LevelData< FArrayBox > & *a_cellData*)

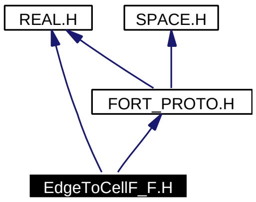
averages cell-centered cellData to edge-centered edgeData

8.64 EdgeToCellF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for EdgeToCellF_F.H:



Defines

- #define **FORT_EDGETOCELL** FORTRAN_NAME(EDGETOCELL ,edgetocell)

Functions

- void **FORT_EDGETOCELL** (CHFp_CONST_FRA1(edgeData),CHFp_FRA1(cellData),CHFp_BOX(cellBox),CHFp_CONST_INT(dir))

8.64.1 Define Documentation

8.64.1.1 #define FORT_EDGETOCELL FORTRAN_NAME(EDGETOCELL ,edgetocell)

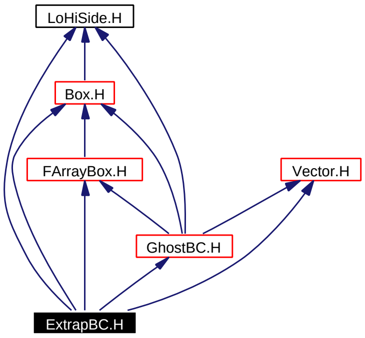
8.64.2 Function Documentation

8.64.2.1 void FORT_EDGETOCELL (CHFp_CONST_FRA1(edgeData), CHFp_FRA1(cellData), CHFp_BOX(cellBox), CHFp_CONST_INT(dir))

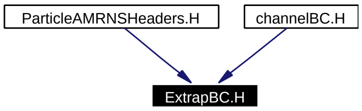
8.65 ExtrapBC.H File Reference

```
#include "Box.H"
#include "FArrayBox.H"
#include "Vector.H"
#include "LoHiSide.H"
#include "GhostBC.H"
```

Include dependency graph for ExtrapBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

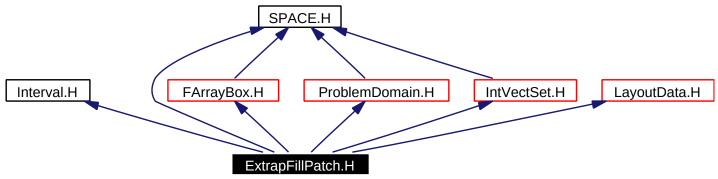
- class **ExtrapBC**

  Class to enforce homogeneous neumann boundary conditions.

8.66 ExtrapFillPatch.H File Reference

```
#include "Interval.H"
#include "SPACE.H"
#include "FArrayBox.H"
#include "LayoutData.H"
#include "ProblemDomain.H"
#include "IntVectSet.H"
```

Include dependency graph for ExtrapFillPatch.H:



Compounds

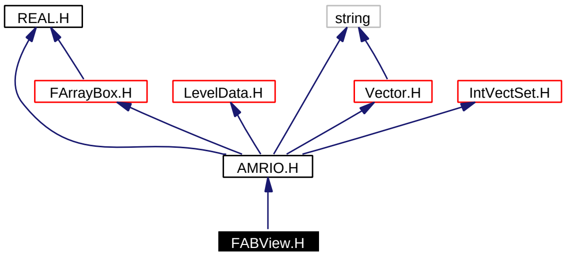
- class **ExtrapFillPatch**

  fills outer ghost cells by copying from inner ghost cells.

8.67 FABView.H File Reference

`#include "AMRIO.H"`

Include dependency graph for FABView.H:



Functions

- int **InitWriteFAB** ()

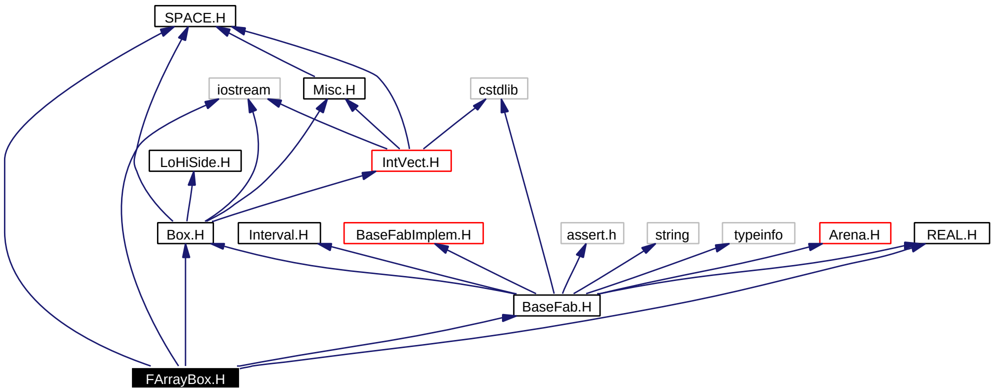
8.67.1 Function Documentation

8.67.1.1 int InitWriteFAB ()

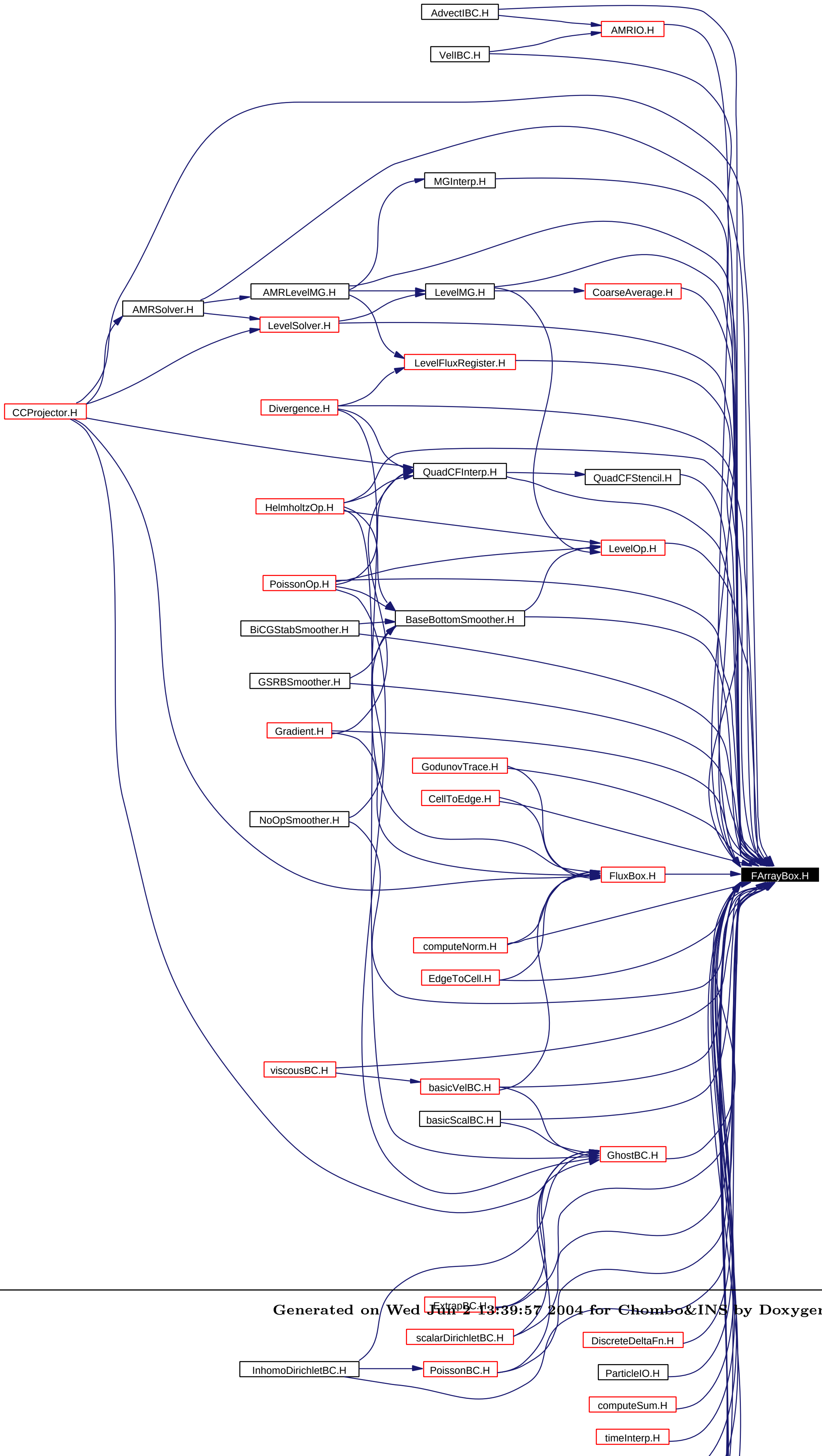
8.68 FArrayBox.H File Reference

```
#include <iostream>
#include "Box.H"
#include "BaseFab.H"
#include "REAL.H"
#include "SPACE.H"
```

Include dependency graph for FArrayBox.H:



This graph shows which files directly or indirectly include this file:


AdvectIBC.H
AMRIO.H
VelIBC.H
MGInterp.H
AMRLevelMG.H
LevelMG.H
CoarseAverage.H
AMRSolver.H
LevelSolver.H
LevelFluxRegister.H
CCProjector.H
Divergence.H
QuadCFInterp.H
QuadCFStencil.H
HelmholtzOp.H
LevelOp.H
PoissonOp.H
BaseBottomSmoother.H
BiCGStabSmoother.H
GSRBSmoother.H
Gradient.H
GodunovTrace.H
CellToEdge.H
NoOpSmoother.H
FluxBox.H
FArrayBox.H
computeNorm.H
EdgeToCell.H
viscousBC.H
basicVelBC.H
basicScalBC.H
GhostBC.H
ExtrapBC.H
scalarDirichletBC.H
DiscreteDeltaFn.H
InhomoDirichletBC.H
PoissonBC.H
ParticleIO.H
computeSum.H
timeInterp.H

Compounds

- class **FArrayBox**

8.69 FineInterp.H File Reference

`#include "REAL.H"`

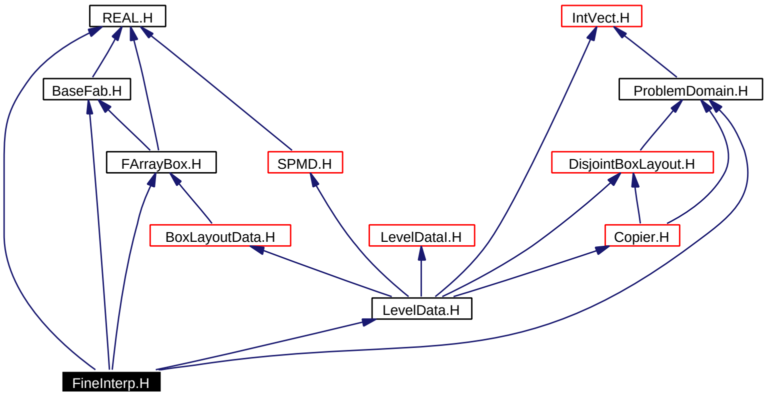
`#include "LevelData.H"`

`#include "BaseFab.H"`

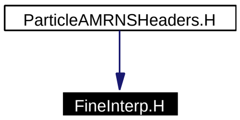
`#include "FArrayBox.H"`

`#include "ProblemDomain.H"`

Include dependency graph for FineInterp.H:



This graph shows which files directly or indirectly include this file:



Compounds

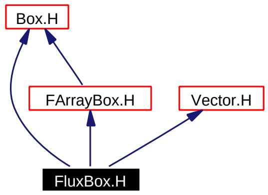
- class **FineInterp**

  replaces fine level data with interpolation of coarse level data.

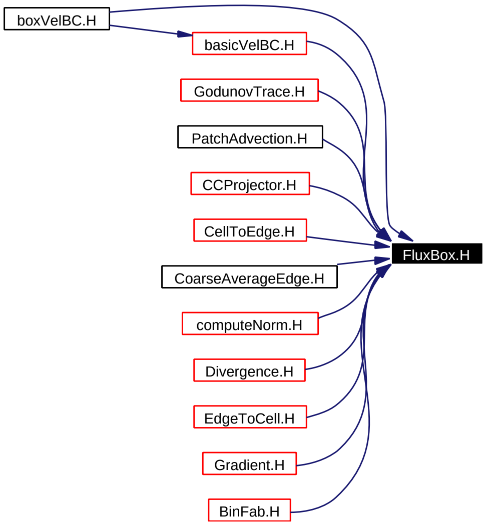
8.70 FluxBox.H File Reference

```
#include "Box.H"
#include "Vector.H"
#include "FArrayBox.H"
```

Include dependency graph for FluxBox.H:



This graph shows which files directly or indirectly include this file:



Compounds

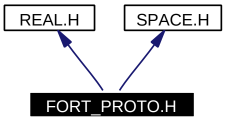
- class **FluxBox**

  A FArrayBox-like container for face-centered fluxes.

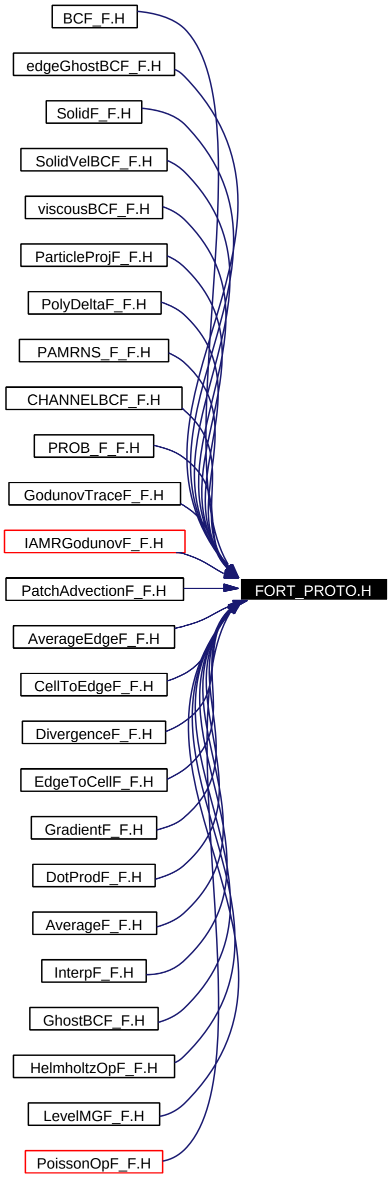
8.71 FORT_PROTO.H File Reference

```
#include "REAL.H"
#include "SPACE.H"
```

Include dependency graph for FORT_PROTO.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **chfptr**

Defines

- #define **FORTRAN_NAME**(NAME,name) name
- #define **CHF_INT**(i) &i
- #define **CHF_CONST_INT**(i) &i
- #define **CHF_REAL**(r) &r
- #define **CHF_CONST_REAL**(r) &r
- #define **CHF_I1D**(ia, dimen) ia,(const int*)**chfptr**(dimen-1)
- #define **CHF_CONST_I1D**(ia, dimen) ia,(const int*)**chfptr**(dimen-1)
- #define **CHF_R1D**(ra, dimen) ra,(const int*)**chfptr**(dimen-1)
- #define **CHF_CONST_R1D**(ra, dimen) ra,(const int*)**chfptr**(dimen-1)
- #define **CHF_INTVECT**(a) a.dataPtr( )
- #define **CHF_CONST_INTVECT**(a) a.dataPtr( )
- #define **CHF_REALVECT**(a) a.dataPtr( )
- #define **CHF_CONST_REALVECT**(a) a.dataPtr( )
- #define **CHF_VI**(v) &v[0],(const int*)**chfptr**(v.size()-1)
- #define **CHF_CONST_VI**(v) &v[0],(const int*)**chfptr**(v.size()-1)
- #define **CHF_VR**(v) &v[0],(const int*)**chfptr**(v.size()-1)
- #define **CHF_CONST_VR**(v) &v[0],(const int*)**chfptr**(v.size()-1)
- #define **CHF_BOX**(b)
- #define **CHF_FIA**(a)
- #define **CHF_FRA**(a)
- #define **CHF_CONST_FIA**(a)
- #define **CHF_CONST_FRA**(a)
- #define **CHF_FIA1**(a, n)
- #define **CHF_FRA1**(a, n)
- #define **CHF_CONST_FIA1**(a, n)
- #define **CHF_CONST_FRA1**(a, n)
- #define **CHFp_INT**(i) int* const i
- #define **CHFp_CONST_INT**(i) const int* const i
- #define **CHFp_REAL**(r) **Real*** const r
- #define **CHFp_CONST_REAL**(r) const **Real*** const r
- #define **CHFp_I1D**(ia, dimen) int* const ia, const int* const i ## ia ## hi0
- #define **CHFp_CONST_I1D**(ia, dimen) const int* const ia, const int* const i ## ia ## hi0
- #define **CHFp_R1D**(ra, dimen) **Real*** const ra, const int* const i ## ra ## hi0
- #define **CHFp_CONST_R1D**(ra, dimen) const **Real*** const ra, const int* const i ## ra ## hi0
- #define **CHFp_INTVECT**(r) int* const r
- #define **CHFp_CONST_INTVECT**(r) const int* const r
- #define **CHFp_REALVECT**(r) **Real*** const r
- #define **CHFp_CONST_REALVECT**(r) const **Real*** const r
- #define **CHFp_VI**(v) int * const v, const int* const i ## v ## hi0
- #define **CHFp_CONST_VI**(v) const int * const v, const int* const i ## v ## hi0
- #define **CHFp_VR**(v) **Real** * const v, const int* const i ## v ## hi0
- #define **CHFp_CONST_VR**(v) const **Real** * const v, const int* const i ## v ## hi0

- #define **CHFp_BOX**(b)
- #define **CHFp_FIA**(a)
- #define **CHFp_FRA**(a)
- #define **CHFp_CONST_FIA**(a)
- #define **CHFp_CONST_FRA**(a)
- #define **CHFp_FIA1**(a)
- #define **CHFp_FRA1**(a)
- #define **CHFp_CONST_FIA1**(a)
- #define **CHFp_CONST_FRA1**(a)

8.71.1 Define Documentation

8.71.1.1 #define CHF_BOX(b)

Value:

```
D_DECL( &b.loVect()[0] \
                            ,&b.loVect()[1] \
                            ,&b.loVect()[2] ) \
                    ,D_DECL( &b.hiVect()[0] \
                            ,&b.hiVect()[1] \
                            ,&b.hiVect()[2] )
```

8.71.1.2 #define CHF_CONST_FIA(a)

Value:

```
a.dataPtr( 0 ) \
                     ,D_DECL( &a.loVect()[0] \
                             ,&a.loVect()[1] \
                             ,&a.loVect()[2] ) \
                     ,D_DECL( &a.hiVect()[0] \
                             ,&a.hiVect()[1] \
                             ,&a.hiVect()[2] ) \
                     ,a.nCompPtr()
```

8.71.1.3 #define CHF_CONST_FIA1(a, n)

Value:

```
a.dataPtr( n ) \
                     ,D_DECL( &a.loVect()[0] \
                             ,&a.loVect()[1] \
                             ,&a.loVect()[2] ) \
                     ,D_DECL( &a.hiVect()[0] \
                             ,&a.hiVect()[1] \
                             ,&a.hiVect()[2] )
```

8.71.1.4 #define CHF_CONST_FRA(a)

Value:

```
a.dataPtr( 0 ) \
                      ,D_DECL( &a.loVect()[0] \
                              ,&a.loVect()[1] \
                              ,&a.loVect()[2] ) \
                      ,D_DECL( &a.hiVect()[0] \
                              ,&a.hiVect()[1] \
                              ,&a.hiVect()[2] ) \
                      ,a.nCompPtr()
```

8.71.1.5 #define CHF_CONST_FRA1(a, n)

Value:

```
a.dataPtr( n ) \
                      ,D_DECL( &a.loVect()[0] \
                              ,&a.loVect()[1] \
                              ,&a.loVect()[2] ) \
                      ,D_DECL( &a.hiVect()[0] \
                              ,&a.hiVect()[1] \
                              ,&a.hiVect()[2] )
```

8.71.1.6 #define CHF_CONST_I1D(ia, dimen) ia,(const int*)chfptr(dimen-1)

8.71.1.7 #define CHF_CONST_INT(i) &i

8.71.1.8 #define CHF_CONST_INTVECT(a) a.dataPtr( )

8.71.1.9 #define CHF_CONST_R1D(ra, dimen) ra,(const int*)chfptr(dimen-1)

8.71.1.10 #define CHF_CONST_REAL(r) &r

8.71.1.11 #define CHF_CONST_REALVECT(a) a.dataPtr( )

8.71.1.12 #define CHF_CONST_VI(v) &v[0],(const int*)chfptr(v.size()-1)

8.71.1.13 #define CHF_CONST_VR(v) &v[0],(const int*)chfptr(v.size()-1)

8.71.1.14 #define CHF_FIA(a)

Value:

```
a.dataPtr( 0 ) \
                      ,D_DECL( &a.loVect()[0] \
                              ,&a.loVect()[1] \
                              ,&a.loVect()[2] ) \
                      ,D_DECL( &a.hiVect()[0] \
                              ,&a.hiVect()[1] \
                              ,&a.hiVect()[2] ) \
                      ,a.nCompPtr()
```

8.71.1.15 #define CHF_FIA1(a, n)

Value:

```
a.dataPtr( n ) \
                      ,D_DECL( &a.loVect()[0] \
                              ,&a.loVect()[1] \
                              ,&a.loVect()[2] ) \
                      ,D_DECL( &a.hiVect()[0] \
                              ,&a.hiVect()[1] \
                              ,&a.hiVect()[2] )
```

8.71.1.16 #define CHF_FRA(a)

Value:

```
a.dataPtr( 0 ) \
                      ,D_DECL( &a.loVect()[0] \
                              ,&a.loVect()[1] \
                              ,&a.loVect()[2] ) \
                      ,D_DECL( &a.hiVect()[0] \
                              ,&a.hiVect()[1] \
                              ,&a.hiVect()[2] ) \
                      ,a.nCompPtr()
```

8.71.1.17 #define CHF_FRA1(a, n)

Value:

```
a.dataPtr( n ) \
                      ,D_DECL( &a.loVect()[0] \
                              ,&a.loVect()[1] \
                              ,&a.loVect()[2] ) \
                      ,D_DECL( &a.hiVect()[0] \
                              ,&a.hiVect()[1] \
                              ,&a.hiVect()[2] )
```

8.71.1.18 #define CHF_I1D(ia, dimen) ia,(const int*)chfptr(dimen-1)

8.71.1.19 #define CHF_INT(i) &i

8.71.1.20 #define CHF_INTVECT(a) a.dataPtr( )

8.71.1.21 #define CHF_R1D(ra, dimen) ra,(const int*)chfptr(dimen-1)

8.71.1.22 #define CHF_REAL(r) &r

8.71.1.23 #define CHF_REALVECT(a) a.dataPtr( )

8.71.1.24 #define CHF_VI(v) &v[0],(const int*)chfptr(v.size()-1)

8.71.1.25 #define CHF_VR(v) &v[0],(const int*)chfptr(v.size()-1)

8.71.1.26 #define CHFp_BOX(b)

Value:

```
D_DECL( const int* const i ## b ## lo0 \
                             ,const int* const i ## b ## lo1 \
                             ,const int* const i ## b ## lo2 ) \
                     ,D_DECL( const int* const i ## b ## hi0 \
                             ,const int* const i ## b ## hi1 \
                             ,const int* const i ## b ## hi2 )
```

8.71.1.27 #define CHFp_CONST_FIA(a)

Value:

```
const int *const a \
                     ,D_DECL( const int* const i ## a ## lo0 \
                             ,const int* const i ## a ## lo1 \
                             ,const int* const i ## a ## lo2 ) \
                     ,D_DECL( const int* const i ## a ## hi0 \
                             ,const int* const i ## a ## hi1 \
                             ,const int* const i ## a ## hi2 ) \
                             ,const int* const n ## a ## comp
```

8.71.1.28 #define CHFp_CONST_FIA1(a)

Value:

```
const int *const a \
                     ,D_DECL( const int* const i ## a ## lo0 \
                             ,const int* const i ## a ## lo1 \
                             ,const int* const i ## a ## lo2 ) \
                     ,D_DECL( const int* const i ## a ## hi0 \
                             ,const int* const i ## a ## hi1 \
                             ,const int* const i ## a ## hi2 )
```

8.71.1.29 #define CHFp_CONST_FRA(a)

Value:

```
const Real *const a \
                     ,D_DECL( const int* const i ## a ## lo0 \
                             ,const int* const i ## a ## lo1 \
                             ,const int* const i ## a ## lo2 ) \
                     ,D_DECL( const int* const i ## a ## hi0 \
                             ,const int* const i ## a ## hi1 \
                             ,const int* const i ## a ## hi2 ) \
                             ,const int* const n ## a ## comp
```

8.71.1.30 #define CHFp_CONST_FRA1(a)

Value:

```
const Real *const a \
                     ,D_DECL( const int* const i ## a ## lo0 \
                             ,const int* const i ## a ## lo1 \
                             ,const int* const i ## a ## lo2 ) \
                     ,D_DECL( const int* const i ## a ## hi0 \
                             ,const int* const i ## a ## hi1 \
                             ,const int* const i ## a ## hi2 )
```

8.71.1.31 #define CHFp_CONST_I1D(ia, dimen) const int* const ia, const int* const i ## ia ## hi0

8.71.1.32 #define CHFp_CONST_INT(i) const int* const i

8.71.1.33 #define CHFp_CONST_INTVECT(r) const int* const r

8.71.1.34 #define CHFp_CONST_R1D(ra, dimen) const Real* const ra, const int* const i ## ra ## hi0

8.71.1.35 #define CHFp_CONST_REAL(r) const Real* const r

8.71.1.36 #define CHFp_CONST_REALVECT(r) const Real* const r

8.71.1.37 #define CHFp_CONST_VI(v) const int * const v, const int* const i ## v ## hi0

8.71.1.38 #define CHFp_CONST_VR(v) const Real * const v, const int* const i ## v ## hi0

8.71.1.39 #define CHFp_FIA(a)

Value:

```
int *const a \
                     ,D_DECL( const int* const i ## a ## lo0 \
                             ,const int* const i ## a ## lo1 \
                             ,const int* const i ## a ## lo2 ) \
                     ,D_DECL( const int* const i ## a ## hi0 \
```

```
                              ,const int* const i ## a ## hi1 \
                              ,const int* const i ## a ## hi2 ) \
                              ,const int* const n ## a ## comp
```

8.71.1.40 #define CHFp_FIA1(a)

Value:

```
int *const a \
                      ,D_DECL( const int* const i ## a ## lo0 \
                              ,const int* const i ## a ## lo1 \
                              ,const int* const i ## a ## lo2 ) \
                      ,D_DECL( const int* const i ## a ## hi0 \
                              ,const int* const i ## a ## hi1 \
                              ,const int* const i ## a ## hi2 )
```

8.71.1.41 #define CHFp_FRA(a)

Value:

```
Real *const a \
                      ,D_DECL( const int* const i ## a ## lo0 \
                              ,const int* const i ## a ## lo1 \
                              ,const int* const i ## a ## lo2 ) \
                      ,D_DECL( const int* const i ## a ## hi0 \
                              ,const int* const i ## a ## hi1 \
                              ,const int* const i ## a ## hi2 ) \
                              ,const int* const n ## a ## comp
```

8.71.1.42 #define CHFp_FRA1(a)

Value:

```
Real *const a \
                      ,D_DECL( const int* const i ## a ## lo0 \
                              ,const int* const i ## a ## lo1 \
                              ,const int* const i ## a ## lo2 ) \
                      ,D_DECL( const int* const i ## a ## hi0 \
                              ,const int* const i ## a ## hi1 \
                              ,const int* const i ## a ## hi2 )
```

8.71.1.43 #define CHFp_I1D(ia, dimen) int* const ia, const int* const i ## ia ## hi0

8.71.1.44 #define CHFp_INT(i) int* const i

8.71.1.45 #define CHFp_INTVECT(r) int* const r

8.71.1.46 #define CHFp_R1D(ra, dimen) Real* const ra, const int* const i ## ra ## hi0

8.71.1.47 #define CHFp_REAL(r) Real* const r

8.71.1.48 #define CHFp_REALVECT(r) Real* const r

8.71.1.49 #define CHFp_VI(v) int * const v, const int* const i ## v ## hi0

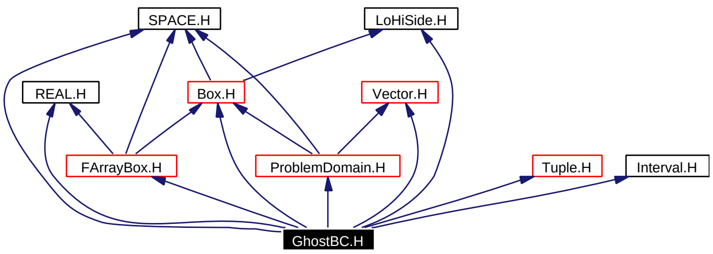
8.71.1.50 #define CHFp_VR(v) Real * const v, const int* const i ## v ## hi0

8.71.1.51 #define FORTRAN_NAME(NAME, name) name

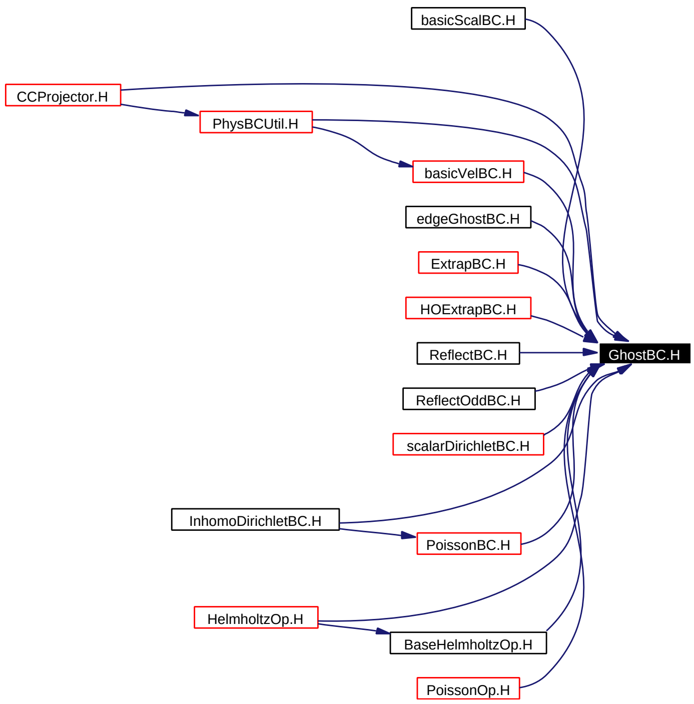
8.72 GhostBC.H File Reference

```
#include "Box.H"
#include "FArrayBox.H"
#include "REAL.H"
#include "SPACE.H"
#include "Tuple.H"
#include "Interval.H"
#include "Vector.H"
#include "LoHiSide.H"
#include "ProblemDomain.H"
```

Include dependency graph for GhostBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BoxGhostBC**

  Class to encapsulate the operations of ghost-cell boundary conditions.

- class **DomainGhostBC**

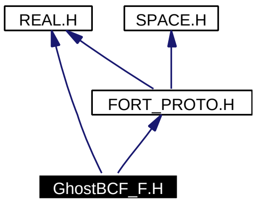
  Class to enforce ghost-cell based boundary conditions.

8.73 GhostBCF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for GhostBCF_F.H:



Defines

- #define **FORT_BOXGHOSTBC** FORTRAN_NAME(BOXGHOSTBC ,boxghostbc)

Functions

- void **FORT_BOXGHOSTBC** (CHFp_FRA(state),CHFp_CONST_FRA(neumfac),CHFp_-CONST_FRA(dircfac),CHFp_CONST_FRA(inhmval),CHFp_BOX(facebox),CHFp_-BOX(interiorBox),CHFp_CONST_INT(idir),CHFp_CONST_INT(side),CHFp_CONST_-REAL(dx),CHFp_CONST_INT(startcomp),CHFp_CONST_INT(endcomp))

8.73.1 Define Documentation

8.73.1.1 #define FORT_BOXGHOSTBC FORTRAN_NAME(BOXGHOSTBC ,boxghostbc)

8.73.2 Function Documentation

8.73.2.1 void FORT_BOXGHOSTBC (CHFp_FRA(state), CHFp_-CONST_FRA(neumfac), CHFp_CONST_FRA(dircfac), CHFp_CONST_FRA(inhmval), CHFp_BOX(facebox), CHFp_-BOX(interiorBox), CHFp_CONST_INT(idir), CHFp_CONST_INT(side), CHFp_CONST_REAL(dx), CHFp_CONST_INT(startcomp), CHFp_CONST_INT(endcomp))

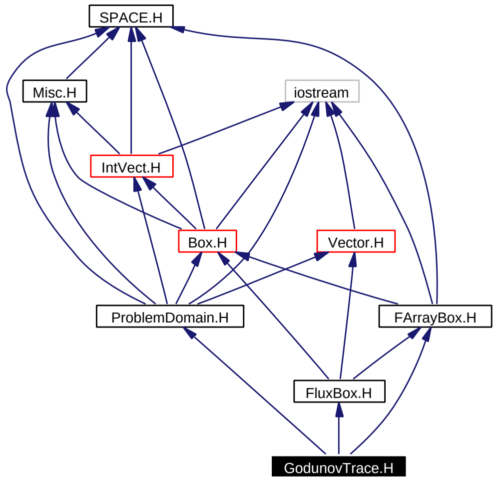
8.74 GodunovTrace.H File Reference

`#include "FluxBox.H"`

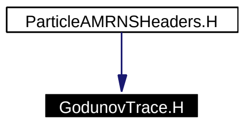
`#include "FArrayBox.H"`

`#include "ProblemDomain.H"`

Include dependency graph for GodunovTrace.H:



This graph shows which files directly or indirectly include this file:



Functions

- void **TraceState** (**FArrayBox** &a_stateHalf, const **FArrayBox** &a_state, const **FArrayBox** &a_cellVel, const **FluxBox** &a_advectionVel, const **FArrayBox** &a_source, const **ProblemDomain** &a_dProblem, const **Box** &a_gridBox, const **Real** a_dt, const **Real** a_dx, const int dir, const **Interval** &a_srcComps, const **Interval** &a_destComps)
- void **TraceState** (**FArrayBox** &a_stateHalf, const **FArrayBox** &a_state, const **FArrayBox** &a_cellVel, const **FluxBox** &a_advectionVel, const **FArrayBox** &a_source, const **Box** &a_dProblem, const **Box** &a_gridBox, const **Real** a_dt, const **Real** a_dx, const int dir, const **Interval** &a_srcComps, const **Interval** &a_destComps)
- void **TraceState** (**FArrayBox** &a_stateHalf, const **FArrayBox** &a_state, const **FArrayBox** &a_cellVel, const **FArrayBox** &a_source, const **ProblemDomain** &a_dProblem, const **Box** &a_gridBox, const **Real** a_dt, const **Real** a_dx, const int dir, const **Interval** &a_srcComps, const **Interval** &a_destComps)
- void **TraceState** (**FArrayBox** &a_stateHalf, const **FArrayBox** &a_state, const **FArrayBox** &a_cellVel, const **FArrayBox** &a_source, const **Box** &a_dProblem, const **Box** &a_gridBox, const **Real** a_dt, const **Real** a_dx, const int dir, const **Interval** &a_srcComps, const **Interval** &a_destComps)

8.74.1 Function Documentation

8.74.1.1 void TraceState (FArrayBox & *a_stateHalf*, const FArrayBox & *a_state*, const FArrayBox & *a_cellVel*, const FArrayBox & *a_source*, const Box & *a_dProblem*, const Box & *a_gridBox*, const Real *a_dt*, const Real *a_dx*, const int *dir*, const Interval & *a_srcComps*, const Interval & *a_destComps*)

Parameters:

a_stateHalf state at time t+dt/2 on edges in direction dir

a_state cell-centered state at time t

a_cellVel cell-centered velocity at time t

a_source cell-centered source

a_dProblem Physical domain

a_gridBox interior of grid patch

a_dt timeStep

a_dx cell-spacing

dir direction in which to perform tracing

a_srcComps which components to trace

a_destComps where to put traced components in a_stateHalf,

8.74.1.2 void TraceState (FArrayBox & *a_stateHalf*, const FArrayBox & *a_state*, const FArrayBox & *a_cellVel*, const FArrayBox & *a_source*, const ProblemDomain & *a_dProblem*, const Box & *a_gridBox*, const Real *a_dt*, const Real *a_dx*, const int *dir*, const Interval & *a_srcComps*, const Interval & *a_destComps*)

Parameters:

a_stateHalf state at time t+dt/2 on edges in direction dir

a_state cell-centered state at time t

a_cellVel cell-centered velocity at time t

a_source cell-centered source

a_dProblem Physical domain

a_gridBox interior of grid patch

a_dt timeStep

a_dx cell-spacing

dir direction in which to perform tracing

a_srcComps which components to trace

a_destComps where to put traced components in a_stateHalf,

8.74.1.3 void TraceState (FArrayBox & *a_stateHalf*, const FArrayBox & *a_state*, const FArrayBox & *a_cellVel*, const FluxBox & *a_advectionVel*, const FArrayBox & *a_source*, const Box & *a_dProblem*, const Box & *a_gridBox*, const Real *a_dt*, const Real *a_dx*, const int *dir*, const Interval & *a_srcComps*, const Interval & *a_destComps*)

Parameters:

a_stateHalf state at time t+dt/2 on edges in direction dir

a_state cell-centered state at time t

a_cellVel cell-centered velocity at time t

a_advectionVel edge-centered advection velocity at time t+dt/2

a_source cell-centered source

a_dProblem Physical domain

a_gridBox interior of grid patch

a_dt timeStep

a_dx cell-spacing

dir direction in which to perform tracing

a_srcComps which components to trace

a_destComps where to put traced components in a_stateHalf,

8.74.1.4 void TraceState (FArrayBox & *a_stateHalf*, const FArrayBox & *a_state*, const FArrayBox & *a_cellVel*, const FluxBox & *a_advectionVel*, const FArrayBox & *a_source*, const ProblemDomain & *a_dProblem*, const Box & *a_gridBox*, const Real *a_dt*, const Real *a_dx*, const int *dir*, const Interval & *a_srcComps*, const Interval & *a_destComps*)

Parameters:

a_stateHalf state at time t+dt/2 on edges in direction dir

a_state cell-centered state at time t

a_cellVel cell-centered velocity at time t

a_advectionVel edge-centered advection velocity at time t+dt/2

a_source cell-centered source

a_dProblem Physical domain

a_gridBox interior of grid patch

a_dt timeStep

a_dx cell-spacing

dir direction in which to perform tracing

a_srcComps which components to trace

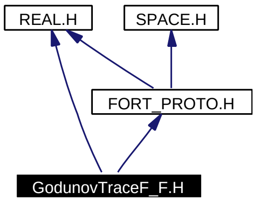
a_destComps where to put traced components in a_stateHalf,

8.75 GodunovTraceF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for GodunovTraceF_F.H:



Defines

- #define **FORT_UPWINDCELLTOEDGE** FORTRAN_NAME(UPWINDCELLTOEDGE ,upwindcelltoedge)
- #define **FORT_SLOPES** FORTRAN_NAME(SLOPES ,slopes)
- #define **FORT_TRANSVERSE** FORTRAN_NAME(TRANSVERSE ,transverse)
- #define **FORT_TRANSVERSECROSS** FORTRAN_NAME(TRANSVERSECROSS ,transversecross)
- #define **FORT_PREDICT** FORTRAN_NAME(PREDICT ,predict)

Functions

- void **FORT_UPWINDCELLTOEDGE** (CHFp_FRA1(edgeState),CHFp_CONST_FRA1(cellState),CHFp_CONST_FRA(edgeVel),CHFp_BOX(edgeBox),CHFp_CONST_INT(dir))
- void **FORT_SLOPES** (CHFp_FRA1(delS),CHFp_CONST_FRA1(s),CHFp_BOX(gridBox),CHFp_CONST_INT(dir),CHFp_CONST_INT(doLimit))
- void **FORT_TRANSVERSE** (CHFp_FRA1(sHat),CHFp_CONST_FRA1(s),CHFp_CONST_FRA1(vel),CHFp_BOX(gridBox),CHFp_CONST_REAL(dt),CHFp_CONST_REAL(dx),CHFp_CONST_INT(dir))
- void **FORT_TRANSVERSECROSS** (CHFp_FRA1(sHat),CHFp_CONST_FRA1(s),CHFp_CONST_FRA(vel),CHFp_BOX(gridBox),CHFp_CONST_REAL(dt),CHFp_CONST_REAL(dx),CHFp_CONST_INT(dir))
- void **FORT_PREDICT** (CHFp_FRA1(sHalf),CHFp_CONST_FRA1(sHat),CHFp_CONST_FRA1(delS),CHFp_CONST_FRA1(vel),CHFp_CONST_FRA1(advVel),CHFp_BOX(edgeBox),CHFp_CONST_REAL(dt),CHFp_CONST_REAL(dx),CHFp_CONST_INT(dir))

8.75.1 Define Documentation

8.75.1.1 #define FORT_PREDICT FORTRAN_NAME(PREDICT ,predict)

8.75.1.2 #define FORT_SLOPES FORTRAN_NAME(SLOPES ,slopes)

8.75.1.3 #define FORT_TRANSVERSE FORTRAN_NAME(TRANSVERSE ,transverse)

8.75.1.4 #define FORT_TRANSVERSECROSS FORTRAN_NAME(TRANSVERSECROSS ,transversecross)

8.75.1.5 #define FORT_UPWINDCELLTOEDGE FORTRAN_NAME(UPWINDCELLTOEDGE ,upwindcelltoedge)

8.75.2 Function Documentation

8.75.2.1 void FORT_PREDICT (CHFp_FRA1(sHalf), CHFp_CONST_FRA1(sHat), CHFp_CONST_FRA1(delS), CHFp_CONST_FRA1(vel), CHFp_CONST_FRA1(advVel), CHFp_BOX(edgeBox), CHFp_CONST_REAL(dt), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir))

8.75.2.2 void FORT_SLOPES (CHFp_FRA1(delS), CHFp_CONST_FRA1(s), CHFp_BOX(gridBox), CHFp_CONST_INT(dir), CHFp_CONST_INT(doLimit))

8.75.2.3 void FORT_TRANSVERSE (CHFp_FRA1(sHat), CHFp_CONST_FRA1(s), CHFp_CONST_FRA1(vel), CHFp_BOX(gridBox), CHFp_CONST_REAL(dt), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir))

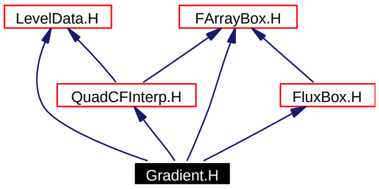
8.75.2.4 void FORT_TRANSVERSECROSS (CHFp_FRA1(sHat), CHFp_CONST_FRA1(s), CHFp_CONST_FRA(vel), CHFp_BOX(gridBox), CHFp_CONST_REAL(dt), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir))

8.75.2.5 void FORT_UPWINDCELLTOEDGE (CHFp_FRA1(edgeState), CHFp_CONST_FRA1(cellState), CHFp_CONST_FRA(edgeVel), CHFp_BOX(edgeBox), CHFp_CONST_INT(dir))

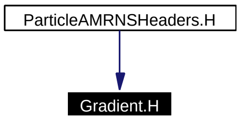
8.76 Gradient.H File Reference

```
#include "LevelData.H"
#include "FArrayBox.H"
#include "FluxBox.H"
#include "QuadCFInterp.H"
```

Include dependency graph for Gradient.H:



This graph shows which files directly or indirectly include this file:



Functions

- void **levelGradientCC** (**LevelData**< **FArrayBox** > &a_grad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phiCrsePtr, const **Real** a_dx, const int a_nRefCrse, const **ProblemDomain** &a_dProblem)
- void **levelGradientCC** (**LevelData**< **FArrayBox** > &a_grad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phiCrsePtr, const **Real** a_dx, const int a_nRefCrse, const **Box** &a_dProblem)
- void **levelGradientCC** (**LevelData**< **FArrayBox** > &a_grad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phiCrsePtr, const **Real** a_dx, **QuadCFInterp** &a_cfInterp)
- void **levelGradientCC** (**LevelData**< **FArrayBox** > &a_grad, const **LevelData**< **FArrayBox** > &a_phi, const **Real** a_dx)
- void **compGradientCC** (**LevelData**< **FArrayBox** > &a_grad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phiCrsePtr, const **LevelData**< **FArrayBox** > *a_phiFinePtr, const **Real** a_dx, const int a_nRefCrse, const int a_nRefFine, const **ProblemDomain** &a_dProblem)
- void **compGradientCC** (**LevelData**< **FArrayBox** > &a_grad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phiCrsePtr, const **LevelData**< **FArrayBox** > *a_phiFinePtr, const **Real** a_dx, const int a_nRefCrse, const int a_nRefFine, const **Box** &a_dProblem)
- void **compGradientCC** (**LevelData**< **FArrayBox** > &a_grad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phiCrsePtr, const **LevelData**< **FArrayBox** > *a_phiFinePtr, const **Real** a_dx, const int a_nRefFine, const **ProblemDomain** &a_dProblem, **QuadCFInterp** &a_cfInterpCrse)

- void **compGradientCC** (**LevelData**< **FArrayBox** > &a_grad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiCrsePtr, const **LevelData**< **FArrayBox** > ∗a_phiFinePtr, const **Real** a_dx, const int a_nRefFine, const **Box** &a_dProblem, **QuadCFInterp** &a_cfInterpCrse)

- void **compGradientCC** (**LevelData**< **FArrayBox** > &a_Grad, const **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiFinePtr, const **Real** a_dx, const int a_nRefFine, const **ProblemDomain** &a_dProblem)

- void **compGradientCC** (**LevelData**< **FArrayBox** > &a_Grad, const **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiFinePtr, const **Real** a_dx, const int a_nRefFine, const **Box** &a_dProblem)

- void **levelGradientMAC** (**LevelData**< **FluxBox** > &a_edgeGrad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiCrsePtr, const **Real** a_dx, const int a_nRefCrse, const **ProblemDomain** &a_dProblem)

- void **levelGradientMAC** (**LevelData**< **FluxBox** > &a_edgeGrad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiCrsePtr, const **Real** a_dx, const int a_nRefCrse, const **Box** &a_dProblem)

- void **levelGradientMAC** (**LevelData**< **FluxBox** > &a_edgeGrad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiCrsePtr, const **Real** a_dx, **QuadCFInterp** &a_cfInterpCrse)

- void **levelGradientMAC** (**LevelData**< **FluxBox** > &a_edgeGrad, const **LevelData**< **FArrayBox** > &a_phi, const **Real** a_dx)

- void **compGradientMAC** (**LevelData**< **FluxBox** > &a_edgeGrad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiCrse, const **LevelData**< **FArrayBox** > ∗a_phiFine, const **Real** a_dx, const int a_nRefCrse, const int a_nRefFine, const **Box** &a_dProblem)

- void **compGradientMAC** (**LevelData**< **FluxBox** > &a_edgeGrad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiCrse, const **LevelData**< **FArrayBox** > ∗a_phiFine, const **Real** a_dx, const int a_nRefCrse, const int a_nRefFine, const **ProblemDomain** &a_dProblem)

- void **compGradientMAC** (**LevelData**< **FluxBox** > &a_edgeGrad, **LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > ∗a_phiCrse, const **LevelData**< **FArrayBox** > ∗a_phiFine, const **Real** a_dx, const int a_nRefFine, **QuadCFInterp** &a_cfInterpCrse)

8.76.1 Function Documentation

8.76.1.1 void compGradientCC (LevelData< FArrayBox > & *a_Grad*, const LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiFinePtr*, const Real *a_dx*, const int *a_nRefFine*, const Box & *a_dProblem*)

8.76.1.2 void compGradientCC (LevelData< FArrayBox > & *a_Grad*, const LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiFinePtr*, const Real *a_dx*, const int *a_nRefFine*, const ProblemDomain & *a_dProblem*)

8.76.1.3 void compGradientCC (LevelData< FArrayBox > & *a_grad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const LevelData< FArrayBox > ∗ *a_phiFinePtr*, const Real *a_dx*, const int *a_nRefFine*, const Box & *a_dProblem*, QuadCFInterp & *a_cfInterpCrse*)

8.76.1.4 void compGradientCC (LevelData< FArrayBox > & *a_grad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const LevelData< FArrayBox > ∗ *a_phiFinePtr*, const Real *a_dx*, const int *a_nRefFine*, const ProblemDomain & *a_dProblem*, QuadCFInterp & *a_cfInterpCrse*)

8.76.1.5 void compGradientCC (LevelData< FArrayBox > & *a_grad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const LevelData< FArrayBox > ∗ *a_phiFinePtr*, const Real *a_dx*, const int *a_nRefCrse*, const int *a_nRefFine*, const Box & *a_dProblem*)

8.76.1.6 void compGradientCC (LevelData< FArrayBox > & *a_grad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const LevelData< FArrayBox > ∗ *a_phiFinePtr*, const Real *a_dx*, const int *a_nRefCrse*, const int *a_nRefFine*, const ProblemDomain & *a_dProblem*)

8.76.1.7 void compGradientMAC (LevelData< FluxBox > & *a_edgeGrad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrse*, const LevelData< FArrayBox > ∗ *a_phiFine*, const Real *a_dx*, const int *a_nRefFine*, QuadCFInterp & *a_cfInterpCrse*)

8.76.1.8 void compGradientMAC (LevelData< FluxBox > & *a_edgeGrad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrse*, const LevelData< FArrayBox > ∗ *a_phiFine*, const Real *a_dx*, const int *a_nRefCrse*, const int *a_nRefFine*, const ProblemDomain & *a_dProblem*)

8.76.1.9 void compGradientMAC (LevelData< FluxBox > & *a_edgeGrad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrse*, const LevelData< FArrayBox > ∗ *a_phiFine*, const Real *a_dx*, const int *a_nRefCrse*, const int *a_nRefFine*, const Box & *a_dProblem*)

8.76.1.10 void levelGradientCC (LevelData< FArrayBox > & *a_grad*, const LevelData< FArrayBox > & *a_phi*, const Real *a_dx*)

8.76.1.11 void levelGradientCC (LevelData< FArrayBox > & *a_grad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const Real *a_dx*, QuadCFInterp & *a_cfInterp*)

8.76.1.12 void levelGradientCC (LevelData< FArrayBox > & *a_grad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const Real *a_dx*, const int *a_nRefCrse*, const Box & *a_dProblem*)

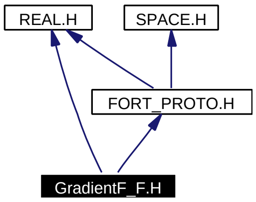
8.76.1.13 void levelGradientCC (LevelData< FArrayBox > & *a_grad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*,

8.77 GradientF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for GradientF_F.H:



Defines

- #define **FORT_MACGRAD** FORTRAN_NAME(MACGRAD ,macgrad)
- #define **FORT_CRSEONESIDEGRAD** FORTRAN_NAME(CRSEONESIDEGRAD ,crseonesidegrad)
- #define **FORT_GRADCC** FORTRAN_NAME(GRADCC ,gradcc)

Functions

- void **FORT_MACGRAD** (CHFp_FRA1(edgeGrad),CHFp_CONST_FRA1(phi),CHFp_BOX(edgeGrid),CHFp_CONST_REAL(dx),CHFp_INT(dir),CHFp_INT(edgeDir),CHFp_BOX(BLcornerBox),CHFp_BOX(BRcornerBox),CHFp_BOX(ULcornerBox),CHFp_BOX(URcornerBox))
- void **FORT_CRSEONESIDEGRAD** (CHFp_FRA1(edgeGrad),CHFp_FIA1(mask),CHFp_BOX(loEdgeBox),CHFp_BOX(hiEdgeBox),CHFp_INT(dir),CHFp_INT(doLo),CHFp_INT(doHi))
- void **FORT_GRADCC** (CHFp_FRA1(gradPhi),CHFp_CONST_FRA1(phi),CHFp_BOX(gridBox),CHFp_CONST_REAL(dx),CHFp_INT(dir))

8.77.1 Define Documentation

8.77.1.1 #define FORT_CRSEONESIDEGRAD FORTRAN_NAME(CRSEONESIDEGRAD ,crseonesidegrad)

8.77.1.2 #define FORT_GRADCC FORTRAN_NAME(GRADCC ,gradcc)

8.77.1.3 #define FORT_MACGRAD FORTRAN_NAME(MACGRAD ,macgrad)

8.77.2 Function Documentation

8.77.2.1 void FORT_CRSEONESIDEGRAD (CHFp_FRA1(edgeGrad), CHFp_FIA1(mask), CHFp_BOX(loEdgeBox), CHFp_BOX(hiEdgeBox), CHFp_INT(dir), CHFp_INT(doLo), CHFp_INT(doHi))

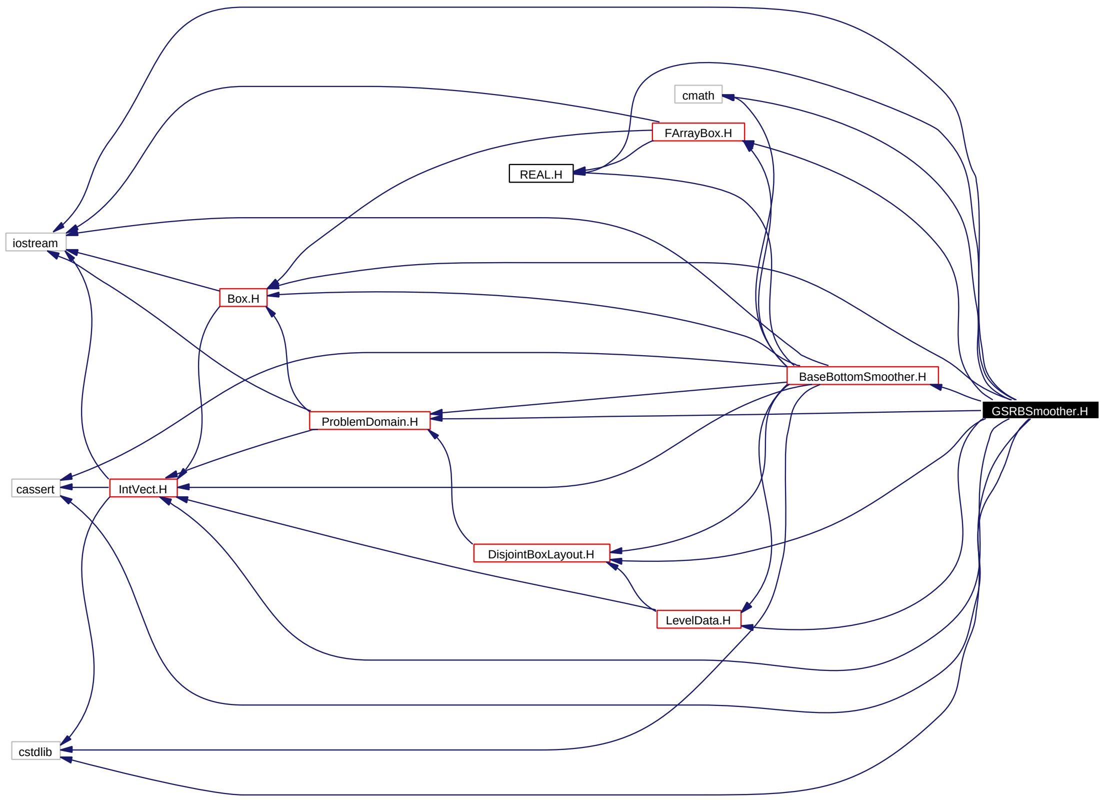
8.77.2.2 void FORT_GRADCC (CHFp_FRA1(gradPhi), CHFp_CONST_-FRA1(phi), CHFp_BOX(gridBox), CHFp_CONST_REAL(dx), CHFp_INT(dir))

8.77.2.3 void FORT_MACGRAD (CHFp_FRA1(edgeGrad), CHFp_CONST_-FRA1(phi), CHFp_BOX(edgeGrid), CHFp_CONST_REAL(dx), CHFp_INT(dir), CHFp_INT(edgeDir), CHFp_BOX(BLcornerBox), CHFp_BOX(BRcornerBox), CHFp_BOX(ULcornerBox), CHFp_BOX(URcornerBox))

8.78 GSRBSmoother.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "FArrayBox.H"
#include "ProblemDomain.H"
#include "BaseBottomSmoother.H"
```

Include dependency graph for GSRBSmoother.H:



Compounds

- class **GSRBSmoother**

GSRB smoothing as a bottom smoother.

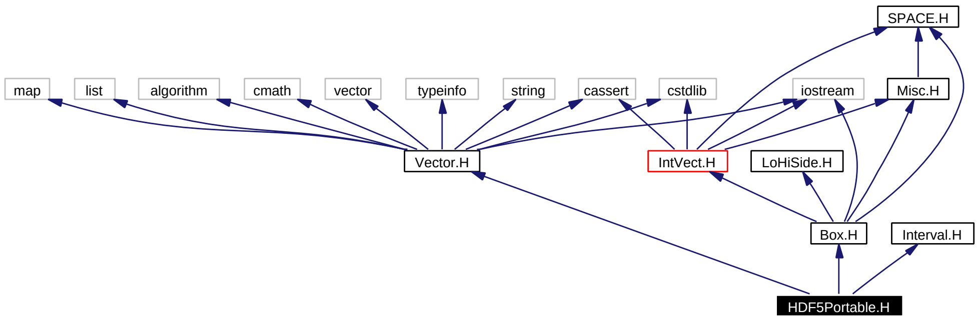
8.79 HDF5Portable.H File Reference

`#include "Vector.H"`

`#include "Box.H"`

`#include "Interval.H"`

Include dependency graph for HDF5Portable.H:



Typedefs

- typedef long **hid_t**

Functions

- template<class T> void **dataTypes** (**Vector**< **hid_t** > &a_types, const T &a_dummySpecializationArg)
- template<class T> void **dataSize** (const T &item, **Vector**< int > &a_sizes, const **Box** &box, const **Interval** &comps)
- template<class T> void **write** (const T &item, **Vector**< void * > &a_allocatedBuffers, const **Box** &box, const **Interval** &comps)
- template<class T> void **read** (T &item, **Vector**< void * > &a_allocatedBuffers, const **Box** &box, const **Interval** &comps)
- template<class T> const char * **name** (const T &a_dummySpecializationArg)

8.79.1 Typedef Documentation

8.79.1.1 typedef long hid_t

8.79.2 Function Documentation

8.79.2.1 template<class T> void dataSize (const T & *item*, Vector< int > & *a_sizes*, const Box & *box*, const Interval & *comps*)

8.79.2.2 template<class T> void dataTypes (Vector< hid_t > & *a_types*, const T & *a_dummySpecializationArg*)

8.79.2.3 template<class T> const char* name (const T & *a_dummySpecializationArg*)

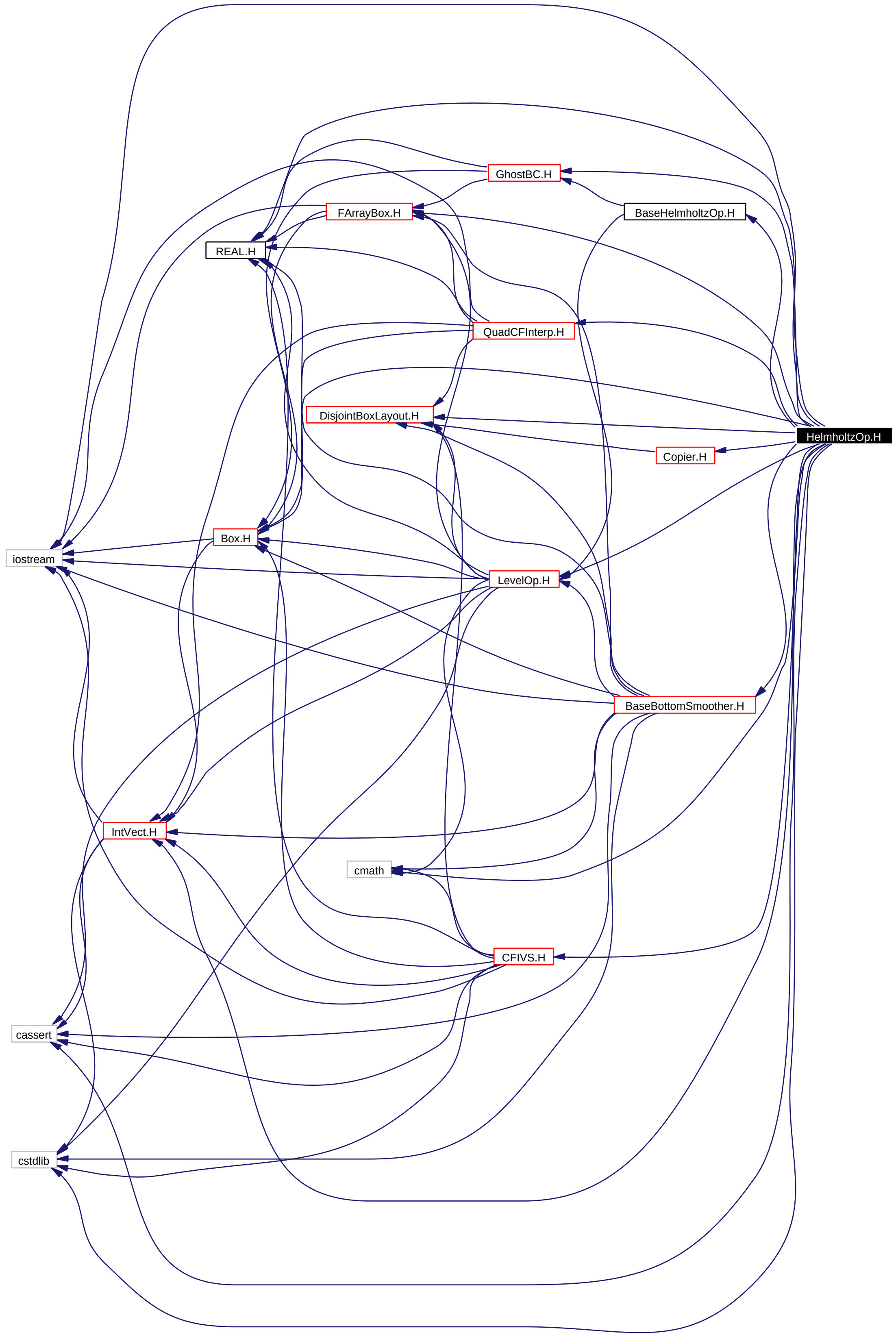
8.79.2.4 template<class T> void read (T & *item*, Vector< void * > & *a_allocatedBuffers*, const Box & *box*, const Interval & *comps*)

8.79.2.5 template<class T> void write (const T & *item*, Vector< void * > & *a_allocatedBuffers*, const Box & *box*, const Interval & *comps*)

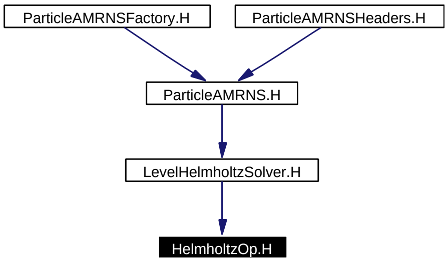
8.80 HelmholtzOp.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "DisjointBoxLayout.H"
#include "FArrayBox.H"
#include "QuadCFInterp.H"
#include "Copier.H"
#include "GhostBC.H"
#include "LevelOp.H"
#include "CFIVS.H"
#include "BaseBottomSmoother.H"
#include "BaseHelmholtzOp.H"
```

Include dependency graph for HelmholtzOp.H:


GhostBC.H
BaseHelmholtzOp.H
FArrayBox.H
REAL.H
QuadCFInterp.H
DisjointBoxLayout.H
HelmholtzOp.H
Copier.H
Box.H
iostream
LevelOp.H
BaseBottomSmoother.H
IntVect.H
cmath
CFIVS.H
cassert
cstdlib

This graph shows which files directly or indirectly include this file:



Compounds

- class **HelmholtzOp**

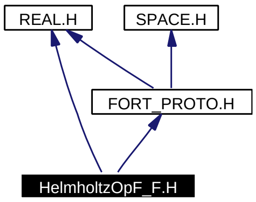
  Helmholtz operator class.

8.81 HelmholtzOpF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for HelmholtzOpF_F.H:



Defines

- #define **FORT_GSRBLEVELHELM** FORTRAN_NAME(GSRBLEVELHELM ,gsrblevelhelm)
- #define **FORT_OPERATORHELM** FORTRAN_NAME(OPERATORHELM ,operatorhelm)

Functions

- void **FORT_GSRBLEVELHELM** (CHFp_FRA(phi),CHFp_CONST_FRA(rhs),CHFp_BOX(region),CHFp_CONST_REAL(dx),CHFp_CONST_REAL(alpha),CHFp_CONST_REAL(beta),CHFp_CONST_INT(redBlack))
- void **FORT_OPERATORHELM** (CHFp_FRA(lofphi),CHFp_CONST_FRA(phi),CHFp_BOX(region),CHFp_CONST_REAL(dx),CHFp_CONST_REAL(alpha),CHFp_CONST_REAL(beta))

8.81.1 Define Documentation

8.81.1.1 #define FORT_GSRBLEVELHELM FORTRAN_NAME(GSRBLEVELHELM ,gsrblevelhelm)

8.81.1.2 #define FORT_OPERATORHELM FORTRAN_NAME(OPERATORHELM ,operatorhelm)

8.81.2 Function Documentation

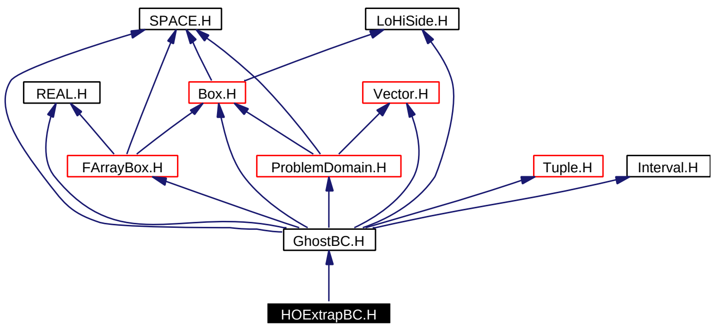
8.81.2.1 void FORT_GSRBLEVELHELM (CHFp_FRA(phi), CHFp_CONST_FRA(rhs), CHFp_BOX(region), CHFp_CONST_REAL(dx), CHFp_CONST_REAL(alpha), CHFp_CONST_REAL(beta), CHFp_CONST_INT(redBlack))

8.81.2.2 void FORT_OPERATORHELM (CHFp_FRA(lofphi), CHFp_CONST_FRA(phi), CHFp_BOX(region), CHFp_CONST_REAL(dx), CHFp_CONST_REAL(alpha), CHFp_CONST_REAL(beta))

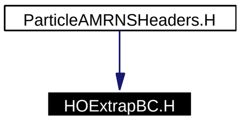
8.82 HOExtrapBC.H File Reference

`#include "GhostBC.H"`

Include dependency graph for HOExtrapBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **HOExtrapBC**

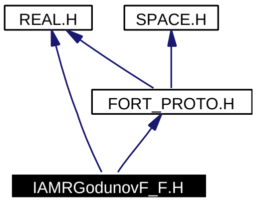
  class to do extrapolation BC's

8.83 IAMRGodunovF_F.H File Reference

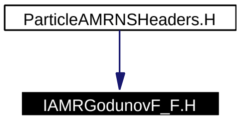
`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for IAMRGodunovF_F.H:



This graph shows which files directly or indirectly include this file:



Defines

- #define **FORT_UDELS** FORTRAN_NAME(UDELS ,udels)

Functions

- void **FORT_UDELS** (CHFp_FRA(uDotDelS),CHFp_FRA(sEdge),CHFp_FRA(vel),CHFp_BOX(gridBox),CHFp_REAL(dx),CHFp_INT(dir))

8.83.1 Define Documentation

8.83.1.1 #define FORT_UDELS FORTRAN_NAME(UDELS ,udels)

8.83.2 Function Documentation

8.83.2.1 void FORT_UDELS (CHFp_FRA(uDotDelS), CHFp_FRA(sEdge), CHFp_FRA(vel), CHFp_BOX(gridBox), CHFp_REAL(dx), CHFp_INT(dir))

8.84 InhomoDirichletBC.H File Reference

`#include "Box.H"`

`#include "FArrayBox.H"`

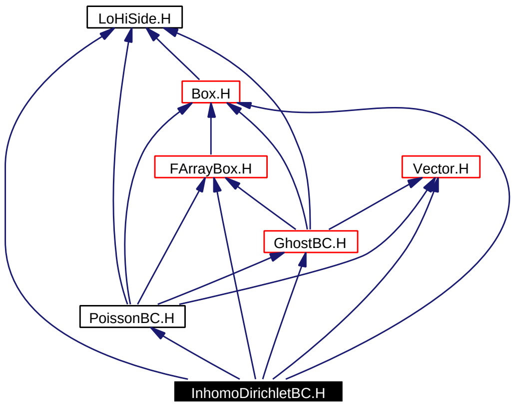
`#include "Vector.H"`

`#include "LoHiSide.H"`

`#include "GhostBC.H"`

`#include "PoissonBC.H"`

Include dependency graph for InhomoDirichletBC.H:



Compounds

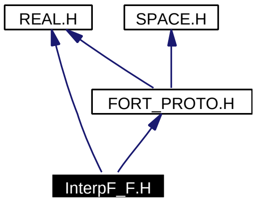
- class **InhomoDirichletBC**

  implements dirichlet BCs for scalars

8.85 InterpF_F.H File Reference

```
#include "FORT_PROTO.H"
#include "REAL.H"
```

Include dependency graph for InterpF_F.H:



Defines

- #define **FORT_INTERPCONSTANT** FORTRAN_NAME(INTERPCONSTANT ,interpconstant)
- #define **FORT_INTERPCENTRALSLOPE** FORTRAN_NAME(INTERPCENTRALSLOPE ,interpcentralslope)
- #define **FORT_INTERPHISIDESLOPE** FORTRAN_NAME(INTERPHISIDESLOPE ,interphisideslope)
- #define **FORT_INTERPLOSIDESLOPE** FORTRAN_NAME(INTERPLOSIDESLOPE ,interplosideslope)
- #define **FORT_INTERPLIMIT** FORTRAN_NAME(INTERPLIMIT ,interplimit)
- #define **FORT_INTERPLINEAR** FORTRAN_NAME(INTERPLINEAR ,interplinear)

Functions

- void **FORT_INTERPCONSTANT** (CHFp_FRA(fine),CHFp_CONST_FRA(coarse),CHFp_BOX(b),CHFp_CONST_INT(ref_ratio),CHFp_BOX(bref))
- void **FORT_INTERPCENTRALSLOPE** (CHFp_FRA(slope),CHFp_CONST_FRA(state),CHFp_BOX(b),CHFp_CONST_INT(dir))
- void **FORT_INTERPHISIDESLOPE** (CHFp_FRA(slope),CHFp_CONST_FRA(state),CHFp_BOX(b),CHFp_CONST_INT(dir))
- void **FORT_INTERPLOSIDESLOPE** (CHFp_FRA(slope),CHFp_CONST_FRA(state),CHFp_BOX(b),CHFp_CONST_INT(dir))
- void **FORT_INTERPLIMIT** (CHFp_FRA(islope),CHFp_FRA(jslope),CHFp_FRA(kslope),CHFp_CONST_FRA(state),CHFp_BOX(bcoarse),CHFp_BOX(bn),CHFp_BOX(physdomain))
- void **FORT_INTERPLINEAR** (CHFp_FRA(fine),CHFp_CONST_FRA(slope),CHFp_BOX(b),CHFp_CONST_INT(dir),CHFp_CONST_INT(ref_ratio),CHFp_BOX(bref))

8.85.1 Define Documentation

8.85.1.1 #define FORT_INTERPCENTRALSLOPE FORTRAN_NAME(INTERPCENTRALSLOPE ,interpcentralslope)

8.85.1.2 #define FORT_INTERPCONSTANT FORTRAN_NAME(INTERPCONSTANT ,interpconstant)

8.85.1.3 #define FORT_INTERPHISIDESLOPE FORTRAN_NAME(INTERPHISIDESLOPE ,interphisideslope)

8.85.1.4 #define FORT_INTERPLIMIT FORTRAN_NAME(INTERPLIMIT ,interplimit)

8.85.1.5 #define FORT_INTERPLINEAR FORTRAN_NAME(INTERPLINEAR ,interplinear)

8.85.1.6 #define FORT_INTERPLOSIDESLOPE FORTRAN_NAME(INTERPLOSIDESLOPE ,interplosideslope)

8.85.2 Function Documentation

8.85.2.1 void FORT_INTERPCENTRALSLOPE (CHFp_FRA(slope), CHFp_CONST_FRA(state), CHFp_BOX(b), CHFp_CONST_INT(dir))

8.85.2.2 void FORT_INTERPCONSTANT (CHFp_FRA(fine), CHFp_-CONST_FRA(coarse), CHFp_BOX(b), CHFp_CONST_INT(ref_ratio), CHFp_BOX(bref))

8.85.2.3 void FORT_INTERPHISIDESLOPE (CHFp_FRA(slope), CHFp_CONST_FRA(state), CHFp_BOX(b), CHFp_CONST_INT(dir))

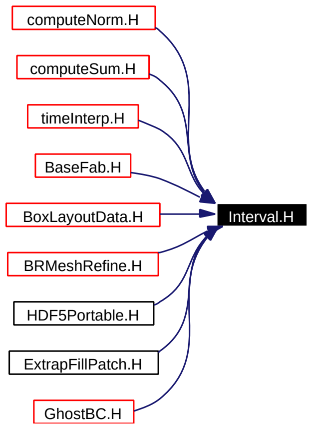
8.85.2.4 void FORT_INTERPLIMIT (CHFp_FRA(islope), CHFp_FRA(jslope), CHFp_FRA(kslope), CHFp_CONST_FRA(state), CHFp_BOX(bcoarse), CHFp_BOX(bn), CHFp_BOX(physdomain))

8.85.2.5 void FORT_INTERPLINEAR (CHFp_FRA(fine), CHFp_-CONST_FRA(slope), CHFp_BOX(b), CHFp_CONST_INT(dir), CHFp_CONST_INT(ref_ratio), CHFp_BOX(bref))

8.85.2.6 void FORT_INTERPLOSIDESLOPE (CHFp_FRA(slope), CHFp_CONST_FRA(state), CHFp_BOX(b), CHFp_CONST_INT(dir))

8.86 Interval.H File Reference

This graph shows which files directly or indirectly include this file:



Compounds

- struct **Interval**

    Structure for passing component ranges in code.

8.87 IntVect.H File Reference

#include <cstddef>

#include <cstdlib>

#include <cstring>

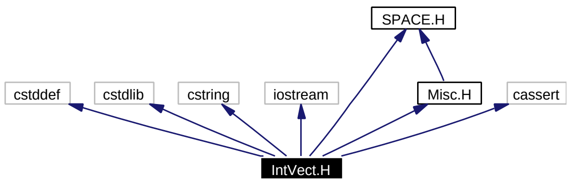
#include <iostream>

#include "Misc.H"

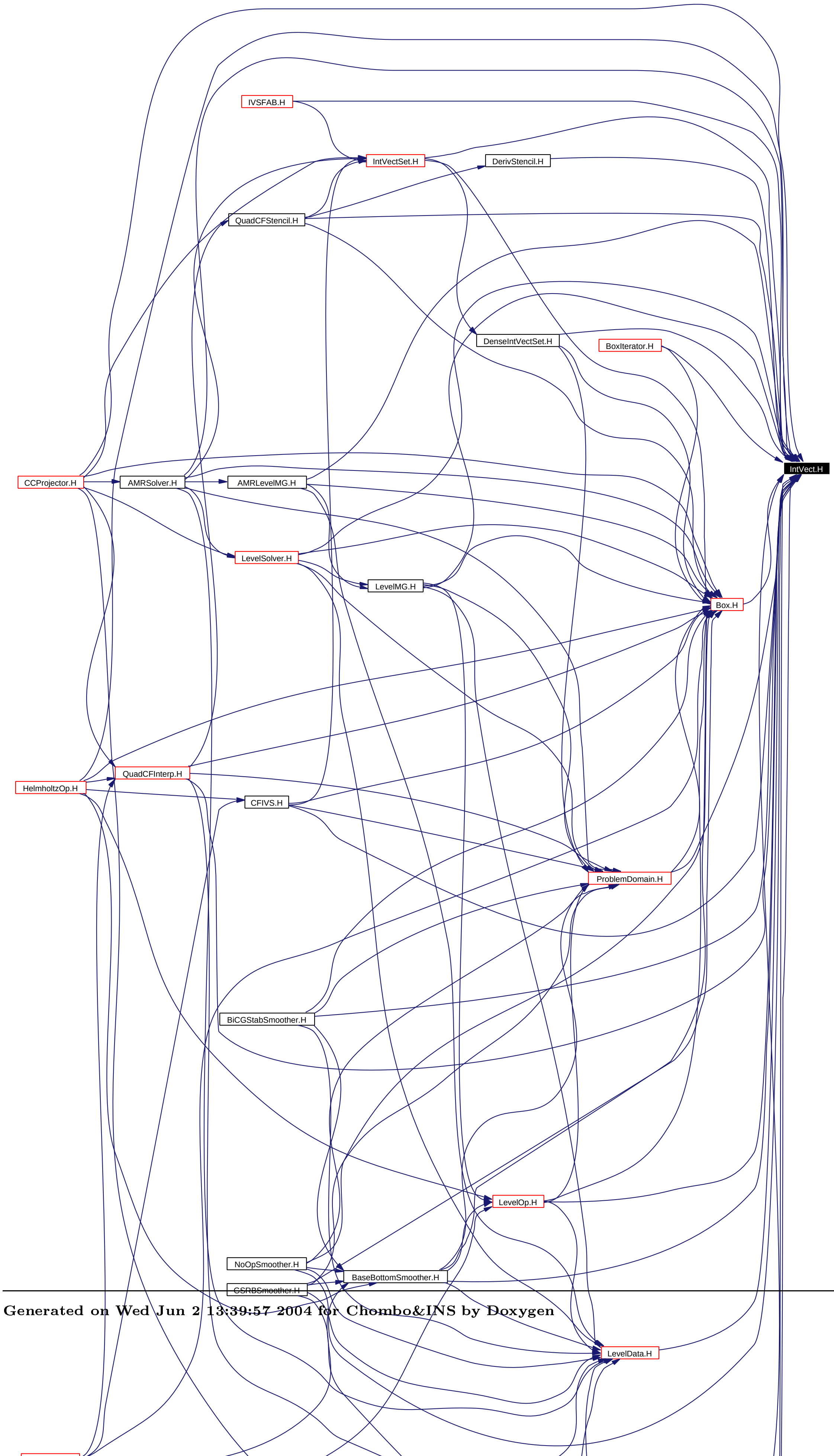
#include <cassert>

#include "SPACE.H"

Include dependency graph for IntVect.H:



This graph shows which files directly or indirectly include this file:


IVSFAB.H
IntVectSet.H
DerivStencil.H
QuadCFStencil.H
DenseIntVectSet.H
BoxIterator.H
IntVect.H
CCProjector.H
AMRSolver.H
AMRLevelMG.H
LevelSolver.H
LevelMG.H
Box.H
QuadCFInterp.H
HelmholtzOp.H
CFIVS.H
ProblemDomain.H
BiCGStabSmoother.H
LevelOp.H
NoOpSmoother.H
BaseBottomSmoother.H
GSRBSmoother.H
LevelData.H

Compounds

- class **IntVect**

  An integer **Vector** *in SpaceDim-dimensional space.*

Defines

- #define **LLT0** (vect[0] < s[0])
- #define **LLT1** ((vect[0] == s[0]) && (vect[1] < s[1]))
- #define **LLT2** ((vect[1] == s[1]) && (vect[2] < s[2]))
- #define **LGT0** (vect[0] > s[0])
- #define **LGT1** ((vect[0] == s[0]) && (vect[1] > s[1]))
- #define **LGT2** ((vect[1] == s[1]) && (vect[2] > s[2]))

Functions

- **IntVect UnitVector** ()
- **IntVect ZeroVector** ()
- **IntVect operator+** (int s, const **IntVect** &p)
- **IntVect operator-** (int s, const **IntVect** &p)
- **IntVect operator** $*$ (int s, const **IntVect** &p)
- **IntVect scale** (const **IntVect** &p, int s)
- **IntVect diagShift** (const **IntVect** &p, int s)
- **IntVect min** (const **IntVect** &p1, const **IntVect** &p2)
- **IntVect max** (const **IntVect** &p1, const **IntVect** &p2)

8.87.1 Define Documentation

8.87.1.1 #define LGT0 (vect[0] > s[0])

8.87.1.2 #define LGT1 ((vect[0] == s[0]) && (vect[1] > s[1]))

8.87.1.3 #define LGT2 ((vect[1] == s[1]) && (vect[2] > s[2]))

8.87.1.4 #define LLT0 (vect[0] < s[0])

8.87.1.5 #define LLT1 ((vect[0] == s[0]) && (vect[1] < s[1]))

8.87.1.6 #define LLT2 ((vect[1] == s[1]) && (vect[2] < s[2]))

8.87.2 Function Documentation

8.87.2.1 IntVect diagShift (const IntVect & *p*, int *s*) `[inline]`

Returns **IntVect** obtained by adding a scalar to each of the components of the given **IntVect**.

8.87.2.2 IntVect max (const IntVect & *p1*, const IntVect & *p2*) `[inline]`

Returns the **IntVect** that is the component-wise maximum of two argument IntVects.

8.87.2.3 IntVect min (const IntVect & *p1*, const IntVect & *p2*) [inline]

Returns the **IntVect** that is the component-wise minimum of two argument IntVects.

8.87.2.4 IntVect operator * (int *s*, const IntVect & *p*) [inline]

Returns an **IntVect** that is an **IntVect** p with each component multiplied by a scalar s.

8.87.2.5 IntVect operator+ (int *s*, const IntVect & *p*) [inline]

Returns an **IntVect** that is an **IntVect** p with a scalar s added to each component.

8.87.2.6 IntVect operator- (int *s*, const IntVect & *p*) [inline]

Returns s - p.

8.87.2.7 IntVect scale (const IntVect & *p*, int *s*) [inline]

Returns an **IntVect** obtained by multiplying each of the components of the given **IntVect** by a scalar.

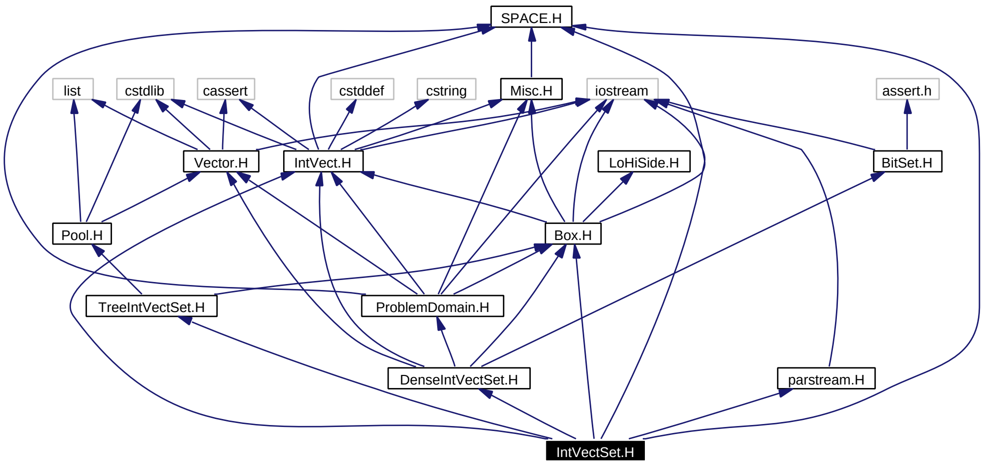
8.87.2.8 IntVect UnitVector ()

8.87.2.9 IntVect ZeroVector ()

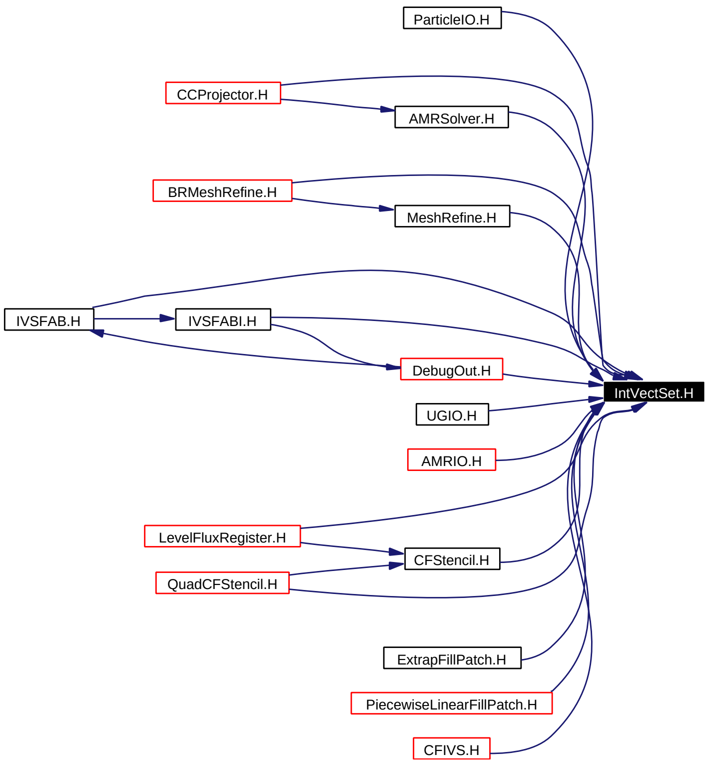
8.88 IntVectSet.H File Reference

```
#include "SPACE.H"
#include <iostream>
#include "Box.H"
#include "IntVect.H"
#include "TreeIntVectSet.H"
#include "DenseIntVectSet.H"
#include "parstream.H"
```

Include dependency graph for IntVectSet.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **IntVectSet**

  An irregular domain on an integer lattice.

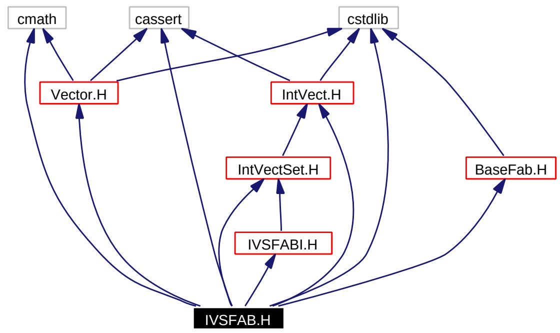
- class **IVSIterator**

  Iterator for an **IntVectSet**.

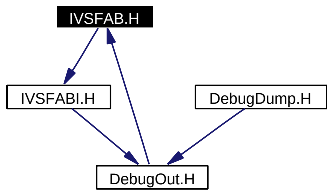
8.89 IVSFAB.H File Reference

```
#include <cmath>
#include <cstdlib>
#include <cassert>
#include "Vector.H"
#include "IntVectSet.H"
#include "IntVect.H"
#include "BaseFab.H"
#include "IVSFABI.H"
```

Include dependency graph for IVSFAB.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **IVSFAB**

8.90 IVSFABI.H File Reference

`#include "MayDay.H"`

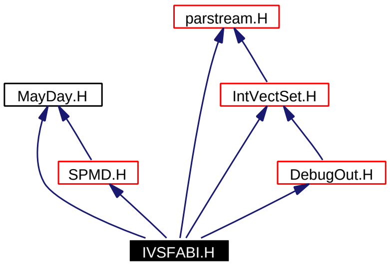
`#include "IntVectSet.H"`

`#include "parstream.H"`

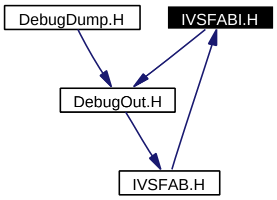
`#include "SPMD.H"`

`#include "DebugOut.H"`

Include dependency graph for IVSFABI.H:



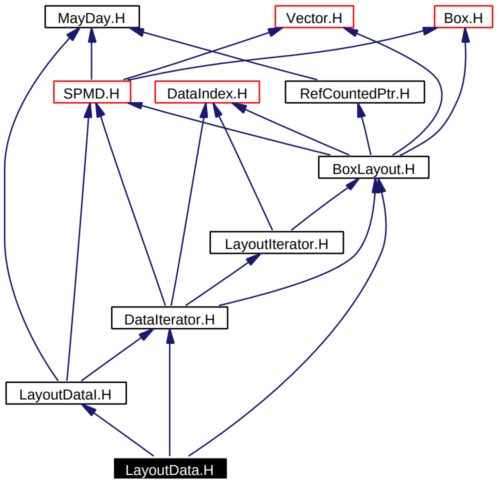
This graph shows which files directly or indirectly include this file:



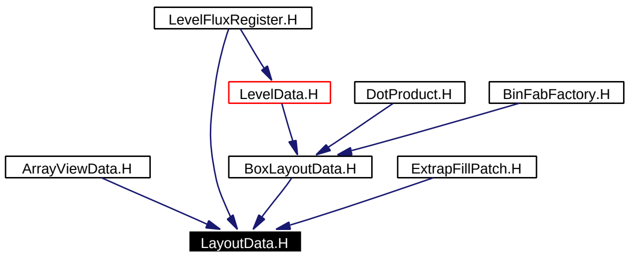
8.91 LayoutData.H File Reference

```
#include "BoxLayout.H"
#include "DataIterator.H"
#include "LayoutDataI.H"
```

Include dependency graph for LayoutData.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **LayoutData**

    Data that maintains a one-to-one mapping to the boxes in a **BoxLayout**.

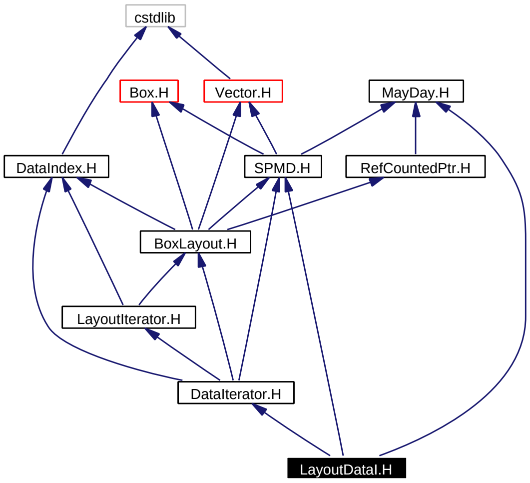
8.92 LayoutDataI.H File Reference

`#include "MayDay.H"`

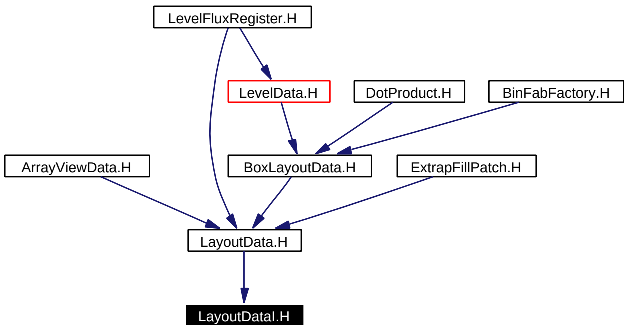
`#include "DataIterator.H"`

`#include "SPMD.H"`

Include dependency graph for LayoutDataI.H:



This graph shows which files directly or indirectly include this file:

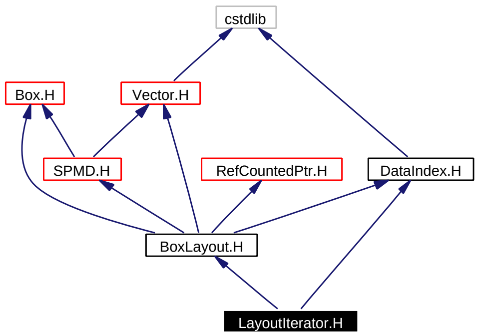


8.93 LayoutIterator.H File Reference

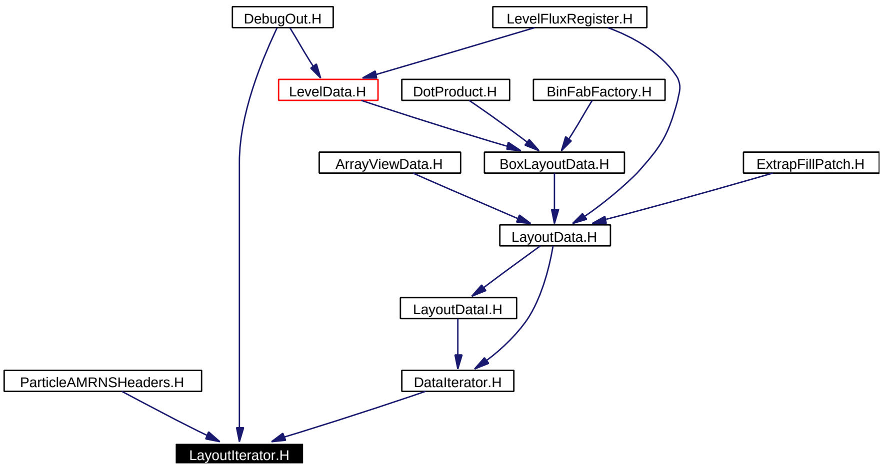
`#include "DataIndex.H"`

`#include "BoxLayout.H"`

Include dependency graph for LayoutIterator.H:



This graph shows which files directly or indirectly include this file:



Compounds

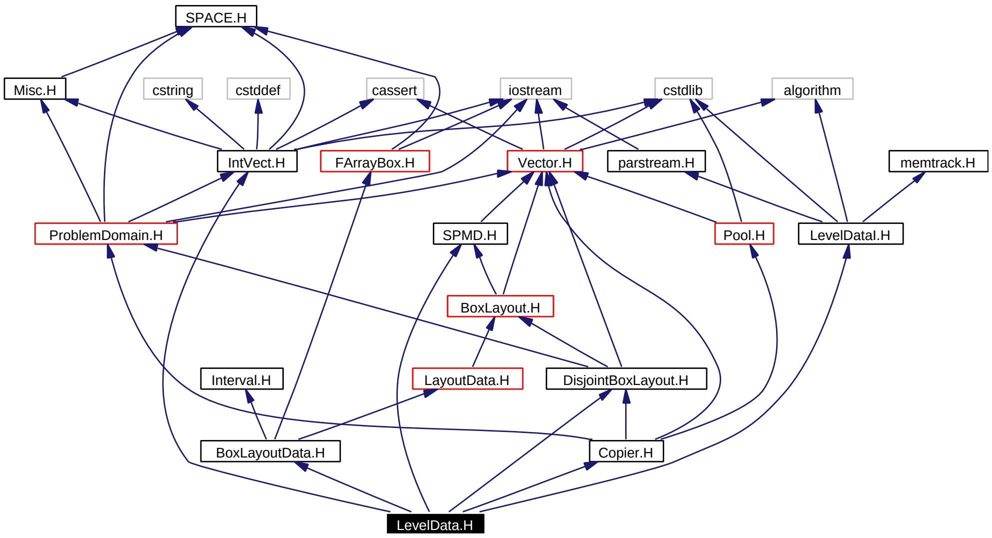
- class **LayoutIterator**

  An Iterator based on a **BoxLayout** *object.*

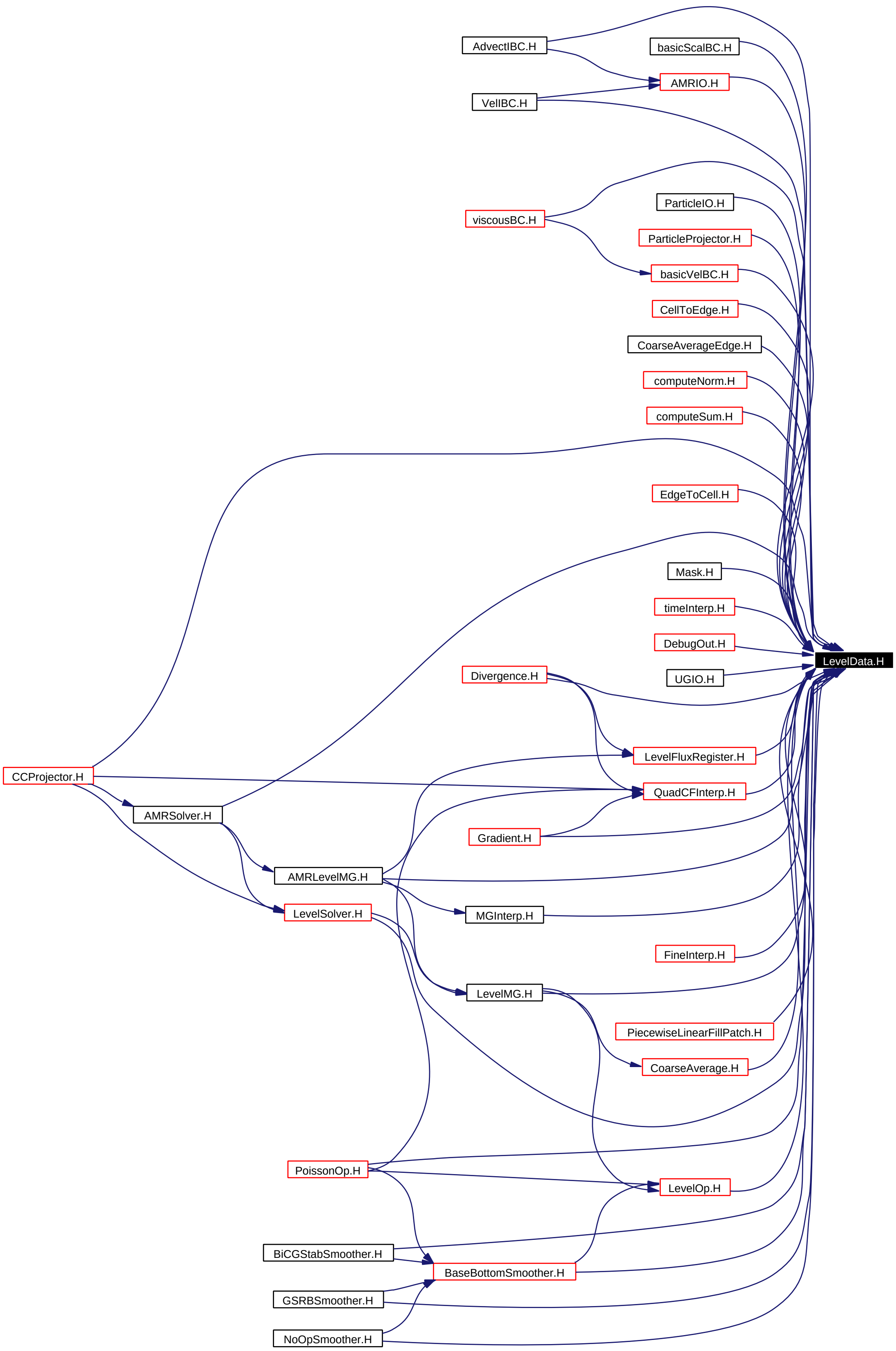
8.94 LevelData.H File Reference

```
#include "IntVect.H"
#include "BoxLayoutData.H"
#include "DisjointBoxLayout.H"
#include "Copier.H"
#include "SPMD.H"
#include "LevelDataI.H"
```

Include dependency graph for LevelData.H:



This graph shows which files directly or indirectly include this file:


AdvectIBC.H
basicScalBC.H
AMRIO.H
VelIBC.H
viscousBC.H
ParticleIO.H
ParticleProjector.H
basicVelBC.H
CellToEdge.H
CoarseAverageEdge.H
computeNorm.H
computeSum.H
EdgeToCell.H
Mask.H
timeInterp.H
DebugOut.H
LevelData.H
Divergence.H
UGIO.H
LevelFluxRegister.H
CCProjector.H
QuadCFInterp.H
AMRSolver.H
Gradient.H
AMRLevelMG.H
LevelSolver.H
MGInterp.H
FineInterp.H
LevelMG.H
PiecewiseLinearFillPatch.H
CoarseAverage.H
PoissonOp.H
LevelOp.H
BiCGStabSmoother.H
BaseBottomSmoother.H
GSRBSmoother.H
NoOpSmoother.H

Compounds

- class **LDOperator**
- class **LevelData**

  Data over a disjoint union of rectangles.

Functions

- template<class T> void **aliasLevelData** (**LevelData**< T > &a_alias, **LevelData**< T > *a_original, const **Interval** &a_interval)

  LevelData *aliasing function.*

8.94.1 Function Documentation

8.94.1.1 template<class T> void aliasLevelData (LevelData< T > & *a_alias*, LevelData< T > * *a_original*, const Interval & *a_interval*)

LevelData aliasing function.

Parameters:

a_alias aliased LevelData<T> object. original data in a_alias is destroyed and new aliasing role assumed.

a_original pointer to LevelData<T> that will be aliased.

a_interval range of components of each T in a_original that will be created in the a_alias argument.

```
LevelData<FArrayBox> original(dbl, 4, 2*IntVect::Unit);
Interval             subcomps(2, 3);
LevelData<FArrayBox> alias;
aliasLevelData<FArrayBox>(alias, &original, subcomps);
// component 0 of every FArrayBox in alias references the same data as
// component 2 in original
```

The template class T must have an 'alias' constructor

```
class A
 {
public:
  A(const Interval& subcomps, A& original);// alias constructor
  ...
 };
```

8.95 LevelDataI.H File Reference

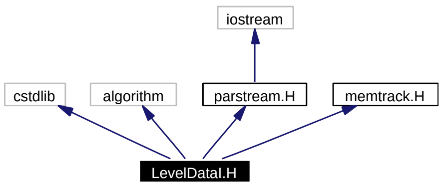
`#include <cstdlib>`

`#include <algorithm>`

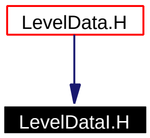
`#include "parstream.H"`

`#include "memtrack.H"`

Include dependency graph for LevelDataI.H:



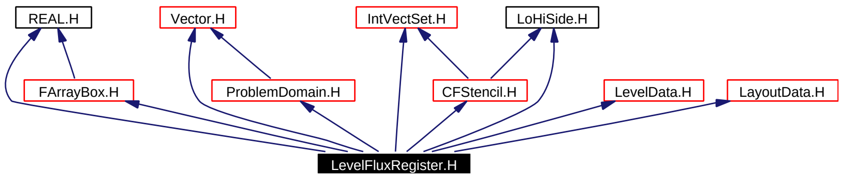
This graph shows which files directly or indirectly include this file:



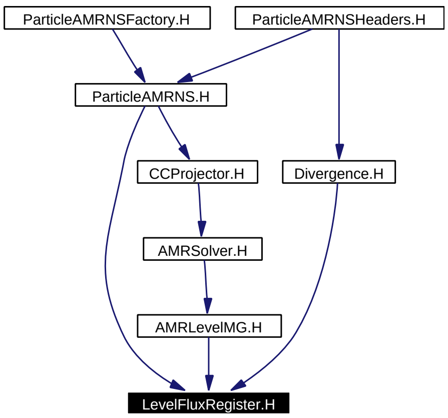
8.96 LevelFluxRegister.H File Reference

```
#include "REAL.H"
#include "Vector.H"
#include "FArrayBox.H"
#include "IntVectSet.H"
#include "CFStencil.H"
#include "LoHiSide.H"
#include "LevelData.H"
#include "LayoutData.H"
#include "ProblemDomain.H"
```

Include dependency graph for LevelFluxRegister.H:



This graph shows which files directly or indirectly include this file:



Compounds

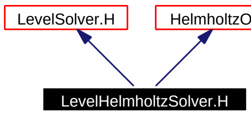
- class **LevelFluxRegister**

  LevelFluxRegister-A class to encapsulate a levels worth of flux registers.

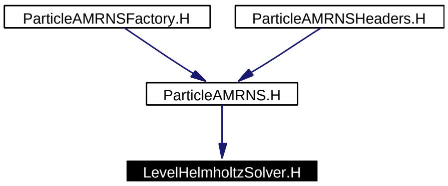
8.97 LevelHelmholtzSolver.H File Reference

```
#include <LevelSolver.H>
#include <HelmholtzOp.H>
```

Include dependency graph for LevelHelmholtzSolver.H:



This graph shows which files directly or indirectly include this file:



Compounds

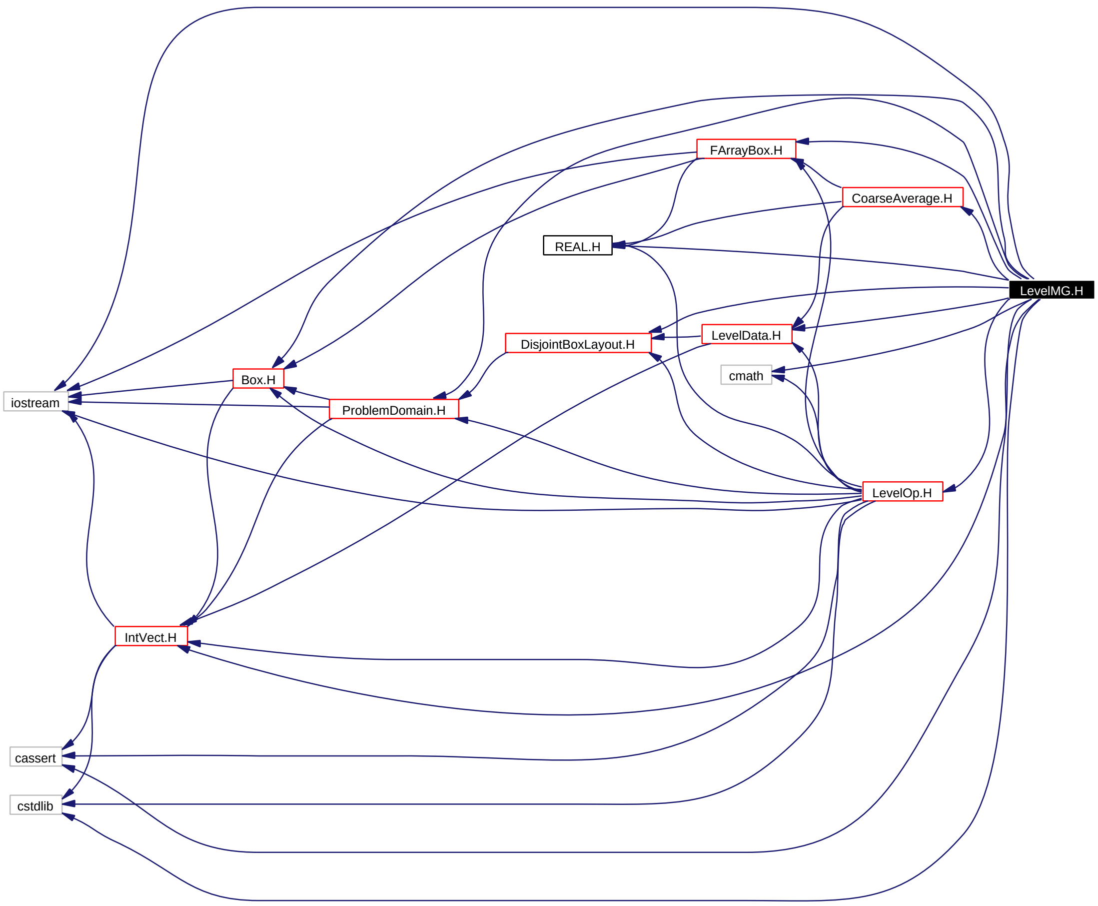
- class **LevelHelmholtzSolver**

  special case **LevelSolver** *to solve Helmholtz eqn*

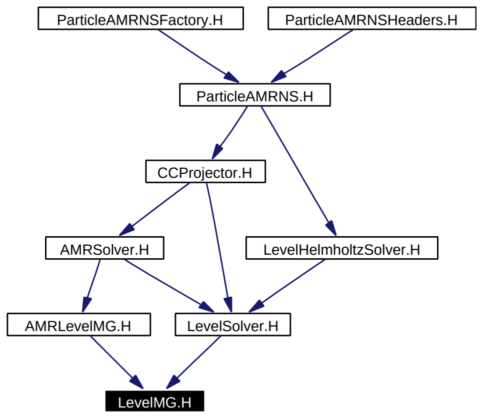
8.98 LevelMG.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "DisjointBoxLayout.H"
#include "FArrayBox.H"
#include "LevelData.H"
#include "CoarseAverage.H"
#include "LevelOp.H"
#include "ProblemDomain.H"
```

Include dependency graph for LevelMG.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **LevelMG**

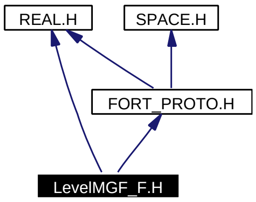
  Internal Chombo class to manage multigrid solve on a level.

8.99 LevelMGF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for LevelMGF_F.H:



Defines

- #define **FORT_INTERPMG** FORTRAN_NAME(INTERPMG ,interpmg)

Functions

- void **FORT_INTERPMG** (CHFp_FRA(fine),CHFp_CONST_FRA(crse),CHFp_BOX(region),CHFp_CONST_INT(nref),CHFp_BOX(bref))

8.99.1 Define Documentation

8.99.1.1 #define FORT_INTERPMG FORTRAN_NAME(INTERPMG ,interpmg)

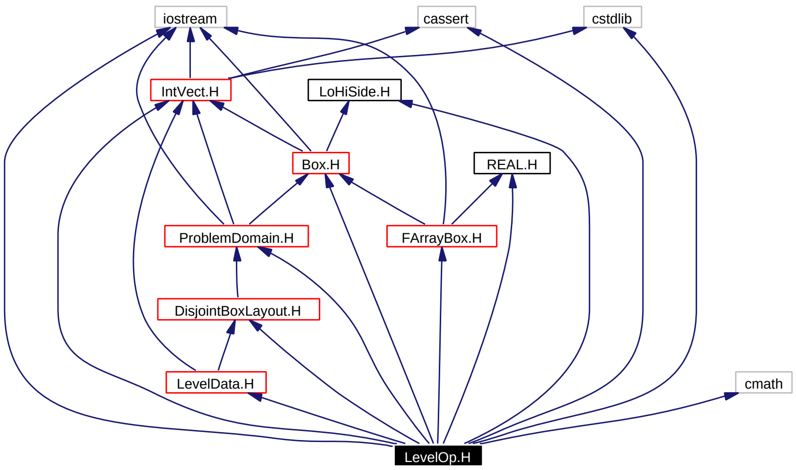
8.99.2 Function Documentation

8.99.2.1 void FORT_INTERPMG (CHFp_FRA(fine), CHFp_CONST_FRA(crse), CHFp_BOX(region), CHFp_CONST_INT(nref), CHFp_BOX(bref))

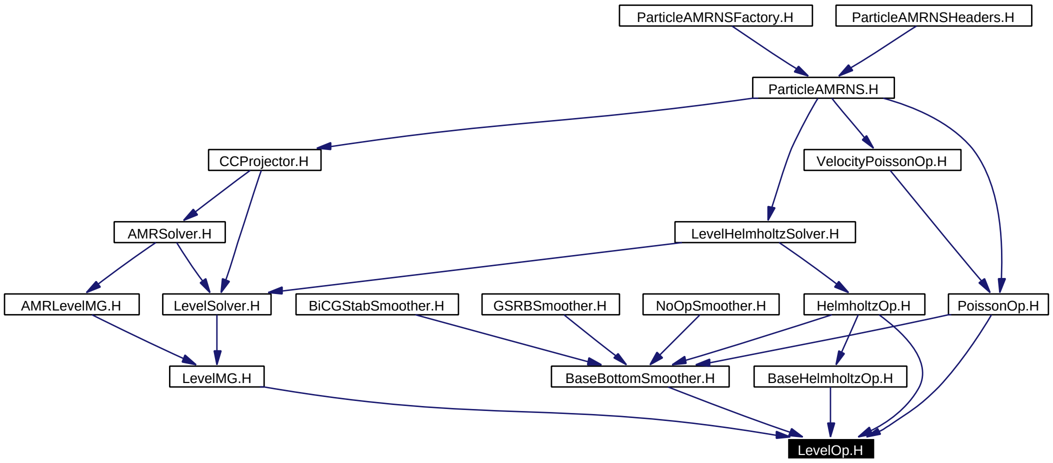
8.100 LevelOp.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "FArrayBox.H"
#include "LoHiSide.H"
#include "ProblemDomain.H"
```

Include dependency graph for LevelOp.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **LevelOp**

    Pure base class to encapsulate level operations API for elliptic solvers.

8.101 LevelSolver.H File Reference

`#include <cstdlib>`

`#include <iostream>`

`#include <cassert>`

`#include <cmath>`

`#include "REAL.H"`

`#include "IntVect.H"`

`#include "Box.H"`

`#include "FArrayBox.H"`

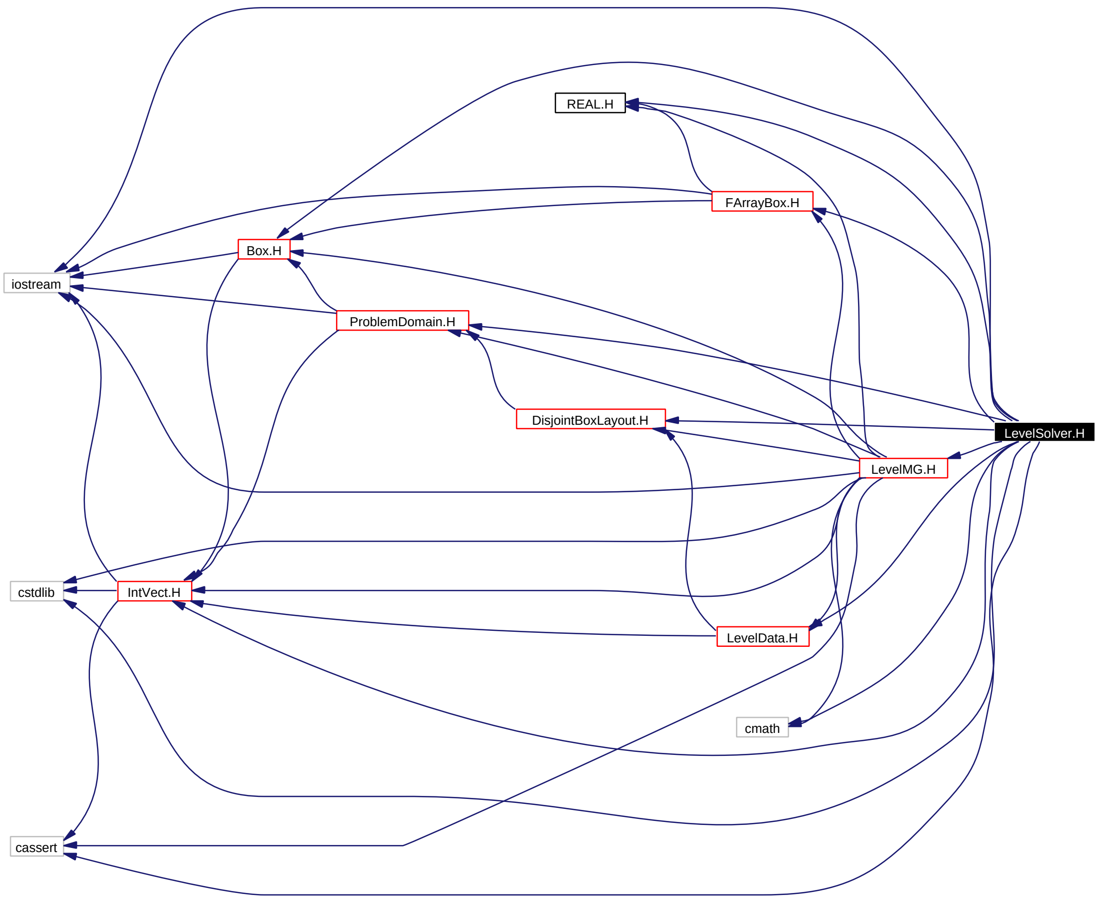
`#include "DisjointBoxLayout.H"`

`#include "LevelData.H"`

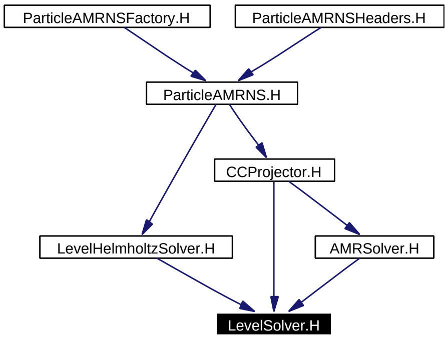
`#include "LevelMG.H"`

`#include "ProblemDomain.H"`

Include dependency graph for LevelSolver.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **LevelSolver**

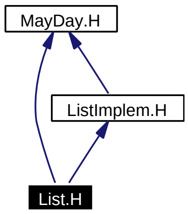
  Solves elliptic equations on a level.

8.102 List.H File Reference

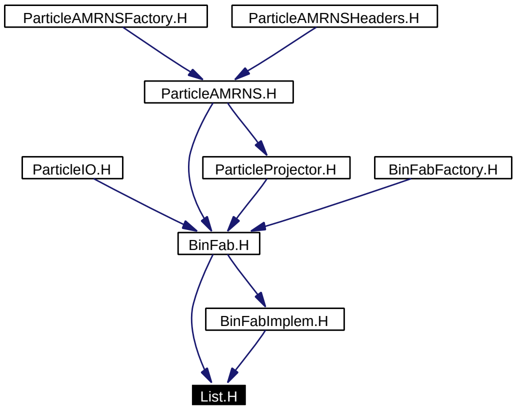
`#include "MayDay.H"`

`#include "ListImplem.H"`

Include dependency graph for List.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **List**

  A Doubly-Linked List Class.

- class **ListIterator**

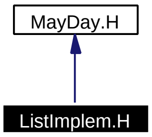
  Iterator over a **List**.

- class **ListLink**

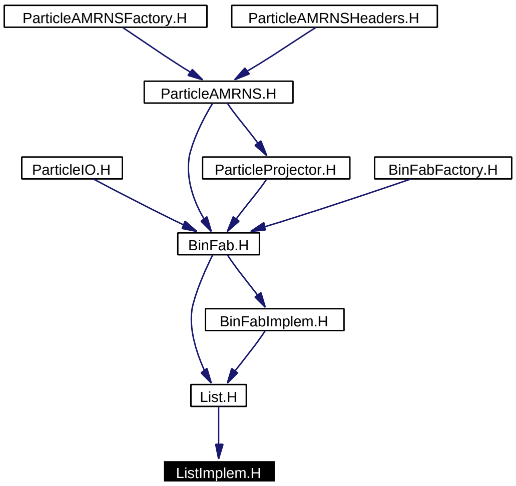
8.103 ListImplem.H File Reference

`#include "MayDay.H"`

Include dependency graph for ListImplem.H:



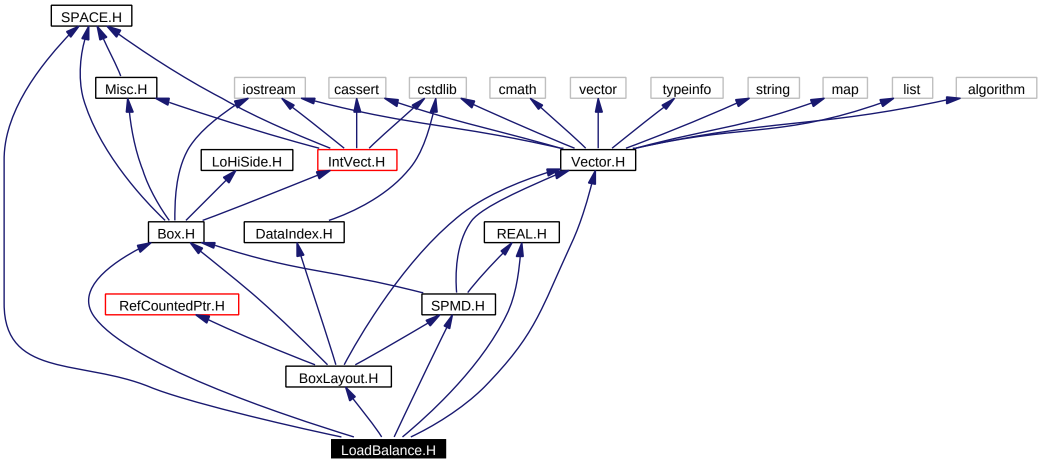
This graph shows which files directly or indirectly include this file:



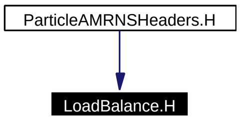
8.104 LoadBalance.H File Reference

```
#include "SPACE.H"
#include "REAL.H"
#include "Box.H"
#include "Vector.H"
#include "BoxLayout.H"
#include "SPMD.H"
```

Include dependency graph for LoadBalance.H:



This graph shows which files directly or indirectly include this file:



Functions

- int **LoadBalance** (**Vector**< **Vector**< int > > &a_procAssignments,**Real** &a_effRatio,const **Vector**< **Vector**< **Box** > > &a_Grids,const **Vector**< **Vector**< long > > &a_ComputeLoads,const **Vector**< int > &a_RefRatios,int a_nProc=numProc())
- int **LoadBalance** (**Vector**< **BoxLayout** > &Grids,**Real** &effRatio,const **Vector**< **Vector**< long > > &ComputeLoads,const **Vector**< int > &RefRatios,int nProc=numProc())
- int **LoadBalance** (**Vector**< int > &procAssignments, const **Vector**< **Box** > &boxes, int a_numProc=numProc())
- int **LoadBalance** (**Vector**< int > &procAssignments,**Real** &effRatio,const **Vector**< **Box** > &Grids,const **Vector**< long > &ComputeLoads)

8.104.1 Function Documentation

8.104.1.1 int LoadBalance (Vector< int > & *procAssignments*, Real & *effRatio*, const Vector< Box > & *Grids*, const Vector< long > & *ComputeLoads*)

8.104.1.2 int LoadBalance (Vector< int > & *procAssignments*, const Vector< Box > & *boxes*, int *a_numProc* = numProc())

convenience function to load balance a **Vector** of Boxes based on load=box.numPts()

8.104.1.3 int LoadBalance (Vector< BoxLayout > & *Grids*, Real & *effRatio*, const Vector< Vector< long > > & *ComputeLoads*, const Vector< int > & *RefRatios*, int *nProc* = numProc())

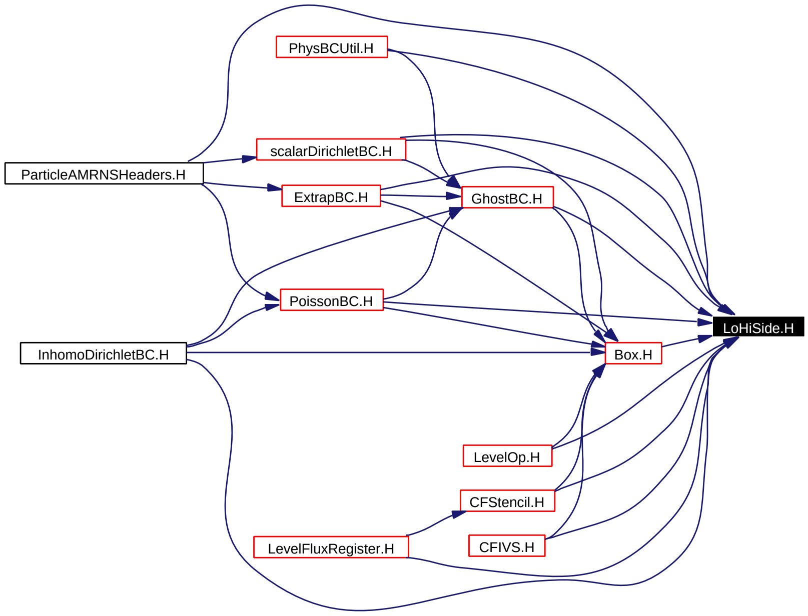
Grids in-out: input grids to balance and output proc. numbers effRatio output: ratio of min load to max load ComputeLoads input: computational cost of each box RefRatios input: refinement ratio for each level

8.104.1.4 int LoadBalance (Vector< Vector< int > > & *a_procAssignments*, Real & *a_effRatio*, const Vector< Vector< Box > > & *a_Grids*, const Vector< Vector< long > > & *a_ComputeLoads*, const Vector< int > & *a_RefRatios*, int *a_nProc* = numProc())

procAssignments output: processor number for each box effRatio output: ratio of min load to max load Grids input: meshes to balance ComputeLoads input: computational cost of each box RefRatios input: refinement ratio for each level

8.105 LoHiSide.H File Reference

This graph shows which files directly or indirectly include this file:



Compounds

- class **Side**

  Encapsulation of high or low side.

- class **SideIterator**

  Iterator for low and high side.

8.106 Mask.H File Reference

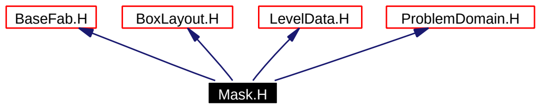
`#include "BaseFab.H"`

`#include "BoxLayout.H"`

`#include "LevelData.H"`

`#include "ProblemDomain.H"`

Include dependency graph for Mask.H:



Compounds

- class **Mask**

8.107 MASKVAL.H File Reference

Defines

- #define **MASKCOVERED** -2
- #define **MASKPHYS** -1
- #define **MASKCOPY** 0
- #define **MASKCOARSE** 1

8.107.1 Define Documentation

8.107.1.1 #define MASKCOARSE 1

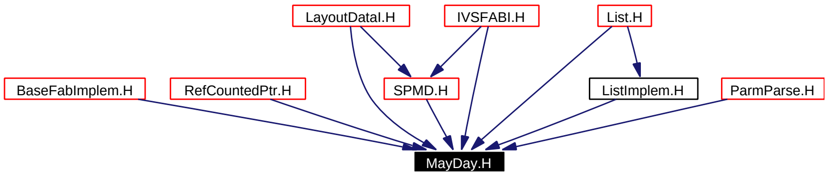
8.107.1.2 #define MASKCOPY 0

8.107.1.3 #define MASKCOVERED -2

8.107.1.4 #define MASKPHYS -1

8.108 MayDay.H File Reference

This graph shows which files directly or indirectly include this file:



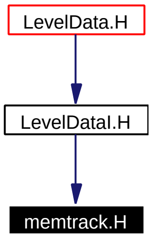
Compounds

- class **MayDay**

  Error-reporting Functions.

8.109 memtrack.H File Reference

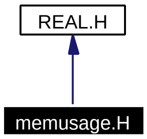
This graph shows which files directly or indirectly include this file:



8.110 memusage.H File Reference

`#include "REAL.H"`

Include dependency graph for memusage.H:



Functions

- **Real get_memory_usage** (void)
- void **gather_memory_from_procs** (**Real** end_memory, **Real** &avg_memory, **Real** &min_-memory, **Real** &max_memory)

8.110.1 Function Documentation

8.110.1.1 void gather_memory_from_procs (Real *end_memory*, Real & *avg_memory*, Real & *min_memory*, Real & *max_memory*)

8.110.1.2 Real get_memory_usage (void)

8.111 MeshRefine.H File Reference

`#include <climits>`

`#include "Vector.H"`

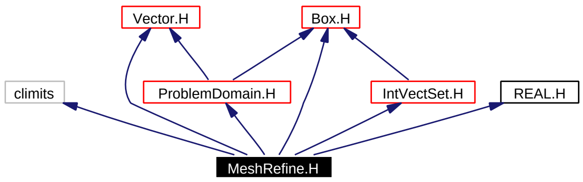
`#include "Box.H"`

`#include "IntVectSet.H"`

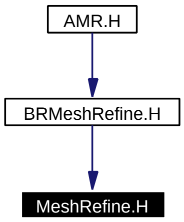
`#include "REAL.H"`

`#include "ProblemDomain.H"`

Include dependency graph for MeshRefine.H:



This graph shows which files directly or indirectly include this file:



Compounds

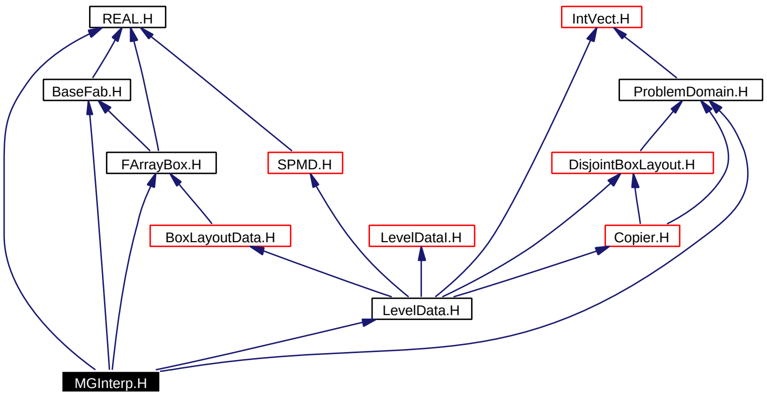
- class **MeshRefine**

    Class which manages grid generation.

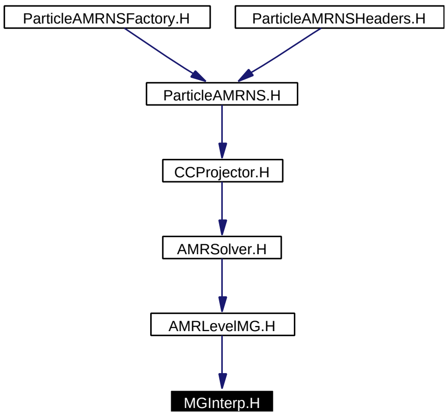
8.112 MGInterp.H File Reference

```
#include "REAL.H"
#include "LevelData.H"
#include "BaseFab.H"
#include "FArrayBox.H"
#include "ProblemDomain.H"
```

Include dependency graph for MGInterp.H:



This graph shows which files directly or indirectly include this file:



Compounds

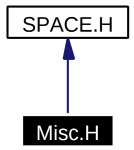
- class **MGInterp**

    Internal Chombo class to interpolate coarse level data to a fine level.

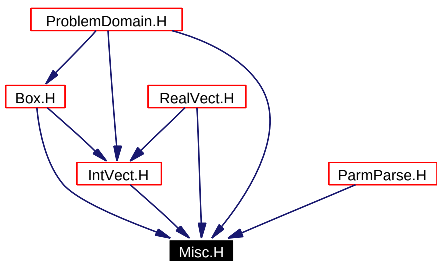
8.113 Misc.H File Reference

`#include "SPACE.H"`

Include dependency graph for Misc.H:



This graph shows which files directly or indirectly include this file:



Functions

- template<class T> T **Min** (const T &a, const T &b)
- template<class T> T **Max** (const T &a, const T &b)
- template<class T> T **Abs** (const T &a)
- template<class T> void **Swap** (T &a, T &b)

8.113.1 Function Documentation

8.113.1.1 template<class T> T Abs (const T & *a*) `[inline]`

Returns a copy of the absolute value of the value. Will work with any type T that can be compared against zero with T::operator>(const T, const T), or an equivalent construct, as well as a copy constructor, and an **T::operator-()** or equivalent. In particular, it will work with any integral or floating-point intrinsic type.

8.113.1.2 template<class T> T Max (const T & *a*, const T & *b*) `[inline]`

Returns a copy of the maximum of the two values. Will work with any type T that supplies a T::operator>(const T, const T), or an equivalent construct, as well as a copy constructor. In particular, it will work with any integral or floating-point intrinsic type.

8.113.1.3 template<class T> T Min (const T & *a*, const T & *b*) [inline]

Returns a copy of the minimum of the two values. Will work with any type T that supplies a T::operator<(const T, const T), or an equivalent construct, as well as a copy constructor. In particular, it will work with any integral or floating-point intrinsic type.

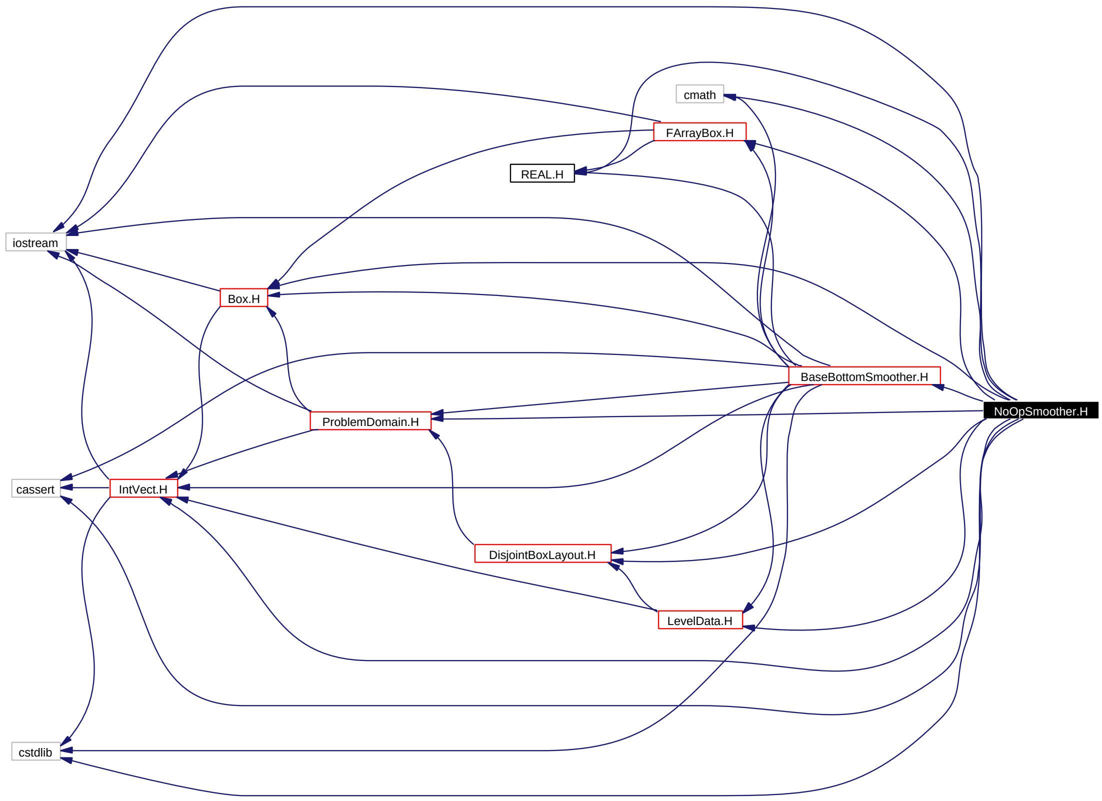
8.113.1.4 template<class T> void Swap (T & *a*, T & *b*) [inline]

Swaps the two values. Type T must have a copy constructor and an assignment operator.

8.114 NoOpSmoother.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "FArrayBox.H"
#include "ProblemDomain.H"
#include "BaseBottomSmoother.H"
```

Include dependency graph for NoOpSmoother.H:



Compounds

- class **NoOpSmoother**

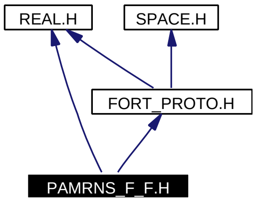
Does nothing as a bottom smoother.

8.115 PAMRNS_F_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for PAMRNS_F_F.H:



Defines

- #define **FORT_COMPUTEVORT** FORTRAN_NAME(COMPUTEVORT ,computevort)
- #define **FORT_KINETICENERGY** FORTRAN_NAME(KINETICENERGY ,kineticenergy)
- #define **FORT_MAGVECT** FORTRAN_NAME(MAGVECT ,magvect)
- #define **FORT_CENTERED_DERIV** FORTRAN_NAME(CENTERED_DERIV ,centered_deriv)

Functions

- void **FORT_COMPUTEVORT** (CHFp_FRA1(vort),CHFp_FRA(vel),CHFp_BOX(gridBox),CHFp_CONST_REAL(dx),CHFp_CONST_INT(dir))
- void **FORT_KINETICENERGY** (CHFp_FRA1(energy),CHFp_CONST_FRA(vel),CHFp_BOX(gridBox))
- void **FORT_MAGVECT** (CHFp_FRA1(magVector),CHFp_FRA(vect),CHFp_BOX(gridBox))
- void **FORT_CENTERED_DERIV** (CHFp_FRA1(deriv),CHFp_CONST_FRA1(phi),CHFp_BOX(grid),CHFp_CONST_REAL(dx),CHFp_INT(dir))

8.115.1 Define Documentation

8.115.1.1 #define FORT_CENTERED_DERIV FORTRAN_NAME(CENTERED_DERIV ,centered_deriv)

8.115.1.2 #define FORT_COMPUTEVORT FORTRAN_NAME(COMPUTEVORT ,computevort)

8.115.1.3 #define FORT_KINETICENERGY FORTRAN_NAME(KINETICENERGY ,kineticenergy)

8.115.1.4 #define FORT_MAGVECT FORTRAN_NAME(MAGVECT ,magvect)

8.115.2 Function Documentation

8.115.2.1 void FORT_CENTERED_DERIV (CHFp_FRA1(deriv), CHFp_CONST_FRA1(phi), CHFp_BOX(grid), CHFp_CONST_REAL(dx), CHFp_INT(dir))

8.115.2.2 void FORT_COMPUTEVORT (CHFp_FRA1(vort), CHFp_FRA(vel), CHFp_BOX(gridBox), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir))

8.115.2.3 void FORT_KINETICENERGY (CHFp_FRA1(energy), CHFp_CONST_FRA(vel), CHFp_BOX(gridBox))

8.115.2.4 void FORT_MAGVECT (CHFp_FRA1(magVector), CHFp_FRA(vect), CHFp_BOX(gridBox))

8.116 ParmParse.H File Reference

`#include <iostream>`

`#include <iomanip>`

`#include <cstdlib>`

`#include <string>`

`#include <vector>`

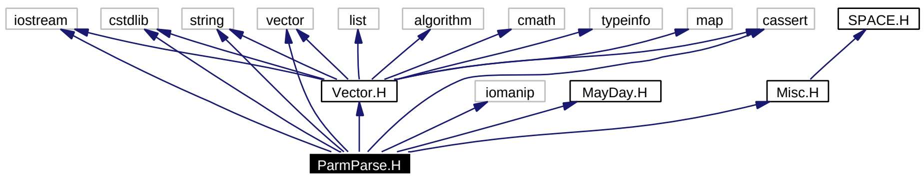
`#include <cassert>`

`#include "MayDay.H"`

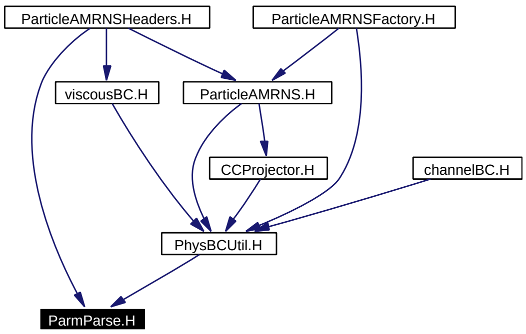
`#include "Misc.H"`

`#include "Vector.H"`

Include dependency graph for ParmParse.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **ParmParse**

  Parse Parameters From Command Line and Input Files.

- class **PP_Array**

  An Array of Objects of Type T for **ParmParse**.

- class **PP_CpClassPtr**

  A Smart Pointer for User-Defined Types for **ParmParse**.

- class **PP_CpPtr**

  : A Smart Pointer for Intrinsic or User-Defined Types for **ParmParse**

- class **PP_entry**
- class **PP_List**

  A Doubly-Linked **List** *for* **ParmParse**.

- class **PP_ListIterator**

  Iterate over a **List** *for* **ParmParse**.

- class **PP_ListLink**
- class **PP_LnClassPtr**

  A Smart Reference Counted Pointer for User-Defined Types for **ParmParse**.

- class **PP_LnPtr**

  A Reference Counted Smart Pointer for Intrinsic or User-Defined Types for **ParmParse**.

- class **PP_String**

  A String Class for **ParmParse**.

- class **PP_StringRep**
- class **PP_UseCount**

Functions

- **PP_String operator+** (const **PP_String** &left, const **PP_String** &right)
- bool **operator<** (const **PP_String** &left, const **PP_String** &right)
- bool **operator<=** (const **PP_String** &left, const **PP_String** &right)
- bool **operator!=** (const **PP_String** &left, const **PP_String** &right)
- bool **operator==** (const **PP_String** &left, const **PP_String** &right)
- bool **operator>=** (const **PP_String** &left, const **PP_String** &right)
- bool **operator>** (const **PP_String** &left, const **PP_String** &right)
- bool **operator<** (const **PP_String** &left, const char *right)
- bool **operator<=** (const **PP_String** &left, const char *right)
- bool **operator!=** (const **PP_String** &left, const char *right)
- bool **operator==** (const **PP_String** &left, const char *right)
- bool **operator>=** (const **PP_String** &left, const char *right)
- bool **operator>** (const **PP_String** &left, const char *right)
- bool **operator<** (const char *left, const **PP_String** &right)
- bool **operator<=** (const char *left, const **PP_String** &right)
- bool **operator!=** (const char *left, const **PP_String** &right)
- bool **operator==** (const char *left, const **PP_String** &right)
- bool **operator>=** (const char *left, const **PP_String** &right)
- bool **operator>** (const char *left, const **PP_String** &right)

8.116.1 Function Documentation

8.116.1.1 bool operator!= (const char * *left*, const PP_String & *right*) `[inline]`

8.116.1.2 bool operator!= (const PP_String & *left*, const char * *right*) `[inline]`

8.116.1.3 bool operator!= (const PP_String & *left*, const PP_String & *right*) `[inline]`

8.116.1.4 PP_String operator+ (const PP_String & *left*, const PP_String & *right*) `[inline]`

8.116.1.5 bool operator< (const char * *left*, const PP_String & *right*) `[inline]`

8.116.1.6 bool operator< (const PP_String & *left*, const char * *right*) `[inline]`

8.116.1.7 bool operator< (const PP_String & *left*, const PP_String & *right*) `[inline]`

8.116.1.8 bool operator<= (const char * *left*, const PP_String & *right*) `[inline]`

8.116.1.9 bool operator<= (const PP_String & *left*, const char * *right*) `[inline]`

8.116.1.10 bool operator<= (const PP_String & *left*, const PP_String & *right*) `[inline]`

8.116.1.11 bool operator== (const char * *left*, const PP_String & *right*) `[inline]`

8.116.1.12 bool operator== (const PP_String & *left*, const char * *right*) `[inline]`

8.116.1.13 bool operator== (const PP_String & *left*, const PP_String & *right*) `[inline]`

8.116.1.14 bool operator> (const char * *left*, const PP_String & *right*) `[inline]`

8.116.1.15 bool operator> (const PP_String & *left*, const char * *right*) `[inline]`

8.116.1.16 bool operator> (const PP_String & *left*, const PP_String & *right*) `[inline]`

8.116.1.17 bool operator>= (const char * *left*, const PP_String & *right*) `[inline]`

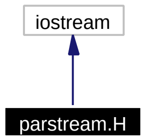
8.116.1.18 bool operator>= (const PP_String & *left*, const char * *right*) `[inline]`

8.116.1.19 bool operator>= (const PP_String & *left*, const PP_String & *right*) `[inline]`

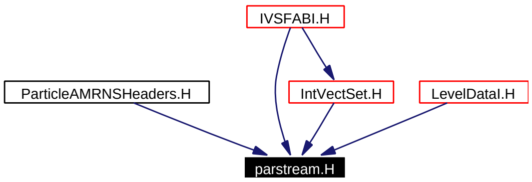
8.117 parstream.H File Reference

`#include <iostream>`

Include dependency graph for parstream.H:



This graph shows which files directly or indirectly include this file:



Functions

- std::ostream & **pout** ()
  used in place of std::cout for stream based output

8.117.1 Function Documentation

8.117.1.1 std::ostream& pout ()

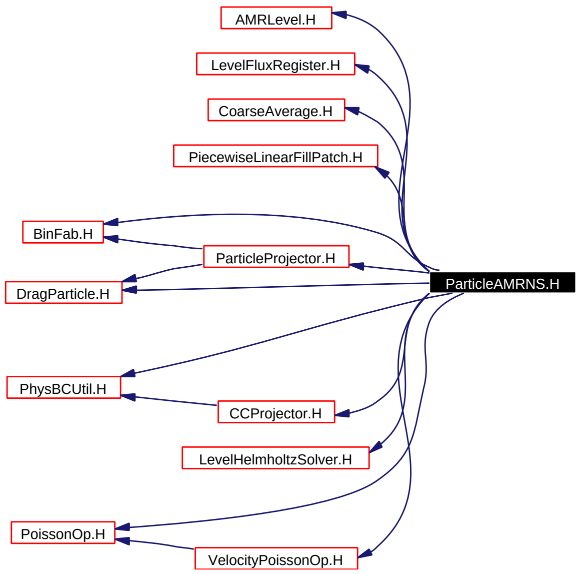
used in place of std::cout for stream based output

replaces std::cout in most of the Chombo code. In serial this just returns std::cout. in parallel, this creates a file called pout.n where n is the procID of the given processor. Output is then directed to these files. This keeps the output from different processors from getting all jumbled up.

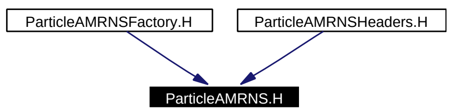
8.118 ParticleAMRNS.H File Reference

```
#include "AMRLevel.H"
#include "LevelFluxRegister.H"
#include "CoarseAverage.H"
#include "PiecewiseLinearFillPatch.H"
#include "BinFab.H"
#include "DragParticle.H"
#include "ParticleProjector.H"
#include "CCProjector.H"
#include "LevelHelmholtzSolver.H"
#include "PoissonOp.H"
#include "VelocityPoissonOp.H"
#include "PhysBCUtil.H"
```

Include dependency graph for ParticleAMRNS.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **ParticleAMRNS**

Enumerations

- enum **viscousSolverTypes** { **backwardEuler** = 0, **CrankNicolson**, **TGA**, **NUM_SOLVER_TYPES** }
- enum **scalTypes** { **xMarker** = 0, **yMarker**, **NUM_SCAL_TYPES** }

  scalar types – defined to faciliate lagrangian tracking

8.118.1 Enumeration Type Documentation

8.118.1.1 enum scalTypes

scalar types – defined to faciliate lagrangian tracking

Enumeration values:

xMarker

yMarker

NUM_SCAL_TYPES

8.118.1.2 enum viscousSolverTypes

Enumeration values:

backwardEuler

CrankNicolson

TGA

NUM_SOLVER_TYPES

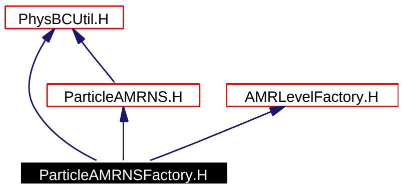
8.119 ParticleAMRNSFactory.H File Reference

`#include "ParticleAMRNS.H"`

`#include "AMRLevelFactory.H"`

`#include "PhysBCUtil.H"`

Include dependency graph for ParticleAMRNSFactory.H:



Compounds

- class **ParticleAMRNSFactory**

8.120 ParticleAMRNSHeaders.H File Reference

#include <iostream>

#include <cmath>

#include <cstdio>

#include "parstream.H"

#include "LoadBalance.H"

#include "LayoutIterator.H"

#include "FineInterp.H"

#include "ParmParse.H"

#include "LoHiSide.H"

#include "BoxIterator.H"

#include "ProblemDomain.H"

#include "AMRIO.H"

#include "ParticleAMRNS.H"

#include "GodunovTrace.H"

#include "basicVelBC.H"

#include "ExtrapBC.H"

#include "HOExtrapBC.H"

#include "PoissonBC.H"

#include "scalarDirichletBC.H"

#include "viscousBC.H"

#include "Divergence.H"

#include "CellToEdge.H"

#include "EdgeToCell.H"

#include "computeNorm.H"

#include "computeSum.H"

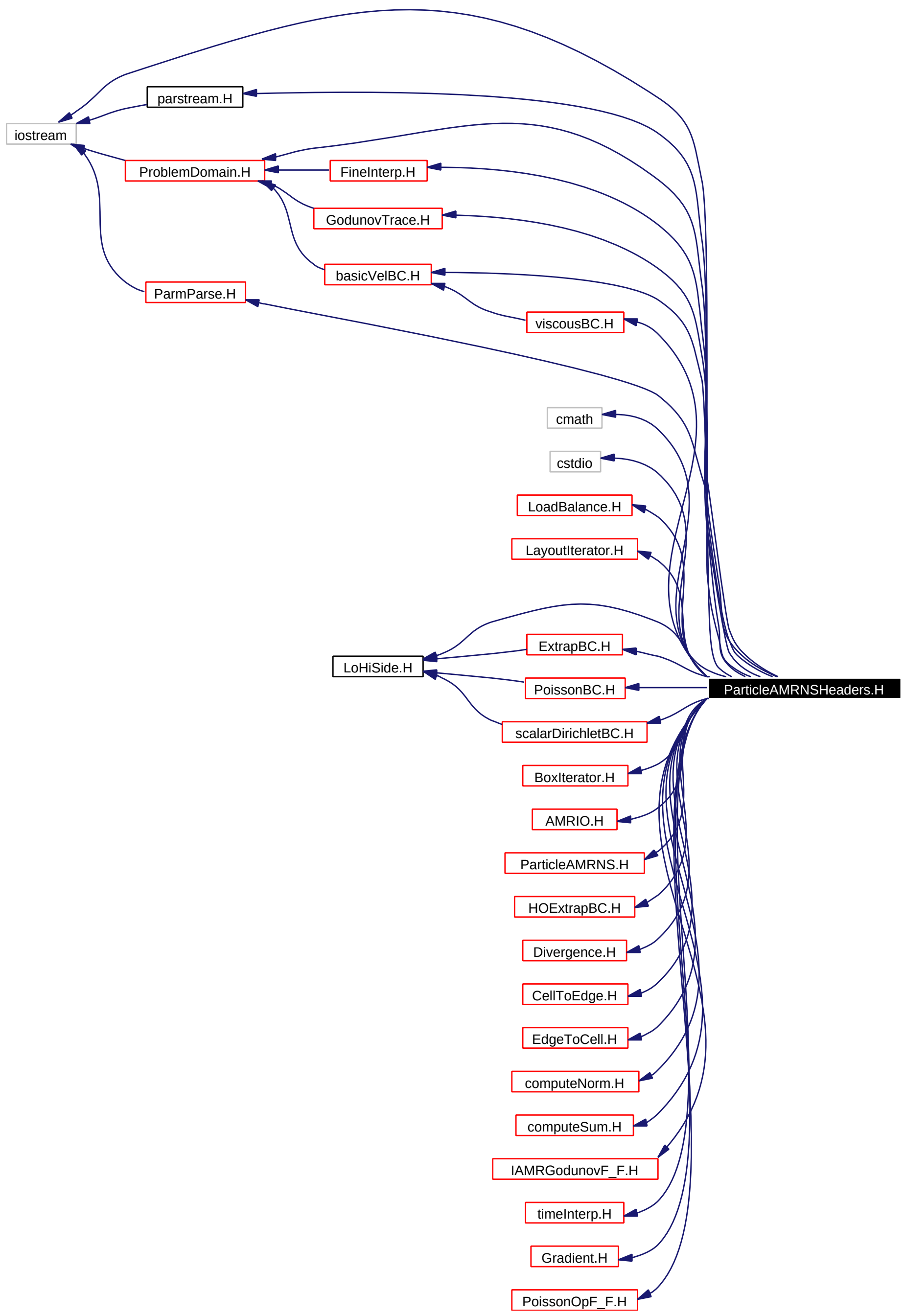
#include "IAMRGodunovF_F.H"

#include "timeInterp.H"

#include "Gradient.H"

#include "PoissonOpF_F.H"

Include dependency graph for ParticleAMRNSHeaders.H:



Defines

- #define **ADVECT_GROW** 2
- #define **TIME_EPS** 1.0e-5
- #define **TGA_EPS** 1.0e-8

8.120.1 Define Documentation

8.120.1.1 #define ADVECT_GROW 2

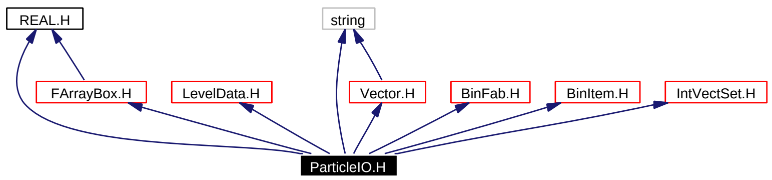
8.120.1.2 #define TGA_EPS 1.0e-8

8.120.1.3 #define TIME_EPS 1.0e-5

8.121 ParticleIO.H File Reference

```
#include "REAL.H"
#include "LevelData.H"
#include "Vector.H"
#include "FArrayBox.H"
#include "BinFab.H"
#include "BinItem.H"
#include "IntVectSet.H"
#include <string>
```

Include dependency graph for ParticleIO.H:



Functions

- void **writeAFabASCII** (std::ostream &os, const **FArrayBox** &fab)
- void **WriteAMRHierarchyASCII** (const string &filename, const **Vector**< **DisjointBoxLayout** > &a_vectGrids, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_vectData, const **Vector**< **LevelData**< **BinFab**< **BinItem** > ∗ > ∗ > &a_vectParticles, const **Vector**< string > &a_vectNames, const **Box** &a_domain, const **Real** &a_dx, const **Real** &a_dt, const **Real** &a_time, const **Vector**< int > &a_vectRatio, const int &a_numLevels)
- int **incrementParticleList** (**List**< **BinItem** > &a_particleList, const **BaseFab**< **BinItem** > &a_binFab)

  helper function

8.121.1 Function Documentation

8.121.1.1 int incrementParticleList (List< BinItem > & *a_particleList*, const BaseFab< BinItem > & *a_binFab*)

helper function

8.121.1.2 void writeAFabASCII (std::ostream & *os*, const FArrayBox & *fab*)

8.121.1.3 void WriteAMRHierarchyASCII (const string & *filename*, const Vector< DisjointBoxLayout > & *a_vectGrids*, const Vector< LevelData< FArrayBox > * > & *a_vectData*, const Vector< LevelData< BinFab< BinItem > * > * > & *a_vectParticles*, const Vector< string > & *a_vectNames*, const Box & *a_domain*, const Real & *a_dx*, const Real & *a_dt*, const Real & *a_time*, const Vector< int > & *a_vectRatio*, const int & *a_numLevels*)

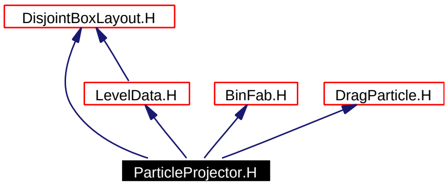
Writes hierarchy of levels in ASCII format.

{ Arguments:}\ filename : file to output to.\ a_vectGrids : grids at each level.\ a_vectData : data at each level.\ a_vectNames: names of variables.\ a_domain : domain at coarsest level.\ a_dx : grid spacing at coarsest level.\ a_dt : time step at coarsest level.\ a_time : time.\ a_vectRatio : refinement ratio at all levels (ith entry is refinement ratio between levels i and i + 1).\ a_numLevels : number of levels to output.\

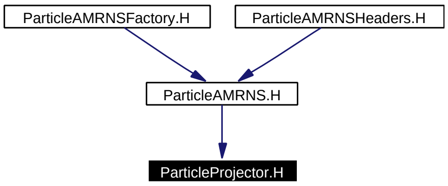
8.122 ParticleProjector.H File Reference

```
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "BinFab.H"
#include "DragParticle.H"
```

Include dependency graph for ParticleProjector.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **ParticleProjector**

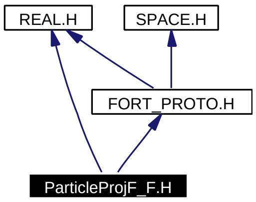
    class to manage projection of particle drag force onto the fluid

8.123 ParticleProjF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for ParticleProjF_F.H:



Defines

- #define **FORT_SETINFDOMAINPATCHSIZE** FORTRAN_NAME(SETINFDOMAINPATCHSIZE ,setinfdomainpatchsize)

Functions

- void **FORT_SETINFDOMAINPATCHSIZE** (CHFp_INT(ptchsz),CHFp_INT(s),CHFp_INT(n))

8.123.1 Define Documentation

8.123.1.1 #define FORT_SETINFDOMAINPATCHSIZE FORTRAN_NAME(SETINFDOMAINPATCHSIZE ,setinfdomainpatchsize)

8.123.2 Function Documentation

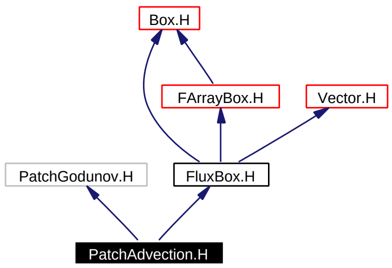
8.123.2.1 void FORT_SETINFDOMAINPATCHSIZE (CHFp_INT(ptchsz), CHFp_INT(s), CHFp_INT(n))

8.124 PatchAdvection.H File Reference

`#include "PatchGodunov.H"`

`#include "FluxBox.H"`

Include dependency graph for PatchAdvection.H:



Compounds

- class **PatchAdvection**

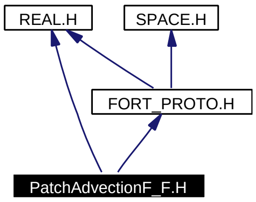
    An PatchGodunov-derived class for simple advection-diffusion problems.

8.125 PatchAdvectionF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for PatchAdvectionF_F.H:



Defines

- #define **FORT_NORMALPREDF** FORTRAN_NAME(NORMALPREDF ,normalpredf)
- #define **FORT_RIEMANNF** FORTRAN_NAME(RIEMANNF ,riemannf)
- #define **FORT_UPDATECONSADF** FORTRAN_NAME(UPDATECONSADF ,updateconsadf)

Functions

- void **FORT_NORMALPREDF** (CHFp_FRA(WMinus),CHFp_FRA(WPlus),CHFp_CONST_FRA(W),CHFp_CONST_FRA(dW),CHFp_CONST_FRA(cellVel),CHFp_CONST_REAL(dtbydx),CHFp_CONST_INT(normdir),CHFp_BOX(box))
- void **FORT_RIEMANNF** (CHFp_FRA(Wgdnv),CHFp_CONST_FRA(WLeft),CHFp_CONST_FRA(WRight),CHFp_CONST_FRA1(advVel),CHFp_CONST_INT(idir),CHFp_BOX(box))
- void **FORT_UPDATECONSADF** (CHFp_FRA(U),CHFp_CONST_FRA(F),CHFp_CONST_FRA(vel),CHFp_CONST_REAL(dtbydx),CHFp_CONST_INT(idir),CHFp_BOX(box))

8.125.1 Define Documentation

8.125.1.1 #define FORT_NORMALPREDF FORTRAN_NAME(NORMALPREDF ,normalpredf)

8.125.1.2 #define FORT_RIEMANNF FORTRAN_NAME(RIEMANNF ,riemannf)

8.125.1.3 #define FORT_UPDATECONSADF FORTRAN_NAME(UPDATECONSADF ,updateconsadf)

8.125.2 Function Documentation

8.125.2.1 void FORT_NORMALPREDF (CHFp_FRA(WMinus), CHFp_FRA(WPlus), CHFp_CONST_FRA(W), CHFp_CONST_FRA(dW), CHFp_CONST_FRA(cellVel), CHFp_CONST_REAL(dtbydx), CHFp_CONST_INT(normdir), CHFp_BOX(box))

8.125.2.2 void FORT_RIEMANNF (CHFp_FRA(Wgdnv), CHFp_CONST_FRA(WLeft), CHFp_CONST_FRA(WRight), CHFp_CONST_FRA1(advVel), CHFp_CONST_INT(idir), CHFp_BOX(box))

8.125.2.3 void FORT_UPDATECONSADF (CHFp_FRA(U), CHFp_CONST_FRA(F), CHFp_CONST_FRA(vel), CHFp_CONST_REAL(dtbydx), CHFp_CONST_INT(idir), CHFp_BOX(box))

8.126 PhysBCUtil.H File Reference

`#include "GhostBC.H"`

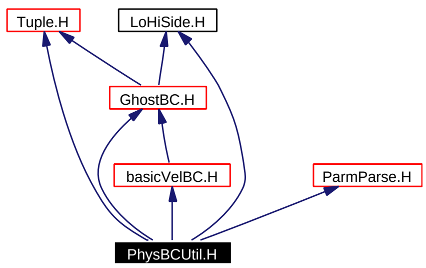
`#include "Tuple.H"`

`#include "ParmParse.H"`

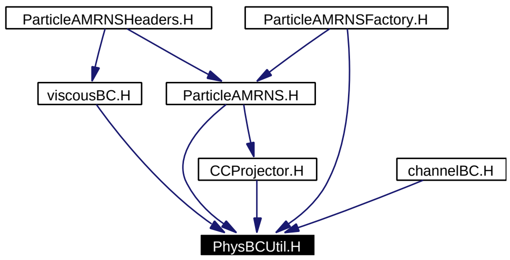
`#include "LoHiSide.H"`

`#include "basicVelBC.H"`

Include dependency graph for PhysBCUtil.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **PhysBCUtil**

  Encapsulates all physical BC's for IAMR solves.

8.127 PiecewiseLinearFillPatch.H File Reference

`#include <iostream>`

`#include <fstream>`

`#include "REAL.H"`

`#include "Box.H"`

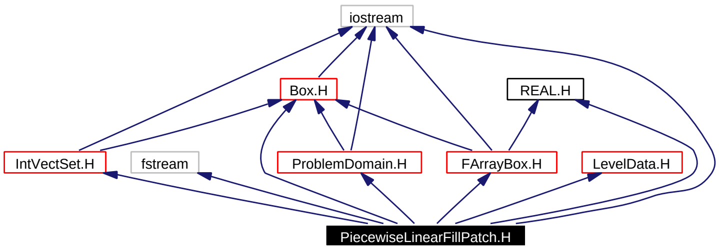
`#include "FArrayBox.H"`

`#include "LevelData.H"`

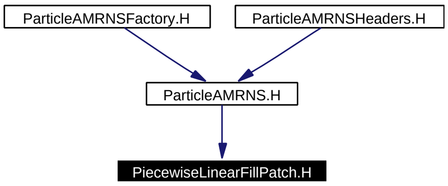
`#include "IntVectSet.H"`

`#include "ProblemDomain.H"`

Include dependency graph for PiecewiseLinearFillPatch.H:



This graph shows which files directly or indirectly include this file:



Compounds

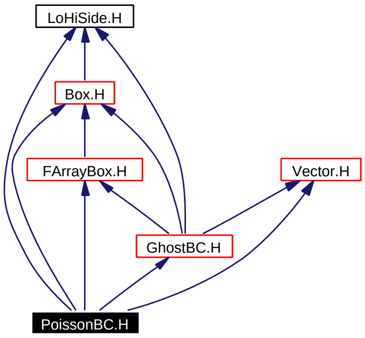
- class **PiecewiseLinearFillPatch**

  Fills ghost cells by linear interpolation in space and time.

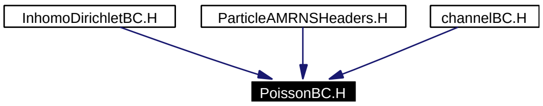
8.128 PoissonBC.H File Reference

```
#include "Box.H"
#include "FArrayBox.H"
#include "Vector.H"
#include "LoHiSide.H"
#include "GhostBC.H"
```

Include dependency graph for PoissonBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DirichletBC**

  Class to enforce homogeneous Dirichlet boundary conditions.

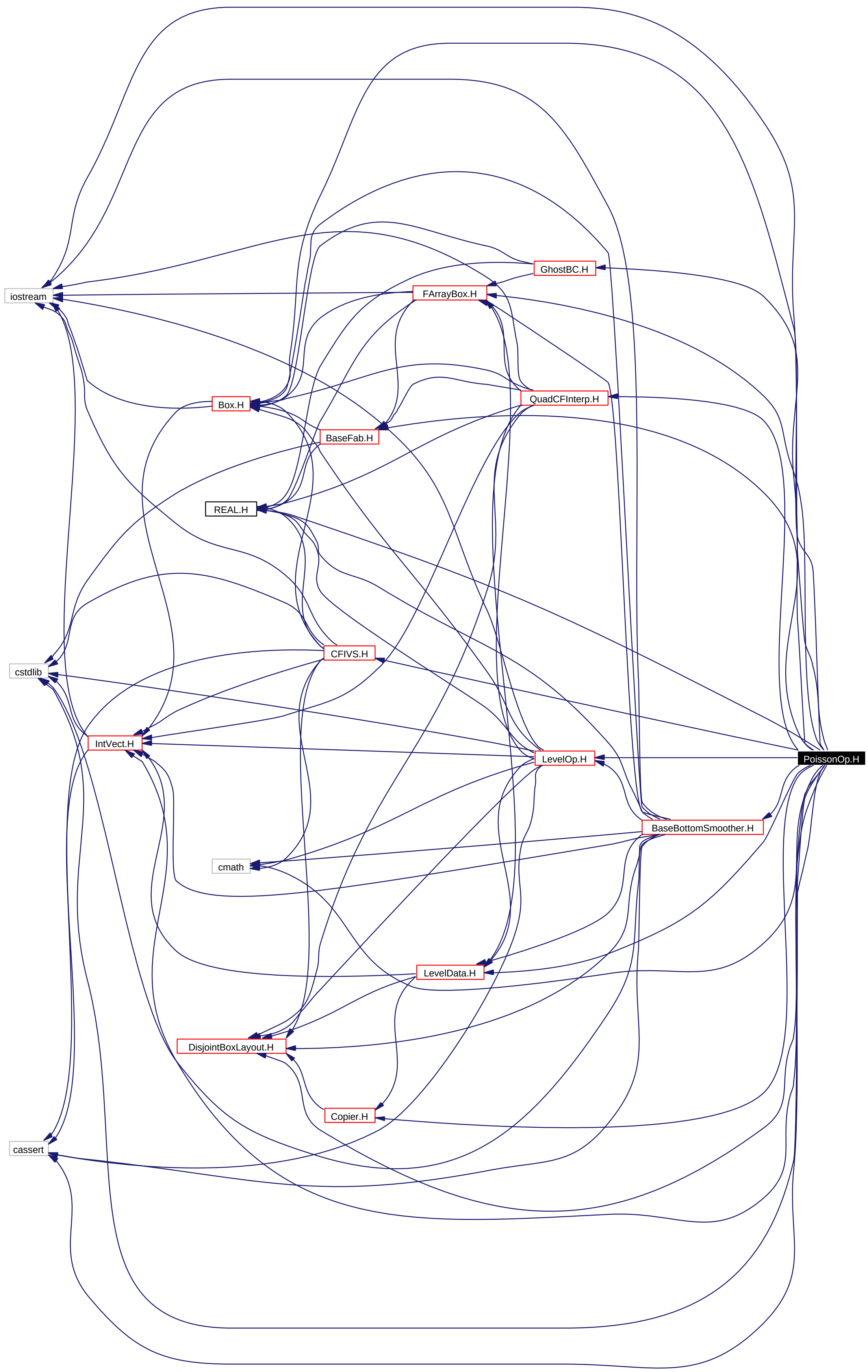
- class **NeumannBC**

  Class to enforce homogeneous Neumann boundary conditions.

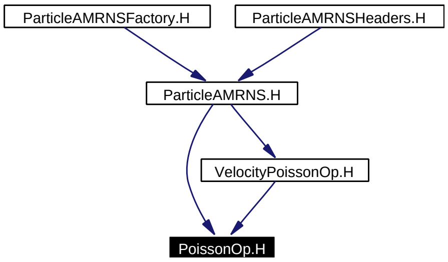
8.129 PoissonOp.H File Reference

```
#include <iostream>
#include <cmath>
#include <cassert>
#include <cstdlib>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "BaseFab.H"
#include "FArrayBox.H"
#include "QuadCFInterp.H"
#include "Copier.H"
#include "GhostBC.H"
#include "LevelOp.H"
#include "CFIVS.H"
#include "BaseBottomSmoother.H"
```

Include dependency graph for PoissonOp.H:


GhostBC.H
FArrayBox.H
iostream
QuadCFInterp.H
Box.H
BaseFab.H
REAL.H
CFIVS.H
cstdlib
IntVect.H
LevelOp.H
PoissonOp.H
BaseBottomSmoother.H
cmath
LevelData.H
DisjointBoxLayout.H
Copier.H
cassert

This graph shows which files directly or indirectly include this file:



Compounds

- class **PoissonOp**

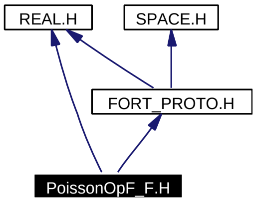
  one-sided differencing version of poisson operator

8.130 PoissonOpF_F.H File Reference

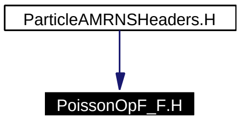
`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for PoissonOpF_F.H:



This graph shows which files directly or indirectly include this file:



Defines

- #define **FORT_GSRBLEVELLAP** FORTRAN_NAME(GSRBLEVELLAP ,gsrblevellap)
- #define **FORT_OPERATORLAP** FORTRAN_NAME(OPERATORLAP ,operatorlap)
- #define **FORT_INTERPHOMO** FORTRAN_NAME(INTERPHOMO ,interphomo)

Functions

- void **FORT_GSRBLEVELLAP** (CHFp_FRA(phi),CHFp_CONST_FRA(rhs),CHFp_BOX(region),CHFp_CONST_REAL(dx),CHFp_CONST_INT(redBlack))
- void **FORT_OPERATORLAP** (CHFp_FRA(lofphi),CHFp_CONST_FRA(phi),CHFp_BOX(region),CHFp_CONST_REAL(dx))
- void **FORT_INTERPHOMO** (CHFp_FRA(phi),CHFp_BOX(region),CHFp_CONST_REAL(x1),CHFp_CONST_REAL(m_dxCrse),CHFp_CONST_INT(a_idir),CHFp_CONST_INT(ihilo))

8.130.1 Define Documentation

8.130.1.1 #define FORT_GSRBLEVELLAP FORTRAN_NAME(GSRBLEVELLAP ,gsrblevellap)

8.130.1.2 #define FORT_INTERPHOMO FORTRAN_NAME(INTERPHOMO ,interphomo)

8.130.1.3 #define FORT_OPERATORLAP FORTRAN_NAME(OPERATORLAP ,operatorlap)

8.130.2 Function Documentation

8.130.2.1 void FORT_GSRBLEVELLAP (CHFp_FRA(phi), CHFp_CONST_FRA(rhs), CHFp_BOX(region), CHFp_CONST_REAL(dx), CHFp_CONST_INT(redBlack))

8.130.2.2 void FORT_INTERPHOMO (CHFp_FRA(phi), CHFp_BOX(region), CHFp_CONST_REAL(x1), CHFp_CONST_REAL(m_dxCrse), CHFp_CONST_INT(a_idir), CHFp_CONST_INT(ihilo))

8.130.2.3 void FORT_OPERATORLAP (CHFp_FRA(lofphi), CHFp_CONST_FRA(phi), CHFp_BOX(region), CHFp_CONST_REAL(dx))

8.131 POLYDELTADEF.H File Reference

Defines

- #define **NUM_POLY_COEFF** 6

8.131.1 Define Documentation

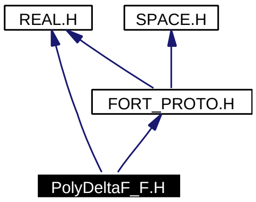
8.131.1.1 #define NUM_POLY_COEFF 6

8.132 PolyDeltaF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for PolyDeltaF_F.H:



Defines

- #define **FORT_POLYKSUM** FORTRAN_NAME(POLYKSUM ,polyksum)

Functions

- void **FORT_POLYKSUM** (CHFp_FRA1(sum),CHFp_CONST_REALVECT(force),CHFp_CONST_REALVECT(position),CHFp_CONST_REALVECT(origin),CHFp_BOX(box),CHFp_CONST_REAL(dx),CHFp_CONST_REAL(epsilon),CHFp_CONST_REAL(scaleFactor),CHFp_INT(comp))

8.132.1 Define Documentation

8.132.1.1 #define FORT_POLYKSUM FORTRAN_NAME(POLYKSUM ,polyksum)

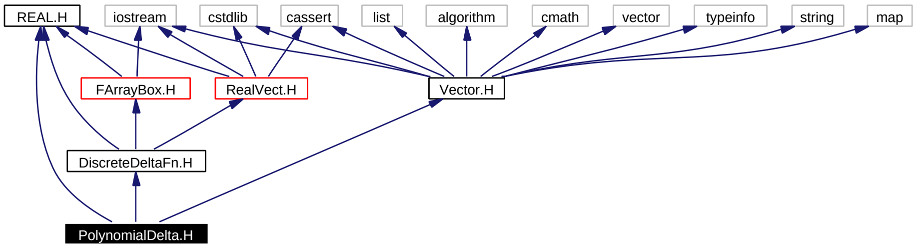
8.132.2 Function Documentation

8.132.2.1 void FORT_POLYKSUM (CHFp_FRA1(sum), CHFp_CONST_REALVECT(force), CHFp_CONST_REALVECT(position), CHFp_CONST_REALVECT(origin), CHFp_BOX(box), CHFp_CONST_REAL(dx), CHFp_CONST_REAL(epsilon), CHFp_CONST_REAL(scaleFactor), CHFp_INT(comp))

8.133 PolynomialDelta.H File Reference

```
#include "REAL.H"
#include "DiscreteDeltaFn.H"
#include "Vector.H"
```

Include dependency graph for PolynomialDelta.H:



Compounds

- class **PolynomialDelta**

  Instantiation of discrete delta function using a polynomial.

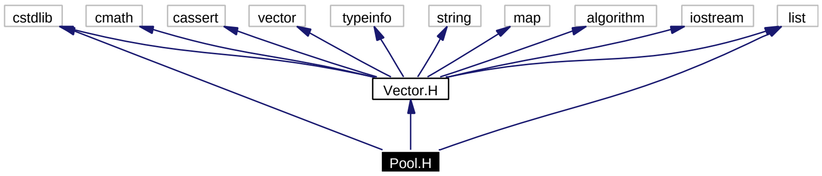
8.134 Pool.H File Reference

`#include <cstdlib>`

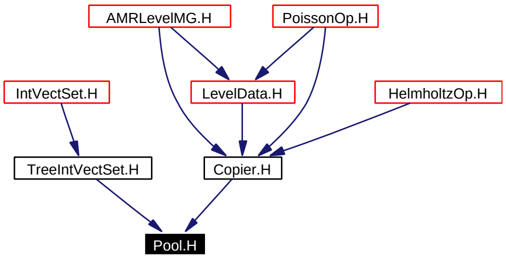
`#include "Vector.H"`

`#include <list>`

Include dependency graph for Pool.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Pool**

  Pool is a class to optimize memory allocation.

Typedefs

- typedef std::list< **Pool** ∗ > **PoolList**

8.134.1 Typedef Documentation

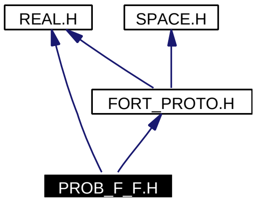
8.134.1.1 typedef std::list<Pool∗> PoolList

8.135 PROB_F_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for PROB_F_F.H:



Defines

- #define **FORT_INITVEL** FORTRAN_NAME(INITVEL ,initvel)
- #define **FORT_SETTAGGINGVAR** FORTRAN_NAME(SETTAGGINGVAR ,settaggingvar)
- #define **FORT_INITSCAL** FORTRAN_NAME(INITSCAL ,initscal)
- #define **FORT_INITVORT** FORTRAN_NAME(INITVORT ,initvort)
- #define **FORT_COMPUTEEXACT** FORTRAN_NAME(COMPUTEEXACT ,computeexact)

Functions

- void **FORT_INITVEL** (CHFp_FRA(vel),CHFp_REAL(dx))
- void **FORT_SETTAGGINGVAR** (CHFp_FRA1(tagvar),CHFp_CONST_REAL(dx))
- void **FORT_INITSCAL** (CHFp_FRA(scal),CHFp_REAL(dx),CHFp_REAL(time),CHFp_INT(comp),CHFp_BOX(domain),CHFp_INT(isEC))
- void **FORT_INITVORT** (CHFp_FRA1(vort),CHFp_INT(comp),CHFp_REAL(dx))
- void **FORT_COMPUTEEXACT** (CHFp_FRA(exact),CHFp_CONST_REAL(nu),CHFp_CONST_REAL(time),CHFp_CONST_REAL(dx))

8.135.1 Define Documentation

8.135.1.1 #define FORT_COMPUTEEXACT FORTRAN_NAME(COMPUTEEXACT ,computeexact)

8.135.1.2 #define FORT_INITSCAL FORTRAN_NAME(INITSCAL ,initscal)

8.135.1.3 #define FORT_INITVEL FORTRAN_NAME(INITVEL ,initvel)

8.135.1.4 #define FORT_INITVORT FORTRAN_NAME(INITVORT ,initvort)

8.135.1.5 #define FORT_SETTAGGINGVAR FORTRAN_NAME(SETTAGGINGVAR ,settaggingvar)

8.135.2 Function Documentation

8.135.2.1 void FORT_COMPUTEEXACT (CHFp_FRA(exact), CHFp_CONST_REAL(nu), CHFp_CONST_REAL(time), CHFp_CONST_REAL(dx))

8.135.2.2 void FORT_INITSCAL (CHFp_FRA(scal), CHFp_REAL(dx), CHFp_REAL(time), CHFp_INT(comp), CHFp_BOX(domain), CHFp_INT(isEC))

8.135.2.3 void FORT_INITVEL (CHFp_FRA(vel), CHFp_REAL(dx))

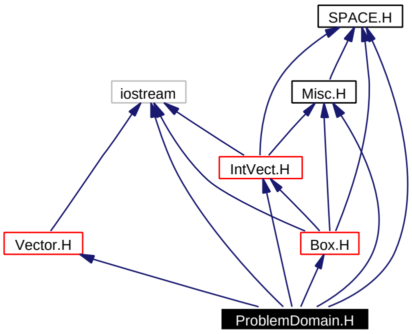
8.135.2.4 void FORT_INITVORT (CHFp_FRA1(vort), CHFp_INT(comp), CHFp_REAL(dx))

8.135.2.5 void FORT_SETTAGGINGVAR (CHFp_FRA1(tagvar), CHFp_CONST_REAL(dx))

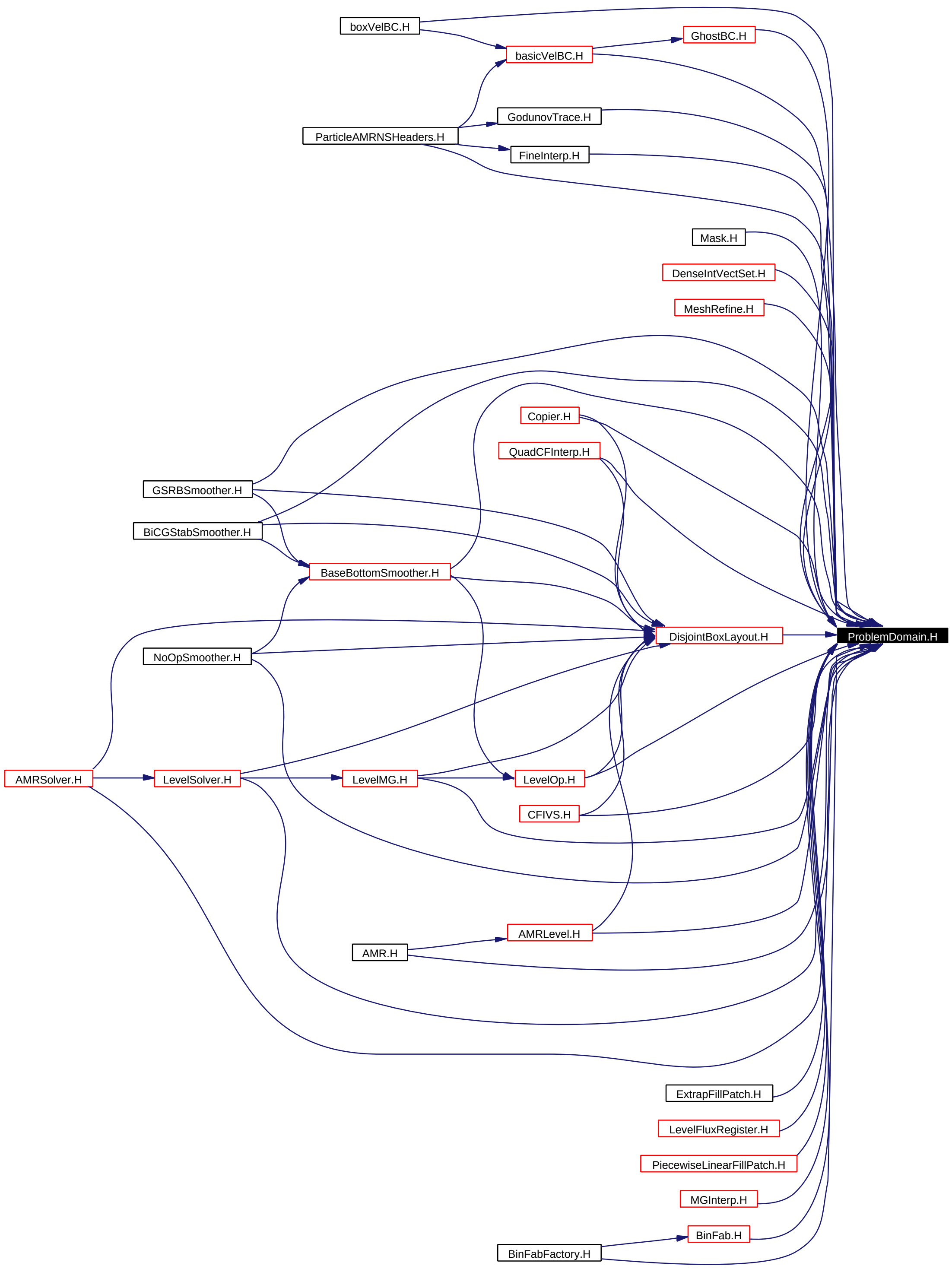
8.136 ProblemDomain.H File Reference

```
#include <iostream>
#include "Vector.H"
#include "IntVect.H"
#include "Box.H"
#include "Misc.H"
#include "SPACE.H"
```

Include dependency graph for ProblemDomain.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **ProblemDomain**

  A class to facilitate interaction with physical boundary conditions.

- class **ShiftIterator**

    Class to manage box-shifting used to enforce periodic BC's.

Functions

- **ProblemDomain grow** (const **ProblemDomain** &pd, int i)
- **ProblemDomain grow** (const **ProblemDomain** &pd, const **IntVect** &v)

8.136.1 Function Documentation

8.136.1.1 ProblemDomain grow (const ProblemDomain & *pd*, const IntVect & *v*) [inline]

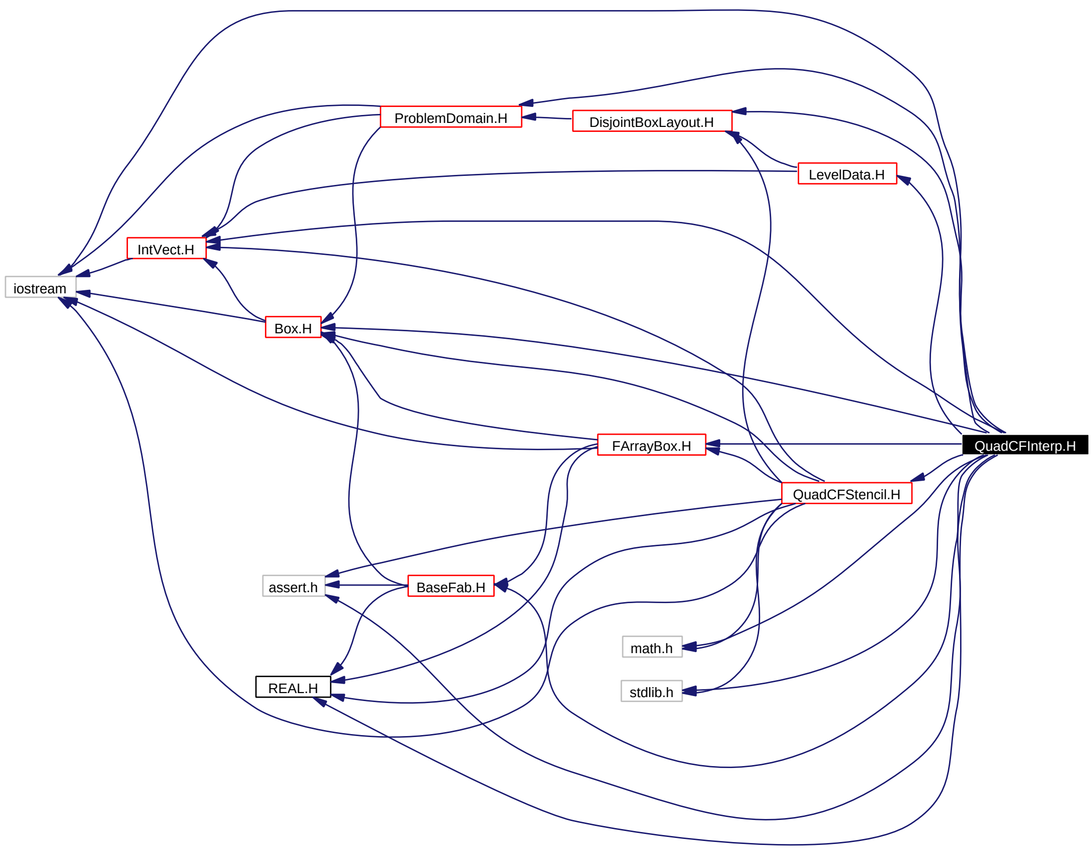
Returns a **ProblemDomain** that is the argument **ProblemDomain** with a DomainBox grown by the given amount.

8.136.1.2 ProblemDomain grow (const ProblemDomain & *pd*, int *i*) [inline]

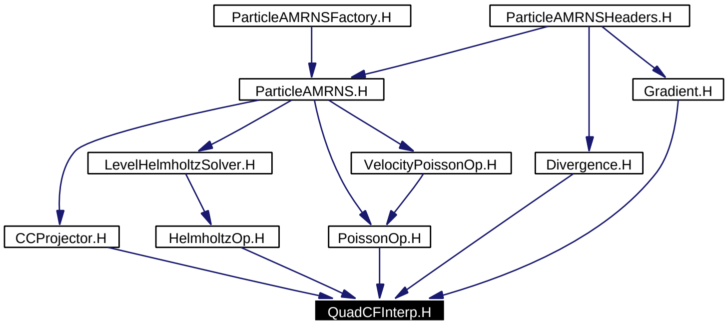
8.137 QuadCFInterp.H File Reference

```
#include <iostream>
#include <math.h>
#include <assert.h>
#include <stdlib.h>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "BaseFab.H"
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "FArrayBox.H"
#include "QuadCFStencil.H"
#include "ProblemDomain.H"
```

Include dependency graph for QuadCFInterp.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **QuadCFInterp**

  Quadratic coarse-fine interpolation utility.

8.138 QuadCFStencil.H File Reference

`#include <iostream>`

`#include <math.h>`

`#include <assert.h>`

`#include <stdlib.h>`

`#include "REAL.H"`

`#include "IntVect.H"`

`#include "Box.H"`

`#include "DisjointBoxLayout.H"`

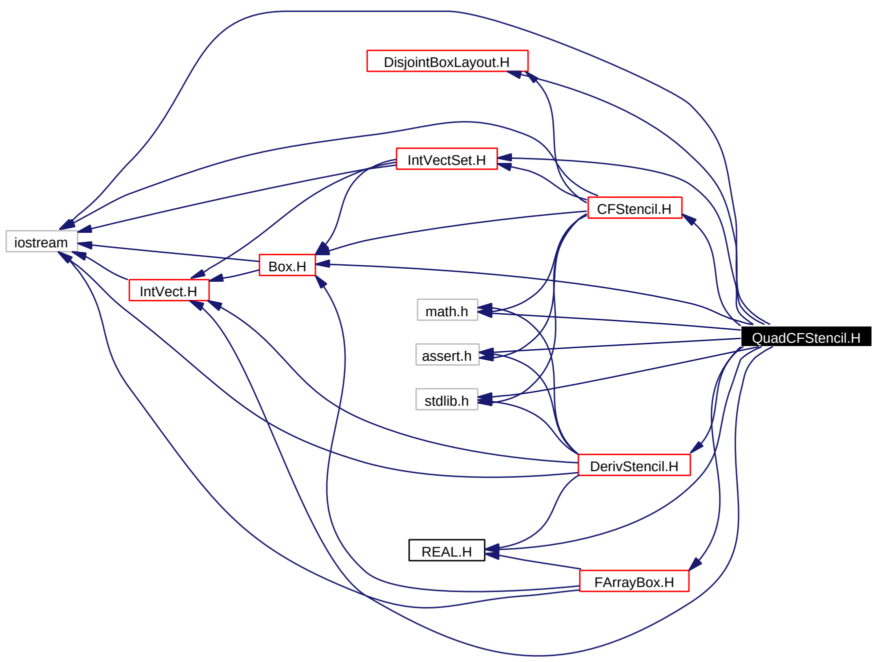
`#include "FArrayBox.H"`

`#include "IntVectSet.H"`

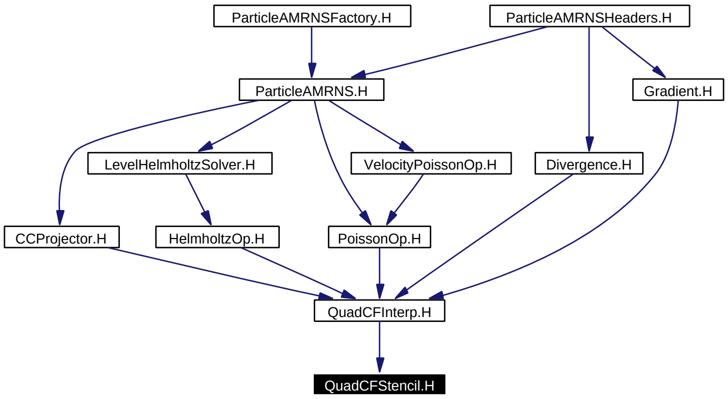
`#include "CFStencil.H"`

`#include "DerivStencil.H"`

Include dependency graph for QuadCFStencil.H:



This graph shows which files directly or indirectly include this file:



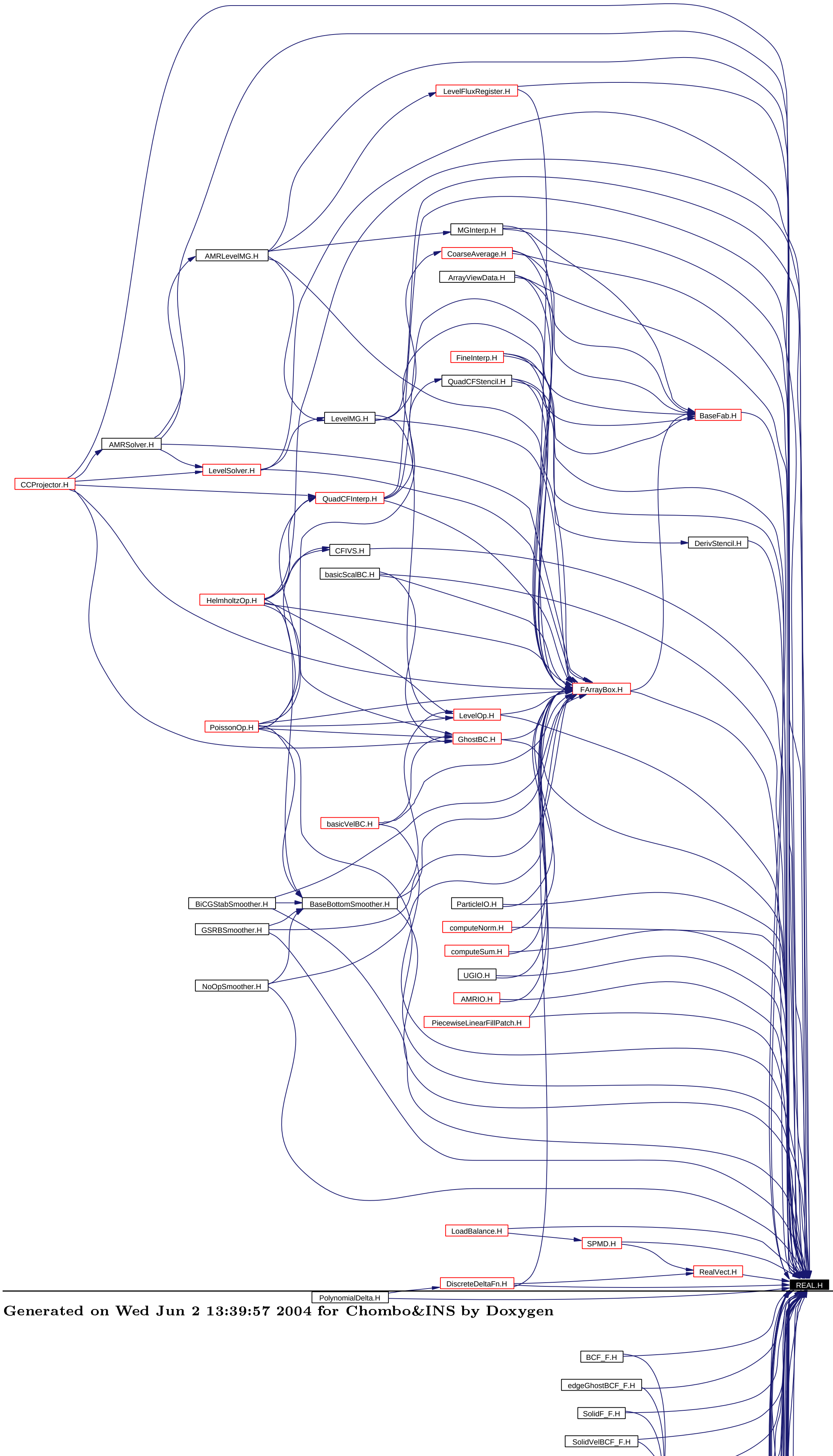
Compounds

- class **QuadCFStencil**

  class to encapsulate CF info for quadratic interpolation

8.139 REAL.H File Reference

This graph shows which files directly or indirectly include this file:


CCProjector.H
AMRSolver.H
AMRLevelMG.H
LevelFluxRegister.H
MGInterp.H
CoarseAverage.H
ArrayViewData.H
FineInterp.H
QuadCFStencil.H
LevelMG.H
BaseFab.H
LevelSolver.H
QuadCFInterp.H
DerivStencil.H
CFIVS.H
basicScalBC.H
HelmholtzOp.H
FArrayBox.H
LevelOp.H
PoissonOp.H
GhostBC.H
basicVelBC.H
BiCGStabSmoother.H
BaseBottomSmoother.H
ParticleIO.H
GSRBSmoother.H
computeNorm.H
computeSum.H
NoOpSmoother.H
UGIO.H
AMRIO.H
PiecewiseLinearFillPatch.H
LoadBalance.H
SPMD.H
RealVect.H
DiscreteDeltaFn.H
REAL.H
PolynomialDelta.H
BCF_F.H
edgeGhostBCF_F.H
SolidF_F.H
SolidVelBCF_F.H

Defines

- #define **CH_USE_DOUBLE** 1
- #define **MPI_CH_REAL** MPI_DOUBLE
- #define **H5T_NATIVE_REAL** H5T_NATIVE_DOUBLE

Typedefs

- typedef double **Real**

8.139.1 Define Documentation

8.139.1.1 #define CH_USE_DOUBLE 1

8.139.1.2 #define H5T_NATIVE_REAL H5T_NATIVE_DOUBLE

8.139.1.3 #define MPI_CH_REAL MPI_DOUBLE

8.139.2 Typedef Documentation

8.139.2.1 typedef double Real

8.140 RealVect.H File Reference

`#include <cstddef>`

`#include <cstdlib>`

`#include <cstring>`

`#include <iostream>`

`#include "SPACE.H"`

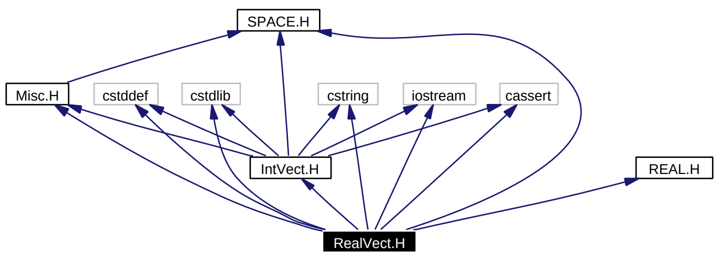
`#include "Misc.H"`

`#include "REAL.H"`

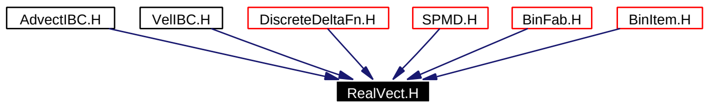
`#include <cassert>`

`#include "IntVect.H"`

Include dependency graph for RealVect.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **RealVect**

  A Real vector in SpaceDim-dimensional space.

Functions

- **RealVect BASISREALV** (int idir)

8.140.1 Function Documentation

8.140.1.1 RealVect BASISREALV (int *idir*)

Returns a basis vector in the given coordinate direction.

In 2-D:

BASISREALV(0) == (1.,0.); BASISREALV(1) == (0.,1.).

In 3-D:

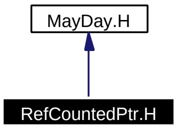
BASISREALV(0) == (1.,0.,0.); BASISREALV(1) == (0.,1.,0.); BASISREALV(2) == (0.,0.,1.).

Note that the coordinate directions are based at zero.

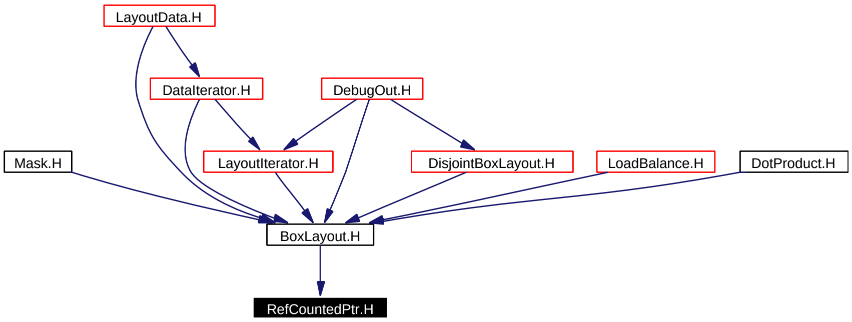
8.141 RefCountedPtr.H File Reference

`#include "MayDay.H"`

Include dependency graph for RefCountedPtr.H:



This graph shows which files directly or indirectly include this file:



Compounds

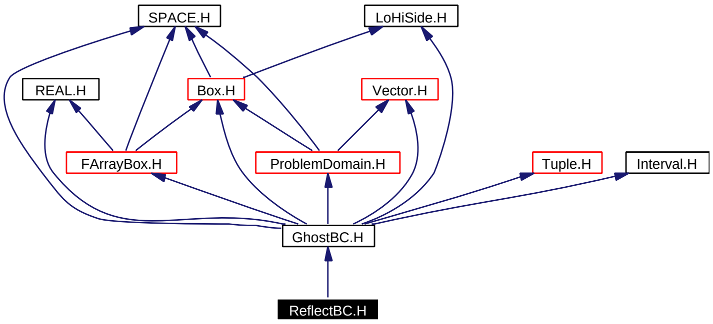
- class **RefCountedPtr**

  A reference-counting handle class.

8.142 ReflectBC.H File Reference

`#include "GhostBC.H"`

Include dependency graph for ReflectBC.H:



Compounds

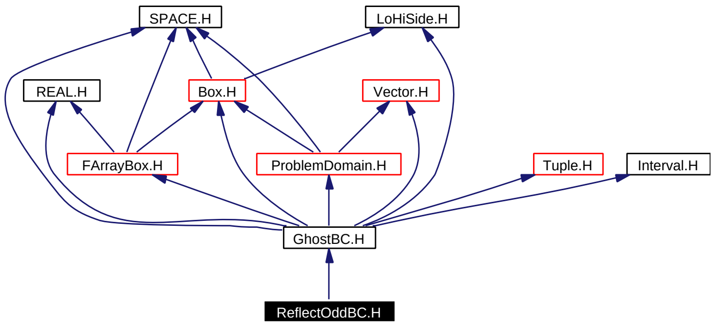
- class **ReflectBC**

  class to do extrapolation BC's

8.143 ReflectOddBC.H File Reference

`#include "GhostBC.H"`

Include dependency graph for ReflectOddBC.H:



Compounds

- class **ReflectOddBC**

  class to do extrapolation BC's

8.144 scalarDirichletBC.H File Reference

`#include "Box.H"`

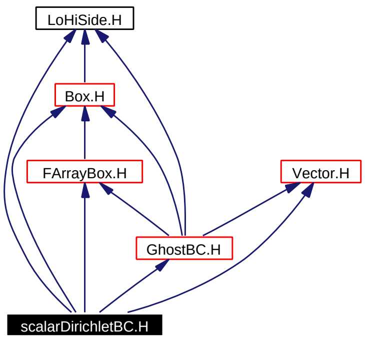
`#include "FArrayBox.H"`

`#include "Vector.H"`

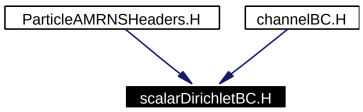
`#include "LoHiSide.H"`

`#include "GhostBC.H"`

Include dependency graph for scalarDirichletBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **scalarDirichletBC**

  implements dirichlet BCs for scalars

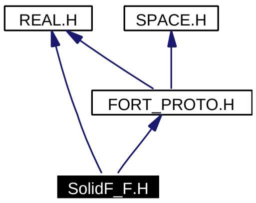
- class **scalarInflowBC**

8.145 SolidF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for SolidF_F.H:



Defines

- #define **FORT_SOLIDBCF** FORTRAN_NAME(SOLIDBCF ,solidbcf)
- #define **FORT_SLOPEBCSF** FORTRAN_NAME(SLOPEBCSF ,slopebcsf)

Functions

- void **FORT_SOLIDBCF** (CHFp_FRA(F),CHFp_CONST_FRA(Wextrap),CHFp_CONST_REAL(bcVal),CHFp_CONST_INT(lohisign),CHFp_CONST_REAL(dx),CHFp_CONST_INT(dir),CHFp_BOX(box))
- void **FORT_SLOPEBCSF** (CHFp_FRA(dW),CHFp_CONST_FRA(W),CHFp_CONST_REAL(dx),CHFp_CONST_INT(dir),CHFp_CONST_REAL(loVal),CHFp_BOX(loBox),CHFp_CONST_INT(hasLo),CHFp_CONST_REAL(hiVal),CHFp_BOX(hiBox),CHFp_CONST_INT(hasHi))

8.145.1 Define Documentation

8.145.1.1 #define FORT_SLOPEBCSF FORTRAN_NAME(SLOPEBCSF ,slopebcsf)

8.145.1.2 #define FORT_SOLIDBCF FORTRAN_NAME(SOLIDBCF ,solidbcf)

8.145.2 Function Documentation

8.145.2.1 void FORT_SLOPEBCSF (CHFp_FRA(dW), CHFp_CONST_FRA(W), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir), CHFp_CONST_REAL(loVal), CHFp_BOX(loBox), CHFp_CONST_INT(hasLo), CHFp_CONST_REAL(hiVal), CHFp_BOX(hiBox), CHFp_CONST_INT(hasHi))

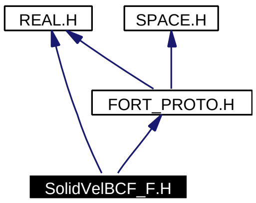
8.145.2.2 void FORT_SOLIDBCF (CHFp_FRA(F), CHFp_CONST_FRA(Wextrap), CHFp_CONST_REAL(bcVal), CHFp_CONST_INT(lohisign), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir), CHFp_BOX(box))

8.146 SolidVelBCF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for SolidVelBCF_F.H:



Defines

- #define **FORT_SOLIDVELBCF** FORTRAN_NAME(SOLIDVELBCF ,solidvelbcf)

Functions

- void **FORT_SOLIDVELBCF** (CHFp_FRA(F),CHFp_CONST_FRA(Wextrap),CHFp_-CONST_REAL(bcVal),CHFp_CONST_INT(lohisign),CHFp_CONST_REAL(dx),CHFp_-CONST_INT(dir),CHFp_BOX(box))

8.146.1 Define Documentation

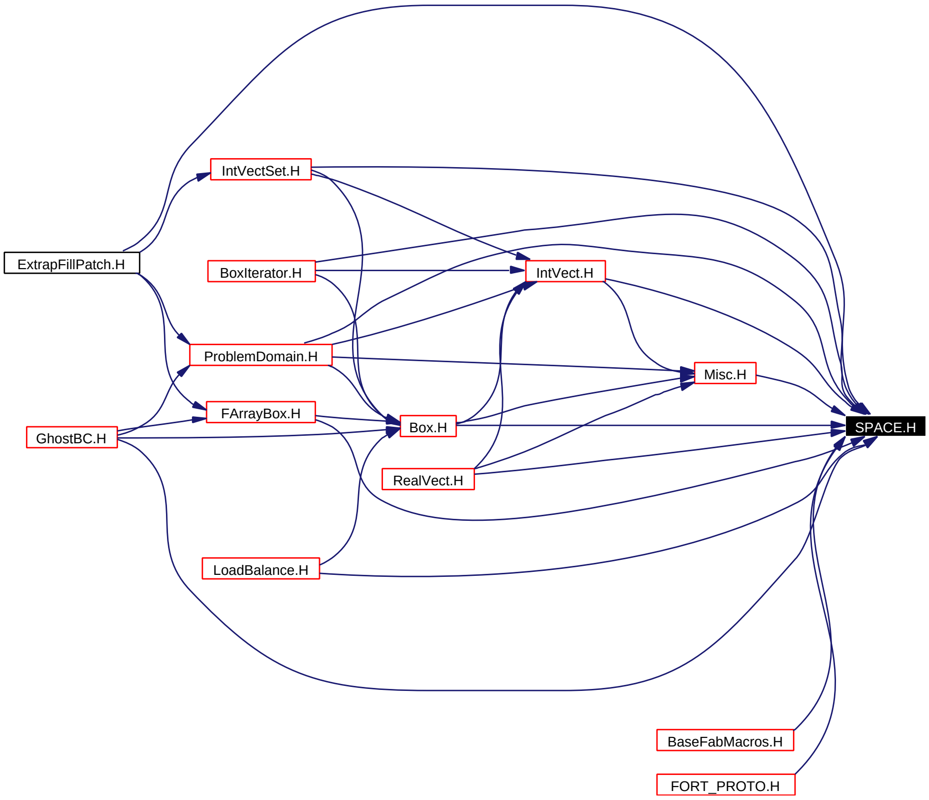
8.146.1.1 #define FORT_SOLIDVELBCF FORTRAN_NAME(SOLIDVELBCF ,solidvelbcf)

8.146.2 Function Documentation

8.146.2.1 void FORT_SOLIDVELBCF (CHFp_FRA(F), CHFp_-CONST_FRA(Wextrap), CHFp_CONST_REAL(bcVal), CHFp_CONST_INT(lohisign), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir), CHFp_BOX(box))

8.147 SPACE.H File Reference

This graph shows which files directly or indirectly include this file:



Variables

- const int **SpaceDim** = CH_SPACEDIM

8.147.1 Variable Documentation

8.147.1.1 const int SpaceDim = CH_SPACEDIM

The SpaceDim C++ integer constant specifies the dimension of the Chombo library to be built. It is initialized to the value of the macro CH_SPACEDIM. Hence it will have one of the values 1, 2, or 3.

C++ code that depends on the dimensionality of the problem that you're trying to solve should be cast in terms of this constant, instead of the CH_SPACEDIM macro, if possible, as constants are visible to debuggers while macros are not.

8.148 SPMD.H File Reference

`#include "REAL.H"`

`#include "Vector.H"`

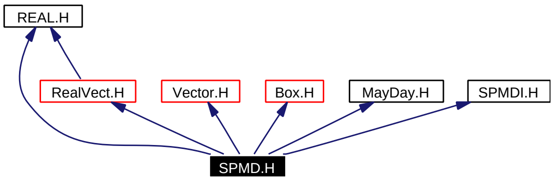
`#include "Box.H"`

`#include "MayDay.H"`

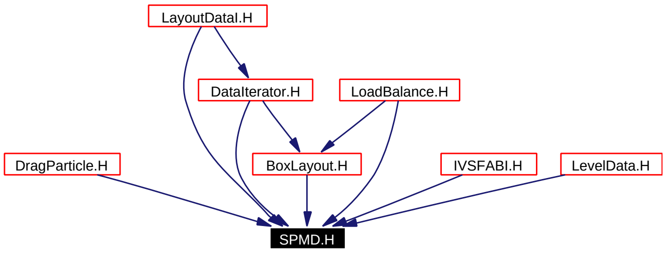
`#include "RealVect.H"`

`#include "SPMDI.H"`

Include dependency graph for SPMD.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **SerialTask**

Functions

- int **procID** ()
  local process ID
- unsigned int **numProc** ()
  number of parallel processes
- template<class T> int **linearSize** (const T &inputT)
- template<class T> void **linearIn** (T &a_outputT, const void *const inBuf)
- template<class T> void **linearOut** (void *const a_outBuf, const T &inputT)
- template<class T> void **broadcast** (T &a_inAndOut, int a_src)
  broadcast to every process

- template<> int **linearSize** (const int &a_input)
- template<> void **linearIn** (int &a_outputT, const void ∗const inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const int &a_inputT)
- template<> int **linearSize** (const **Real** &a_input)
- template<> void **linearIn** (**Real** &a_outputT, const void ∗const a_inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const **Real** &a_inputT)
- template<> int **linearSize** (const **Box** &a_input)
- template<> void **linearIn** (**Box** &a_outputT, const void ∗const a_inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const **Box** &a_inputT)
- template<> int **linearSize** (const **Vector**< int > &a_input)
- template<> void **linearIn** (**Vector**< int > &a_outputT, const void ∗const inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const **Vector**< int > &a_inputT)
- template<> int **linearSize** (const **Vector**< **Real** > &a_input)
- template<> void **linearIn** (**Vector**< **Real** > &a_outputT, const void ∗const inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const **Vector**< **Real** > &a_inputT)
- template<> int **linearSize** (const **Vector**< **Box** > &a_input)
- template<> void **linearIn** (**Vector**< **Box** > &a_outputT, const void ∗const inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const **Vector**< **Box** > &a_inputT)
- template<> int **linearSize** (const **Vector**< **Vector**< **Box** > > &a_input)
- template<> void **linearIn** (**Vector**< **Vector**< **Box** > > &a_outputT, const void ∗const inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const **Vector**< **Vector**< **Box** > > &a_inputT)
- template<> int **linearSize** (const **RealVect** &vindex)
- template<> void **linearIn** (**RealVect** &a_outputT, const void ∗const inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const **RealVect** &a_inputT)
- template<> int **linearSize** (const **Vector**< **Vector**< int > > &a_input)
- template<> void **linearIn** (**Vector**< **Vector**< int > > &a_outputT, const void ∗const inBuf)
- template<> void **linearOut** (void ∗const a_outBuf, const **Vector**< **Vector**< int > > &a_inputT)
- template<class T> int **linearListSize** (const **Vector**< T > &a_input)
- template<class T> void **linearListIn** (**Vector**< T > &a_outputT, const void ∗const a_inBuf)
- template<class T> void **linearListOut** (void ∗const a_outBuf, const **Vector**< T > &a_inputT)
- int **uniqueProc** (const **SerialTask::task** &a_task)

Variables

- int **num_procs**

8.148.1 Function Documentation

8.148.1.1 template<class T> void broadcast (T & *a_inAndOut*, int *a_src*) `[inline]`

broadcast to every process

Broadcasts a_inAndOut to every process from process number a_src. It is an error if a_src < 0 or a_src >= **numProc()**. \

T must have have the following functions: \

int **linearSize<T>(const T& inputT)**; \ void **linearIn<T>(T& a_outputT, const void∗ const inBuf)**; \ void linearOut<T>(void∗ a_outBuf, const T& inputT); \

8.148.1.2 template<> void linearIn (Vector< Vector< int > > & *a_outputT*, const void *const *inBuf*)

8.148.1.3 template<> void linearIn (RealVect & *a_outputT*, const void *const *inBuf*)

8.148.1.4 template<> void linearIn (Vector< Vector< Box > > & *a_outputT*, const void *const *inBuf*)

8.148.1.5 template<> void linearIn (Vector< Box > & *a_outputT*, const void *const *inBuf*)

8.148.1.6 template<> void linearIn (Vector< Real > & *a_outputT*, const void *const *inBuf*)

8.148.1.7 template<> void linearIn (Vector< int > & *a_outputT*, const void *const *inBuf*)

8.148.1.8 template<> void linearIn (Box & *a_outputT*, const void *const *a_inBuf*)

8.148.1.9 template<> void linearIn (Real & *a_outputT*, const void *const *a_inBuf*)

8.148.1.10 template<> void linearIn (int & *a_outputT*, const void *const *inBuf*)

8.148.1.11 template<class T> void linearIn (T & *a_outputT*, const void *const *inBuf*)

8.148.1.12 template<class T> void linearListIn (Vector< T > & *a_outputT*, const void *const *a_inBuf*)

8.148.1.13 template<class T> void linearListOut (void *const *a_outBuf*, const Vector< T > & *a_inputT*)

8.148.1.14 template<class T> int linearListSize (const Vector< T > & *a_input*)

8.148.1.15 template<> void linearOut (void *const *a_outBuf*, const Vector< Vector< int > > & *a_inputT*)

8.148.1.16 template<> void linearOut (void *const *a_outBuf*, const RealVect & *a_inputT*)

8.148.1.17 template<> void linearOut (void *const *a_outBuf*, const Vector< Vector< Box > > & *a_inputT*)

8.148.1.18 template<> void linearOut (void *const *a_outBuf*, const Vector< Box > & *a_inputT*)

8.148.1.19 template<> void linearOut (void *const *a_outBuf*, const Vector< Real > & *a_inputT*)

8.148.1.20 template<> void linearOut (void *const *a_outBuf*, const Vector< int > & *a_inputT*)

8.148.1.21 template<> void linearOut (void *const *a_outBuf*, const Box & *a_inputT*)

8.148.1.22 template<> void linearOut (void *const *a_outBuf*, const Real & *a_inputT*)

8.148.1.23 template<> void linearOut (void *const *a_outBuf*, const int & *a_inputT*)

8.148.1.24 template<class T> void linearOut (void *const *a_outBuf*, const T &

Returns the number of parallel processes running. Always returns at least 1.

8.148.1.36 int procID ()

local process ID

Returns the ID of the locally running process in the range 0 <= **procID()** < **numProc()**. This has no relation to the operating system pid. There is always a **procID()** == 0.

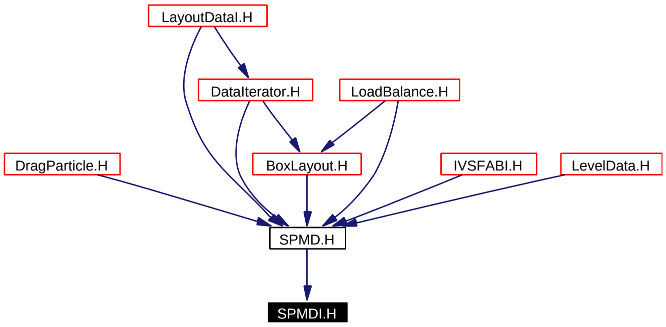
8.148.1.37 int uniqueProc (const SerialTask::task & *a_task*)

8.148.2 Variable Documentation

8.148.2.1 int num_procs

8.149 SPMDI.H File Reference

This graph shows which files directly or indirectly include this file:



Functions

- template<class T> int **linearSize** (const T &inputT)
- template<class T> void **linearIn** (T &a_outputT, const void *const inBuf)
- template<class T> void **linearOut** (void *const a_outBuf, const T &inputT)
- template<class T> void **gather** (**Vector**< T > &a_outVec, const T &a_input, int a_dest)
- template<class T> void **broadcast** (T &a_inAndOut, int a_src)

  broadcast to every process

- template<class T> void **linearListIn** (**Vector**< T > &a_outputT, const void *const a_inBuf)
- template<class T> void **linearListOut** (void *const a_outBuf, const **Vector**< T > &a_input)
- template<class T> int **linearListSize** (const **Vector**< T > &a_input)

8.149.1 Function Documentation

8.149.1.1 template<class T> void broadcast (T & *a_inAndOut*, int *a_src*) [inline]

broadcast to every process

Broadcasts a_inAndOut to every process from process number a_src. It is an error if a_src < 0 or a_src >= **numProc()**. \

T must have have the following functions: \

int **linearSize<T>(const T& inputT)**; \ void **linearIn<T>(T& a_outputT, const void* const inBuf)**; \ void linearOut<T>(void* a_outBuf, const T& inputT); \

8.149.1.2 template<class T> void gather (Vector< T > & *a_outVec*, const T & *a_input*, int *a_dest*) [inline]

8.149.1.3 template<class T> void linearIn (T & *a_outputT*, const void *const *inBuf*)

8.149.1.4 template<class T> void linearListIn (Vector< T > & *a_outputT*, const void *const *a_inBuf*)

8.149.1.5 template<class T> void linearListOut (void *const *a_outBuf*, const Vector< T > & *a_input*)

8.149.1.6 template<class T> int linearListSize (const Vector< T > & *a_input*)

8.149.1.7 template<class T> void linearOut (void *const *a_outBuf*, const T & *inputT*)

8.149.1.8 template<class T> int linearSize (const T & *inputT*)

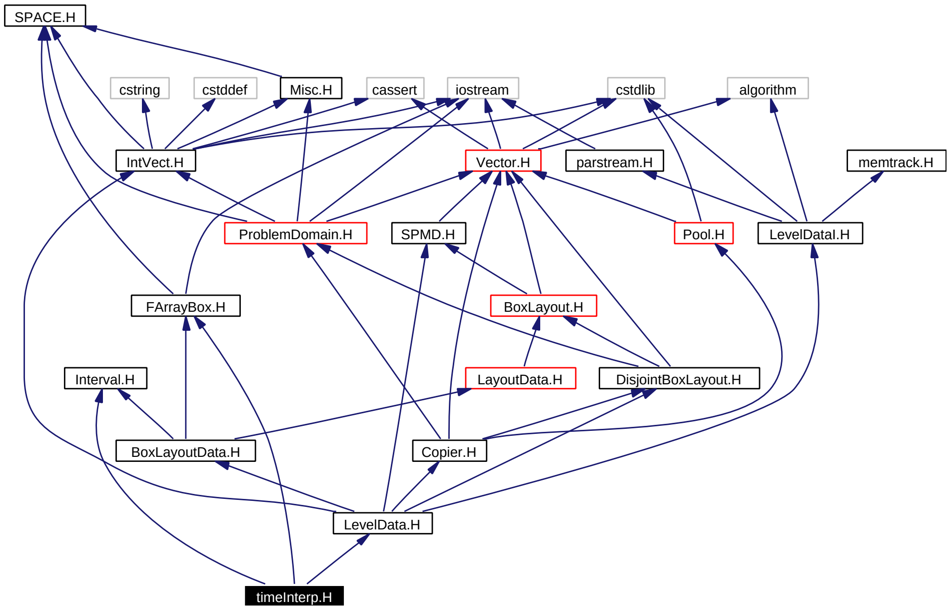
8.150 timeInterp.H File Reference

`#include "LevelData.H"`

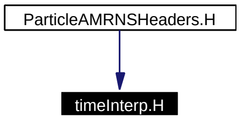
`#include "FArrayBox.H"`

`#include "Interval.H"`

Include dependency graph for timeInterp.H:



This graph shows which files directly or indirectly include this file:



Functions

- void **timeInterp** (**LevelData**< **FArrayBox** > &a_dest, **Real** a_time, const **LevelData**< **FArrayBox** > &a_old_phi, **Real** a_old_time, const **LevelData**< **FArrayBox** > &a_new_phi, **Real** a_new_time, const **Interval** &a_comps)

  interpolate data in time – assumes one-to-one correspondence in comps

- void **timeInterp** (**LevelData**< **FArrayBox** > &a_dest, **Real** a_time, const **LevelData**< **FArrayBox** > &a_old_phi, **Real** a_old_time, const **LevelData**< **FArrayBox** > &a_new_phi, **Real** a_new_time, const **Interval** &a_src_comps, const **Interval** &a_dest_comps)

  interpolate data in time – dest comps can be misaligned with src comps

8.150.1 Function Documentation

8.150.1.1 void timeInterp (LevelData< FArrayBox > & *a_dest*, Real *a_time*, const LevelData< FArrayBox > & *a_old_phi*, Real *a_old_time*, const LevelData< FArrayBox > & *a_new_phi*, Real *a_new_time*, const Interval & *a_src_comps*, const Interval & *a_dest_comps*)

interpolate data in time – dest comps can be misaligned with src comps

8.150.1.2 void timeInterp (LevelData< FArrayBox > & *a_dest*, Real *a_time*, const LevelData< FArrayBox > & *a_old_phi*, Real *a_old_time*, const LevelData< FArrayBox > & *a_new_phi*, Real *a_new_time*, const Interval & *a_comps*)

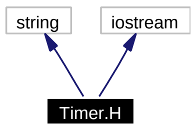
interpolate data in time – assumes one-to-one correspondence in comps

8.151 Timer.H File Reference

`#include <string>`

`#include <iostream>`

Include dependency graph for Timer.H:



Namespaces

- namespace **std**

Compounds

- class **Timer**

Defines

- #define **DIAG_TABLE1** 1
- #define **DIAG_TABLE2** 2
- #define **DIAG_TABLE3** 3
- #define **DIAG_TABLE4** 4

8.151.1 Define Documentation

8.151.1.1 #define DIAG_TABLE1 1

8.151.1.2 #define DIAG_TABLE2 2

8.151.1.3 #define DIAG_TABLE3 3

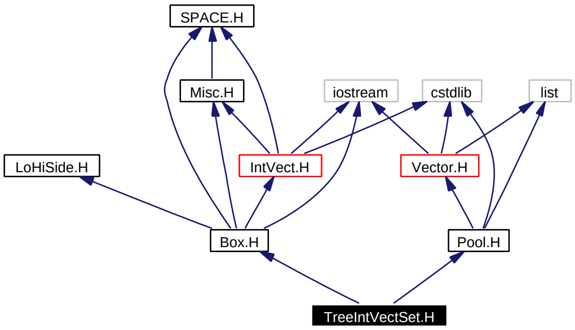
8.151.1.4 #define DIAG_TABLE4 4

8.152 TreeIntVectSet.H File Reference

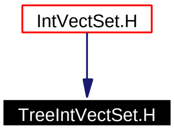
`#include "Box.H"`

`#include "Pool.H"`

Include dependency graph for TreeIntVectSet.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **TreeIntVectSet**

  IntVectSet *implementation based on tree representation.*

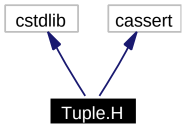
- struct **TreeIntVectSet::TreeNode**
- class **TreeIntVectSetIterator**

8.153 Tuple.H File Reference

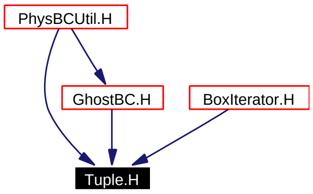
`#include <cstdlib>`

`#include <cassert>`

Include dependency graph for Tuple.H:



This graph shows which files directly or indirectly include this file:



Compounds

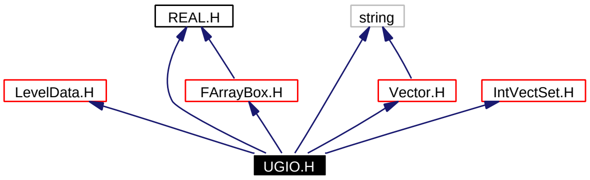
- class **Tuple**

  Ordered Tuples for Types T.

8.154 UGIO.H File Reference

```
#include "LevelData.H"
#include "REAL.H"
#include "Vector.H"
#include "FArrayBox.H"
#include "IntVectSet.H"
#include <string>
```

Include dependency graph for UGIO.H:



8.155 Vector.H File Reference

`#include <cmath>`

`#include <cstdlib>`

`#include <cassert>`

`#include <vector>`

`#include <typeinfo>`

`#include <string>`

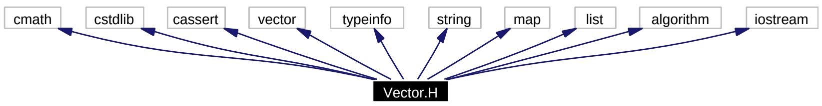
`#include <map>`

`#include <list>`

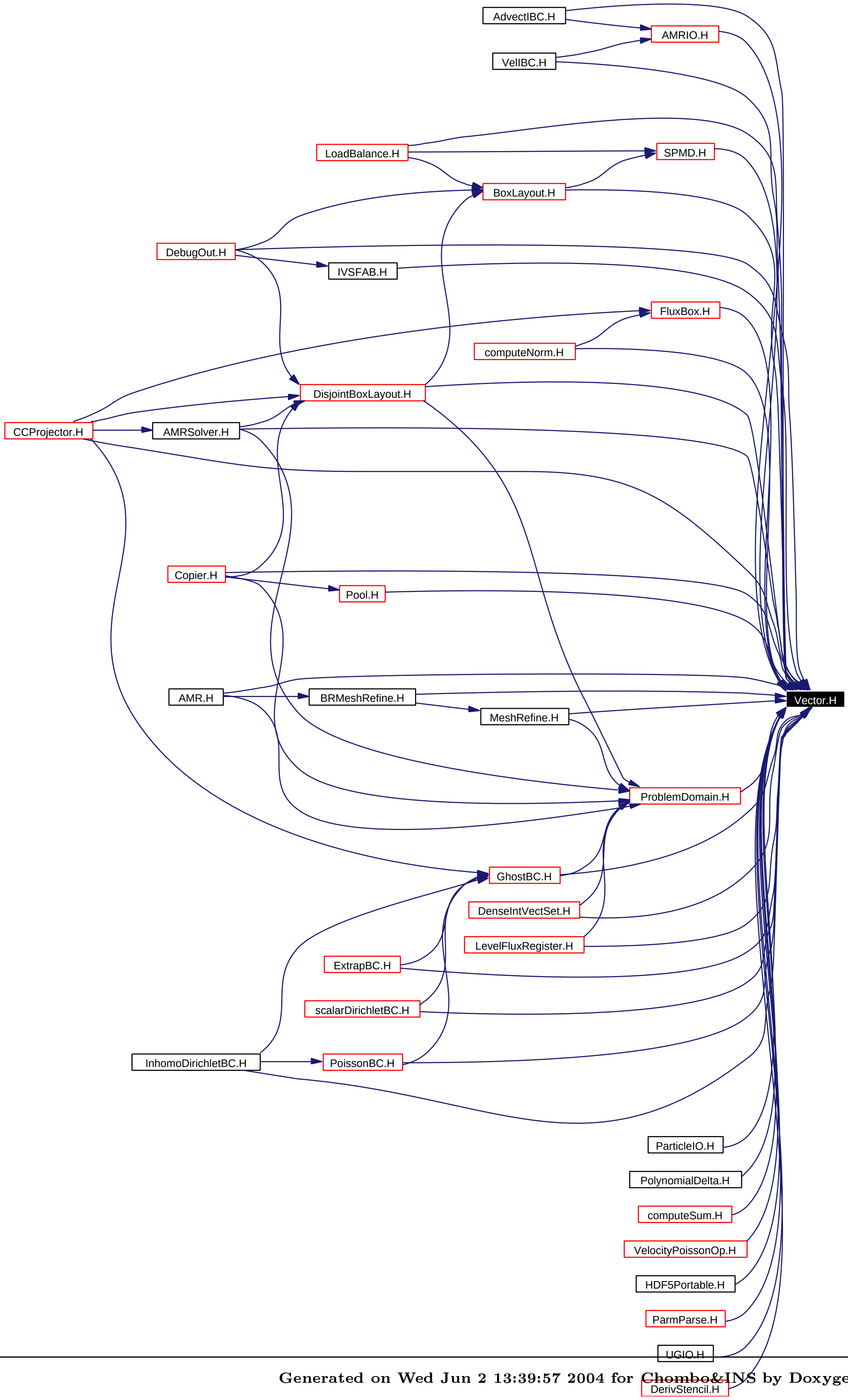
`#include <algorithm>`

`#include <iostream>`

Include dependency graph for Vector.H:



This graph shows which files directly or indirectly include this file:



AdvectIBC.H
AMRIO.H
VelIBC.H
LoadBalance.H
SPMD.H
BoxLayout.H
DebugOut.H
IVSFAB.H
FluxBox.H
computeNorm.H
DisjointBoxLayout.H
CCProjector.H
AMRSolver.H
Copier.H
Pool.H
AMR.H
BRMeshRefine.H
Vector.H
MeshRefine.H
ProblemDomain.H
GhostBC.H
DenseIntVectSet.H
LevelFluxRegister.H
ExtrapBC.H
scalarDirichletBC.H
InhomoDirichletBC.H
PoissonBC.H
ParticleIO.H
PolynomialDelta.H
computeSum.H
VelocityPoissonOp.H
HDF5Portable.H
ParmParse.H
UGIO.H
DerivStencil.H

Compounds

- class **Vector**

  one dimensional dynamic array

Functions

- template<class T> ostream & **operator**<< (ostream &os, const **Vector**< T > &vec)

8.155.1 Function Documentation

8.155.1.1 template<class T> ostream& operator<< (ostream & *os*, const Vector< T > & *vec*)

8.156 VelIBC.H File Reference

`#include <iostream>`

`#include "LevelData.H"`

`#include "FArrayBox.H"`

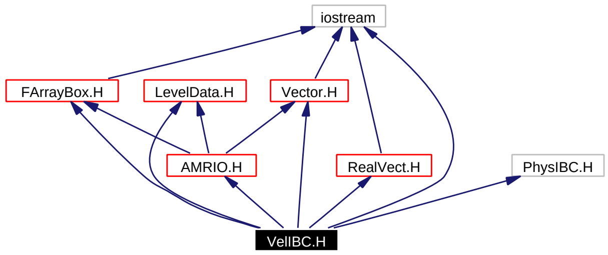
`#include "Vector.H"`

`#include "RealVect.H"`

`#include "AMRIO.H"`

`#include "PhysIBC.H"`

Include dependency graph for VelIBC.H:



Compounds

- class **VelIBC**

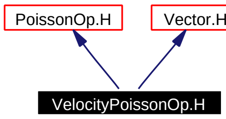
  IBC for simple Velocity advection (solid wall) boundary conditions.

8.157 VelocityPoissonOp.H File Reference

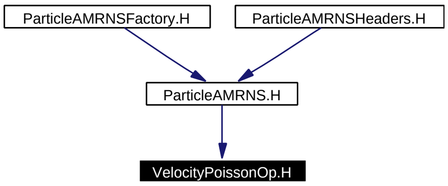
`#include "PoissonOp.H"`

`#include "Vector.H"`

Include dependency graph for VelocityPoissonOp.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **VelocityPoissonOp**

8.158 viscousBC.H File Reference

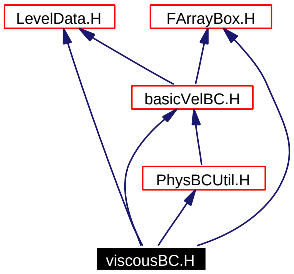
`#include <LevelData.H>`

`#include <FArrayBox.H>`

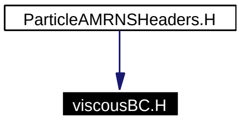
`#include "PhysBCUtil.H"`

`#include "basicVelBC.H"`

Include dependency graph for viscousBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **viscousBC**

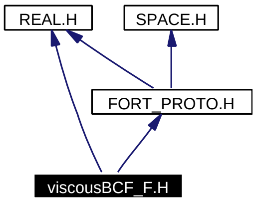
  implements boundary conditions on viscous terms themselves

8.159 viscousBCF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for viscousBCF_F.H:



Defines

- #define **FORT_VISCOUSBC** FORTRAN_NAME(VISCOUSBC ,viscousbc)

Functions

- void **FORT_VISCOUSBC** (CHFp_FRA(viscous),CHFp_BOX(gridBox),CHFp_BOX(loBox),CHFp_BOX(hiBox),CHFp_INT(dir),CHFp_INT(nComp))

8.159.1 Define Documentation

8.159.1.1 #define FORT_VISCOUSBC FORTRAN_NAME(VISCOUSBC ,viscousbc)

8.159.2 Function Documentation

8.159.2.1 void FORT_VISCOUSBC (CHFp_FRA(viscous), CHFp_BOX(gridBox), CHFp_BOX(loBox), CHFp_BOX(hiBox), CHFp_INT(dir), CHFp_INT(nComp))