Chombo&INS Reference Manual

1.2

Generated by Doxygen 1.2.16

Thu Jun 27 14:38:53 2002

Contents

Chapter 1

Chombo&INS Namespace Index

1.1 Chombo&INS Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Chombo&INS Hierarchical Index

2.1 Chombo&INS Class Hierarchy

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

Chapter 3

Chombo&INS Compound Index

3.1 Chombo&INS Compound List

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

Chapter 4

Chombo&INS File Index

4.1 Chombo&INS File List

Here is a list of all files with brief descriptions:

Chapter 5

Chombo&INS Namespace Documentation

5.1 std Namespace Reference

Chapter 6

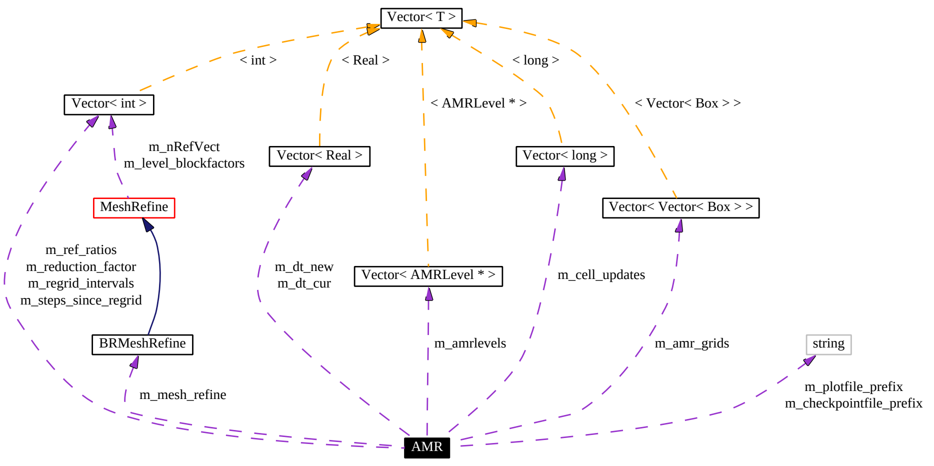
Chombo&INS Class Documentation

6.1 AMR Class Reference

framework for Berger-Oliger timestepping for AMR

`#include <AMR.H>`

Collaboration diagram for AMR:



Public Methods

- **AMR** ()

  {\bf *constructors, destructors and defines*}

- ~**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

  {\bf Setup functions}

- 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)

  {\bf Administrative functions}

- 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 **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
- void **verbosity** (int a_verbosity)
- void **regridIntervals** (const **Vector**< int > &a_regridIntervals)
- int **verbosity** () const
- void **maxDtGrow** (**Real** a_dtGrowFactor)
- **Real maxDtGrow** () const
- void **fixedDt** (**Real** a_dt)
- **Real fixedDt** () const

Protected Methods

- int **timeStep** (int a_level, int a_stepsLeft, bool timeBoundary)
- 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**
- **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**
- **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**

6.1.1 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 {\tt **AMRLevel** (p. 33)}s. 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) \ 1.5. Modify any parameters you like (blocking factor and so forth using access functions.\ 2. Call any one of the three setup functions so AMR can set up all its internal data structures.\ 3. Call run to run the calculation.\ 4. Call conclude to output (to stdout) how many cells were updated and so forth.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 AMR::AMR ()

{\bf constructors, destructors and defines}

Default constructor. Use must subsequently call a **define**() (p. 20) function and a setUp function.

6.1.2.2 AMR::~AMR ()

Destructor.

6.1.3 Member Function Documentation

6.1.3.1 void AMR::assignDt () [protected]

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

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

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

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

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

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

6.1.3.5 void AMR::clearMemory () [protected]

6.1.3.6 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.

6.1.3.7 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. {\bf 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 AMRLevels in this AMR as objects of the physics class type. It can be destructed after this **define**() (p. 20) returns.

6.1.3.8 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. {\bf 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 AMRLevels in this AMR as objects of the physics class type. It can be destructed after this **define**() (p. 20) returns.

6.1.3.9 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).

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

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

6.1.3.11 Real AMR::fixedDt () const

Returns the fixed timestep.

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

Set a fixed timestep.

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

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

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

6.1.3.15 bool AMR::isDefined () const

{\bf Setup functions}

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

6.1.3.16 bool AMR::isSetUp () const

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

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

6.1.3.18 Real AMR::maxDtGrow () const

Returns current maximum factor by which a timestep can grow.

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

Set maximum factor by which a timestep can grow.

6.1.3.20 int AMR::maxGridSize () const

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

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

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

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

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

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

{\bf Administrative functions}

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

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

6.1.3.26 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.

6.1.3.27 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.

6.1.3.28 void AMR::setDefaultValues () `[protected]`

6.1.3.29 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;

6.1.3.30 void AMR::setupForNewAMRRun ()

Sets up this object for cold start. User must have previously called **define**() (p. 20). Need to call this or setupForRestart or setupforFixedHierarchyRun before you run.

6.1.3.31 int AMR::timeStep (int *a_level*, int *a_stepsLeft*, bool *timeBoundary*) [protected]

6.1.3.32 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.

6.1.3.33 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.

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

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

6.1.4 Member Data Documentation

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

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

6.1.4.3 int AMR::m_blockFactor [protected]

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

6.1.4.5 int AMR::m_checkpoint_interval [protected]

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

6.1.4.7 int AMR::m_cur_step [protected]

6.1.4.8 Real AMR::m_cur_time [protected]

6.1.4.9 Real AMR::m_dt_base [protected]

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

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

6.1.4.12 Real AMR::m_dt_tolerance_factor [protected]

6.1.4.13 Real AMR::m_fillRatio [protected]

6.1.4.14 int AMR::m_finest_level [protected]

6.1.4.15 int AMR::m_finest_level_old [protected]

6.1.4.16 Real AMR::m_fixedDt [protected]

6.1.4.17 bool AMR::m_isDefined [protected]

6.1.4.18 bool AMR::m_isSetUp [protected]

6.1.4.19 int AMR::m_lastcheck_step [protected]

6.1.4.20 int AMR::m_max_grid_size [protected]

6.1.4.21 int AMR::m_max_level [protected]

6.1.4.22 Real AMR::m_maxDtGrow [protected]

6.1.4.23 BRMeshRefine AMR::m_mesh_refine [protected]

6.1.4.24 int AMR::m_plot_interval [protected]

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

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

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

6.1.4.28 Vector<int> AMR::m_regrid_intervals [protected]

6.1.4.29 int AMR::m_restart_step [protected]

6.1.4.30 Vector<int> AMR::m_steps_since_regrid [protected]

6.1.4.31 bool AMR::m_use_meshrefine [protected]

6.1.4.32 int AMR::m_verbosity [protected]

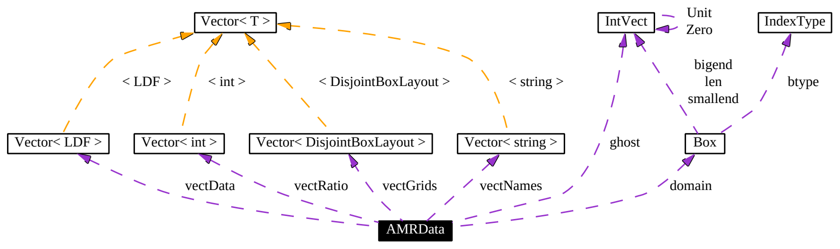
The documentation for this class was generated from the following file:

- **AMR.H**

6.2 AMRData Class Reference

`#include <AMRIO.H>`

Collaboration diagram for AMRData:



Public Methods

- **AMRData** ()

Public Attributes

- **Vector**< **DisjointBoxLayout** > **vectGrids**
- **Vector**< **LDF** > **vectData**
- **Vector**< string > **vectNames**
- **Box domain**
- **Real dx**
- **Real dt**
- **Real time**
- **Vector**< int > **vectRatio**
- int **numLevels**
- **IntVect ghost**

6.2.1 Constructor & Destructor Documentation

6.2.1.1 AMRData::AMRData () `[inline]`

6.2.2 Member Data Documentation

6.2.2.1 Box AMRData::domain

6.2.2.2 Real AMRData::dt

6.2.2.3 Real AMRData::dx

6.2.2.4 IntVect AMRData::ghost

6.2.2.5 int AMRData::numLevels

6.2.2.6 Real AMRData::time

6.2.2.7 Vector< LDF > AMRData::vectData

6.2.2.8 Vector< DisjointBoxLayout > AMRData::vectGrids

6.2.2.9 Vector< string > AMRData::vectNames

6.2.2.10 Vector< int > AMRData::vectRatio

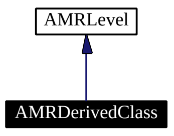
The documentation for this class was generated from the following file:

- **AMRIO.H**

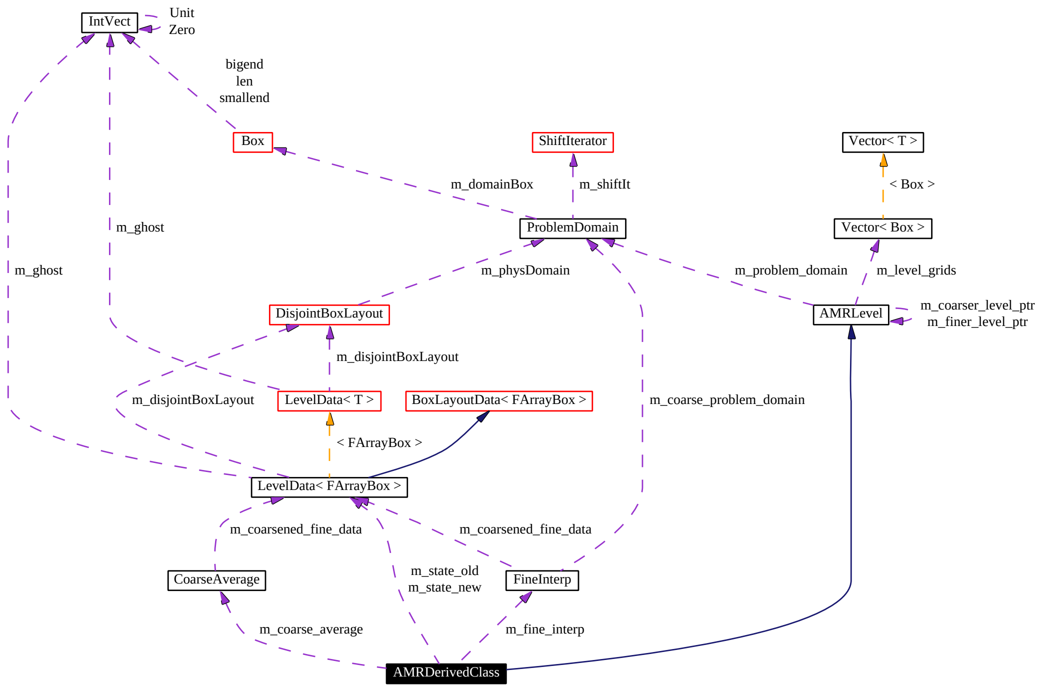
6.3 AMRDerivedClass Class Reference

`#include <AMRDerivedClass.H>`

Inheritance diagram for AMRDerivedClass:



Collaboration diagram for AMRDerivedClass:



Public Methods

- **AMRDerivedClass** ()
- virtual ~**AMRDerivedClass** ()
- 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 **Real advance** ()
- virtual void **postTimeStep** ()
- virtual void **tagCells** (**IntVectSet** &a_tags)
- virtual void **tagCellsInit** (**IntVectSet** &a_tags)
- virtual void **regrid** (const **Vector**< **Box** > &a_new_grids)
- virtual void **initialGrid** (const **Vector**< **Box** > &a_new_grids)

- virtual void **initialData** ()
- virtual void **postInitialize** ()
- virtual void **regridData** ()
- virtual **Real computeDt** ()
- virtual **Real computeInitialDt** ()

Protected Methods

- **DisjointBoxLayout loadBalance** (const **Vector**< **Box** > &a_grids)

Protected Attributes

- **LevelData< FArrayBox > m_state_old**
- **LevelData< FArrayBox > m_state_new**
- **Real m_dx**
- **CoarseAverage m_coarse_average**
- **FineInterp m_fine_interp**

Static Protected Attributes

- const int **s_num_comps** = 2
- const char ∗ **s_state_names** [**s_num_comps**]

6.3.1 Constructor & Destructor Documentation

6.3.1.1 AMRDerivedClass::AMRDerivedClass ()

6.3.1.2 virtual AMRDerivedClass::~AMRDerivedClass () `[virtual]`

6.3.2 Member Function Documentation

6.3.2.1 virtual Real AMRDerivedClass::advance () `[virtual]`

6.3.2.2 virtual Real AMRDerivedClass::computeDt () `[virtual]`

Returns maximum stable time step for this level.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 35).

6.3.2.3 virtual Real AMRDerivedClass::computeInitialDt () `[virtual]`

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

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 35).

6.3.2.4 virtual void AMRDerivedClass::define (AMRLevel * *a_coarser_level_ptr*, const ProblemDomain & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) [virtual]

Defines this **AMRLevel** (p. 33).

{\bf Arguments:}\ 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. 35).

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

Defines this **AMRLevel** (p. 33).

{\bf Arguments:}\ 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. 35).

6.3.2.6 virtual void AMRDerivedClass::initialData () [virtual]

Initializes data.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 36).

6.3.2.7 virtual void AMRDerivedClass::initialGrid (const Vector< Box > & *a_new_grids*) [virtual]

Initializes this level to have the specified domain a_new_grids.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 36).

6.3.2.8 DisjointBoxLayout AMRDerivedClass::loadBalance (const Vector< Box > & *a_grids*) [protected]

6.3.2.9 virtual void AMRDerivedClass::postInitialize () [virtual]

Things to do after initialization.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 37).

6.3.2.10 virtual void AMRDerivedClass::postTimeStep () [virtual]

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

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 37).

6.3.2.11 virtual void AMRDerivedClass::regrid (const Vector< Box > & *a_new_grids*) [virtual]

Redefines this level to have the specified domain a_new_grids.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 37).

6.3.2.12 virtual void AMRDerivedClass::regridData () [virtual]

6.3.2.13 virtual void AMRDerivedClass::tagCells (IntVectSet & *a_tags*) [virtual]

Creates tagged cells for dynamic mesh refinement.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 37).

6.3.2.14 virtual void AMRDerivedClass::tagCellsInit (IntVectSet & *a_tags*) [virtual]

Creates tagged cells for mesh refinement at initialization.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implements **AMRLevel** (p. 37).

6.3.3 Member Data Documentation

6.3.3.1 CoarseAverage AMRDerivedClass::m_coarse_average [protected]

6.3.3.2 Real AMRDerivedClass::m_dx [protected]

6.3.3.3 FineInterp AMRDerivedClass::m_fine_interp [protected]

6.3.3.4 LevelData<FArrayBox> AMRDerivedClass::m_state_new [protected]

6.3.3.5 LevelData<FArrayBox> AMRDerivedClass::m_state_old [protected]

6.3.3.6 const int AMRDerivedClass::s_num_comps = 2 [static, protected]

6.3.3.7 const char* AMRDerivedClass::s_state_names[s_num_comps] [static, protected]

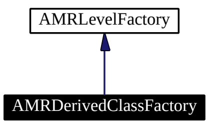
The documentation for this class was generated from the following file:

- **AMRDerivedClass.H**

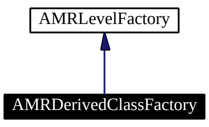
6.4 AMRDerivedClassFactory Class Reference

`#include <AMRDerivedClassFactory.H>`

Inheritance diagram for AMRDerivedClassFactory:



Collaboration diagram for AMRDerivedClassFactory:



Public Methods

- **AMRDerivedClassFactory** ()
- virtual **AMRLevel** ∗ **new_amrlevel** () const
- virtual ∼**AMRDerivedClassFactory** ()

6.4.1 Constructor & Destructor Documentation

6.4.1.1 AMRDerivedClassFactory::AMRDerivedClassFactory ()

6.4.1.2 virtual AMRDerivedClassFactory::∼AMRDerivedClassFactory () [virtual]

6.4.2 Member Function Documentation

6.4.2.1 virtual AMRLevel∗ AMRDerivedClassFactory::new_amrlevel () const [virtual]

Factory function. creates a new **AMRLevel** (p. 33) object. In your instantiation you should include defining all the stuff that is not part of the **AMRLevel** (p. 33) API.

Implements **AMRLevelFactory** (p. 39).

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

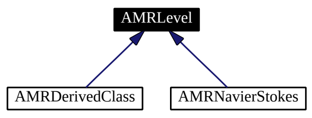
- **AMRDerivedClassFactory.H**

6.5 AMRLevel Class Reference

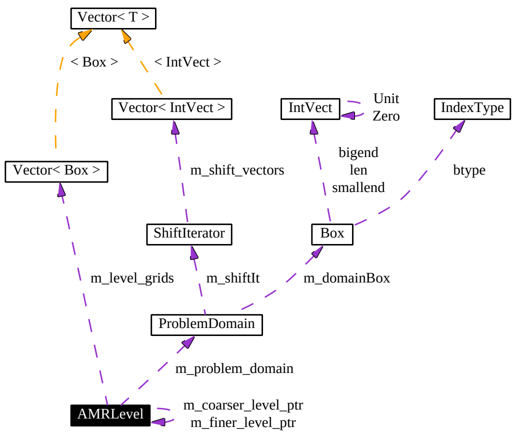
abstract base class for time-dependent data at a level of refinement.

`#include <AMRLevel.H>`

Inheritance diagram for AMRLevel:



Collaboration diagram for AMRLevel:



Public Methods

- Default constructor ∗ **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)
- Sets the pointer to finer level member to a_finer_level_ptr ∗virtual void **finerLevelPtr** (AMRLevel ∗a_finer_level_ptr)
- virtual void **dt** (**Real** a_dt)
- virtual void **time** (**Real** a_time)
- virtual **Real dt** () const

  {\bf *Access functions*}

- virtual **Real time** () const
- virtual void **initialDtMultiplier** (**Real** a_initial_dt_multiplier)
- virtual **Real initialDtMultiplier** () const

  {\bf *Access functions*}

- virtual const **ProblemDomain** & **problemDomain** () const
- virtual **Vector**< **Box** > **boxes** () const
- bool **isDefined** () const
- virtual int **refRatio** () const
- 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 Methods

- 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**

6.5.1 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.

{\tt AMRLevel} is intimately entangled with the class {\tt **AMR** (p. 17)}. The {\tt **AMR** (p. 17)} contains a collection of {\tt AMRLevel}s which represent a hierarchy of levels of refinement. The class {\tt **AMR** (p. 17)} is responsible for calling the correct member functions of {\tt AMRLevel}. The user is responsible for implementing the physics class, and might never call any {\tt AMRLevel} functions described here.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Default constructor* AMRLevel::AMRLevel ()

{\bf constructors, destructor and defines}

6.5.2.2 virtual AMRLevel::~AMRLevel () `[virtual]`

Destructor.

6.5.3 Member Function Documentation

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

Returns the domain of this level.

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

Returns maximum stable time step for this level.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 55), and **AMRDerivedClass** (p. 29).

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

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

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 55), and **AMRDerivedClass** (p. 29).

6.5.3.4 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.

{\bf Arguments:}\ 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 **AMRNavierStokes** (p. 56), and **AMRDerivedClass** (p. 30).

6.5.3.5 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.

{\bf Arguments:}\ 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 **AMRNavierStokes** (p. 56), and **AMRDerivedClass** (p. 30).

6.5.3.6 virtual Real AMRLevel::dt () const [virtual]

{\bf Access functions}

Returns the current value of the time step.

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

Sets the time step to a_dt.

6.5.3.8 Sets the pointer to finer level member to a_finer_level_ptr∗ virtual void AMRLevel::finerLevelPtr (AMRLevel ∗ *a_finer_level_ptr*) [virtual]

{\bf Modification functions}

6.5.3.9 virtual void AMRLevel::initialData () [pure virtual]

Initializes data.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 57), and **AMRDerivedClass** (p. 30).

6.5.3.10 virtual Real AMRLevel::initialDtMultiplier () const [virtual]

{\bf Access functions}

Returns the initial dt multiplier.

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

Sets the initial dt multiplier to a_initial_dt_multiplier.

6.5.3.12 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 {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 57), and **AMRDerivedClass** (p. 30).

6.5.3.13 bool AMRLevel::isDefined () const

Returns true if any **AMRLevel::define** (p. 35) function has been called, false otherwise.

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

Things to do after initialization.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 59), and **AMRDerivedClass** (p. 30).

6.5.3.15 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**() (p. 37) instantiation is a no-op.

Reimplemented in **AMRNavierStokes** (p. 59).

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

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

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 59), and **AMRDerivedClass** (p. 30).

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

Returns the problem domain of this level.

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

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

6.5.3.19 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 {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 60), and **AMRDerivedClass** (p. 31).

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

Creates tagged cells for dynamic mesh refinement.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 60), and **AMRDerivedClass** (p. 31).

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

Creates tagged cells for mesh refinement at initialization.

This is a pure virtual function and {\bf MUST} be defined in the derived class.

Implemented in **AMRNavierStokes** (p. 60), and **AMRDerivedClass** (p. 31).

6.5.3.22 virtual Real AMRLevel::time () const `[virtual]`

Returns the current value of the time on this level virtual

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

Sets the time to a_time.

6.5.3.24 int AMRLevel::verbosity () `[static]`

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

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

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

6.5.4 Member Data Documentation

6.5.4.1 AMRLevel* AMRLevel::m_coarser_level_ptr `[protected]`

6.5.4.2 Real AMRLevel::m_dt `[protected]`

6.5.4.3 AMRLevel* AMRLevel::m_finer_level_ptr `[protected]`

6.5.4.4 Real AMRLevel::m_initial_dt_multiplier `[protected]`

6.5.4.5 bool AMRLevel::m_isDefined `[protected]`

6.5.4.6 int AMRLevel::m_level `[protected]`

6.5.4.7 Vector<Box> AMRLevel::m_level_grids `[protected]`

6.5.4.8 ProblemDomain AMRLevel::m_problem_domain `[protected]`

6.5.4.9 int AMRLevel::m_ref_ratio `[protected]`

6.5.4.10 Real AMRLevel::m_time `[protected]`

6.5.4.11 int AMRLevel::s_verbosity `[static, protected]`

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

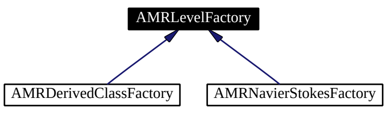
- **AMRLevel.H**

6.6 AMRLevelFactory Class Reference

Factory class to produce BaseEBFaceFABs.

`#include <AMRLevelFactory.H>`

Inheritance diagram for AMRLevelFactory:



Public Methods

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

6.6.1 Detailed Description

Factory class to produce BaseEBFaceFABs.

Factory class to produce BaseEBFaceFABs. This is needed for **LevelData** (p. 295) construction.

6.6.2 Member Function Documentation

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

Factory function. creates a new **AMRLevel** (p. 33) object. In your instantiation you should include defining all the stuff that is not part of the **AMRLevel** (p. 33) API.

Implemented in **AMRNavierStokesFactory** (p. 68), and **AMRDerivedClassFactory** (p. 32).

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

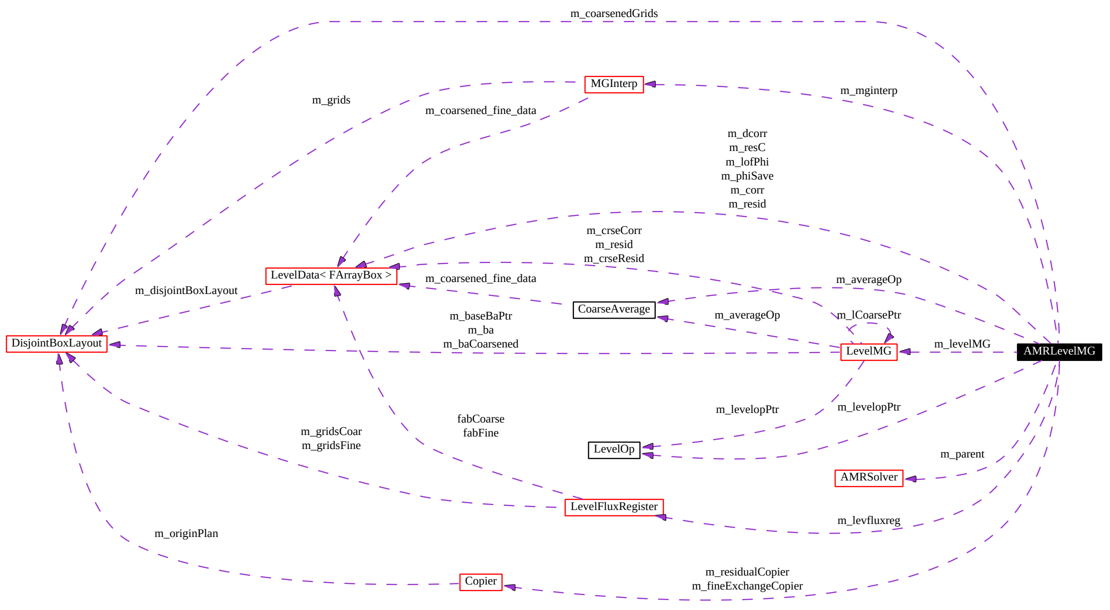
- **AMRLevelFactory.H**

6.7 AMRLevelMG Class Reference

class which manages data and operations on a level for **AMRSolver** (p. 69)

`#include <AMRLevelMG.H>`

Collaboration diagram for AMRLevelMG:



Public Methods

- **AMRLevelMG** ()
- ~**AMRLevelMG** ()
- **AMRLevelMG** (const **AMRSolver** ∗const a_parent, int a_level, const **LevelOp** ∗const a_opin, int ncomp=1)
- void **define** (const **AMRSolver** ∗const a_parent, int a_level, const **LevelOp** ∗const a_opin, int ncomp=1)
- void **setnumSmoothUp** (int a_numSmoothUp)
- void **setnumSmoothDown** (int a_numSmoothDown)
- void **applyAMROperator** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, **LevelData**< **FArrayBox** > &Lofphi)
- void **applyAMROperatorHphys** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, **LevelData**< **FArrayBox** > &LofPhi)
- void **computeAMRResidual** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)
- void **computeAMRResidualHphys** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)
- void **computeAMRResidual** (**LevelData**< **FArrayBox** > &a_resid, **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)
- void **upSweep** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)

- void **downSweep** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)
- **Vector**< **Real** > **computeResidualNorm** (int normType) const
- **LevelOp** ∗ **levelOpPtr** () const
- bool **isDefined** () const

Protected Methods

- **Vector**< **Real** > **computeNorm** (const **LevelData**< **FArrayBox** > &a_mfinput, int a_normType) const
- void **crseCorrect** (**LevelData**< **FArrayBox** > &a_soln, const **LevelData**< **FArrayBox** > &a_crseCorr, int a_nref)
- void **smooth** (**LevelData**< **FArrayBox** > &a_soln, const **LevelData**< **FArrayBox** > &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**

6.7.1 Detailed Description

class which manages data and operations on a level for **AMRSolver** (p. 69)

AMRLevelMG manages the operations on a level for **AMRSolver** (p. 69). **AMRSolver** (p. 69) contains a vector of AMRLevelMG's (one for each level). This class should be considered internal to **AMRSolver** (p. 69) and should not be considered part of the Chombo API.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 AMRLevelMG::AMRLevelMG ()

6.7.2.2 AMRLevelMG::~AMRLevelMG ()

6.7.2.3 AMRLevelMG::AMRLevelMG (const AMRSolver ∗const *a_parent*, int *a_level*, const LevelOp ∗const *a_opin*, int *ncomp* = 1)

Define AMRLevelMG from the inputs.

6.7.3 Member Function Documentation

6.7.3.1 void AMRLevelMG::applyAMROperator (Vector< LevelData< FArrayBox > ∗ > & *a_phiLevel*, LevelData< FArrayBox > & *Lofphi*)

Complete, 3-level operator. Applies operator with inhomogeneous coarse-fine boundary conditions (and inhomogeneous domain boundary conditions. Refluxing is used to compute the operator at coarse-fine boundaries.

6.7.3.2 void AMRLevelMG::applyAMROperatorHphys (Vector< LevelData< FArrayBox > ∗ > & *a_phiLevel*, LevelData< FArrayBox > & *LofPhi*)

Complete, 3-level operator with homogeneous physical boundary conditions. Applies operator with inhomogeneous physical boundary conditions. Refluxing is used to compute the operator at coarse-fine boundaries.

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

6.7.3.4 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 the argument **LevelData** (p. 295)<**FArrayBox** (p. 229)>

6.7.3.5 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

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

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

6.7.3.7 Vector<Real> AMRLevelMG::computeNorm (const LevelData< FArrayBox > & *a_mfinput*, int *a_normType*) const `[protected]`

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

compute and return norm of internal data m_resid.

6.7.3.9 void AMRLevelMG::crseCorrect (LevelData< FArrayBox > & *a_soln*, const LevelData< FArrayBox > & *a_crseCorr*, int *a_nref*) `[protected]`

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

Define AMRLevelMG from the inputs. The AMRLevelMG will only work for the input **AMRSolver** (p. 69). A **LevelMG** (p. 310) is created.

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

Sweep down v-cycle

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

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

6.7.3.14 bool AMRLevelMG::isDefined () const

6.7.3.15 LevelOp* AMRLevelMG::levelOpPtr () const

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

6.7.3.17 void AMRLevelMG::setDefaultValues () `[protected]`

6.7.3.18 void AMRLevelMG::setnumSmoothDown (int *a_numSmoothDown*)

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

6.7.3.19 void AMRLevelMG::setnumSmoothUp (int *a_numSmoothUp*)

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

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

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

Sweep up multigrid v-cycle. Computes residual on

6.7.4 Friends And Related Function Documentation

6.7.4.1 friend class AMRSolver [friend]

6.7.5 Member Data Documentation

6.7.5.1 bool AMRLevelMG::m_arrayViewVerbose [protected]

6.7.5.2 CoarseAverage AMRLevelMG::m_averageOp [protected]

6.7.5.3 DisjointBoxLayout AMRLevelMG::m_coarsenedGrids [protected]

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

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

6.7.5.6 Copier AMRLevelMG::m_fineExchangeCopier [protected]

6.7.5.7 bool AMRLevelMG::m_isDefined [protected]

6.7.5.8 int AMRLevelMG::m_level [protected]

6.7.5.9 LevelMG AMRLevelMG::m_levelMG [protected]

6.7.5.10 LevelOp* AMRLevelMG::m_levelopPtr [protected]

6.7.5.11 LevelFluxRegister AMRLevelMG::m_levfluxreg [protected]

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

6.7.5.13 MGInterp AMRLevelMG::m_mginterp [protected]

6.7.5.14 const AMRSolver* AMRLevelMG::m_parent [protected]

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

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

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

6.7.5.18 Copier AMRLevelMG::m_residualCopier [protected]

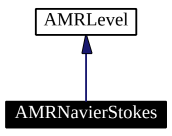
The documentation for this class was generated from the following file:

- **AMRLevelMG.H**

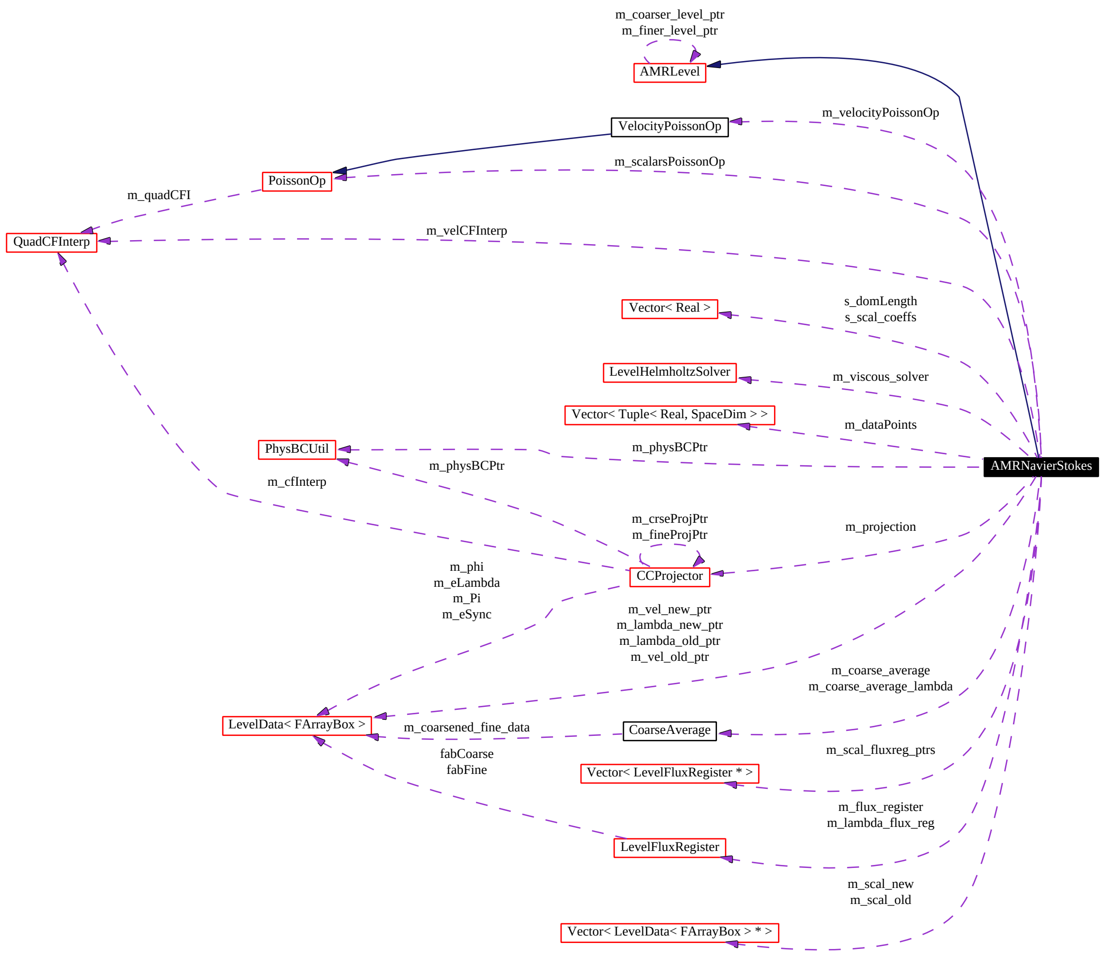
6.8 AMRNavierStokes Class Reference

`#include <AMRNavierStokes.H>`

Inheritance diagram for AMRNavierStokes:



Collaboration diagram for AMRNavierStokes:



Public Methods

- **AMRNavierStokes** ()
- **AMRNavierStokes** (**AMRLevel** ∗a_coarser_level_ptr, const **Box** &a_prob_domain, int a_level, int a_ref_ratio)

- **AMRNavierStokes** (**AMRLevel** *a_coarser_level_ptr, const **ProblemDomain** &a_problem_domain, int a_level, int a_ref_ratio)
- virtual ~**AMRNavierStokes** ()
- 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 **refinementThreshold** (**Real** a_refine_threshold)

  set refinement threshold

- bool **finestLevel** () const

  is this this finest level?

- bool **isEmpty** () const

  is this an empty level?

- void **computeVorticity** (**LevelData**< **FArrayBox** > &a_vorticity) const
- **Real Dx** () const

  access functions

- **Real Nu** () const

  viscous coefficient

- AMRNavierStokes * **finerNSPtr** () const

  encapsulates the finer-level-pointer casting

- AMRNavierStokes * **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**< **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 Methods

- **DisjointBoxLayout loadBalance** (const **Vector**< **Box** > &a_grids)
- void **levelSetup** (const **DisjointBoxLayout** &level_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)

  compute advection velocities

- void **predictVelocities** (**LevelData**< **FArrayBox** > &a_uDelU, **LevelData**< **FluxBox** > &a_advVel)
- void **computeUStar** (**LevelData**< **FArrayBox** > &a_uStar)
- 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**< **DisjointBoxLayout** > &a_grids, const **Vector**< **ProblemDomain** > &a_domains, const **Vector**< **Real** > &a_amrDx, const **Vector**< int > &a_refRatios, const int &a_lBase)

  defines (helmholtz) **AMRSolver** (p. 69) *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**< **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_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
- **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 (p. 383) *for velocities.*
- **PoissonOp m_scalarsPoissonOp**
  PoissonOp (p. 383) *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_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?
- bool **s_tag_theta**
  tag on scalar (theta)?
- **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_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)
- **Real s_nu**
  viscosity
- bool **s_boussinesq**
  use Boussinesq approximation – theta (temperature) provides forcing

- **Real s_theta_coeff**

  factor to use in theta forcing

- **Real s_rayleighNo**

  Rayleigh number, if boussinesq.

- **Real s_prandtlNo**

  Prandtl number, if boussinesq.

- **Real s_wallTemp**

  wall temperature, if boussinesq

- 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)

6.8.1 Constructor & Destructor Documentation

6.8.1.1 AMRNavierStokes::AMRNavierStokes ()

6.8.1.2 AMRNavierStokes::AMRNavierStokes (AMRLevel * *a_coarser_level_ptr*, const Box & *a_prob_domain*, int *a_level*, int *a_ref_ratio*)

6.8.1.3 AMRNavierStokes::AMRNavierStokes (AMRLevel * *a_coarser_level_ptr*, const ProblemDomain & *a_prob_domain*, int *a_level*, int *a_ref_ratio*)

6.8.1.4 virtual AMRNavierStokes::~AMRNavierStokes () `[virtual]`

6.8.2 Member Function Documentation

6.8.2.1 virtual Real AMRNavierStokes::advance () `[virtual]`

advance by one timestep (returns what?)

6.8.2.2 void AMRNavierStokes::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?

6.8.2.3 void AMRNavierStokes::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

6.8.2.4 virtual void AMRNavierStokes::CFL (Real *a_cfl*) `[virtual]`

set CFL number

6.8.2.5 void AMRNavierStokes::computeAdvectionVelocities (LevelData< FluxBox > & *a_adv_vel*) `[protected]`

compute advection velocities

6.8.2.6 virtual Real AMRNavierStokes::computeDt () `[virtual]`

compute dt

Implements **AMRLevel** (p. 35).

6.8.2.7 virtual Real AMRNavierStokes::computeInitialDt () `[virtual]`

compute dt with initial data

Implements **AMRLevel** (p. 35).

6.8.2.8 void AMRNavierStokes::computeLapScal (LevelData< FArrayBox > & *a_lapScal*, LevelData< FArrayBox > & *a_scal*, DomainGhostBC & *a_physBC*, const LevelData< FArrayBox > * *a_crseScalPtr*) `[protected]`

compute Laplacian(scalars)

6.8.2.9 void AMRNavierStokes::computeLapVel (LevelData< FArrayBox > & *a_lapVel*, LevelData< FArrayBox > & *a_vel*, const LevelData< FArrayBox > * *a_crseVelPtr*) `[protected]`

compute Laplacian(velocity)

6.8.2.10 void AMRNavierStokes::computeUStar (LevelData< FArrayBox > & *a_uStar*) [protected]

6.8.2.11 void AMRNavierStokes::computeVorticity (LevelData< FArrayBox > & *a_vorticity*) const

6.8.2.12 AMRNavierStokes* AMRNavierStokes::crseNSPtr () const

encapsulates the coarser-level pointer casting

6.8.2.13 virtual void AMRNavierStokes::define (AMRLevel * *a_coarse_level_ptr*, const ProblemDomain & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) [virtual]

Defines this **AMRLevel** (p. 33).

{\bf Arguments:}\ 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. 35).

6.8.2.14 virtual void AMRNavierStokes::define (AMRLevel * *a_coarse_level_ptr*, const Box & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) [virtual]

Defines this **AMRLevel** (p. 33).

{\bf Arguments:}\ 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. 35).

6.8.2.15 void AMRNavierStokes::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** (p. 69) used in smoothing during regridding

6.8.2.16 Real AMRNavierStokes::Dx () const

access functions

6.8.2.17 void AMRNavierStokes::fillLambda (LevelData< FArrayBox > & *a_lambda*, Real *a_time*) const [protected]

resize temporary lambda for advection tracing computation and fill interior and boundary cells

6.8.2.18 void AMRNavierStokes::fillScalars (LevelData< FArrayBox > & *a_scal*, Real *a_time*, const int *a_comp*) const [protected]

fill interior and boundary cells (assume a_scal already sized)

6.8.2.19 void AMRNavierStokes::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

6.8.2.20 void AMRNavierStokes::fillVelocity (LevelData< FArrayBox > & *a_vel*, Real *a_time*) [protected]

fill grown velocity field using piecewise-constant interp

6.8.2.21 AMRNavierStokes* AMRNavierStokes::finerNSPtr () const

encapsulates the finer-level-pointer casting

6.8.2.22 void AMRNavierStokes::finestLevel (bool *a_finest_level*) [protected]

set whether this is the finest level or not

6.8.2.23 bool AMRNavierStokes::finestLevel () const

is this this finest level?

6.8.2.24 PhysBCUtil* AMRNavierStokes::getPhysBCPtr () const

gets physical BC object

6.8.2.25 virtual void AMRNavierStokes::getPlotData (LevelData< FArrayBox > & *a_plot_data*) const [protected, virtual]

stuff plotfile data

6.8.2.26 virtual void AMRNavierStokes::initialData () [virtual]

initialize data

Implements **AMRLevel** (p. 36).

6.8.2.27 virtual void AMRNavierStokes::initialGrid (const Vector< Box > & *a_new_grids*) [virtual]

initialize grids

Implements **AMRLevel** (p. 36).

6.8.2.28 void AMRNavierStokes::initializeGlobalPressure () [protected]

manages pressure initialization after init or regrid

6.8.2.29 void AMRNavierStokes::initializeLevelPressure (Real *a_currentTime*, Real *a_dtInit*) [protected]

does level-based pressure initalization

6.8.2.30 bool AMRNavierStokes::isEmpty () const

is this an empty level?

6.8.2.31 void AMRNavierStokes::lambda (LevelData< FArrayBox > & *a_lambda*, Real *a_time*) const

returns lambda at time t

6.8.2.32 void AMRNavierStokes::levelSetup (const DisjointBoxLayout & *level_domain*) [protected]

6.8.2.33 DisjointBoxLayout AMRNavierStokes::loadBalance (const Vector< Box > & *a_grids*) [protected]

6.8.2.34 virtual AMRLevel* AMRNavierStokes::makeAMRLevel (AMRLevel * *a_coarser_level_ptr*, const ProblemDomain & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) const [virtual]

6.8.2.35 virtual AMRLevel* AMRNavierStokes::makeAMRLevel (AMRLevel * *a_coarser_level_ptr*, const Box & *a_problem_domain*, int *a_level*, int *a_ref_ratio*) const [virtual]

6.8.2.36 const LevelData<FArrayBox>& AMRNavierStokes::newLambda () const

6.8.2.37 LevelData<FArrayBox>& AMRNavierStokes::newLambda ()

6.8.2.38 const LevelData<FArrayBox>& AMRNavierStokes::newScal (const int *a_comp*) const

6.8.2.39 LevelData<FArrayBox>& AMRNavierStokes::newScal (const int *a_comp*)

6.8.2.40 const LevelData<FArrayBox>& AMRNavierStokes::newVel () const

6.8.2.41 LevelData<FArrayBox>& AMRNavierStokes::newVel ()

6.8.2.42 Real AMRNavierStokes::Nu () const

viscous coefficient

6.8.2.43 virtual int AMRNavierStokes::numPlotComps () const [protected, virtual]

number of components in plotfiles

6.8.2.44 const LevelData<FArrayBox>& AMRNavierStokes::oldLambda () const

6.8.2.45 LevelData<FArrayBox>& AMRNavierStokes::oldLambda ()

6.8.2.46 const LevelData<FArrayBox>& AMRNavierStokes::oldScal (const int *a_comp*) const

6.8.2.47 LevelData<FArrayBox>& AMRNavierStokes::oldScal (const int *a_comp*)

6.8.2.48 const LevelData<FArrayBox>& AMRNavierStokes::oldVel () const

6.8.2.49 LevelData<FArrayBox>& AMRNavierStokes::oldVel ()

6.8.2.50 virtual void AMRNavierStokes::postInitialize () [virtual]

things do do after initialization

Implements **AMRLevel** (p. 37).

6.8.2.51 virtual void AMRNavierStokes::postRegrid (int *a_lBase*) [virtual]

perform any post-regridding ops – lBase is the finest unchanged level

Reimplemented from **AMRLevel** (p. 37).

6.8.2.52 virtual void AMRNavierStokes::postTimeStep () [virtual]

things to do after a timestep

Implements **AMRLevel** (p. 37).

6.8.2.53 void AMRNavierStokes::predictVelocities (LevelData< FArrayBox > & *a_uDelU*, LevelData< FluxBox > & *a_advVel*) [protected]

6.8.2.54 void AMRNavierStokes::readParameters () [protected]

read parameters from parmParse database

6.8.2.55 virtual void AMRNavierStokes::refinementThreshold (Real *a_refine_threshold*) [virtual]

set refinement threshold

6.8.2.56 virtual void AMRNavierStokes::regrid (const Vector< Box > & *a_new_grids*) [virtual]

regrid

Implements **AMRLevel** (p. 37).

6.8.2.57 void AMRNavierStokes::resetStates (const Real *a_time*) [protected]

exchange old and new states, resets time to a_time

6.8.2.58 void AMRNavierStokes::scalars (LevelData< FArrayBox > & *a_scalars*, const Real *a_time*, const int *a_comp*) const

returns scalars at time t

6.8.2.59 void AMRNavierStokes::setPhysBC (const PhysBCUtil & *a_BC*)

sets physical BC object

6.8.2.60 void AMRNavierStokes::smoothVelocityField (int *lbase*) [protected]

smooths composite velocity field after regridding

6.8.2.61 void AMRNavierStokes::swapOldAndNewStates () [protected]

move new-time state into old-time state – also advances time by dt

6.8.2.62 virtual void AMRNavierStokes::tagCells (IntVectSet & *a_tags*) [virtual]

create tags

Implements **AMRLevel** (p. 37).

6.8.2.63 virtual void AMRNavierStokes::tagCellsInit (IntVectSet & *a_tags*) [virtual]

create tags at initialization

Implements **AMRLevel** (p. 37).

6.8.2.64 void AMRNavierStokes::velocity (LevelData< FArrayBox > & *a_vel*, Real *a_time*) const

returns velocity at time t

6.8.3 Member Data Documentation

6.8.3.1 Real AMRNavierStokes::m_cfl [protected]

cfl number

6.8.3.2 CoarseAverage AMRNavierStokes::m_coarse_average [protected]

averaging from fine to coarse level for velocity

6.8.3.3 CoarseAverage AMRNavierStokes::m_coarse_average_lambda [protected]

averaging from fine to coarse level for freestream preservation qty

6.8.3.4 Vector<Tuple<Real, SpaceDim> > AMRNavierStokes::m_dataPoints [protected]

points where data is dumped out

6.8.3.5 Real AMRNavierStokes::m_dt_save [protected]

dt from most recent timestep

6.8.3.6 Real AMRNavierStokes::m_dx [protected]

grid spacing

6.8.3.7 bool AMRNavierStokes::m_finest_level [protected]

true if this is the finest extant level

6.8.3.8 LevelFluxRegister AMRNavierStokes::m_flux_register [protected]

momentum flux register

6.8.3.9 bool AMRNavierStokes::m_is_empty [protected]

true if this level is an empty level (no grids)

6.8.3.10 LevelFluxRegister AMRNavierStokes::m_lambda_flux_reg [protected]

freestream preservation qty flux register

6.8.3.11 LevelData<FArrayBox>* AMRNavierStokes::m_lambda_new_ptr [protected]

freestream preservation qty at new time

6.8.3.12 LevelData<FArrayBox>* AMRNavierStokes::m_lambda_old_ptr [protected]

freestream preservation qty at old time

6.8.3.13 int AMRNavierStokes::m_level_steps [protected]

number of steps on this level

6.8.3.14 PhysBCUtil* AMRNavierStokes::m_physBCPtr [protected]

physical BC object

6.8.3.15 CCProjector AMRNavierStokes::m_projection [protected]

cell-centered projection object

6.8.3.16 Real AMRNavierStokes::m_refine_threshold [protected]

refinement threshold for regridding

6.8.3.17 bool AMRNavierStokes::m_regrid_smoothing_done [protected]

internal indicator whether post-regrid smoothing stuff has been done

6.8.3.18 Vector<LevelFluxRegister*> AMRNavierStokes::m_scal_fluxreg_ptrs [protected]

conserved scalar flux register

6.8.3.19 Vector<LevelData<FArrayBox>*> AMRNavierStokes::m_scal_new [protected]

scalars at new time

6.8.3.20 Vector<LevelData<FArrayBox>*> AMRNavierStokes::m_scal_old [protected]

scalars at old time

6.8.3.21 PoissonOp AMRNavierStokes::m_scalarsPoissonOp [protected]

PoissonOp (p. 383) for advected scalars (for diffusion).

6.8.3.22 LevelData<FArrayBox>* AMRNavierStokes::m_vel_new_ptr [protected]

velocity vector at new time

6.8.3.23 LevelData<FArrayBox>* AMRNavierStokes::m_vel_old_ptr [protected]

velocity vector at old time

6.8.3.24 QuadCFInterp AMRNavierStokes::m_velCFInterp [protected]

does quadratic coarse-fine interpolation for velocities

6.8.3.25 VelocityPoissonOp AMRNavierStokes::m_velocityPoissonOp [protected]

PoissonOp (p. 383) for velocities.

6.8.3.26 LevelHelmholtzSolver AMRNavierStokes::m_viscous_solver [protected]

levelsolver for viscous solves

6.8.3.27 Real AMRNavierStokes::s_backgroundVel [static, protected]

background velocity (in the z-direction)

6.8.3.28 Real AMRNavierStokes::s_bogus_value [static, protected]

debugging option – value to which arrays are initially set

6.8.3.29 bool AMRNavierStokes::s_boussinesq [static, protected]

use Boussinesq approximation – theta (temperature) provides forcing

6.8.3.30 Vector<Real> AMRNavierStokes::s_domLength [static, protected]

size of physical domain

6.8.3.31 bool AMRNavierStokes::s_dump_smoothing_plots [static, protected]

dump before-and-after plotfiles for post-regrid smoothing (for debug)

6.8.3.32 bool AMRNavierStokes::s_implicit_reflux [static, protected]

do implicit refluxing for velocity?

6.8.3.33 bool AMRNavierStokes::s_implicit_scal_reflux [static, protected]

do implicit refluxing for advected/diffused scalars?

6.8.3.34 Real AMRNavierStokes::s_init_shrink [static, protected]

factor to shrink initial timestep

6.8.3.35 bool AMRNavierStokes::s_init_vel_from_vorticity [static, protected]

initialize velocity field from vorticity field

6.8.3.36 string AMRNavierStokes::s_initialGridFile [static, protected]

gridfile where initial grids are spec'd, if relevant

6.8.3.37 bool AMRNavierStokes::s_initialize_pressures [static, protected]

should we initialize pressures after grid generation & regridding?

6.8.3.38 Real AMRNavierStokes::s_max_dt [static, protected]

maximum allowable timestep

6.8.3.39 const int AMRNavierStokes::s_max_scal_comps = SpaceDim [static, protected]

maximum possible number of scalar components

6.8.3.40 Real AMRNavierStokes::s_nu [static, protected]

viscosity

6.8.3.41 int AMRNavierStokes::s_num_init_passes [static, protected]

number of times to iterate when initializing pressures

6.8.3.42 int AMRNavierStokes::s_num_scal_comps [static, protected]

number of scalars

6.8.3.43 const int AMRNavierStokes::s_num_vel_comps = SpaceDim [static, protected]

number of components of m_state (vel comps + lambda)

6.8.3.44 bool AMRNavierStokes::s_ppInit [static, protected]

have we initialized static variables from parmParse?

6.8.3.45 Real AMRNavierStokes::s_prandtlNo [static, protected]

Prandtl number, if boussinesq.

6.8.3.46 Real AMRNavierStokes::s_prescribedDt [static, protected]

6.8.3.47 bool AMRNavierStokes::s_project_initial_vel [static, protected]

should we project initial velocity field (default is yes)

6.8.3.48 Real AMRNavierStokes::s_rayleighNo [static, protected]

Rayleigh number, if boussinesq.

6.8.3.49 bool AMRNavierStokes::s_reflux_momentum [static, protected]

should we reflux momentum?

6.8.3.50 bool AMRNavierStokes::s_reflux_scal [static, protected]

should we reflux scalars?

6.8.3.51 Real AMRNavierStokes::s_regrid_smoothing_coeff [static, protected]

antidiffusive smoothing coefficient (multiplies nu*dtCrse)

6.8.3.52 Vector<Real> AMRNavierStokes::s_scal_coeffs [static, protected]

diffusion coefficients for scalar advection-diffusion

6.8.3.53 const char* AMRNavierStokes::s_scal_names[s_max_scal_comps] [static, protected]

names of scalars

6.8.3.54 bool AMRNavierStokes::s_set_bogus_values [static, protected]

debugging option – initially set arrays to s_bogus_value

6.8.3.55 bool AMRNavierStokes::s_smooth_after_regrid [static, protected]

do antidiffusive smoothing after regridding?

6.8.3.56 bool AMRNavierStokes::s_specifyInitialGrids [static, protected]

do we specify initial grids in a gridFile?

6.8.3.57 bool AMRNavierStokes::s_tag_theta [static, protected]

tag on scalar (theta)?

6.8.3.58 bool AMRNavierStokes::s_tag_vorticity [static, protected]

tag on vorticity?

6.8.3.59 int AMRNavierStokes::s_tags_grow [static, protected]

amount by which to grow tagged regions before calling meshrefine

6.8.3.60 Real AMRNavierStokes::s_theta_coeff [static, protected]

factor to use in theta forcing

6.8.3.61 const char* AMRNavierStokes::s_vel_names[s_num_vel_comps] [static, protected]

names of components

6.8.3.62 Real AMRNavierStokes::s_viscous_solver_tol [static, protected]

viscous solver tolerance (defaults to 1e-7)

6.8.3.63 int AMRNavierStokes::s_viscous_solver_type [static, protected]

viscous solver type: 0 = backwards euler, 1=Crank-Nicolson, 2=TGA

6.8.3.64 Real AMRNavierStokes::s_vort_factor [static, protected]

factor for tagging – tag if vorticity greater than factor*max_vort

6.8.3.65 Real AMRNavierStokes::s_wallTemp [static, protected]

wall temperature, if boussinesq

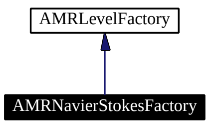
The documentation for this class was generated from the following file:

- **AMRNavierStokes.H**

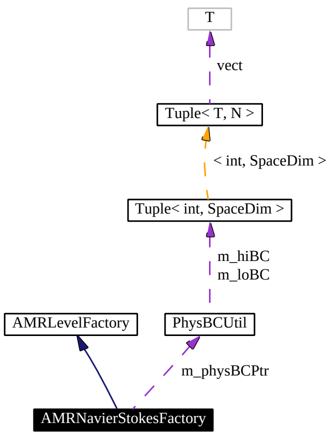
6.9 AMRNavierStokesFactory Class Reference

`#include <AMRNavierStokesFactory.H>`

Inheritance diagram for AMRNavierStokesFactory:



Collaboration diagram for AMRNavierStokesFactory:



Public Methods

- **AMRNavierStokesFactory** ()
- virtual **AMRLevel** ∗ **new_amrlevel** () const
- virtual ∼**AMRNavierStokesFactory** ()
- void **CFL** (const **Real** a_cfl)
- void **refinementThreshold** (const **Real** a_refine_threshold)
- void **setPhysBC** (const **PhysBCUtil** &a_physBC)

Protected Methods

- void **setDefaultValues** ()

Protected Attributes

- **Real m_cfl**
- **Real m_refine_thresh**
- **PhysBCUtil** ∗ **m_physBCPtr**

6.9.1 Constructor & Destructor Documentation

6.9.1.1 AMRNavierStokesFactory::AMRNavierStokesFactory ()

6.9.1.2 virtual AMRNavierStokesFactory::~AMRNavierStokesFactory () [virtual]

6.9.2 Member Function Documentation

6.9.2.1 void AMRNavierStokesFactory::CFL (const Real *a_cfl*)

6.9.2.2 virtual AMRLevel* AMRNavierStokesFactory::new_amrlevel () const [virtual]

Factory function. creates a new **AMRLevel** (p. 33) object. In your instantiation you should include defining all the stuff that is not part of the **AMRLevel** (p. 33) API.

Implements **AMRLevelFactory** (p. 39).

6.9.2.3 void AMRNavierStokesFactory::refinementThreshold (const Real *a_refine_threshold*)

6.9.2.4 void AMRNavierStokesFactory::setDefaultValues () [protected]

6.9.2.5 void AMRNavierStokesFactory::setPhysBC (const PhysBCUtil & *a_physBC*)

6.9.3 Member Data Documentation

6.9.3.1 Real AMRNavierStokesFactory::m_cfl [protected]

6.9.3.2 PhysBCUtil* AMRNavierStokesFactory::m_physBCPtr [protected]

6.9.3.3 Real AMRNavierStokesFactory::m_refine_thresh [protected]

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

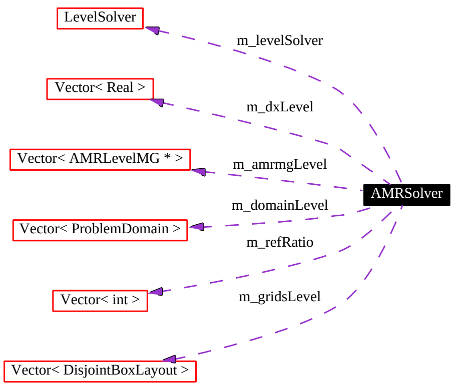
- **AMRNavierStokesFactory.H**

6.10 AMRSolver Class Reference

Class which manages grid hierarchy and composite elliptic solution.

`#include <AMRSolver.H>`

Collaboration diagram for AMRSolver:



Public Methods

- **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 ncomp=1)
- **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 ncomp=1)
- 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 ncomp=1)
- 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 ncomp=1)
- **~AMRSolver** ()
- bool **isDefined** () const
- void **setNumSmoothUp** (int a_numSmoothUp)
- void **setNumSmoothDown** (int a_numSmoothDown)
- void **setTolerance** (**Real** a_tolerance)
- void **setOperatorTolerance** (**Real** a_operatorTolerance)
- void **setMaxIter** (int a_maxIter)
- void **setMinIter** (int a_minIter)
- void **setNumVCyclesBottom** (int a_numVCycleBottom)
- void **solveAMR** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel)

  Solves on hierarchy to tolerance m_tolerance.

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

- **Vector**< **Real** > **computeResidualNorm** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel, int normNtype)
- **Vector**< **Real** > **computeResidualNorm** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_residLevel, **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel, int normNtype)
- void **computeAMRResidual** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_rhsLevel, **LevelData**< **FArrayBox** > &res, int ilev)
- void **applyAMROperator** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, **LevelData**< **FArrayBox** > &a_LofPhi, int a_ilev)
- void **applyAMROperatorHphys** (**Vector**< **LevelData**< **FArrayBox** > ∗ > &a_phiLevel, **LevelData**< **FArrayBox** > &a_LofPhi, int a_ilev)
- void **setVerbose** (bool a_verbose)

Protected Methods

- void **setDefaultValues** ()
- void **clear** ()

Protected Attributes

- **Real m_errorTolerance**
- **Real m_tolerance**
- **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**

6.10.1 Detailed Description

Class which manages grid hierarchy and composite elliptic solution.

AMRSolver manages the **AMR** (p. 17)/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** (p. 17) / multigrid V-cycle as a preconditioner for some other iterative method, such as a Krylov method.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 AMRSolver::AMRSolver ()

Creates a AMRSolver whose internal state is undefined. Need to call define(...) function to use any of the functionality of the class

6.10.2.2 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 *ncomp* = 1)

Creates a fully-defined AMRSolver. Calls define function with identical arguments

6.10.2.3 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 *ncomp* = 1)

Creates a fully-defined AMRSolver. Calls define function with identical arguments

6.10.2.4 AMRSolver::∼AMRSolver ()

6.10.3 Member Function Documentation

6.10.3.1 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. \ {\bf Inputs:} \ corrLevel - pointers to current guess at the correction values for levels lMin = max(lBase-1,0) ... lMax. **Vector** (p. 471) index corresponds to level number. \ residLevel - pointers to **AMR** (p. 17) residual for levels lMin ... lMax. **Vector** (p. 471) index corresponds to level number. \{\bf Outputs:} \ corrLevel - **LevelData** (p. 295) <**FArrayBox** (p. 229) >s pointed to for levels lMin,...,lMax are updated in place.

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

Calculate multilevel L(phi). includes refluxing and all that This is the three-level operator.

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

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

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

6.10.3.5 void AMRSolver::computeAMRResidual (Vector< LevelData< FArrayBox > * > & *a_phiLevel*, const Vector< LevelData< FArrayBox > * > & *a_rhsLevel*, LevelData< FArrayBox > & *res*, int *ilev*)

Calculate multilevel residual on level ilev.

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

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.

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

Calculate norm of multilevel residual on levels lBase to lmax. Does not include data covered by finer levels.

6.10.3.8 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 *ncomp* = 1)

Defines AMRSolvers' internal state \ {\bf 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** (p. 471) 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** (p. 471) index corresponds to level number. \ dxLevel — The grid spacing at all levels. Each element in the vector is a level's $\dx$. dxLevel[0]— is $\dx$ for level 0 and so forth. **Vector** (p. 471) index corresponds to level number. \ ref_ratio— The refinement ratio between all levels. ref_ratio[0] is the refinement ratio between level 0 and level 1; **Vector** (p. 471) index corresponds to level number. \ numlevels The number of **AMR** (p. 17) levels in the calculation. The length of the Vector~s 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** (p. 319). \ opin_a The levelop to use in the solution.

6.10.3.9 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 *ncomp* = 1)

Defines AMRSolvers' internal state \ {\bf 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** (p. 471) 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** (p. 471) index corresponds to level number. \ dxLevel — The grid spacing at all levels. Each element in the vector is a level's $\dx$. dxLevel[0]— is $\dx$ for level 0 and so forth. **Vector** (p. 471) index corresponds to level number. \ ref_ratio— The refinement ratio between all levels. ref_ratio[0] is the refinement ratio between level 0 and level 1; **Vector** (p. 471) index corresponds to level number. \ numlevels The number of **AMR** (p. 17) levels in the calculation. The length of the Vector∼s 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** (p. 319). \ opin_a The levelop to use in the solution.

6.10.3.10 bool AMRSolver::isDefined () const

Returns true if full define function has been called.

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

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

Set max number of iterations. Default is 42.

6.10.3.13 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.

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

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

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

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

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

6.10.3.17 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.

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

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

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

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

6.10.3.20 void AMRSolver::solveAMR (Vector< LevelData< FArrayBox > ∗ > & *a_phiLevel*, const Vector< LevelData< FArrayBox > ∗ > & *a_rhsLevel*)

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. \{\bf Inputs:\} \ phiLevel - pointers to current guess at the solution values for levels (lMin = max(lBase-1,0)) ... lMax. **Vector** (p. 471) index corresponds to level number. \ rhsLevel - pointers to right-hand side for levels lmin ... lMax. **Vector** (p. 471) index corresponds to level number. \{\bf Outputs:\} \ phiLevel - **LevelData** (p. 295)<**FArrayBox** (p. 229)>s pointed to for levels lMin,...,lMax are updated in place.

6.10.4 Friends And Related Function Documentation

6.10.4.1 friend class AMRLevelMG [friend]

6.10.5 Member Data Documentation

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

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

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

6.10.5.4 Real AMRSolver::m_errorTolerance [protected]

6.10.5.5 int AMRSolver::m_finestLevel [protected]

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

6.10.5.7 bool AMRSolver::m_isDefined [protected]

6.10.5.8 int AMRSolver::m_lBase [protected]

6.10.5.9 LevelSolver AMRSolver::m_levelSolver [protected]

6.10.5.10 int AMRSolver::m_maxIter [protected]

6.10.5.11 int AMRSolver::m_minIter [protected]

6.10.5.12 int AMRSolver::m_ncomp [protected]

6.10.5.13 int AMRSolver::m_numLevels [protected]

6.10.5.14 int AMRSolver::m_numSmoothDown [protected]

6.10.5.15 int AMRSolver::m_numSmoothUp [protected]

6.10.5.16 int AMRSolver::m_numVCyclesBottom [protected]

6.10.5.17 Real AMRSolver::m_operatorTolerance [protected]

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

6.10.5.19 Real AMRSolver::m_tolerance [protected]

6.10.5.20 bool AMRSolver::m_verbose [protected]

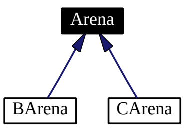
The documentation for this class was generated from the following file:

- **AMRSolver.H**

6.11 Arena Class Reference

`#include <Arena.H>`

Inheritance diagram for Arena:



Public Types

- typedef void(* **FP**)()

Public Methods

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

Static Public Methods

- size_t **align** (size_t sz)

6.11.1 Member Typedef Documentation

6.11.1.1 typedef void(* Arena::FP)()

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Arena::Arena ()

base class constructor. registers Arena object with memory tracking system

6.11.2.2 virtual Arena::~Arena () `[virtual]`

base class destructor. unregisters Arena object with memory tracking system

6.11.3 Member Function Documentation

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

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

Implemented in **BArena** (p. 80), and **CArena** (p. 159).

6.11.3.3 virtual void Arena::free (void * *pt*) [pure virtual]

Implemented in **BArena** (p. 80), and **CArena** (p. 159).

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

- **Arena.H**

6.12 Arena::Word Union Reference

`#include <Arena.H>`

Public Attributes

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

6.12.1 Member Data Documentation

6.12.1.1 double Arena::Word::d

6.12.1.2 FP Arena::Word::f

6.12.1.3 long Arena::Word::l

6.12.1.4 void* Arena::Word::p

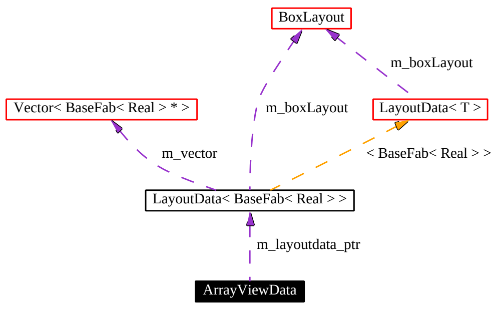
The documentation for this union was generated from the following file:

- **Arena.H**

6.13 ArrayViewData Class Reference

`#include <ArrayViewData.H>`

Collaboration diagram for ArrayViewData:



Public Methods

- **ArrayViewData** ()
- ~**ArrayViewData** ()
- **ArrayViewData** (**LayoutData**< **BaseFab**< **Real** > > ∗a_layoutdata_ptr)
- **ArrayViewData** (**LayoutData**< **FArrayBox** > ∗a_layoutdata_ptr)
- **BaseFab**< **Real** > & **operator**[] (int a_elem)
- void **set** (int a_elem, **BaseFab**< **Real** > ∗a_fab_ptr)

6.13.1 Constructor & Destructor Documentation

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

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

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

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

6.13.2 Member Function Documentation

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

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

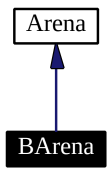
The documentation for this class was generated from the following file:

- **ArrayViewData.H**

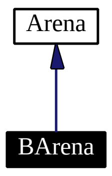
6.14 BArena Class Reference

`#include <Arena.H>`

Inheritance diagram for BArena:



Collaboration diagram for BArena:



Public Methods

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

6.14.1 Constructor & Destructor Documentation

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

6.14.2 Member Function Documentation

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

Implements **Arena** (p. 76).

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

Implements **Arena** (p. 77).

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

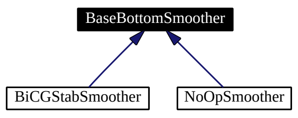
- **Arena.H**

6.15 BaseBottomSmoother Class Reference

Pure base class to encapsulate bottom smooth for **LevelOp** (p. 315)-derived objects.

`#include <BaseBottomSmoother.H>`

Inheritance diagram for BaseBottomSmoother:



Public Methods

- **BaseBottomSmoother** ()

  constructor, destructor default constructor

- virtual ~**BaseBottomSmoother** ()

  destructor

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

  do smoothing

6.15.1 Detailed Description

Pure base class to encapsulate bottom smooth for **LevelOp** (p. 315)-derived objects.

The BaseBottomSmoother class is a virtual base class for implementing different bottomSmoother algorithms for **LevelOp** (p. 315)-derived classes. It takes a **LevelOp** (p. 315) pointer as an argument during its definition, in order to access the problem-dependent (equation-dependent) functionality of the specific **LevelOp** (p. 315) derivative class.

6.15.2 Constructor & Destructor Documentation

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

constructor, destructor default constructor

6.15.2.2 virtual BaseBottomSmoother::~BaseBottomSmoother () `[inline, virtual]`

destructor

6.15.3 Member Function Documentation

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

do smoothing

Implemented in **BiCGStabSmoother** (p. 101), and **NoOpSmoother** (p. 354).

6.15.3.2 virtual BaseBottomSmoother* BaseBottomSmoother::new_bottomSmoother () const `[pure virtual]`

this gets around the "no virtual constructor" rule

Implemented in **BiCGStabSmoother** (p. 101), and **NoOpSmoother** (p. 354).

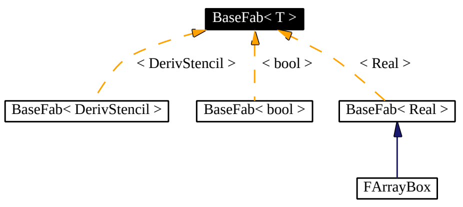
The documentation for this class was generated from the following file:

- **BaseBottomSmoother.H**

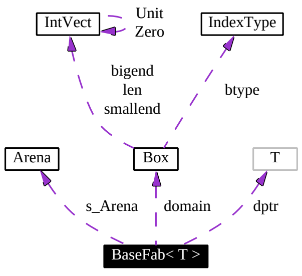
6.16 BaseFab< T > Class Template Reference

`#include <BaseFab.H>`

Inheritance diagram for BaseFab< T >:



Collaboration diagram for BaseFab< T >:



Public Methods

- **BaseFab** ()

    {\bf *constructors, destructor and defines*}

- **BaseFab** (const **Box** &bx, int n)
- virtual ~**BaseFab** ()
- void **resize** (const **Box** &b, int N=1)
- virtual void **define** (const **Box** &box, int comps)
- void **clear** ()
- int **nComp** () const

    {\bf *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

  {\bf *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

  {\bf *comparison functions*}

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

  {\bf *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)

  {\bf *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

  {\bf *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 Methods

- int **preAllocatable** ()

Protected Methods

- 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 Methods

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

Protected Attributes

- **Box domain**
- int **nvar**
- long **numpts**
- long **truesize**
- T ∗ **dptr**

Static Protected Attributes

- **Arena** ∗ **s_Arena** = NULL

template<class T> class BaseFab< T >

6.16.1 Constructor & Destructor Documentation

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

{\bf constructors, destructor and defines}

6.16.1.2 template<class T> BaseFab< T >::BaseFab (const Box & *bx*, int *n*) `[inline]`

Constructs a BaseFab with desired domain and number of components.

6.16.1.3 template<class T> BaseFab< T >::∼BaseFab () `[inline, virtual]`

The destructor deletes the array memory.

6.16.2 Member Function Documentation

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

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

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

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

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

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

{\bf comparison functions}

6.16.2.6 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.

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

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

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

6.16.2.10 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]

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

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

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

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

Reimplemented in **FArrayBox** (p. 232).

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

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

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

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

Returns an **Interval** (p. 263) for the entire range on components.

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

6.16.2.20 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.

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

{\bf Fortran interface functions}

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

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

{\bf accessors}

Returns the number of components.

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

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

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

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

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

6.16.2.29 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. 237).

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

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

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

6.16.2.33 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.

6.16.2.34 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.

6.16.2.35 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** (p. 106) bx are set to the given value x.

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

{\bf data modification functions}

6.16.2.37 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.

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

{\bf 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.

6.16.2.39 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.

6.16.2.40 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** (p. 106) 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_halfs is EVEN the shift is num_halfs/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.

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

{\bf 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** (p. 106) R and the component **Interval** (p. 263) comps. The size does not include the size of R and comps.

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

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

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

6.16.3 Member Data Documentation

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

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

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

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

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

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

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

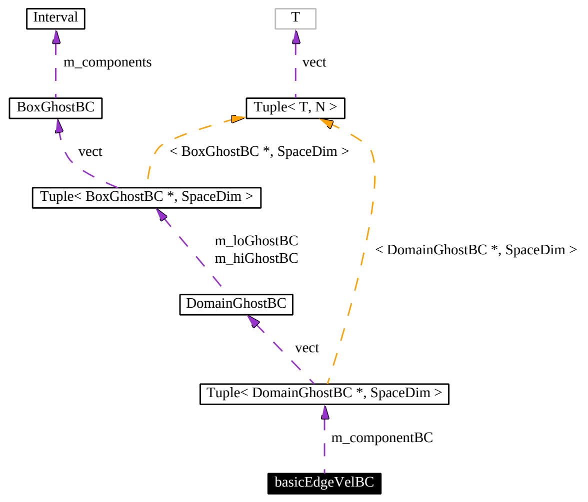
- **BaseFab.H**
- **BaseFabImplem.H**

6.17 basicEdgeVelBC Class Reference

this is a basic physical BC class designed to handle velocities

`#include <basicVelBC.H>`

Collaboration diagram for basicEdgeVelBC:



Public Methods

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

    one **DomainGhostBC** (p. 215) *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** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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** (p. 106) *instead of* **ProblemDomain** (p. 418)

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

  returns **DomainGhostBC** (p. 215) *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

6.17.1 Detailed Description

this is a basic physical BC class designed to handle velocities

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

6.17.2 Constructor & Destructor Documentation

6.17.2.1 basicEdgeVelBC::basicEdgeVelBC ()

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

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

6.17.3 Member Function Documentation

6.17.3.1 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** (p. 106) instead of **ProblemDomain** (p. 418)

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

single-component version

6.17.3.3 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** (p. 106) instead of **ProblemDomain** (p. 418)

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

single-component version

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

old interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

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

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

old interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

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

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

old interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

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

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

returns **DomainGhostBC** (p. 215) for direction Dir

6.17.3.12 bool basicEdgeVelBC::isSingleComponent () const

will this be applied to single-component FluxBoxes?

6.17.3.13 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.

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

one **DomainGhostBC** (p. 215) is set for each velocity component

6.17.4 Member Data Documentation

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

6.17.4.2 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:

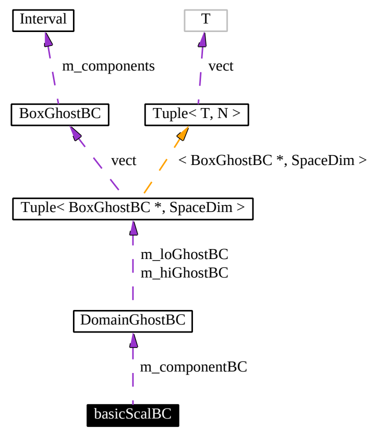
- **basicVelBC.H**

6.18 basicScalBC Class Reference

this is a basic physical BC class designed to handle scalars

`#include <basicScalBC.H>`

Collaboration diagram for basicScalBC:



Public Methods

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

    DomainGhostBC (p. 215) *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**

6.18.1 Detailed Description

this is a basic physical BC class designed to handle scalars

6.18.2 Constructor & Destructor Documentation

6.18.2.1 basicScalBC::basicScalBC ()

6.18.2.2 virtual basicScalBC::~basicScalBC () `[virtual]`

6.18.2.3 basicScalBC::basicScalBC ()

6.18.3 Member Function Documentation

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

deprecated interface

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

deprecated interface

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

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

6.18.3.5 virtual void basicScalBC::setAllBCTypes () `[pure virtual]`

or the derived class can do it for you

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

DomainGhostBC (p. 215) is set.

6.18.4 Member Data Documentation

6.18.4.1 DomainGhostBC* basicScalBC::m_componentBC `[protected]`

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

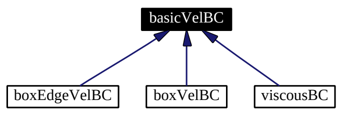
- **basicScalBC.H**

6.19 basicVelBC Class Reference

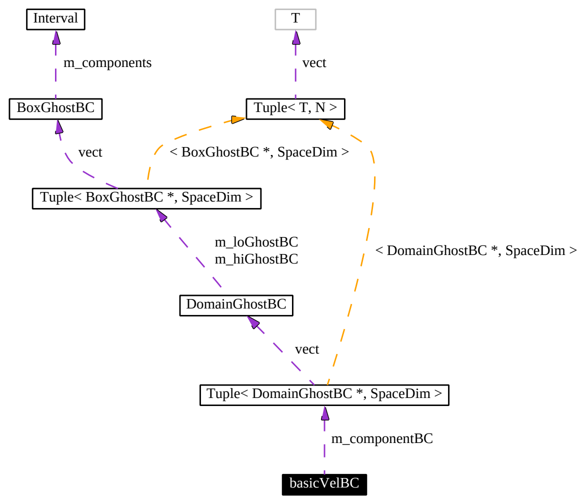
this is a basic physical BC class designed to handle velocities

`#include <basicVelBC.H>`

Inheritance diagram for basicVelBC:



Collaboration diagram for basicVelBC:



Public Methods

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

  one **DomainGhostBC** (p. 215) *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** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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** (p. 106) *instead of* **ProblemDomain** (p. 418)

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

  returns **DomainGhostBC** (p. 215) *for direction Dir*

Protected Attributes

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

6.19.1 Detailed Description

this is a basic physical BC class designed to handle velocities

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

6.19.2 Constructor & Destructor Documentation

6.19.2.1 basicVelBC::basicVelBC ()

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

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

6.19.3 Member Function Documentation

6.19.3.1 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** (p. 106) instead of **ProblemDomain** (p. 418)

Reimplemented in **boxEdgeVelBC** (p. 129), and **viscousBC** (p. 479).

6.19.3.2 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. 130), and **viscousBC** (p. 480).

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

old interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

Reimplemented in **viscousBC** (p. 479).

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

Reimplemented in **viscousBC** (p. 480).

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

old interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

Reimplemented in **boxEdgeVelBC** (p. 130), and **viscousBC** (p. 479).

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

Reimplemented in **boxEdgeVelBC** (p. 130), and **viscousBC** (p. 480).

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

returns **DomainGhostBC** (p. 215) for direction Dir

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

one **DomainGhostBC** (p. 215) is set for each velocity component

6.19.4 Member Data Documentation

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

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

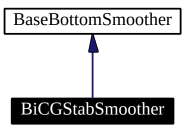
- **basicVelBC.H**

6.20 BiCGStabSmoother Class Reference

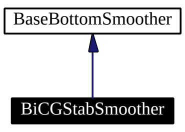
Implements BiCGStab as a bottom smoother for **LevelOp** (p. 315)-derived classes.

`#include <BiCGStabSmoother.H>`

Inheritance diagram for BiCGStabSmoother:



Collaboration diagram for BiCGStabSmoother:



Public Methods

- **BiCGStabSmoother** ()
  constructor, destructor default constructor
- virtual ~**BiCGStabSmoother** ()
  destructor
- virtual **BaseBottomSmoother** ∗ **new_bottomSmoother** () const
- void **setMaxIter** (int a_max_iter)
  set maximum number of BiCGStab iterations
- void **setSolverTol** (**Real** a_solver_tol)
  set solver tolerance
- void **setVerbose** (bool a_verbose)
  set verbosity level
- virtual void **doBottomSmooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs, **LevelOp** ∗a_levelop_ptr)
  do smoothing

Protected Attributes

- int **m_maxIter**
- **Real m_solver_tol**
- **Real m_small**
- bool **m_verbose**

6.20.1 Detailed Description

Implements BiCGStab as a bottom smoother for **LevelOp** (p. 315)-derived classes.

The BiCGStabSmoother class implements the BiCGStab algorithm for **LevelOp** (p. 315)-derived classes

6.20.2 Constructor & Destructor Documentation

6.20.2.1 BiCGStabSmoother::BiCGStabSmoother ()

constructor, destructor default constructor

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

destructor

6.20.3 Member Function Documentation

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

do smoothing

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

Implements **BaseBottomSmoother** (p. 82).

6.20.3.2 virtual BaseBottomSmoother* BiCGStabSmoother::new_bottomSmoother () const `[virtual]`

this gets around the "no virtual constructor" rule

Implements **BaseBottomSmoother** (p. 82).

6.20.3.3 void BiCGStabSmoother::setMaxIter (int *a_max_iter*)

set maximum number of BiCGStab iterations

set max number of BiCGStab iterations

6.20.3.4 void BiCGStabSmoother::setSolverTol (Real *a_solver_tol*)

set solver tolerance

set solver tolerance (amount residual must be decreased, etc)

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

set verbosity level

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

6.20.4 Member Data Documentation

6.20.4.1 int BiCGStabSmoother::m_maxIter [protected]

6.20.4.2 Real BiCGStabSmoother::m_small [protected]

6.20.4.3 Real BiCGStabSmoother::m_solver_tol [protected]

6.20.4.4 bool BiCGStabSmoother::m_verbose [protected]

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

- **BiCGStabSmoother.H**

6.21 BitSet Class Reference

`#include <BitSet.H>`

Public Methods

- **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

Static Public Methods

- int **initialize** ()

Static Public Attributes

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

Friends

- class **BitSetIterator**

6.21.1 Constructor & Destructor Documentation

6.21.1.1 BitSet::BitSet ()

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

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

6.21.1.4 BitSet::~BitSet ()

6.21.2 Member Function Documentation

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

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

6.21.2.3 bool BitSet::isEmpty () const

returns 'true' if the entire bitset is zero

6.21.2.4 bool BitSet::isFull () const

returns 'true' if entire bitset is 1

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

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

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

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

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

6.21.3 Friends And Related Function Documentation

6.21.3.1 friend class BitSetIterator `[friend]`

6.21.4 Member Data Documentation

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

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

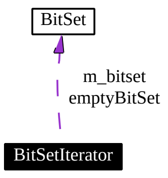
The documentation for this class was generated from the following file:

- **BitSet.H**

6.22 BitSetIterator Class Reference

`#include <BitSet.H>`

Collaboration diagram for BitSetIterator:



Public Methods

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

6.22.1 Constructor & Destructor Documentation

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

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

6.22.2 Member Function Documentation

6.22.2.1 void BitSetIterator::begin ()

6.22.2.2 void BitSetIterator::end ()

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

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

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

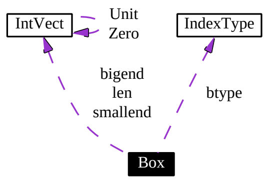
The documentation for this class was generated from the following file:

- **BitSet.H**

6.23 Box Class Reference

#include <Box.H>

Collaboration diagram for Box:



Public Methods

- **Box** ()

  {\bf Constructors}

- ~**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

  {\bf 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

  {\bf *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)

  {\bf *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** ()

  {*bf Box Constants*}

- void **computeBoxLenNotEmpty** ()

Protected Methods

- 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)

  {\bf I/O Functions}

- std::istream & **operator**>> (std::istream &os, Box &bx)

6.23.1 Constructor & Destructor Documentation

6.23.1.1 Box::Box ()

{\bf Constructors}

The default constructor. The constructed Box is empty.

6.23.1.2 Box::~Box () `[inline]`

The destructor.

6.23.1.3 Box::Box (const IntVect & *small*, const IntVect & *big*)

Construct cell-centered type Box. It is an error if small is greater than big.

6.23.1.4 Box::Box (const IntVect & *small*, const int * *vec_len*)

Construct Box with specified lengths. It is an error if the lengths are negative.

6.23.1.5 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** (p. 256).

6.23.1.6 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.

6.23.1.7 Box::Box (const Box & *b*) `[inline]`

The copy constructor.

6.23.2 Member Function Documentation

6.23.2.1 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.

6.23.2.2 const IntVect & Box::bigEnd () const `[inline]`

Returns the upper corner of this Box.

6.23.2.3 bool Box::cellCentered () const `[inline]`

Returns true if this Box is cell-centered in all indexing directions.

6.23.2.4 Box Box::chop (int *dir*, int *chop_pnt*)

misc

Chops the Box at the chop_pnt 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_pnt included in the high end (new Box). It is an ERROR if chop_pnt is the low end of the orig Box. If type(dir) = NODE, the chop_pnt is included in both Boxes but is the only point in common. It is also an error if the chop_pnt is an end node of the Box. Directions are zero-based. It is an error if not 0 <= dir < SpaceDim.

6.23.2.5 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 + ((hiratio)==0 ? 0 : 1).\

That is, refinement of coarsened Box must contain the original Box.

6.23.2.6 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 + ((hiratio)==0 ? 0 : 1).\

That is, refinement of coarsened Box must contain the original Box.

6.23.2.7 void Box::computeBoxLen () `[inline]`

{\bf Box Constants}

This is a canonical empty Box of cell-centered type.

6.23.2.8 void Box::computeBoxLenNotEmpty () `[inline]`

6.23.2.9 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** (p. 264).

6.23.2.10 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.

6.23.2.11 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.

6.23.2.12 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** (p. 256).

6.23.2.13 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.

6.23.2.14 Box Box::copy () const `[inline]`

6.23.2.15 void Box::define (const Box & *b*)

6.23.2.16 void Box::define (const IntVect & *small*, const IntVect & *big*)

Construct cell-centered type Box. It is an error if small is greater than big.

6.23.2.17 void Box::dumpOn (std::ostream & *strm*) const

Gives more detail than printOn. Useful for exiting due to an error.

6.23.2.18 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.

6.23.2.19 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.

6.23.2.20 Box& Box::enclosedCells_int (int *dir*)

6.23.2.21 bool Box::eq (const Box & *b*) const

6.23.2.22 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.

6.23.2.23 Box & Box::grow (int *idir*, int *n_cell*) `[inline]`

Modifies this Box by growing it on the low and high end by n_cell cells in direction idir. The Empty Box is not modified by this function. NOTE: n_cell 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.

6.23.2.24 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.

6.23.2.25 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_cell negative shrinks the Box by that number of cells. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box.

6.23.2.26 Box & Box::growDir (int *a_idir*, const Side::LoHiSide & *a_sd*, int *a_cell*) [inline]

growLo if a_sd == **Side::Lo** (p. 457) \ growHi if a_sd= = **Side::Hi** (p. 457)

6.23.2.27 Box & Box::growHi (int *idir*, int *n_cell* = 1) [inline]

Modifies this Box by growing it on the high end by n_cell cells in direction idir. The Empty Box is not modified by this function. NOTE: n_cell 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.

6.23.2.28 Box & Box::growLo (int *idir*, int *n_cell* = 1) [inline]

Modifies this Box by growing it on the low end by n_cell cells in direction idir. The Empty Box is not modified by this function. NOTE: n_cell 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.

6.23.2.29 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.

6.23.2.30 long Box::index (const IntVect & *v*) const [inline]

Returns offset of point from smallend; i.e. index(smallend) -> 0, bigend would return **volume**() (p. 119)-1. Is used in accessing **FArrayBox** (p. 229).

6.23.2.31 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.

6.23.2.32 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.

6.23.2.33 bool Box::isEmpty () const `[inline]`

{\bf Comparison Functions}

Returns true if this Box is empty.

6.23.2.34 IndexType Box::ixType () const `[inline]`

centering

Return the indexing type of this Box.

6.23.2.35 int Box::longside () const

Returns length of the longest side of this Box. Ignores type.

6.23.2.36 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.

6.23.2.37 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.

6.23.2.38 bool Box::lt (const Box & *rhs*) const

6.23.2.39 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.

6.23.2.40 bool Box::neq (const Box & *b*) const

6.23.2.41 void Box::next (IntVect & *p*, const int $*$ *shv*) const

6.23.2.42 void Box::next (IntVect &) const

6.23.2.43 long Box::numPts () const

Return the volume, in indexing space, of region enclosed by this Box. This is identical to **volume**() (p. 119). Abort()s if the number cannot be represented in a long.\

6.23.2.44 bool Box::numPtsOK (long & *N*) const `[protected]`

6.23.2.45 bool Box::numPtsOK () const `[inline]`

Is the number calculated by **numPts**() (p. 115) representable in a long?

6.23.2.46 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.

6.23.2.47 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.

6.23.2.48 bool Box::operator!= (const Box & *b*) const `[inline]`

Returns true if this Box and the argument differ, including type.

6.23.2.49 Box Box::operator+ (const IntVect & *v*) const `[inline]`

Returns a Box that is this Box shifted by the given **IntVect** (p. 264). The Empty Box is not modified by this function.

6.23.2.50 Box & Box::operator+= (const IntVect & *v*) `[inline]`

Modifies this Box by shifting it by given **IntVect** (p. 264). The Empty Box is not modified by this function.

6.23.2.51 Box Box::operator- (const IntVect & *v*) const `[inline]`

Returns a Box that is this Box shifted by the given **IntVect** (p. 264). The Empty Box is not modified by this function.

6.23.2.52 Box & Box::operator-= (const IntVect & *v*) [inline]

Modifies this Box by shifting it by given **IntVect** (p. 264). The Empty Box is not modified by this function.

6.23.2.53 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.

6.23.2.54 Box & Box::operator= (const Box & *b*) [inline]

{\bf Modification Functions}

The assignment operator.

6.23.2.55 bool Box::operator== (const Box & *b*) const [inline]

Returns true if this Box and the argument are identical, including type.

6.23.2.56 void Box::p () const

print to **pout**() (p. 664)

6.23.2.57 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.

6.23.2.58 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.

6.23.2.59 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.

6.23.2.60 bool Box::sameType (const Box & *b*) const [inline]

Returns true if this Box and the argument have same type.

6.23.2.61 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.

6.23.2.62 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.

6.23.2.63 Box & Box::setRange (int *dir*, int *sm_index*, int *n_cells* = 1) [inline]

Set the entire range in a given direction, starting at sm_index with length n_cells. It is an error if n_cells <= 0.

6.23.2.64 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.

6.23.2.65 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.

6.23.2.66 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.

6.23.2.67 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.

6.23.2.68 Box& Box::shift_intvect (const IntVect & *iv*)

6.23.2.69 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.

6.23.2.70 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_halfs is EVEN the shift is num_halfs/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.

6.23.2.71 Box& Box::shiftHalf_intvect (const IntVect & *iv*)

6.23.2.72 int Box::shortside () const

Returns length of the shortest side of this Box. Ignores type.

6.23.2.73 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.

6.23.2.74 IntVect Box::sideEnd (Side::LoHiSide *a_side*) const `[inline]`

return smallend if side is low, bigend if side is high.

6.23.2.75 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.

6.23.2.76 const IntVect & Box::size () const `[inline]`

size functions

Return an **IntVect** (p. 264) containing the size of this Box in each coordinate direction.

6.23.2.77 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.

6.23.2.78 const IntVect & Box::smallEnd () const [inline]

{\bf Accessors}

Returns the lower corner of this Box.

6.23.2.79 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.

6.23.2.80 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.

6.23.2.81 Box& Box::surroundingNodes_int (int *dir*)

6.23.2.82 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.

6.23.2.83 IntVect Box::type () const [inline]

Return the indexing type of this Box.

6.23.2.84 long Box::volume () const

Return the volume, in indexing space, of region enclosed by this Box. This is identical to **numPts**() (p. 115). Abort()s if the number cannot be represented in a long.\

6.23.2.85 bool Box::volumeOK (long & *N*) const [protected]

6.23.2.86 bool Box::volumeOK () const [inline]

Is the number calculated by **volume**() (p. 119) representable in a long?

6.23.3 Friends And Related Function Documentation

6.23.3.1 Box adjCellBox (const Box & *b*, int *dir*, Side::LoHiSide *a_side*, int *len*) [friend]

6.23.3.2 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.

6.23.3.3 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.

6.23.3.4 Box bdryBox (const Box & *b*, int *dir*, Side::LoHiSide *a_sd*, int *len*) `[friend]`

neighbor box functions

6.23.3.5 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.

6.23.3.6 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.

6.23.3.7 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 + ((hiratio)==0 ? 0 : 1).\

That is, refinement of coarsened Box must contain the original Box.

6.23.3.8 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 + ((hiratio)==0 ? 0 : 1).\

That is, refinement of coarsened Box must contain the original Box.

6.23.3.9 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.

6.23.3.10 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.

6.23.3.11 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.

6.23.3.12 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_cell negative shrinks the Box by that number of cells. If shrinking produces an empty Box, the Box is transformed into the canonical Empty Box.

6.23.3.13 friend class HDF5Handle [friend]

6.23.3.14 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.

6.23.3.15 std::ostream& operator<< (std::ostream & *os*, const Box & *bx*) [friend]

{\bf I/O Functions}

Write an ASCII representation to the ostream.

6.23.3.16 std::istream& operator>> (std::istream & *os*, Box & *bx*) [friend]

Read from istream.

6.23.3.17 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.

6.23.3.18 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.

6.23.3.19 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.

6.23.3.20 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.

6.23.4 Member Data Documentation

6.23.4.1 IntVect Box::bigend [protected]

6.23.4.2 IndexType Box::btype [protected]

6.23.4.3 IntVect Box::len [protected]

6.23.4.4 IntVect Box::smallend [protected]

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

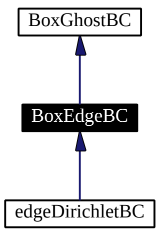
- **Box.H**

6.24 BoxEdgeBC Class Reference

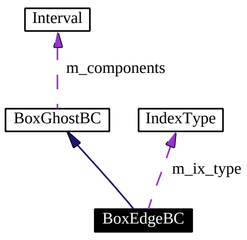
class to do edge-centered physical boundary conditions

`#include <edgeGhostBC.H>`

Inheritance diagram for BoxEdgeBC:



Collaboration diagram for BoxEdgeBC:



Public Methods

- 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 Methods

- 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

6.24.1 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 **BoxGhost-BC** (p. 132) so that it can be used seamlessly with BoxGhostBCs.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 virtual BoxEdgeBC::~BoxEdgeBC () `[virtual]`

6.24.2.2 BoxEdgeBC::BoxEdgeBC ()

6.24.2.3 BoxEdgeBC::BoxEdgeBC (int *a_dir*, Side::LoHiSide *a_sd*)

sets interval to be (0,0)

6.24.2.4 BoxEdgeBC::BoxEdgeBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*)

6.24.2.5 BoxEdgeBC::BoxEdgeBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*, IndexType & *a_ixtype*)

6.24.3 Member Function Documentation

6.24.3.1 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]`

Reimplemented from **BoxGhostBC** (p. 134).

6.24.3.2 virtual void BoxEdgeBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

Reimplemented from **BoxGhostBC** (p. 134).

6.24.3.3 virtual void BoxEdgeBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.24.3.4 virtual void BoxEdgeBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

Reimplemented from **BoxGhostBC** (p. 134).

6.24.3.5 virtual void BoxEdgeBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.24.3.6 void BoxEdgeBC::define (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*, IndexType & *a_ixtype*) `[protected]`

6.24.3.7 void BoxEdgeBC::define (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*) `[protected]`

Reimplemented from **BoxGhostBC** (p. 134).

6.24.3.8 void BoxEdgeBC::define (int *a_dir*, Side::LoHiSide *a_sd*) `[protected]`

sets interval to be (0,0)

Reimplemented from **BoxGhostBC** (p. 134).

6.24.3.9 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. 135).

Reimplemented in **edgeDirichletBC** (p. 219).

6.24.3.10 virtual void BoxEdgeBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const `[protected, virtual]`

Implements **BoxGhostBC** (p. 135).

Reimplemented in **edgeDirichletBC** (p. 219).

6.24.3.11 void BoxEdgeBC::ixType (const IndexType & *a_ixType*)

6.24.3.12 const IndexType& BoxEdgeBC::ixType () const

returns the type of centering of this edge-centered BC

6.24.3.13 virtual BoxGhostBC* BoxEdgeBC::new_boxghostbc () const `[protected, virtual]`

virtual constructor workaround

Implements **BoxGhostBC** (p. 135).

Reimplemented in **edgeDirichletBC** (p. 220).

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

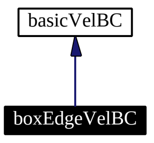
- **edgeGhostBC.H**

6.25 boxEdgeVelBC Class Reference

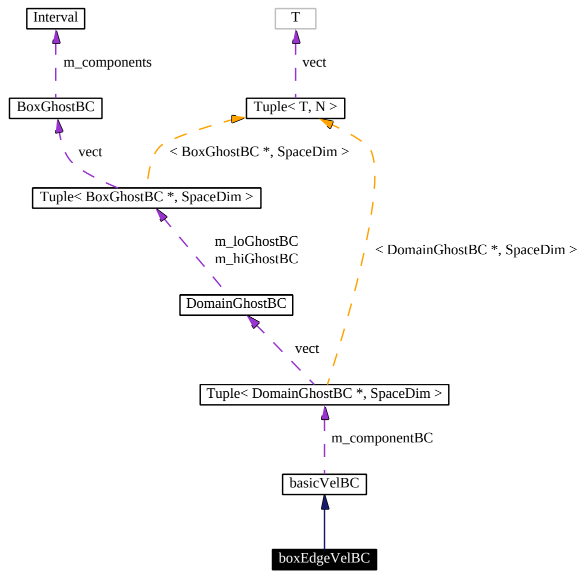
implements edge-centered velocity BC's for flow in a box!

`#include <boxVelBC.H>`

Inheritance diagram for boxEdgeVelBC:



Collaboration diagram for boxEdgeVelBC:



Public Methods

- **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, const 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, const int a_velComp) const

  this one assumes each edge-centered FAB is one component (old interface)

Protected Attributes

- bool **m_isViscous**

6.25.1 Detailed Description

implements edge-centered velocity BC's for flow in a box!

6.25.2 Constructor & Destructor Documentation

6.25.2.1 boxEdgeVelBC::boxEdgeVelBC (bool *isViscous* = false)

6.25.2.2 virtual boxEdgeVelBC::~boxEdgeVelBC () `[inline, virtual]`

6.25.3 Member Function Documentation

6.25.3.1 virtual void boxEdgeVelBC::applyBCs (FArrayBox & *a_state*, const Box & *a_domain*, const Real *a_dx*, const int *a_velComp*) const `[virtual]`

this one assumes each edge-centered FAB is one component (old interface)

Reimplemented from **basicVelBC** (p. 98).

6.25.3.2 virtual void boxEdgeVelBC::applyBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, const Real *a_dx*, const int *a_velComp*) const [virtual]

this one assumes that each edge-centered FAB is one component

Reimplemented from **basicVelBC** (p. 98).

6.25.3.3 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. 99).

6.25.3.4 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. 99).

6.25.3.5 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)

6.25.3.6 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

6.25.3.7 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)

6.25.3.8 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

6.25.3.9 void boxEdgeVelBC::isViscous (bool *a_isViscous*)

6.25.3.10 bool boxEdgeVelBC::isViscous ()

access function

6.25.3.11 virtual void boxEdgeVelBC::setAllBCTypes () [virtual]

6.25.4 Member Data Documentation

6.25.4.1 bool boxEdgeVelBC::m_isViscous [protected]

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

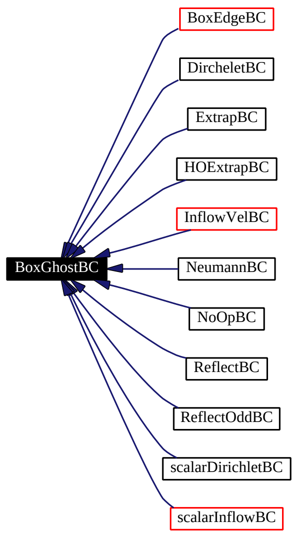
- **boxVelBC.H**

6.26 BoxGhostBC Class Reference

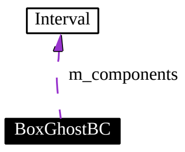
class to encapsulate the operations of ghost-cell boundary conditions.

`#include <GhostBC.H>`

Inheritance diagram for BoxGhostBC:



Collaboration diagram for BoxGhostBC:



Public Methods

- virtual ~**BoxGhostBC** ()
- **BoxGhostBC** ()
- **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)

Protected Methods

- virtual BoxGhostBC ∗ **new_boxghostbc** () const=0

  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 **Box** &a_domain, **Real** a_dx) const
- virtual void **applyHomogeneousBCs** (**FArrayBox** &a_state, const **Box** &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** &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** &domain) const=0

Protected Attributes

- **Side::LoHiSide m_side**
- int **m_direction**
- **Interval m_components**

Friends

- class **DomainGhostBC**

6.26.1 Detailed Description

class to encapsulate the operations of ghost-cell boundary conditions.

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. The derived class provides a function to to fill A B and C. These functions are only meant to be called by **DomainGhostBC** (p. 215).

6.26.2 Constructor & Destructor Documentation

6.26.2.1 virtual BoxGhostBC::~BoxGhostBC () `[inline, virtual]`

6.26.2.2 BoxGhostBC::BoxGhostBC () `[inline]`

6.26.2.3 BoxGhostBC::BoxGhostBC (int *a_dir*, Side::LoHiSide *a_sd*)

sets interval to be (0,0)

6.26.2.4 BoxGhostBC::BoxGhostBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*)

6.26.3 Member Function Documentation

6.26.3.1 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]`

Reimplemented in **BoxEdgeBC** (p. 126).

6.26.3.2 virtual void BoxGhostBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented in **BoxEdgeBC** (p. 126), **NoOpBC** (p. 351), **ExtrapBC** (p. 223), **HOExtrapBC** (p. 255), **ReflectBC** (p. 442), and **ReflectOddBC** (p. 445).

6.26.3.3 virtual void BoxGhostBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented in **BoxEdgeBC** (p. 126), **NoOpBC** (p. 351), **ExtrapBC** (p. 223), **HOExtrapBC** (p. 254), **ReflectBC** (p. 442), and **ReflectOddBC** (p. 445).

6.26.3.4 virtual void BoxGhostBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented in **BoxEdgeBC** (p. 126), **NoOpBC** (p. 352), **ExtrapBC** (p. 223), **HOExtrapBC** (p. 255), **ReflectBC** (p. 442), and **ReflectOddBC** (p. 445).

6.26.3.5 virtual void BoxGhostBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented in **BoxEdgeBC** (p. 126), **NoOpBC** (p. 351), **ExtrapBC** (p. 223), **HOExtrapBC** (p. 255), **ReflectBC** (p. 442), and **ReflectOddBC** (p. 445).

6.26.3.6 void BoxGhostBC::define (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*) `[protected]`

Reimplemented in **BoxEdgeBC** (p. 126), and **NoOpBC** (p. 352).

6.26.3.7 void BoxGhostBC::define (int *a_dir*, Side::LoHiSide *a_sd*) `[protected]`

sets interval to be (0,0)

Reimplemented in **BoxEdgeBC** (p. 126), and **NoOpBC** (p. 352).

6.26.3.8 virtual void BoxGhostBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *domain*) const [protected, pure virtual]

Implemented in **InflowVelBC** (p. 261), **BoxEdgeBC** (p. 127), **edgeDirichletBC** (p. 219), **NoOpBC** (p. 352), **ExtrapBC** (p. 223), **HOExtrapBC** (p. 255), **ReflectBC** (p. 443), **ReflectOddBC** (p. 446), and **NeumannBC** (p. 348).

6.26.3.9 virtual void BoxGhostBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *domain*) const [protected, pure virtual]

Implemented in **InflowVelBC** (p. 261), **BoxEdgeBC** (p. 127), **edgeDirichletBC** (p. 219), **NoOpBC** (p. 352), **ExtrapBC** (p. 223), **HOExtrapBC** (p. 255), **ReflectBC** (p. 443), **ReflectOddBC** (p. 446), **NormalInflowVelBC** (p. 356), and **NeumannBC** (p. 348).

6.26.3.10 virtual BoxGhostBC* BoxGhostBC::new_boxghostbc () const [protected, pure virtual]

virtual constructor workaround.

Implemented in **InflowVelBC** (p. 262), **BoxEdgeBC** (p. 127), **edgeDirichletBC** (p. 220), **NoOpBC** (p. 352), **ExtrapBC** (p. 223), **HOExtrapBC** (p. 255), **ReflectBC** (p. 443), **ReflectOddBC** (p. 446), **NormalInflowVelBC** (p. 356), and **NeumannBC** (p. 348).

6.26.4 Friends And Related Function Documentation

6.26.4.1 friend class DomainGhostBC [friend]

Reimplemented in **InflowVelBC** (p. 262), **edgeDirichletBC** (p. 220), **ExtrapBC** (p. 224), **HOExtrapBC** (p. 255), **ReflectBC** (p. 443), **ReflectOddBC** (p. 446), **scalarDirichletBC** (p. 448), **scalarInflowBC** (p. 450), **scalarInflowChannelBC** (p. 452), **NormalInflowVelBC** (p. 356), **NeumannBC** (p. 348), and **DircheletBC** (p. 209).

6.26.5 Member Data Documentation

6.26.5.1 Interval BoxGhostBC::m_components [protected]

6.26.5.2 int BoxGhostBC::m_direction [protected]

6.26.5.3 Side::LoHiSide BoxGhostBC::m_side [protected]

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

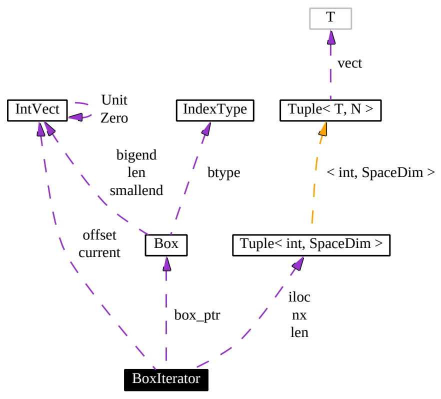
- **GhostBC.H**

6.27 BoxIterator Class Reference

iterates through the IntVects of a **Box** (p. 106)

`#include <BoxIterator.H>`

Collaboration diagram for BoxIterator:



Public Methods

- **BoxIterator** ()

  {\bf *Constructors, destructors, defines*}

- **BoxIterator** (const **Box** &bx)
- void **setBox** (const **Box** &bx)
- void **define** (const **Box** &bx)
- **BoxIterator** (const BoxIterator &tx)
- **~BoxIterator** ()
- void **begin** ()

  {\bf *Modification functions*}

- void **reset** ()
- void **operator++** ()
- void **next** ()
- const **IntVect** & **operator()** () const
- bool **ok** ()

Protected Attributes

- **Box box_ptr**
- **Tuple**< int, **SpaceDim** > **nx**
- **Tuple**< int, **SpaceDim** > **iloc**
- **Tuple**< int, **SpaceDim** > **len**
- **IntVect current**
- **IntVect offset**
- int **icur**

6.27.1 Detailed Description

iterates through the IntVects of a **Box** (p. 106)

BoxIterator iterates through the IntVects of a box. The actual sqeuence of IntVects is implementation-specific. Typical usage:\

\begin{verbatim} **Box** (p. 106) b; ... BoxIterator bit (b); for (bit.begin(); bit.ok(); ++bit) { **IntVect** (p. 264) iv = bit(); (do operations involving iv) } \end{verbatim}

6.27.2 Constructor & Destructor Documentation

6.27.2.1 BoxIterator::BoxIterator () `[inline]`

{\bf Constructors, destructors, defines}

Default constructor. This constructs an invalid iterator. The user must call define before using.

6.27.2.2 BoxIterator::BoxIterator (const Box & *bx*) `[inline]`

Constructs a BoxIterator and associates it with a **Box** (p. 106).\ {\bf Arguments:}\ bx (not modified) the **Box** (p. 106) to iterate over.\ {\bf Returns:} none.\ {\bf This:}\ This object is modified.

6.27.2.3 BoxIterator::BoxIterator (const BoxIterator & *tx*) `[inline]`

Copy constructor.\ {\bf Arguments:}\ tx (not modified) the BoxIterator to copy.\ {\bf Returns:} none.\ {\bf This:}\ This object is modified.

6.27.2.4 BoxIterator::~BoxIterator () `[inline]`

Destructor.

6.27.3 Member Function Documentation

6.27.3.1 void BoxIterator::begin () `[inline]`

{\bf Modification functions}

Sets this BoxIterator to the first **IntVect** (p. 264) in its **Box** (p. 106). The definition of the "first" **IntVect** (p. 264) is implementation-dependent.\ {\bf Arguments:} none.\ {\bf Returns:} none.\ {\bf This:}\ This object is modified.

6.27.3.2 void BoxIterator::define (const Box & *bx*) `[inline]`

Associates a **Box** (p. 106) with this BoxIterator.\ {\bf Arguments:}\ bx (not modified) the **Box** (p. 106) to iterate over. \ {\bf Returns:} none.\ {\bf This:}\ This object is modified.

6.27.3.3 void BoxIterator::next () [inline]

6.27.3.4 bool BoxIterator::ok () [inline]

Returns true if this BoxIterator's location is within its **Box** (p. 106).\ {\bf Arguments:} none.\ {\bf Returns:} \ true if this BoxIterator's current location is in its **Box** (p. 106).\ {\bf This:}\ This object is not modified.

6.27.3.5 const IntVect& BoxIterator::operator() () const [inline]

Returns the value of the InVect for the current location of this BoxIterator.\ {\bf Arguments:} none.\ {\bf Returns:} \ the value of the intvect for the current location.\ {\bf This:}\ This object is not modified.

6.27.3.6 void BoxIterator::operator++ () [inline]

Modifies this BoxIterator to set it to the next location in its **Box** (p. 106). The definition of the "next location" of a **Box** (p. 106) is implementation-dependent.\ {\bf Arguments:} none. \ {\bf Returns:} none. \ {\bf This:} \This object is modified.

6.27.3.7 void BoxIterator::reset () [inline]

Sets this BoxIterator to the first **IntVect** (p. 264) in its **Box** (p. 106). The definition of the "first" **IntVect** (p. 264) is implementation-dependent.\ {\bf Arguments:} none.\ {\bf Returns:} none.\ {\bf This:}\ This object is modified.

6.27.3.8 void BoxIterator::setBox (const Box & *bx*) [inline]

6.27.4 Member Data Documentation

6.27.4.1 Box BoxIterator::box_ptr [protected]

6.27.4.2 IntVect BoxIterator::current [protected]

6.27.4.3 int BoxIterator::icur [protected]

6.27.4.4 Tuple<int, SpaceDim> BoxIterator::iloc [protected]

6.27.4.5 Tuple<int, SpaceDim> BoxIterator::len [protected]

6.27.4.6 Tuple<int, SpaceDim> BoxIterator::nx [protected]

6.27.4.7 IntVect BoxIterator::offset [protected]

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

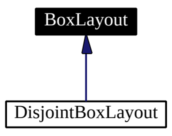
- **BoxIterator.H**

6.28 BoxLayout Class Reference

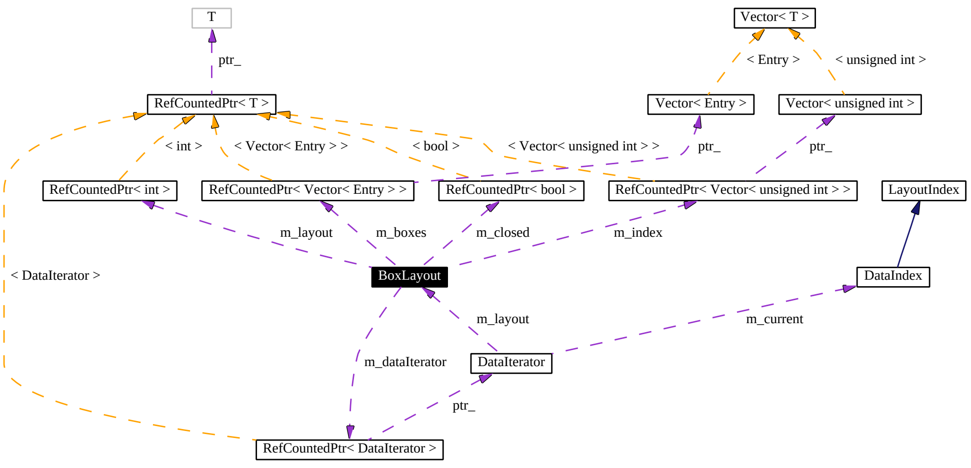
A not-necessarily disjoint collective of boxes.

`#include <BoxLayout.H>`

Inheritance diagram for BoxLayout:



Collaboration diagram for BoxLayout:



Public Methods

- **BoxLayout** ()
- **BoxLayout** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs)
- virtual ~**BoxLayout** ()
- BoxLayout & **operator=** (const BoxLayout &a_rhs)

  refcounted assignment

- virtual void **define** (const **Vector**< **Box** > &a_boxes, const **Vector**< int > &a_procIDs)
- const **Box** & **operator**[] (const **LayoutIndex** &it) const
- **Box** & **operator**[] (const **LayoutIndex** &it)
- bool **operator==** (const BoxLayout &rhs) const
- bool **eq** (const BoxLayout &rhs) const
- bool **check** (const **LayoutIndex** &index) const
- **DataIndex addBox** (const **Box** &box, int procID)
- virtual void **close** ()
- virtual void **deepCopy** (const BoxLayout &a_source)
- **Box get** (const **LayoutIndex** &it) const

- bool **isClosed** () const
- bool **isSorted** () const
- **DataIterator dataIterator** () const
- **LayoutIterator layoutIterator** () const
- void **print** () const

  *invokes cout<<*this; pretty-print dump of BoxLayout*

- void **p** () const
- unsigned int **procID** (const **LayoutIndex** &a_index) const
- int **numBoxes** (const int procID) const
- void **setProcID** (const **LayoutIndex** &a_index, unsigned int a_procID)
- unsigned int **size** () const
- void **sort** ()
- unsigned int **index** (const **LayoutIndex** &index) const

Protected Methods

- 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**
- void **coarsen** (BoxLayout &output, const BoxLayout &input, int refinement)
- void **refine** (BoxLayout &output, const BoxLayout &input, int refinement)

6.28.1 Detailed Description

A not-necessarily disjoint collective of boxes.

A BoxLayout is a collection of **Box** (p. 106) objects that are assigned to a process number. Each box is associated with only one process. processes are numbered from 0 to n-1 (for an n process parallel job)

BoxLayout can be either open or closed. null construction or deepcopy creates an open BoxLayout. Boxes may be added to an open BoxLayout. non-const operations may be performed on the boxes existing in an open BoxLayout. closed BoxLayouts are sorted. Many uses of BoxLayouts require a closed BoxLayout. A closed BoxLayout cannot be modified.

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 instaniations make use of a single implementation of that object and keep a tally of how many instantiations are sharing. Thus

\begin{verbatim} 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 \end{verbatim}

6.28.2 Constructor & Destructor Documentation

6.28.2.1 BoxLayout::BoxLayout ()

6.28.2.2 BoxLayout::BoxLayout (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*)

Construct from a **Vector** (p. 471) of Boxes and a **Vector** (p. 471) of processor assignments. On exit, the BoxLayout will be closed.

6.28.2.3 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

6.28.3 Member Function Documentation

6.28.3.1 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, otherwise a runtime error will occur.

6.28.3.2 bool BoxLayout::check (const LayoutIndex & *index*) const `[inline]`

6.28.3.3 virtual void BoxLayout::close () `[virtual]`

Mark this BoxLayout as complete and unchangeable.

Reimplemented in **DisjointBoxLayout** (p. 212).

6.28.3.4 DataIterator BoxLayout::dataIterator () const

6.28.3.5 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); \begin{verbatim} 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 \end{verbatim}

Reimplemented in **DisjointBoxLayout** (p. 212).

6.28.3.6 virtual void BoxLayout::define (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*) `[virtual]`

Define this BoxLayout from a **Vector** (p. 471) of Boxes and a **Vector** (p. 471) of processor assignments. Any pre-existing layout will be lost (ref-count dropped by one). The processor assignment **Vector** (p. 471) must be the same length as the **Box** (p. 106) **Vector** (p. 471). On exit, the BoxLayout will be closed.

Reimplemented in **DisjointBoxLayout** (p. 213).

6.28.3.7 bool BoxLayout::eq (const BoxLayout & *rhs*) const `[inline]`

6.28.3.8 Box BoxLayout::get (const LayoutIndex & *it*) const `[inline]`

As a consquence 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.

6.28.3.9 unsigned int BoxLayout::index (const LayoutIndex & *index*) const `[inline]`

6.28.3.10 bool BoxLayout::isClosed () const `[inline]`

return true if **close**() (p. 141) has been called. closed BoxLayout is always sorted.

6.28.3.11 bool BoxLayout::isSorted () const `[inline]`

return true if **sort**() (p. 143) has been called

6.28.3.12 LayoutIterator BoxLayout::layoutIterator () const

If BoxLayout is closed, then **LayoutIterator** (p. 291) will return the **Box** (p. 106)'s in sorted order. Processes through ALL the boxes in a BoxLayout

6.28.3.13 int BoxLayout::numBoxes (const int *procID*) const

6.28.3.14 BoxLayout& BoxLayout::operator= (const BoxLayout & *a_rhs*)

refcounted assignment

6.28.3.15 bool BoxLayout::operator== (const BoxLayout & *rhs*) const `[inline]`

Refcounted pointer check. True if these two objects share the same implementation.

6.28.3.16 Box & BoxLayout::operator[] (const LayoutIndex & *it*) `[inline]`

6.28.3.17 const Box & BoxLayout::operator[] (const LayoutIndex & *it*) const `[inline]`

const accessor operator. see also {\tt get(const DataIndex&)}

6.28.3.18 void BoxLayout::p () const `[inline]`

6.28.3.19 void BoxLayout::print () const

invokes cout<<*this; pretty-print dump of BoxLayout

6.28.3.20 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** (p. 196) class.

6.28.3.21 void BoxLayout::setIndexVector () `[protected]`

6.28.3.22 void BoxLayout::setProcID (const LayoutIndex & *a_index*, unsigned int *a_procID*) `[inline]`

Assign a **Box** (p. 106) in the BoxLayout to a processor. Requires the BoxLayout to be open.

6.28.3.23 unsigned int BoxLayout::size () const `[inline]`

6.28.3.24 void BoxLayout::sort ()

Sort the boxes.

6.28.4 Friends And Related Function Documentation

6.28.4.1 void coarsen (BoxLayout & *output*, const BoxLayout & *input*, int *refinement*) [friend]

'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.

6.28.4.2 friend class DataIterator [friend]

6.28.4.3 friend class LayoutIterator [friend]

6.28.4.4 void refine (BoxLayout & *output*, const BoxLayout & *input*, int *refinement*) [friend]

see '**coarsen** (p. 144) ' and substitute the word "refine" for "coarsen"

6.28.5 Member Data Documentation

6.28.5.1 RefCountedPtr<Vector<Entry> > BoxLayout::m_boxes [protected]

6.28.5.2 RefCountedPtr<bool> BoxLayout::m_closed [protected]

6.28.5.3 RefCountedPtr<DataIterator> BoxLayout::m_dataIterator [protected]

6.28.5.4 RefCountedPtr<Vector<unsigned int> > BoxLayout::m_index [protected]

6.28.5.5 RefCountedPtr<int> BoxLayout::m_layout [protected]

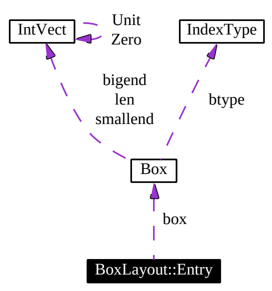
The documentation for this class was generated from the following file:

- **BoxLayout.H**

6.29 BoxLayout::Entry Struct Reference

`#include <BoxLayout.H>`

Collaboration diagram for BoxLayout::Entry:



Public Methods

- **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**

6.29.1 Constructor & Destructor Documentation

6.29.1.1 BoxLayout::Entry::Entry () [inline]

6.29.1.2 BoxLayout::Entry::Entry (const Box & *a_box*) [inline]

6.29.1.3 BoxLayout::Entry::Entry (const Box & *a_box*, const unsigned int *a_index*) [inline]

6.29.2 Member Function Documentation

6.29.2.1 bool BoxLayout::Entry::operator< (const Entry & *rhs*) const [inline]

6.29.3 Member Data Documentation

6.29.3.1 Box BoxLayout::Entry::box

6.29.3.2 unsigned int BoxLayout::Entry::index

6.29.3.3 unsigned int BoxLayout::Entry::m_procID

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

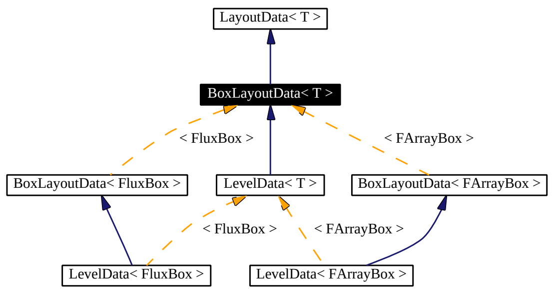
- **BoxLayout.H**

6.30 BoxLayoutData< T > Class Template Reference

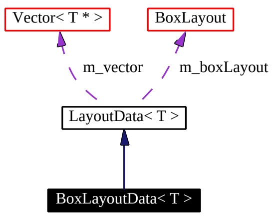
Data on a **BoxLayout** (p. 139).

`#include <BoxLayoutData.H>`

Inheritance diagram for BoxLayoutData< T >:



Collaboration diagram for BoxLayoutData< T >:



Public Methods

- **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
- void **setVal** (void(∗setFunction)(const **Box** &box, int comps, T &t))
- virtual bool **isDefined** () const

Protected Methods

- 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** >

6.30.1 Detailed Description

template<class T> class BoxLayoutData< T >

Data on a **BoxLayout** (p. 139).

A collection of **Box** (p. 106)-oriented objects. The arrangement of Boxes is given by the underlying **BoxLayout** (p. 139) object. BoxLayoutData attempt to prevent users from maniupulating the template class T to become out of synch with the boxes in the **BoxLayout** (p. 139). Caveat emptor.

All access to the data in a BoxLayoutData is forced to be data-parallel, by use of the **DataIterator** (p. 196).

class T must provide the following methods: \begin{itemize} \item {\tt T()} \item {\tt T(const Box& box, int comps)} \item {\tt define(const Box& box, int comps)} \item {\tt 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

\item {\tt static int preAllocatable()} returns 0 if the size(...) function is strictly a function of **Box** (p. 106) and **Interval** (p. 263), 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. \item {\tt int size(const Box& R, const Interval& comps)} \item {\tt void linearOut(void* buf, const Box& R, const Interval& comps)} \item {\tt void linearIn(void* buf, const Box& R, const Interval& comps)} \end{itemize}

In sequential codes, many of these functions will not be called, but in the general parallel case, they all will be.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 template<class T> BoxLayoutData< T >::BoxLayoutData () `[inline]`

6.30.2.2 template<class T> virtual BoxLayoutData< T >::~BoxLayoutData () `[inline, virtual]`

6.30.2.3 template<class T> BoxLayoutData< T >::BoxLayoutData (const BoxLayout & *boxes*, int *comps*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) `[inline]`

6.30.3 Member Function Documentation

6.30.3.1 template<class T> void BoxLayoutData< T >::allocateGhostVector (const DataFactory< T > & *factory*, const IntVect & *ghost* = IntVect::TheZeroVector()) `[inline, protected]`

6.30.3.2 template<class T> void BoxLayoutData< T >::define (const BoxLayout & *boxes*) `[inline, virtual]`

overridden and throws an error.

Reimplemented from **LayoutData< T >** (p. 288).

Reimplemented in **LevelData< T >** (p. 298), **LevelData< FluxBox >** (p. 298), and **LevelData< FArrayBox >** (p. 298).

6.30.3.3 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** (p. 263) comps.

Reimplemented in **LevelData< T >** (p. 298), **LevelData< FluxBox >** (p. 298), and **LevelData< FArrayBox >** (p. 298).

6.30.3.4 template<class T> void BoxLayoutData< T >::define (const BoxLayoutData< T > & *da*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) `[inline, virtual]`

Reimplemented in **LevelData< T >** (p. 298), **LevelData< FluxBox >** (p. 298), and **LevelData< FArrayBox >** (p. 298).

6.30.3.5 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. 298), **LevelData< FluxBox >** (p. 298), and **LevelData< FArrayBox >** (p. 298).

6.30.3.6 template<class T> Interval BoxLayoutData< T >::interval () const [inline]

6.30.3.7 template<class T> bool BoxLayoutData< T >::isDefined () const [inline, virtual]

6.30.3.8 template<class T> int BoxLayoutData< T >::nComp () const [inline]

6.30.3.9 template<class T> void BoxLayoutData< T >::setVal (void(* *setFunction*)(const Box &box, int comps, T &t)) [inline]

6.30.3.10 template<class T> void BoxLayoutData< T >::setVector (const BoxLayoutData< T > & *da*, const Interval & *srcComps*, const Interval & *destComps*) [inline, protected]

6.30.4 Friends And Related Function Documentation

6.30.4.1 template<class T> friend class LevelData< T > [friend]

6.30.5 Member Data Documentation

6.30.5.1 template<class T> int BoxLayoutData< T >::m_comps [protected]

6.30.5.2 template<class T> bool BoxLayoutData< T >::m_isdefined [protected]

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

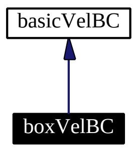
- **BoxLayoutData.H**

6.31 boxVelBC Class Reference

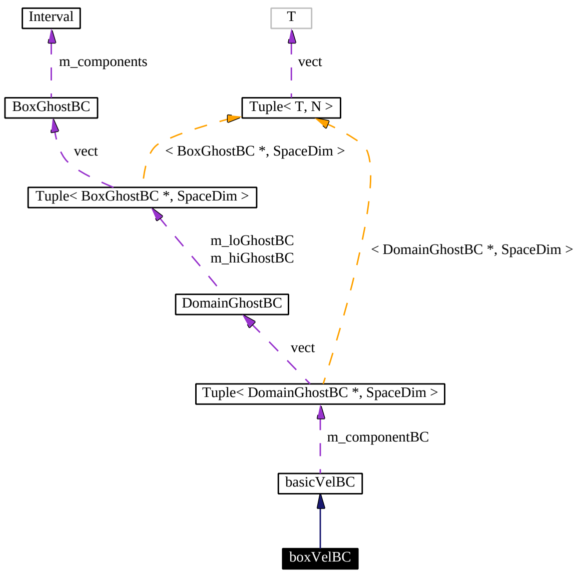
implements cell-centered velocity BC's for flow in a box!

`#include <boxVelBC.H>`

Inheritance diagram for boxVelBC:



Collaboration diagram for boxVelBC:



Public Methods

- **boxVelBC** (bool isViscous=false)
- virtual **~boxVelBC** ()
- virtual void **setAllBCTypes** ()
- bool **isViscous** ()

  access function

- void **isViscous** (bool a_isViscous)

Protected Attributes

- bool **m_isViscous**

6.31.1 Detailed Description

implements cell-centered velocity BC's for flow in a box!

6.31.2 Constructor & Destructor Documentation

6.31.2.1 boxVelBC::boxVelBC (bool *isViscous* = false)

6.31.2.2 virtual boxVelBC::~boxVelBC () `[inline, virtual]`

6.31.3 Member Function Documentation

6.31.3.1 void boxVelBC::isViscous (bool *a_isViscous*)

6.31.3.2 bool boxVelBC::isViscous ()

access function

6.31.3.3 virtual void boxVelBC::setAllBCTypes () `[virtual]`

6.31.4 Member Data Documentation

6.31.4.1 bool boxVelBC::m_isViscous `[protected]`

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

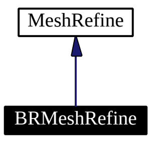
- **boxVelBC.H**

6.32 BRMeshRefine Class Reference

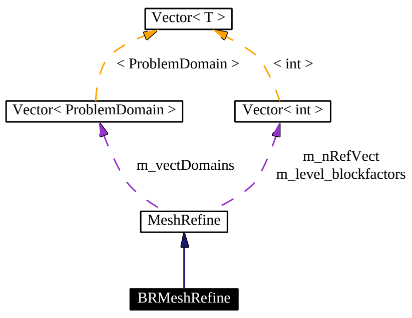
class which manages Berger-Rigoutsos grid generation

`#include <BRMeshRefine.H>`

Inheritance diagram for BRMeshRefine:



Collaboration diagram for BRMeshRefine:



Public Methods

- **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 Methods

- void **splitBox** (**Vector**< **Box** > &a_boxes,const int a_boxindex, const int a_dimension,const int a_maxboxsize) const
- **Vector**< int > **makeTrace** (const **IntVectSet** &a_Ivs,int a_dir) 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

6.32.1 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** (p. 335) class from which it is derived

There are two ways grids can be defined based on tagged cells. one takes a single **IntVectSet** (p. 274) of tags defined on the BaseLevel mesh and uses that set of tags for every level to be refined; the other takes a **Vector** (p. 471)<**IntVectSet** (p. 274)> of tags defined on all the mesh levels to be refined and uses those. \ \

{\bf 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** (p. 274) of tags is passed (instead of a **Vector** (p. 471)<**IntVectSet** (p. 274)>) 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.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 BRMeshRefine::BRMeshRefine ()

default constructor – leaves object in an unusable state

6.32.2.2 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.

6.32.2.3 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.

6.32.2.4 virtual BRMeshRefine::~BRMeshRefine () `[virtual]`

destructor

6.32.3 Member Function Documentation

6.32.3.1 void BRMeshRefine::breakBoxes (Vector< Box > & *a_vboxin*, const int & *a_maxSize*, const int & *a_idir*) const `[protected]`

6.32.3.2 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 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. 339).

6.32.3.3 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. 340).

6.32.3.4 int BRMeshRefine::findMaxInflectionPoint (const Vector< int > & *a_trace*, int & *a_maxVal*) const [protected]

6.32.3.5 int BRMeshRefine::findSplit (const Vector< int > & *a_trace*) const [protected]

6.32.3.6 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 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

Implements **MeshRefine** (p. 340).

6.32.3.7 Vector<int> BRMeshRefine::makeTrace (const IntVectSet & *a_Ivs*, int *a_dir*) const [protected]

6.32.3.8 int BRMeshRefine::maxloc (const int * *a_V*, const int *a_Size*) const [protected]

6.32.3.9 void BRMeshRefine::splitBox (Vector< Box > & *a_boxes*, const int *a_boxindex*, const int *a_dimension*, const int *a_maxboxsize*) const [protected]

6.32.3.10 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]

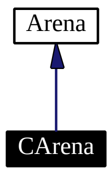
The documentation for this class was generated from the following file:

- **BRMeshRefine.H**

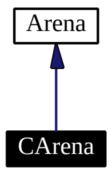
6.33 CArena Class Reference

#include <Arena.H>

Inheritance diagram for CArena:



Collaboration diagram for CArena:



Public Types

- enum { **DefaultHunkSize** = 1024*1024 }

Public Methods

- **CArena** (size_t hunk_size=0)
- virtual ~**CArena** ()
- virtual void * **alloc** (size_t nbytes)
- 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**

6.33.1 Member Typedef Documentation

6.33.1.1 typedef set< Node > CArena::NL [protected]

6.33.2 Member Enumeration Documentation

6.33.2.1 anonymous enum

Enumeration values:
DefaultHunkSize

6.33.3 Constructor & Destructor Documentation

6.33.3.1 CArena::CArena (size_t *hunk_size* = 0)

6.33.3.2 virtual CArena::~CArena () [virtual]

6.33.4 Member Function Documentation

6.33.4.1 virtual void* CArena::alloc (size_t *nbytes*) [virtual]

Implements **Arena** (p. 76).

6.33.4.2 void* CArena::calloc (size_t *nmemb*, size_t *size*)

6.33.4.3 virtual void CArena::free (void * *ap*) [virtual]

Implements **Arena** (p. 77).

6.33.4.4 void* CArena::realloc (void * *ptr*, size_t *size*)

6.33.5 Member Data Documentation

6.33.5.1 std::vector<void*> CArena::m_alloc [protected]

6.33.5.2 NL CArena::m_busylist [protected]

6.33.5.3 NL CArena::m_freelist [protected]

6.33.5.4 size_t CArena::m_hunk [protected]

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

- **Arena.H**

6.34 CArena::Node Class Reference

`#include <Arena.H>`

Public Methods

- **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)

6.34.1 Constructor & Destructor Documentation

6.34.1.1 CArena::Node::Node () `[inline]`

6.34.1.2 CArena::Node::Node (void * *block*, size_t *size*) `[inline]`

6.34.1.3 CArena::Node::Node (const Node & *rhs*) `[inline]`

6.34.2 Member Function Documentation

6.34.2.1 void CArena::Node::block (void * *blk*) `[inline]`

6.34.2.2 void* CArena::Node::block () const `[inline]`

6.34.2.3 bool CArena::Node::operator< (const Node & *rhs*) const `[inline]`

6.34.2.4 Node& CArena::Node::operator= (const Node & *rhs*) `[inline]`

6.34.2.5 bool CArena::Node::operator== (const Node & *rhs*) const `[inline]`

6.34.2.6 void CArena::Node::size (size_t *sz*) `[inline]`

6.34.2.7 size_t CArena::Node::size () const `[inline]`

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

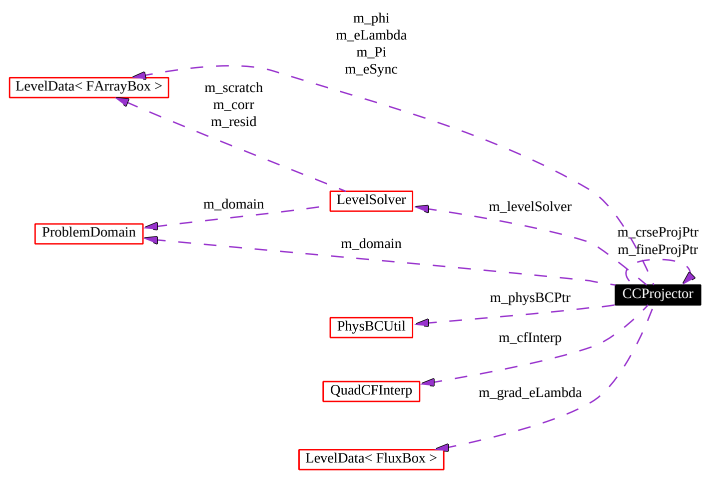
- **Arena.H**

6.35 CCProjector Class Reference

this class manages the various forms of the CC projection

`#include <CCProjector.H>`

Collaboration diagram for CCProjector:



Public Methods

- **CCProjector** ()
- **CCProjector** (const **DisjointBoxLayout** &a_grids, 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 **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 **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 **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 **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 Methods

- 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 **defineSolver** (**AMRSolver** &a_solver, const **Vector**< **LevelData**< **FArrayBox** > ∗ > &a_vel, bool a_freestreamSolver)
- void **postRestart** ()

  takes care of setting ghost cell values after restarting

6.35.1 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.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 CCProjector::CCProjector ()

6.35.2.2 CCProjector::CCProjector (const DisjointBoxLayout & *a_grids*, 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

6.35.2.3 CCProjector::CCProjector (const DisjointBoxLayout & *a_grids*, 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

6.35.2.4 CCProjector::∼CCProjector ()

destructor

6.35.3 Member Function Documentation

6.35.3.1 void CCProjector::applyMacCorrection (LevelData< FluxBox > & *a_uEdge*, Real *CFscale*) `[protected]`

returns uEdge - G(phi_mac); C/F BC is CFscale∗Pi(crse)

6.35.3.2 void CCProjector::applySyncCorrection (Vector< LevelData< FArrayBox > ∗ > & *a_velocity*, const Real *scale*, LevelData< FArrayBox > ∗ *crseCorr*) `[protected]`

6.35.3.3 void CCProjector::computeGrad_eLambda () `[protected]`

compute composite gradient of eLambda – does this recursively

6.35.3.4 void CCProjector::computeVDCorrection (Vector< LevelData< FArrayBox > * > & *a_lambda*, const Real *a_newTime*, const Real *a_dtSync*, AMRSolver & *solver*) [protected]

6.35.3.5 void CCProjector::correctCCVelocities (LevelData< FArrayBox > & *a_velocity*, const Real *scale*) const [protected]

vel := vel - scale*G_CC(pi)

6.35.3.6 CCProjector* CCProjector::crseProjPtr () const

6.35.3.7 void CCProjector::define (const DisjointBoxLayout & *a_grids*, 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

6.35.3.8 void CCProjector::define (const DisjointBoxLayout & *a_grids*, 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

6.35.3.9 void CCProjector::defineSolver (AMRSolver & *a_solver*, const Vector< LevelData< FArrayBox > * > & *a_vel*, bool *a_freestreamSolver*) [protected]

6.35.3.10 void CCProjector::doInitialSyncOperations (Vector< LevelData< FArrayBox > * > & *a_vel*, Vector< LevelData< FArrayBox > * > & *a_lambda*, const Real *a_newTime*, const Real *a_dtSync*)

6.35.3.11 bool CCProjector::doMacSync () const

do volume discrepancy correction?

6.35.3.12 void CCProjector::doPostRegridOps (Vector< LevelData< FArrayBox > * > & *a_velocity*, Vector< LevelData< FArrayBox > * > & *a_lambda*, const Real *a_dt*, const Real *a_time*)

6.35.3.13 bool CCProjector::doQuadInterp () const

use quadratic interpolation (instead of extrap) for c/f bc

6.35.3.14 void CCProjector::doSyncOperations (Vector< LevelData< FArrayBox > * > & *a_velocity*, Vector< LevelData< FArrayBox > * > & *a_lambda*, const Real *a_newTime*, const Real *a_dtSync*)

6.35.3.15 void CCProjector::doSyncProjection (Vector< LevelData< FArrayBox > * > & *a_velocity*, const Real *a_newTime*, const Real *a_dtSync*, AMRSolver & *solver*) `[protected]`

6.35.3.16 bool CCProjector::doSyncProjection () const

do sync projection?

6.35.3.17 const ProblemDomain& CCProjector::dProblem () const

6.35.3.18 Real CCProjector::dx () const

6.35.3.19 const LevelData<FArrayBox>& CCProjector::eLambda () const

returns volume-discrepancy correction (const version)

6.35.3.20 LevelData<FArrayBox>& CCProjector::eLambda ()

returns volume-discrepancy correction

6.35.3.21 const LevelData<FArrayBox>& CCProjector::eSync () const

returns synchronization correction (const version)

6.35.3.22 LevelData<FArrayBox>& CCProjector::eSync ()

returns synchronization correction

6.35.3.23 Real CCProjector::etaLambda () const

returns coefficient for volume-discrepancy solve.

6.35.3.24 CCProjector* CCProjector::fineProjPtr () const

6.35.3.25 const DisjointBoxLayout& CCProjector::getBoxes () const

6.35.3.26 int CCProjector::getLevel () const

access functions

6.35.3.27 PhysBCUtil* CCProjector::getPhysBCPtr () const

6.35.3.28 const LevelData<FluxBox>& CCProjector::grad_eLambda () const

6.35.3.29 LevelData<FluxBox>& CCProjector::grad_eLambda ()

6.35.3.30 void CCProjector::grad_eLambda (LevelData< FArrayBox > & *a_grad_eLambda*, int *a_dir*) const

returns edge-centered grad(eLambda) in direction dir

6.35.3.31 void CCProjector::grad_eSync (LevelData< FArrayBox > & *a_grad_eSync*) const

returns cell-centered G^{comp}(eSync)

6.35.3.32 void CCProjector::grad_eSync (LevelData< FArrayBox > & *a_grad_eSync*, int *a_dir*) const

returns cell-centered grad(eSync) (composite gradient)

6.35.3.33 void CCProjector::gradPhi (LevelData< FluxBox > & *a_gradPhi*) const

returns all components of grad(phi) (gradPhi should be correct size)

6.35.3.34 void CCProjector::gradPhi (LevelData< FArrayBox > & *a_gradPhi*, int *a_dir*) const

returns edge-centered grad(phi) in direction dir

6.35.3.35 void CCProjector::gradPi (LevelData< FArrayBox > & *a_gradPi*) const

6.35.3.36 void CCProjector::gradPi (LevelData< FArrayBox > & *a_gradPi*, int *a_dir*) const

returns grad(pi) in direction dir

6.35.3.37 void CCProjector::init (const CCProjector & *a_oldProj*)

initialize new projection with data from old projection

6.35.3.38 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

6.35.3.39 void CCProjector::initialSyncProjection (Vector< LevelData< FArrayBox > * > & *a_velocity*, const Real *a_newTime*, const Real *a_dtSync*, AMRSolver & *a_solver*) [protected]

perform initial sync projection

6.35.3.40 void CCProjector::initialVelocityProject (Vector< LevelData< FArrayBox > * > & *a_velocity*, AMRSolver & *a_solver*, bool *a_homogeneousCFBC* = true)

6.35.3.41 void CCProjector::initialVelocityProject (Vector< LevelData< FArrayBox > * > & *a_velocity*, bool *a_homogeneousCFBC* = true)

6.35.3.42 void CCProjector::isFinestLevel (bool *a_finest_level*)

set whether this is the finest level

6.35.3.43 bool CCProjector::isFinestLevel () const

is this the finest level?

6.35.3.44 bool CCProjector::isInitialized () const

has this object been completely initialized?

6.35.3.45 void CCProjector::levelMacProject (LevelData< FluxBox > & *uEdge*, Real *a_oldTime*, Real *a_dt*)

6.35.3.46 void CCProjector::LevelProject (LevelData< FArrayBox > & *a_velocity*, LevelData< FArrayBox > * *a_crseVelPtr*, const Real *a_newTime*, const Real *a_dt*)

6.35.3.47 int CCProjector::nRefCrse () const

6.35.3.48 const LevelData<FArrayBox>& CCProjector::phi () const

const version of accessor

6.35.3.49 LevelData<FArrayBox>& CCProjector::phi ()

returns MAC correction

6.35.3.50 LevelData<FArrayBox>& CCProjector::Pi ()

returns level-projection pressure

6.35.3.51 void CCProjector::postRestart () [protected]

takes care of setting ghost cell values after restarting

6.35.3.52 QuadCFInterp& CCProjector::quadCFInterpolator ()

returns predefined quadratic coarse-fine interpolation object

6.35.3.53 void CCProjector::readCheckpointHeader (HDF5Handle & *a_handle*)

read the checkpoint header

6.35.3.54 void CCProjector::readCheckpointLevel (HDF5Handle & *a_handle*)

read this class from a checkpoint file

6.35.3.55 void CCProjector::setCrseProj (CCProjector * *a_crseProj*, int *nRefCrse*)

6.35.3.56 void CCProjector::setFineProj (CCProjector * *a_fineProj*)

6.35.3.57 void CCProjector::setPhysBC (const PhysBCUtil & *a_bc*)

sets physBCs

6.35.3.58 void CCProjector::variableSetUp ()

define static parts

6.35.3.59 int CCProjector::verbosity () const

returns verbosity

6.35.3.60 void CCProjector::verbosity (int *a_verbosity*)

set verbosity

6.35.3.61 void CCProjector::writeCheckpointHeader (HDF5Handle & *a_handle*) const

write checkpoint header

6.35.3.62 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:

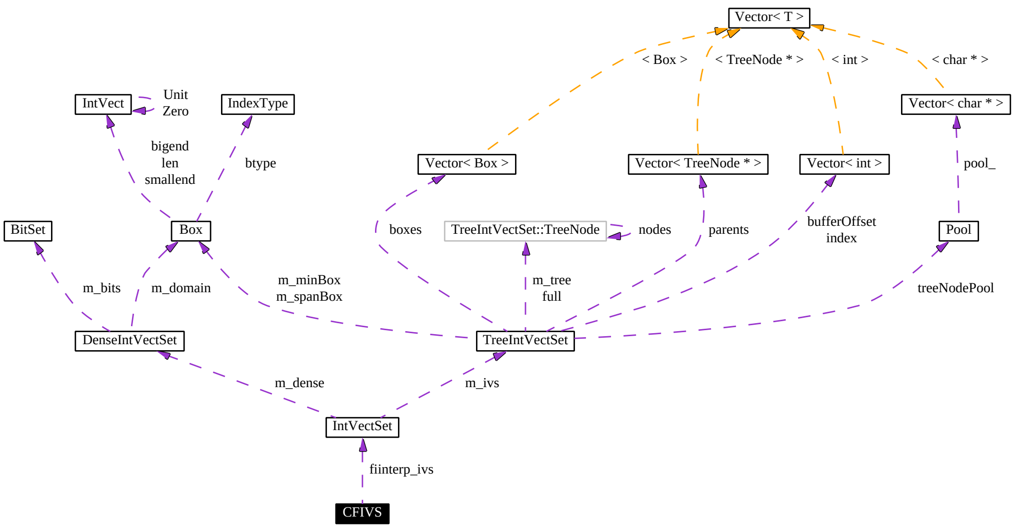
- **CCProjector.H**

6.36 CFIVS Class Reference

Class to get fine intvectset at CF interface.

`#include <CFIVS.H>`

Collaboration diagram for CFIVS:



Public Methods

- bool **isDefined** () const
- const **IntVectSet** & **getFineIVS** () const
- **CFIVS** ()
- ~**CFIVS** ()
- **CFIVS** (const **Box** &Domain, const **Box** &box_in, const **DisjointBoxLayout** &Levboxes, int Direction, **Side::LoHiSide** hiorlo)
- **CFIVS** (const **ProblemDomain** &Domain, const **Box** &box_in, const **DisjointBoxLayout** &Levboxes, int Direction, **Side::LoHiSide** hiorlo)
- void **define** (const **Box** &Domain, const **Box** &box_in, const **DisjointBoxLayout** &fine_boxes, int Direction, **Side::LoHiSide** hiorlo)
- void **define** (const **ProblemDomain** &Domain, const **Box** &box_in, const **DisjointBoxLayout** &fine_boxes, int Direction, **Side::LoHiSide** hiorlo)

Protected Methods

- void **setDefaultValues** ()

Protected Attributes

- **IntVectSet fiinterp_ivs**
- bool **isdefined**

6.36.1 Detailed Description

Class to get fine intvectset at CF interface.

Class to get fine intvectset at cf interface. This class should be considered internal to **AMRSolver** (p. 69) and should not be considered part of the Chombo API.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 CFIVS::CFIVS ()

6.36.2.2 CFIVS::~CFIVS ()

6.36.2.3 CFIVS::CFIVS (const Box & *Domain*, const Box & *box_in*, const DisjointBoxLayout & *Levboxes*, int *Direction*, Side::LoHiSide *hiorlo*)

6.36.2.4 CFIVS::CFIVS (const ProblemDomain & *Domain*, const Box & *box_in*, const DisjointBoxLayout & *Levboxes*, int *Direction*, Side::LoHiSide *hiorlo*)

6.36.3 Member Function Documentation

6.36.3.1 void CFIVS::define (const ProblemDomain & *Domain*, const Box & *box_in*, const DisjointBoxLayout & *fine_boxes*, int *Direction*, Side::LoHiSide *hiorlo*)

6.36.3.2 void CFIVS::define (const Box & *Domain*, const Box & *box_in*, const DisjointBoxLayout & *fine_boxes*, int *Direction*, Side::LoHiSide *hiorlo*)

6.36.3.3 const IntVectSet& CFIVS::getFineIVS () const

Get fine intvects which need to be interpolated. This will be empty if isEmpty() returns true

6.36.3.4 bool CFIVS::isDefined () const

6.36.3.5 void CFIVS::setDefaultValues () `[protected]`

6.36.4 Member Data Documentation

6.36.4.1 IntVectSet CFIVS::fiinterp_ivs `[protected]`

6.36.4.2 bool CFIVS::isdefined `[protected]`

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

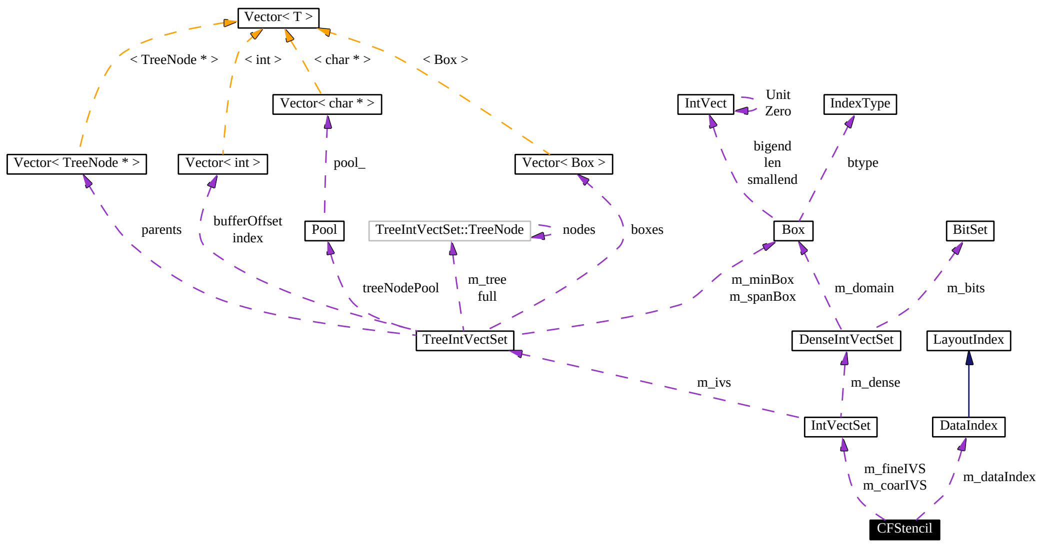
- **CFIVS.H**

6.37 CFStencil Class Reference

Class to encapsulate coarse-fine information for **AMR** (p. 17) operations.

```
#include <CFStencil.H>
```

Collaboration diagram for CFStencil:



Public Methods

- bool **isDefined** () const

  {\bf *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 Methods

- void **setDefaultValues** ()

Protected Attributes

- int **m_direction**
- **Side::LoHiSide m_hiorlo**
- **DataIndex m_dataIndex**
- **IntVectSet m_fineIVS**
- **IntVectSet m_coarIVS**
- bool **m_isDefined**

6.37.1 Detailed Description

Class to encapsulate coarse-fine information for **AMR** (p. 17) 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** (p. 426) and should not be considered part of the Chombo API.

6.37.2 Constructor & Destructor Documentation

6.37.2.1 CFStencil::CFStencil (const CFStencil & *cfs_in*)

6.37.2.2 CFStencil::CFStencil (const Box & *a_b*, int *a_nComp*)

6.37.2.3 CFStencil::CFStencil ()

6.37.2.4 CFStencil::∼CFStencil ()

6.37.2.5 CFStencil::CFStencil (const Box & *FineDomain*, const Box & *grid*, const DisjointBoxLayout & *Levboxes*, const DisjointBoxLayout & *LevCBoxes*, int *RefRatio*, int *Direction*, Side::LoHiSide *hiorlo*)

6.37.2.6 CFStencil::CFStencil (const ProblemDomain & *FineDomain*, const Box & *grid*, const DisjointBoxLayout & *Levboxes*, const DisjointBoxLayout & *LevCBoxes*, int *RefRatio*, int *Direction*, Side::LoHiSide *hiorlo*)

6.37.3 Member Function Documentation

6.37.3.1 void CFStencil::clear ()

6.37.3.2 void CFStencil::define (const ProblemDomain & *FineDomain*, const Box & *grid*, const DisjointBoxLayout & *fine_boxes*, const DisjointBoxLayout & *coar_boxes*, int *Refratio*, int *Direction*, Side::LoHiSide *hiorlo*)

6.37.3.3 void CFStencil::define (const Box & *FineDomain*, const Box & *grid*, const DisjointBoxLayout & *fine_boxes*, const DisjointBoxLayout & *coar_boxes*, int *Refratio*, int *Direction*, Side::LoHiSide *hiorlo*)

6.37.3.4 void CFStencil::define (const Box & *a_b*, int *a_nComp*)

6.37.3.5 const IntVectSet& CFStencil::getCoarIVS () const

get coarse intvects that underly fiinterpivs. This will be empty if **isEmpty**() (p. 175) returns true this is the same as getCoarInterpIVS

6.37.3.6 const IntVectSet& CFStencil::getFineIVS () const

get fine intvects which need to be interpolated \ This will be empty if **isEmpty**() (p. 175) returns true

6.37.3.7 bool CFStencil::isDefined () const

{\bf Access functions}

returns true if CFStencil is defined

6.37.3.8 bool CFStencil::isEmpty () const

are there any interpolation points? \ returns false if so.

6.37.3.9 CFStencil& CFStencil::operator= (const CFStencil & *cfs_in*)

6.37.3.10 void CFStencil::setDefaultValues () `[protected]`

6.37.4 Member Data Documentation

6.37.4.1 IntVectSet CFStencil::m_coarIVS `[protected]`

6.37.4.2 DataIndex CFStencil::m_dataIndex `[protected]`

6.37.4.3 int CFStencil::m_direction `[protected]`

6.37.4.4 IntVectSet CFStencil::m_fineIVS `[protected]`

6.37.4.5 Side::LoHiSide CFStencil::m_hiorlo `[protected]`

6.37.4.6 bool CFStencil::m_isDefined `[protected]`

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

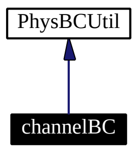
- **CFStencil.H**

6.38 channelBC Class Reference

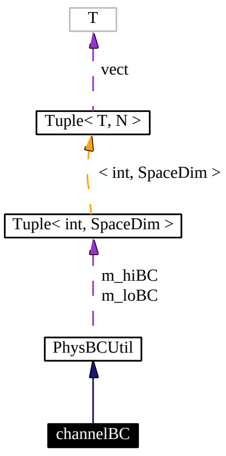
specific BC's for the convection problem (particularly for scalars)

`#include <channelBC.H>`

Inheritance diagram for channelBC:



Collaboration diagram for channelBC:



Public Methods

- **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** (p. 357) *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 Methods

- 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

6.38.1 Detailed Description

specific BC's for the convection problem (particularly for scalars)

6.38.2 Constructor & Destructor Documentation

6.38.2.1 channelBC::channelBC ()

constructor

6.38.2.2 virtual channelBC::∼channelBC () `[virtual]`

destructor

6.38.3 Member Function Documentation

6.38.3.1 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. 371).

6.38.3.2 virtual basicEdgeVelBC* channelBC::basicECVelBC (bool *a_isHomogeneous*, bool *a_isViscous*, bool *a_isSingleComponent*) const [protected, virtual]

Reimplemented from **PhysBCUtil** (p. 372).

6.38.3.3 virtual DomainGhostBC* channelBC::basicScalBC (bool *a_isHomogeneous*, int *a_scalType*) const [protected, virtual]

Reimplemented from **PhysBCUtil** (p. 372).

6.38.3.4 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. 372).

6.38.3.5 virtual DomainGhostBC* channelBC::diffusiveBC (int *a_scalarType*) const [virtual]

physical bc's for diffusion term for scalars

Reimplemented from **PhysBCUtil** (p. 372).

6.38.3.6 virtual DomainGhostBC* channelBC::gradELambdaBC () const [virtual]

BC for grad eLambda.

Reimplemented from **PhysBCUtil** (p. 373).

6.38.3.7 virtual DomainGhostBC* channelBC::gradMacPressureBC () const [virtual]

returns BC applied to MAC pressure before computing gradient

Reimplemented from **PhysBCUtil** (p. 373).

6.38.3.8 Real channelBC::maxInflowVel () const

6.38.3.9 virtual PhysBCUtil* channelBC::newPhysBCUtil () const [virtual]

"virtual" constructor

Reimplemented from **PhysBCUtil** (p. 374).

6.38.3.10 virtual void channelBC::setBCs () [virtual]

interact with **ParmParse** (p. 357) to set physical BC types

Reimplemented from **PhysBCUtil** (p. 374).

6.38.3.11 void channelBC::setMaxInflowVel (Real *a_maxInflowVel*)

6.38.4 Member Data Documentation

6.38.4.1 Real channelBC::m_maxInflowVel [protected]

max inflow velocity

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

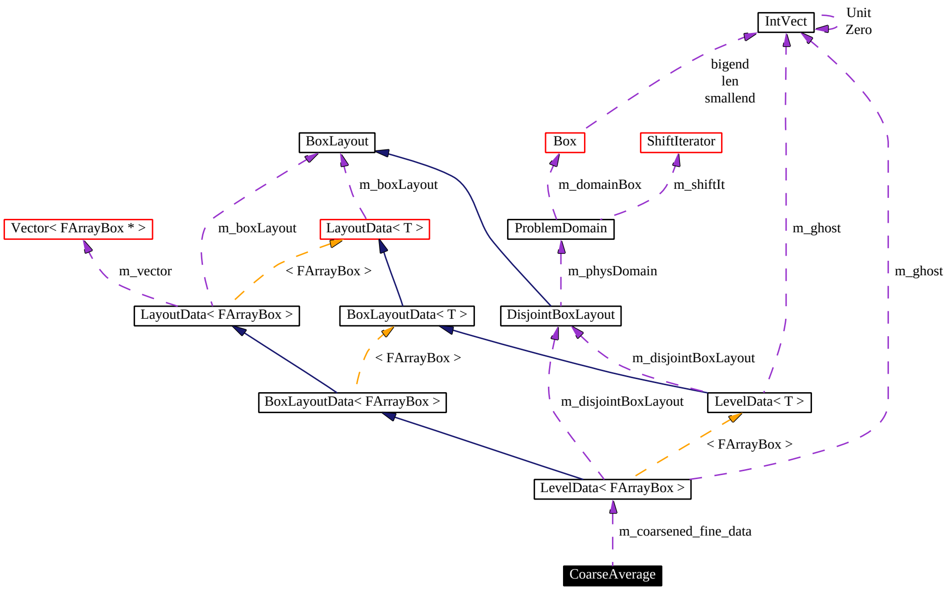
- **channelBC.H**

6.39 CoarseAverage Class Reference

replaces coarse level data with an average of fine level data.

`#include <CoarseAverage.H>`

Collaboration diagram for CoarseAverage:



Public Methods

- **CoarseAverage** ()
- ~**CoarseAverage** ()
- **CoarseAverage** (const **DisjointBoxLayout** &a_fine_domain, int a_numcomps, int a_ref_ratio)
- void **define** (const **DisjointBoxLayout** &a_fine_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 Methods

- 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**

6.39.1 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.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 CoarseAverage::CoarseAverage ()

Default constructor. User must subsequently call **define**() (p. 182).

6.39.2.2 CoarseAverage::~CoarseAverage ()

Destructor.

6.39.2.3 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**() (p. 182).

{\bf 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.\

6.39.3 Member Function Documentation

6.39.3.1 void CoarseAverage::averageGridData (BaseFab< Real > & *a_coarse*, const BaseFab< Real > & *a_fine*, int *a_ref_ratio*) const `[protected]`

6.39.3.2 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**() (p. 183). The domain of a_fine_data should be the same as the fine domain specified in the most recent call to **define**() (p. 182). 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**() (p. 182).

{\bf Arguments:}\ a_coarse_data (modified): coarse data. \ a_fine_data (not modified): fine data. \

{\bf 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.

6.39.3.3 void CoarseAverage::define (const DisjointBoxLayout & *a_fine_domain*, int *a_numcomps*, int *a_ref_ratio*)

Defines this object. Existing information is overriden.

{\bf 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.\

{\bf This:}\ —This object is modified.—

6.39.3.4 bool CoarseAverage::isDefined () const

Returns true if this object was created with the defining constructor or if **define**() (p. 182) has been called.

{\bf This:}\ This object is not modified.

6.39.4 Member Data Documentation

6.39.4.1 bool CoarseAverage::is_defined `[protected]`

6.39.4.2 LevelData<FArrayBox> CoarseAverage::m_coarsened_fine_data `[protected]`

6.39.4.3 int CoarseAverage::m_ref_ratio `[protected]`

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

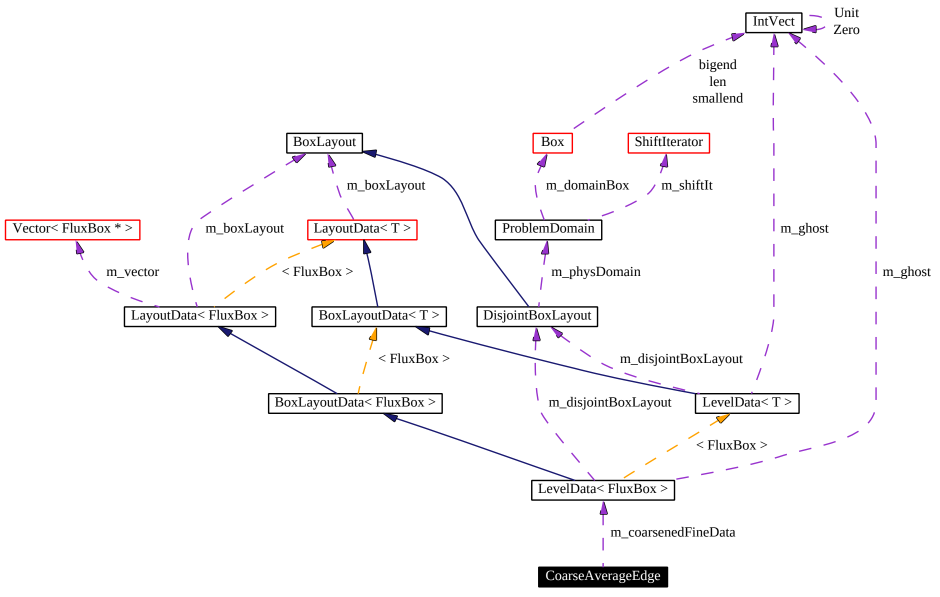
- **CoarseAverage.H**

6.40 CoarseAverageEdge Class Reference

replaces edge-centered coarse-level data w/ averaged fine-level data

`#include <CoarseAverageEdge.H>`

Collaboration diagram for CoarseAverageEdge:



Public Methods

- **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 Methods

- 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**

6.40.1 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** (p. 181)

6.40.2 Constructor & Destructor Documentation

6.40.2.1 CoarseAverageEdge::CoarseAverageEdge ()

Default constructor.

6.40.2.2 CoarseAverageEdge::CoarseAverageEdge (const DisjointBoxLayout & *a_fineGrids*, int *a_nComp*, int *a_nRef*)

defining constructor

6.40.2.3 CoarseAverageEdge::~CoarseAverageEdge ()

destructor

6.40.3 Member Function Documentation

6.40.3.1 void CoarseAverageEdge::averageGridData (FluxBox & *a_coarsenedFine*, const FluxBox & *fine*) const `[protected]`

averages entire single grid data from fine->crse

6.40.3.2 void CoarseAverageEdge::averageToCoarse (LevelData< FluxBox > & *a_coarse_data*, const LevelData< FluxBox > & *a_fine_data*)

averages fine-level data to coarse level

6.40.3.3 void CoarseAverageEdge::define (const DisjointBoxLayout & *a_fineGrids*, int *a_nComp*, int *a_nRef*)

defines the object

6.40.3.4 bool CoarseAverageEdge::isDefined () const

6.40.4 Member Data Documentation

6.40.4.1 LevelData<FluxBox> CoarseAverageEdge::m_coarsenedFineData [protected]

6.40.4.2 bool CoarseAverageEdge::m_isDefined [protected]

6.40.4.3 int CoarseAverageEdge::m_nRef [protected]

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

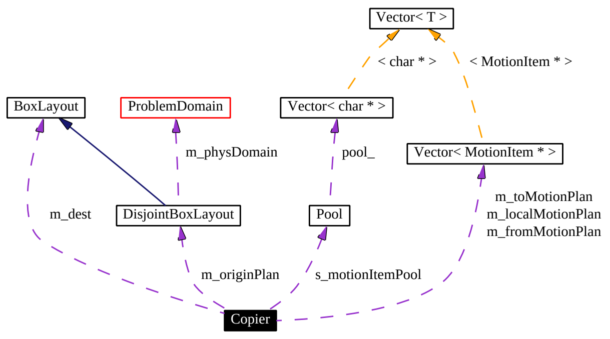
- **CoarseAverageEdge.H**

6.41 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 Methods

- **Copier** ()

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

- **Copier** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest)
- **Copier** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **Problem-Domain** &a_domain)

  this constructor contains support for periodic BC's

- **Copier** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **IntVect** &a_destGhost)

  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 **Problem-Domain** &a_domain, const **IntVect** &a_destGhost)

  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)
- virtual void **define** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **ProblemDomain** &a_domain)

  contains support for periodic BCs

- virtual void **define** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **IntVect** &a_destGhost)
- virtual void **define** (const **DisjointBoxLayout** &a_level, const **BoxLayout** &a_dest, const **ProblemDomain** &a_domain, const **IntVect** &a_destGhost)

  contains support for periodic BCs

- 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

Protected Attributes

- **Vector**< **MotionItem** ∗ > **m_localMotionPlan**
- **Vector**< **MotionItem** ∗ > **m_fromMotionPlan**
- **Vector**< **MotionItem** ∗ > **m_toMotionPlan**

Static Protected Attributes

- **Pool s_motionItemPool**

Friends

- class **CopyIterator**
- void **dumpmemoryatexit** ()

6.41.1 Detailed Description

A strange but true thing to make copying from one boxlayoutdata to another fast.

class to handle the of data from a **LevelData** (p. 295) to a **BoxLayoutData** (p. 147) 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** (p. 295) and **BoxLayoutData** (p. 147) that have unchanging BoxLayouts underneath them. In these cases, **LevelData::copyTo** (p. 298) 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.

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** (p. 295) provides the copyTo methods.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 Copier::Copier () `[inline]`

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

6.41.2.2 Copier::Copier (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*)

6.41.2.3 Copier::Copier (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const ProblemDomain & *a_domain*)

this constructor contains support for periodic BC's

6.41.2.4 Copier::Copier (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const IntVect & *a_destGhost*)

Copier to copy data into the valid and invalid regions of a_dest from a_level.

6.41.2.5 Copier::Copier (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const ProblemDomain & *a_domain*, const IntVect & *a_destGhost*)

Copier to copy data into the valid and invalid regions of a_dest from a_level with support for periodic BCs.

6.41.2.6 virtual Copier::~Copier () `[virtual]`

6.41.3 Member Function Documentation

6.41.3.1 bool Copier::check (const DisjointBoxLayout & *from*, const BoxLayout & *to*) const

check that this Copier object corresponds to these two BoxLayouts.

6.41.3.2 virtual void Copier::clear () `[virtual]`

6.41.3.3 virtual void Copier::define (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const ProblemDomain & *a_domain*, const IntVect & *a_destGhost*) `[virtual]`

contains support for periodic BCs

6.41.3.4 virtual void Copier::define (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const IntVect & *a_destGhost*) `[virtual]`

6.41.3.5 virtual void Copier::define (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*, const ProblemDomain & *a_domain*) `[virtual]`

contains support for periodic BCs

6.41.3.6 virtual void Copier::define (const DisjointBoxLayout & *a_level*, const BoxLayout & *a_dest*) [virtual]

6.41.3.7 int Copier::print () const

6.41.4 Friends And Related Function Documentation

6.41.4.1 friend class CopyIterator [friend]

6.41.4.2 void dumpmemoryatexit () [friend]

6.41.5 Member Data Documentation

6.41.5.1 Vector<MotionItem*> Copier::m_fromMotionPlan [protected]

6.41.5.2 Vector<MotionItem*> Copier::m_localMotionPlan [protected]

6.41.5.3 Vector<MotionItem*> Copier::m_toMotionPlan [protected]

6.41.5.4 Pool Copier::s_motionItemPool [static, protected]

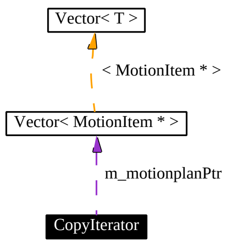
The documentation for this class was generated from the following file:

- **Copier.H**

6.42 CopyIterator Class Reference

`#include <Copier.H>`

Collaboration diagram for CopyIterator:



Public Types

- enum **local_from_to** { **LOCAL**, **FROM**, **TO** }

Public Methods

- **CopyIterator** (const **Copier** &a_copier, **local_from_to**)
- const **MotionItem** & **operator()** () const
- void **operator++** ()
- bool **ok** () const
- void **reset** ()

6.42.1 Member Enumeration Documentation

6.42.1.1 enum CopyIterator::local_from_to

Enumeration values:

LOCAL

FROM

TO

6.42.2 Constructor & Destructor Documentation

6.42.2.1 CopyIterator::CopyIterator (const Copier & *a_copier*, local_from_to) [inline]

6.42.3 Member Function Documentation

6.42.3.1 bool CopyIterator::ok () const [inline]

6.42.3.2 const MotionItem & CopyIterator::operator() () const [inline]

6.42.3.3 void CopyIterator::operator++ () [inline]

6.42.3.4 void CopyIterator::reset () [inline]

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

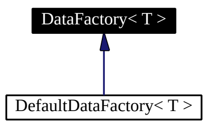
- **Copier.H**

6.43 DataFactory< T > Class Template Reference

Factory object to data members of a **BoxLayoutData** (p. 147) container.

```
#include <BoxLayoutData.H>
```

Inheritance diagram for DataFactory< T >:



Public Methods

- virtual T * **create** (const **Box** &box, int ncomps, const **DataIndex** &a_datInd) const=0
  factory function. creates a new 'T' object

6.43.1 Detailed Description

template<class T> class DataFactory< T >

Factory object to data members of a **BoxLayoutData** (p. 147) container.

6.43.2 Member Function Documentation

6.43.2.1 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. Responsiblitly for calling operator 'delete' on this pointer is passed to the user.

Implemented in **DefaultDataFactory< T >** (p. 198).

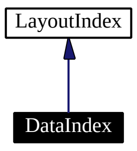
The documentation for this class was generated from the following file:

- **BoxLayoutData.H**

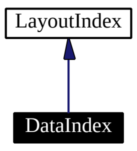
6.44 DataIndex Class Reference

`#include <DataIndex.H>`

Inheritance diagram for DataIndex:



Collaboration diagram for DataIndex:



Public Methods

- **DataIndex** (const **LayoutIndex** &promotion)
- **DataIndex** ()
- ~**DataIndex** ()

Friends

- class **LayoutIterator**
- class **DataIterator**
- class **BoxLayout**
- class **Copier**

6.44.1 Constructor & Destructor Documentation

6.44.1.1 DataIndex::DataIndex (const LayoutIndex & *promotion*) `[inline, explicit]`

6.44.1.2 DataIndex::DataIndex () `[inline]`

6.44.1.3 DataIndex::~DataIndex () `[inline]`

6.44.2 Friends And Related Function Documentation

6.44.2.1 friend class BoxLayout `[friend]`

Reimplemented from **LayoutIndex** (p. 290).

6.44.2.2 friend class Copier `[friend]`

6.44.2.3 friend class DataIterator `[friend]`

Reimplemented from **LayoutIndex** (p. 290).

6.44.2.4 friend class LayoutIterator `[friend]`

Reimplemented from **LayoutIndex** (p. 290).

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

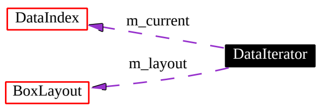
- **DataIndex.H**

6.45 DataIterator Class Reference

An Iterator based on a **BoxLayout** (p. 139) object.

`#include <DataIterator.H>`

Collaboration diagram for DataIterator:



Public Methods

- **DataIterator** ()

  a null constructed DataIterator will return false on **ok()** (p. 197)

- ~**DataIterator** ()
- const **DataIndex** & **operator()** () const
- **DataIndex i** () const
- void **operator++** ()
- void **incr** ()
- bool **ok** () const
- void **reset** ()
- void **begin** ()
- void **end** ()

Friends

- class **BoxLayout**
- class **DisjointBoxLayout**

6.45.1 Detailed Description

An Iterator based on a **BoxLayout** (p. 139) object.

An Iterator based on a **BoxLayout** (p. 139) object. It does not support a dereferencing operation, since it is intended to work with all of BoxLayouts, DisjointBoxLayouts, BoxLayoutDatas LevelDatas, and any object that is built on top of a **BoxLayout** (p. 139) object. DataIterator access the data in a **BoxLayout** (p. 139)-based object in a data-parallel manner. Order is not defined.

6.45.2 Constructor & Destructor Documentation

6.45.2.1 DataIterator::DataIterator () `[inline]`

a null constructed DataIterator will return false on **ok**() (p. 197)

6.45.2.2 DataIterator::~DataIterator () [inline]

6.45.3 Member Function Documentation

6.45.3.1 void DataIterator::begin () [inline]

6.45.3.2 void DataIterator::end ()

skip the iterator to the end. **ok**() (p. 197) will return false after this method is called.

6.45.3.3 DataIndex DataIterator::i () const [inline]

6.45.3.4 void DataIterator::incr () [inline]

6.45.3.5 bool DataIterator::ok () const [inline]

6.45.3.6 const DataIndex & DataIterator::operator() () const [inline]

6.45.3.7 void DataIterator::operator++ () [inline]

6.45.3.8 void DataIterator::reset () [inline]

6.45.4 Friends And Related Function Documentation

6.45.4.1 friend class BoxLayout [friend]

6.45.4.2 friend class DisjointBoxLayout [friend]

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

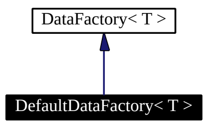
- **DataIterator.H**

6.46 DefaultDataFactory< T > Class Template Reference

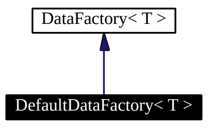
Factory object to data members of a **BoxLayoutData** (p. 147) container.

#include <BoxLayoutData.H>

Inheritance diagram for DefaultDataFactory< T >:



Collaboration diagram for DefaultDataFactory< T >:



Public Methods

- virtual T ∗ **create** (const **Box** &box, int ncomps, const **DataIndex** &a_datInd) const

  factory function. creates a new 'T' object

6.46.1 Detailed Description

template<class T> class DefaultDataFactory< T >

Factory object to data members of a **BoxLayoutData** (p. 147) container.

6.46.2 Member Function Documentation

6.46.2.1 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. 193).

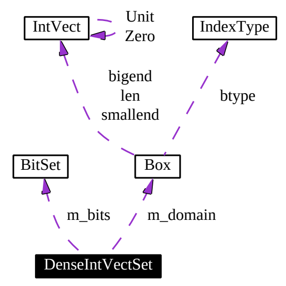
The documentation for this class was generated from the following file:

- **BoxLayoutData.H**

6.47 DenseIntVectSet Class Reference

`#include <DenseIntVectSet.H>`

Collaboration diagram for DenseIntVectSet:



Public Methods

- **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)
- bool **operator**[ ] (const **IntVect** &index) const
- bool **contains** (const **Box** &box) const
- 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
- bool **isFull** () const
- int **numPts** () const
- **Vector**< **Box** > **createBoxes** () const
- void **compact** () const

Friends

- class **DenseIntVectSetIterator**

6.47.1 Detailed Description

Performance-oriented class that reproduces most of the functionality of **IntVectSet** (p. 274). The one function that is not supported is union (|=). This object stores a dense representation of an **IntVectSet** (p. 274) over the box of definition

6.47.2 Constructor & Destructor Documentation

6.47.2.1 DenseIntVectSet::DenseIntVectSet () `[inline]`

6.47.2.2 DenseIntVectSet::DenseIntVectSet (const Box & *a_domain*, bool *init* = true)

you can either have the domain begin all true, or all false

6.47.3 Member Function Documentation

6.47.3.1 const Box & DenseIntVectSet::box () const `[inline]`

6.47.3.2 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** (p. 274). \ The modified DenseIntVectSet is the low end, the returned DenseIntvectSet \ is the high end.

6.47.3.3 void DenseIntVectSet::coarsen (int *iref*)

coarsen the set by the input

6.47.3.4 void DenseIntVectSet::compact () const

set object to its minimum representation. changes data, but is a logically const operation

6.47.3.5 bool DenseIntVectSet::contains (const Box & *box*) const

returns 'true' if the entire set of points specified by the box 'b' is a member of the IntvectSet

6.47.3.6 Vector<Box> DenseIntVectSet::createBoxes () const

turn DenseIntVectSet into a **Vector** (p. 471) of Boxes

6.47.3.7 void DenseIntVectSet::grow (int *idir*, int *igrow*)

grow the set by the input

6.47.3.8 void DenseIntVectSet::grow (int *igrow*)

grow the set by the input

6.47.3.9 bool DenseIntVectSet::isEmpty () const

return true if the set has no points

6.47.3.10 bool DenseIntVectSet::isFull () const

return true if the set consists of a set of a full box

6.47.3.11 void DenseIntVectSet::nestingRegion (int *a_radius*, const ProblemDomain & *a_domain*)

see **IntVectSet** (p. 274)

6.47.3.12 void DenseIntVectSet::nestingRegion (int *a_radius*, const Box & *a_domain*)

see **IntVectSet** (p. 274)

6.47.3.13 int DenseIntVectSet::numPts () const

return the number of points in the set

6.47.3.14 DenseIntVectSet& DenseIntVectSet::operator &= (const DenseIntVectSet & *ivs*)

make this set the intersection of itself and the input

6.47.3.15 DenseIntVectSet& DenseIntVectSet::operator &= (const ProblemDomain & *domain*)

make this set the intersection of itself and the input

6.47.3.16 DenseIntVectSet& DenseIntVectSet::operator &= (const Box & *box*)

make this set the intersection of itself and the input

6.47.3.17 DenseIntVectSet& DenseIntVectSet::operator-= (const DenseIntVectSet & *ivs*)

make this set the complement of itself and the input

6.47.3.18 DenseIntVectSet & DenseIntVectSet::operator-= (const IntVect & *intvect*) `[inline]`

make this set the complement of itself and the input

6.47.3.19 DenseIntVectSet& DenseIntVectSet::operator-= (const Box & *box*)

make this set the complement of itself and the input

6.47.3.20 bool DenseIntVectSet::operator[] (const IntVect & *index*) const

6.47.3.21 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

6.47.3.22 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

6.47.3.23 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

6.47.3.24 void DenseIntVectSet::refine (int *iref*)

refine the set by the input

6.47.3.25 void DenseIntVectSet::shift (const IntVect & *iv*) `[inline]`

Shift every cell in the set by the input.

6.47.4 Friends And Related Function Documentation

6.47.4.1 friend class DenseIntVectSetIterator `[friend]`

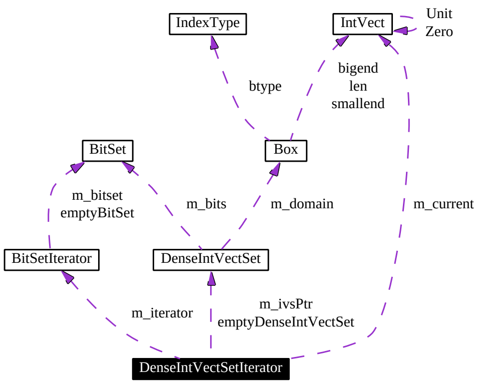
The documentation for this class was generated from the following file:

- **DenseIntVectSet.H**

6.48 DenseIntVectSetIterator Class Reference

`#include <DenseIntVectSet.H>`

Collaboration diagram for DenseIntVectSetIterator:



Public Methods

- **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**

6.48.1 Detailed Description

DenseIntVectSetIterator iterates over all the 'true' members of a **DenseIntVectSet** (p. 199) set.

6.48.2 Constructor & Destructor Documentation

6.48.2.1 DenseIntVectSetIterator::DenseIntVectSetIterator () `[inline]`

6.48.2.2 DenseIntVectSetIterator::DenseIntVectSetIterator (const DenseIntVectSet & *ivs*) `[inline]`

6.48.3 Member Function Documentation

6.48.3.1 void DenseIntVectSetIterator::begin ()

6.48.3.2 void DenseIntVectSetIterator::define (const DenseIntVectSet & *ivs*) `[inline]`

6.48.3.3 void DenseIntVectSetIterator::end () `[inline]`

6.48.3.4 bool DenseIntVectSetIterator::ok () const `[inline]`

6.48.3.5 const IntVect & DenseIntVectSetIterator::operator() () const `[inline]`

6.48.3.6 void DenseIntVectSetIterator::operator++ () `[inline]`

6.48.4 Member Data Documentation

6.48.4.1 DenseIntVectSet DenseIntVectSetIterator::emptyDenseIntVectSet `[static]`

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

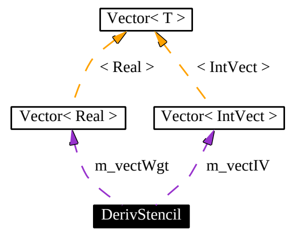
- **DenseIntVectSet.H**

6.49 DerivStencil Class Reference

class to encapsulate the operations to create derivs on irreg stencils

`#include <DerivStencil.H>`

Collaboration diagram for DerivStencil:



Public Methods

- 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**

6.49.1 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.

6.49.2 Constructor & Destructor Documentation

6.49.2.1 DerivStencil::DerivStencil ()

default constructor; creates empty vectors

6.49.2.2 DerivStencil::DerivStencil (const DerivStencil & *a_dsin*)

copy constructor; sets *this = a_dsin

6.49.2.3 DerivStencil::~DerivStencil ()

6.49.3 Member Function Documentation

6.49.3.1 void DerivStencil::accumulate (const IntVect & *a_iv*, Real *a_weight*)

add another set if the **IntVect** (p. 264) is not in the stencil already. \ Add the weight to the existing weight otherwise**

6.49.3.2 void DerivStencil::clear ()

make derivstencil empty

6.49.3.3 void DerivStencil::define ()

6.49.3.4 const IntVect& DerivStencil::getIndex (int *a_ivec*) const

get iv at ivec

6.49.3.5 const Real& DerivStencil::getWeight (int *a_ivec*) const

get weight at ivec

6.49.3.6 bool DerivStencil::isDefined () const `[inline]`

return true if any define function been called.

6.49.3.7 const DerivStencil& DerivStencil::operator *= (Real *a_facin*)

Multiply each weight by a_facin does nothing if vectors are of zero length

6.49.3.8 const DerivStencil& DerivStencil::operator+= (Real *a_facin*)

add a_facin to each weight does nothing if vectors are of zero length

6.49.3.9 const DerivStencil& DerivStencil::operator-= (Real *a_facin*)

subtract a_facin from each weight does nothing if vectors are of zero length

6.49.3.10 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.

6.49.3.11 const DerivStencil& DerivStencil::operator= (const DerivStencil & *a_dsin*)

assignment operator

6.49.3.12 int DerivStencil::size () const

return length of vectors

6.49.4 Member Data Documentation

6.49.4.1 bool DerivStencil::isdefined [protected]

6.49.4.2 Vector<IntVect> DerivStencil::m_vectIV [protected]

6.49.4.3 Vector<Real> DerivStencil::m_vectWgt [protected]

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

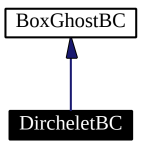
- **DerivStencil.H**

6.50 DircheletBC Class Reference

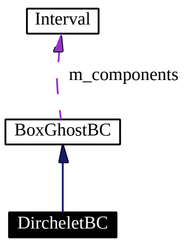
Class to enforce homogeneous dirchelet boundary conditions.

`#include <PoissonBC.H>`

Inheritance diagram for DircheletBC:



Collaboration diagram for DircheletBC:



Public Methods

- **DircheletBC** (int dir, **Side::LoHiSide** sd)
- **DircheletBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- ~**DircheletBC** ()
- **DircheletBC** ()

Friends

- class **DomainGhostBC**

6.50.1 Detailed Description

Class to enforce homogeneous dirchelet boundary conditions.

BoxGhostBC (p. 132) class implentation to enforce homogeneous dirchelet boundary conditions Recall that **BoxGhostBC** (p. 132) 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, B=C=0, A= 1 These functions are only meant to be called by **DomainGhostBC** (p. 215)

6.50.2 Constructor & Destructor Documentation

6.50.2.1 DircheletBC::DircheletBC (int *dir*, Side::LoHiSide *sd*)

6.50.2.2 DircheletBC::DircheletBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.50.2.3 DircheletBC::~DircheletBC ()

6.50.2.4 DircheletBC::DircheletBC ()

6.50.3 Friends And Related Function Documentation

6.50.3.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 135).

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

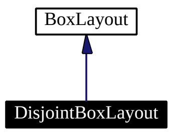
- **PoissonBC.H**

6.51 DisjointBoxLayout Class Reference

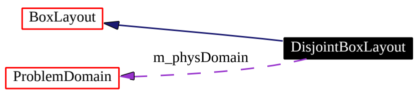
A **BoxLayout** (p. 139) that has a concept of disjointedness.

`#include <DisjointBoxLayout.H>`

Inheritance diagram for DisjointBoxLayout:



Collaboration diagram for DisjointBoxLayout:



Public Methods

- **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)
- 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)
- bool **isDisjoint** () const
- bool **checkPeriodic** (const **ProblemDomain** &a_domain) const
- bool **checkDomains** (const DisjointBoxLayout &a_dbl) const

Protected Methods

- const **ProblemDomain** & **physDomain** () const

Friends

- class **Copier**
- void **coarsen** (DisjointBoxLayout &output, const DisjointBoxLayout &input, int refinement)
- void **refine** (DisjointBoxLayout &output, const DisjointBoxLayout &input, int refinement)
- 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)

6.51.1 Detailed Description

A **BoxLayout** (p. 139) that has a concept of disjointedness.

DisjointBoxLayout is-a **BoxLayout** (p. 139) that has a concept of disjointedness of the boxes. DisjointBoxLayout is only different from **BoxLayout** (p. 139) in that it has a method **isDisjoint**() (p. 213). DisjointBoxLayouts also have a **ProblemDomain** (p. 418) member to ensure disjointness in the case of periodic boundary conditions, among other things.

6.51.2 Constructor & Destructor Documentation

6.51.2.1 DisjointBoxLayout::DisjointBoxLayout ()

6.51.2.2 DisjointBoxLayout::DisjointBoxLayout (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*)

Construct from a **Vector** (p. 471) of Boxes and optionally a **Vector** (p. 471) of processor assignments. If the processor assignment **Vector** (p. 471) is present, it must be either zero-length or the same length as the **Box** (p. 106) **Vector** (p. 471). On exit, the DisjointBoxLayout will be closed.

6.51.2.3 DisjointBoxLayout::DisjointBoxLayout (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*, const ProblemDomain & *a_physDomain*)

Construct from a **Vector** (p. 471) of Boxes and optionally a **Vector** (p. 471) of processor assignments. If the processor assignment **Vector** (p. 471) is present, it must be either zero-length or the same length as the **Box** (p. 106) **Vector** (p. 471). On exit, the DisjointBoxLayout will be closed.

6.51.2.4 virtual DisjointBoxLayout::~DisjointBoxLayout () `[inline, virtual]`

6.51.3 Member Function Documentation

6.51.3.1 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

6.51.3.2 bool DisjointBoxLayout::checkPeriodic (const ProblemDomain & *a_domain*) const

Checks to see that problem domains are compatible To be compatible, a_domain and the **ProblemDomain** (p. 418) in this DisjointBoxLayout must have the same periodicity. They must be periodic in the same dimensions, and 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.

6.51.3.3 virtual void DisjointBoxLayout::close () `[virtual]`

Mark this DisjointBoxLayout as complete and unchangeable.

Reimplemented from **BoxLayout** (p. 141).

6.51.3.4 virtual void DisjointBoxLayout::deepCopy (const BoxLayout & *a_source*, const ProblemDomain & *a_physDomain*) `[virtual]`

checks that the source **BoxLayout** (p. 139) isDisjoint, throws an error if it is not. Otherwise, same as **BoxLayout::deepCopy** (p. 141)

6.51.3.5 virtual void DisjointBoxLayout::deepCopy (const BoxLayout & *a_source*) `[virtual]`

checks that the source **BoxLayout** (p. 139) isDisjoint, throws an error if it is not. Otherwise, same as **BoxLayout::deepCopy** (p. 141)

Reimplemented from **BoxLayout** (p. 141).

6.51.3.6 virtual void DisjointBoxLayout::deepCopy (const DisjointBoxLayout & *a_source*) `[virtual]`

Doesn't check for disjointness, since a_source is already guaranteed to be disjoint. Also copies problem domain info

6.51.3.7 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** (p. 471) of Boxes. This is a wrapper for a call to **BoxLayout::define** (p. 142), required because **DisjointBoxLayout::define** (p. 213) is overloaded and the compiler will not look into the base class for name resolution.

6.51.3.8 void DisjointBoxLayout::define (BoxLayout & *a_layout*, const ProblemDomain & *a_physDomain*)

Shallow define. Only way to promote a **BoxLayout** (p. 139). If **BoxLayout** (p. 139) 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

6.51.3.9 void DisjointBoxLayout::define (BoxLayout & *a_layout*)

Shallow define. Only way to promote a **BoxLayout** (p. 139). If **BoxLayout** (p. 139) 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

6.51.3.10 virtual void DisjointBoxLayout::define (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*) `[virtual]`

Define this DisjointBoxLayout from a **Vector** (p. 471) of Boxes. This is a wrapper for a call to **BoxLayout::define** (p. 142), required because **DisjointBoxLayout::define** (p. 213) is overloaded and the compiler will not look into the base class for name resolution.

Reimplemented from **BoxLayout** (p. 142).

6.51.3.11 virtual void DisjointBoxLayout::define_pd (const Vector< Box > & *a_boxes*, const Vector< int > & *a_procIDs*, const ProblemDomain & *a_physDomain*) `[inline, virtual]`

6.51.3.12 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

6.51.3.13 const ProblemDomain& DisjointBoxLayout::physDomain () const `[protected]`

6.51.4 Friends And Related Function Documentation

6.51.4.1 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** (p. 119) on each box in input. output must be open, will be closed upon exiting this function.

6.51.4.2 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** (p. 120) on each box in input. output must be open, is closed upon exit from this function.

6.51.4.3 void coarsen (DisjointBoxLayout & *output*, const DisjointBoxLayout & *input*, int *refinement*) `[friend]`

'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.

6.51.4.4 friend class Copier [friend]

6.51.4.5 void refine (DisjointBoxLayout & *output*, const DisjointBoxLayout & *input*, int *refinement*) [friend]

see '**coarsen** (p. 144) ' and substitute the word "refine" for "coarsen"

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

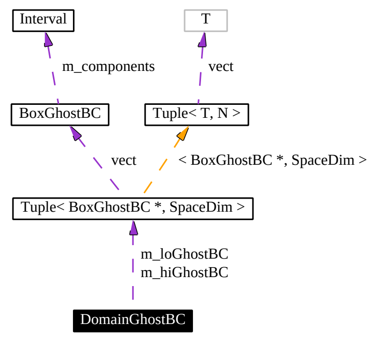
- **DisjointBoxLayout.H**

6.52 DomainGhostBC Class Reference

Class to enforce ghost-cell based boundary conditions.

`#include <GhostBC.H>`

Collaboration diagram for DomainGhostBC:



Public Methods

- **DomainGhostBC** ()
- ~**DomainGhostBC** ()
- void **setBoxGhostBC** (const **BoxGhostBC** &ghost_bc)
- const **BoxGhostBC** & **operator()** (int direction, **Side::LoHiSide** side) const
- void **applyHomogeneousBCs** (**FArrayBox** &state, const **Box** &Domain, **Real** dx) const
- void **applyHomogeneousBCs** (**FArrayBox** &state, const **ProblemDomain** &Domain, **Real** dx) const
- void **applyInhomogeneousBCs** (**FArrayBox** &state, const **Box** &Domain, **Real** dx) const
- void **applyInhomogeneousBCs** (**FArrayBox** &state, const **ProblemDomain** &Domain, **Real** dx) const
- DomainGhostBC & **operator=** (const DomainGhostBC &)
- **DomainGhostBC** (const DomainGhostBC &)

Protected Methods

- bool **isBCDefined** (const int a_dir, const **Side::LoHiSide** a_side) const

  returns true if this GhostBC is defined

- void **resetBoxGhostBC** (const int a_dir, const **Side::LoHiSide** a_side)

Protected Attributes

- **Tuple**< **BoxGhostBC** ∗, **SpaceDim** > **m_loGhostBC**
- **Tuple**< **BoxGhostBC** ∗, **SpaceDim** > **m_hiGhostBC**

6.52.1 Detailed Description

Class to enforce ghost-cell based boundary conditions.

This class which holds a domain's worth of boxghostbcs, one for each face. The user adds a **BoxGhostBC** (p. 132)-derived 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 are expressed as A*phi + B*dphi/dx = C. The class derived from **BoxGhostBC** (p. 132) provides a function to to fill A B and C. DomainGhostBC calls this function and fills the ghost cells outside the domain appropriately.

6.52.2 Constructor & Destructor Documentation

6.52.2.1 DomainGhostBC::DomainGhostBC ()

6.52.2.2 DomainGhostBC::~DomainGhostBC ()

6.52.2.3 DomainGhostBC::DomainGhostBC (const DomainGhostBC &)

6.52.3 Member Function Documentation

6.52.3.1 void DomainGhostBC::applyHomogeneousBCs (FArrayBox & *state*, const ProblemDomain & *Domain*, Real *dx*) const

apply boundary conditions on all sides compares sides of state to domain.

6.52.3.2 void DomainGhostBC::applyHomogeneousBCs (FArrayBox & *state*, const Box & *Domain*, Real *dx*) const

apply boundary conditions on all sides compares sides of state to domain.

6.52.3.3 void DomainGhostBC::applyInhomogeneousBCs (FArrayBox & *state*, const ProblemDomain & *Domain*, Real *dx*) const

apply boundary conditions on all sides compares sides of state to domain.

6.52.3.4 void DomainGhostBC::applyInhomogeneousBCs (FArrayBox & *state*, const Box & *Domain*, Real *dx*) const

apply boundary conditions on all sides compares sides of state to domain.

6.52.3.5 bool DomainGhostBC::isBCDefined (const int *a_dir*, const Side::LoHiSide *a_side*) const `[protected]`

returns true if this GhostBC is defined

6.52.3.6 const BoxGhostBC& DomainGhostBC::operator() (int *direction*, Side::LoHiSide *side*) const

6.52.3.7 DomainGhostBC& DomainGhostBC::operator= (const DomainGhostBC &)

6.52.3.8 void DomainGhostBC::resetBoxGhostBC (const int *a_dir*, const Side::LoHiSide *a_side*) `[protected]`

6.52.3.9 void DomainGhostBC::setBoxGhostBC (const BoxGhostBC & *ghost_bc*)

Set boundary conditions at a face with ghost_bc.

6.52.4 Member Data Documentation

6.52.4.1 Tuple<BoxGhostBC*, SpaceDim> DomainGhostBC::m_hiGhostBC `[protected]`

6.52.4.2 Tuple<BoxGhostBC*, SpaceDim> DomainGhostBC::m_loGhostBC `[protected]`

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

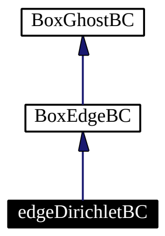
- **GhostBC.H**

6.53 edgeDirichletBC Class Reference

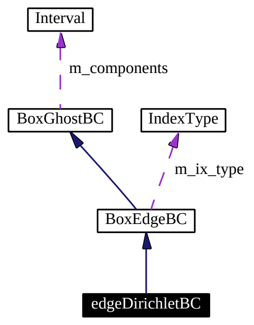
class to do Dirichlet edge-centered BC's

`#include <edgeGhostBC.H>`

Inheritance diagram for edgeDirichletBC:



Collaboration diagram for edgeDirichletBC:



Public Methods

- ~**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 Methods

- 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**

6.53.1 Detailed Description

class to do Dirichlet edge-centered BC's

this class implements the edge-centered instantiation of the edge-centered homogeneous BC's

6.53.2 Constructor & Destructor Documentation

6.53.2.1 edgeDirichletBC::∼edgeDirichletBC ()

6.53.2.2 edgeDirichletBC::edgeDirichletBC ()

6.53.2.3 edgeDirichletBC::edgeDirichletBC (int *a_dir*, Side::LoHiSide *a_sd*)

sets interval to be (0,0)

6.53.2.4 edgeDirichletBC::edgeDirichletBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*)

6.53.3 Member Function Documentation

6.53.3.1 Real edgeDirichletBC::BCVal () const

6.53.3.2 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. 127).

6.53.3.3 virtual void edgeDirichletBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const `[protected, virtual]`

Reimplemented from **BoxEdgeBC** (p. 127).

6.53.3.4 virtual BoxGhostBC* edgeDirichletBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround

Reimplemented from **BoxEdgeBC** (p. 127).

6.53.3.5 void edgeDirichletBC::setBCVal (const Real *a_bcVal*)

set edge-BC value (single-valued)

6.53.4 Friends And Related Function Documentation

6.53.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 135).

6.53.5 Member Data Documentation

6.53.5.1 Real edgeDirichletBC::m_bcVal [protected]

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

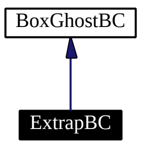
- **edgeGhostBC.H**

6.54 ExtrapBC Class Reference

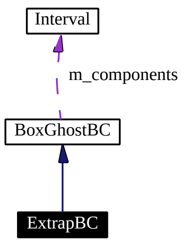
Class to enforce homogeneous neumann boundary conditions.

`#include <ExtrapBC.H>`

Inheritance diagram for ExtrapBC:



Collaboration diagram for ExtrapBC:



Public Methods

- **~ExtrapBC** ()
- **ExtrapBC** ()
- **ExtrapBC** (int dir, **Side::LoHiSide** sd)
- **ExtrapBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)

Protected Methods

- 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

  virtual constructor workaround.

Friends

- class **DomainGhostBC**

6.54.1 Detailed Description

Class to enforce homogeneous neumann boundary conditions.

BoxGhostBC (p. 132) class implentation to enforce homogeneous neumann boundary conditions Recall that **BoxGhostBC** (p. 132) 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** (p. 215)

6.54.2 Constructor & Destructor Documentation

6.54.2.1 ExtrapBC::∼ExtrapBC ()

6.54.2.2 ExtrapBC::ExtrapBC ()

6.54.2.3 ExtrapBC::ExtrapBC (int *dir*, Side::LoHiSide *sd*)

6.54.2.4 ExtrapBC::ExtrapBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.54.3 Member Function Documentation

6.54.3.1 virtual void ExtrapBC::applyExtrapBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

6.54.3.2 virtual void ExtrapBC::applyExtrapBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

6.54.3.3 virtual void ExtrapBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const [protected, virtual]

deprecated interface

Reimplemented from **BoxGhostBC** (p. 134).

6.54.3.4 virtual void ExtrapBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Reimplemented from **BoxGhostBC** (p. 134).

6.54.3.5 virtual void ExtrapBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const [protected, virtual]

deprecated interface

Reimplemented from **BoxGhostBC** (p. 134).

6.54.3.6 virtual void ExtrapBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Reimplemented from **BoxGhostBC** (p. 134).

6.54.3.7 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. 135).

6.54.3.8 virtual void ExtrapBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const [protected, virtual]

Implements **BoxGhostBC** (p. 135).

6.54.3.9 BoxGhostBC* ExtrapBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround.

Implements **BoxGhostBC** (p. 135).

6.54.4 Friends And Related Function Documentation

6.54.4.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 135).

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

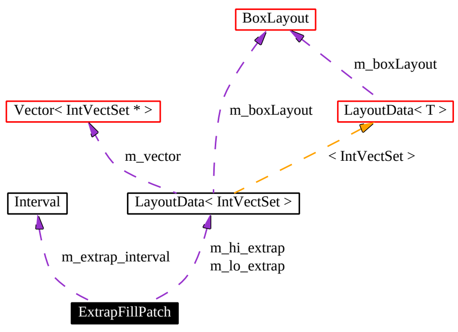
- **ExtrapBC.H**

6.55 ExtrapFillPatch Class Reference

fills outer ghost cells by copying from inner ghost cells.

`#include <ExtrapFillPatch.H>`

Collaboration diagram for ExtrapFillPatch:



Public Methods

- **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**

6.55.1 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).

\begin{verbatim} +—–+——————————————+—–+ | | +————————————+ | | | | | | | | | | | | | | | | | | | | | | | | | 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. | |————————————| \end{verbatim}

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.

\begin{verbatim} + - - - - - - - - - - - + > > > X X X X X X < < < | +——-+ | > > > X| |X < < < | | | | > > > X| |X < < < | | grid 1| | > > > X| |X < < < | | | | > > > X| |X < < < | +—+——-+ | ? ?|^ ^ ^ ^ ^ ^|X < < < + - - - - - - - - - - - + | | | | | grid 0 | | | | | +————+ \end{verbatim}

6.55.2 Constructor & Destructor Documentation

6.55.2.1 ExtrapFillPatch::ExtrapFillPatch ()

Default constructor. User must subsequently call **define**() (p. 227).

6.55.2.2 ExtrapFillPatch::~ExtrapFillPatch ()

Destructor.

6.55.2.3 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**() (p. 227). 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.

{\bf 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. \

6.55.2.4 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**() (p. 227). 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.

{\bf 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. \

6.55.3 Member Function Documentation

6.55.3.1 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**() (p. 227) 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.

{\bf 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. \

{\bf This:}\ —This object is modified.—

6.55.3.2 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**() (p. 227) 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.

{\bf 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. \

{\bf This:}\ —This object is modified.—

6.55.3.3 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**() (p. 228).\

{\bf 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. \

{\bf This:}\ This object is not modified.

6.55.3.4 bool ExtrapFillPatch::isDefined () const

Returns true if this object was created with the defining constructor or if **define**() (p. 227) has called.

{\bf This:}\ This object is not modified.

6.55.4 Member Data Documentation

6.55.4.1 Interval ExtrapFillPatch::m_extrap_interval `[protected]`

6.55.4.2 LayoutData<IntVectSet> ExtrapFillPatch::m_hi_extrap[SpaceDim] `[protected]`

6.55.4.3 bool ExtrapFillPatch::m_is_defined `[protected]`

6.55.4.4 LayoutData<IntVectSet> ExtrapFillPatch::m_lo_extrap[SpaceDim] `[protected]`

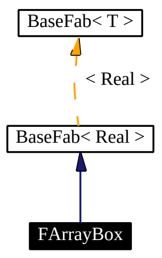
The documentation for this class was generated from the following file:

- **ExtrapFillPatch.H**

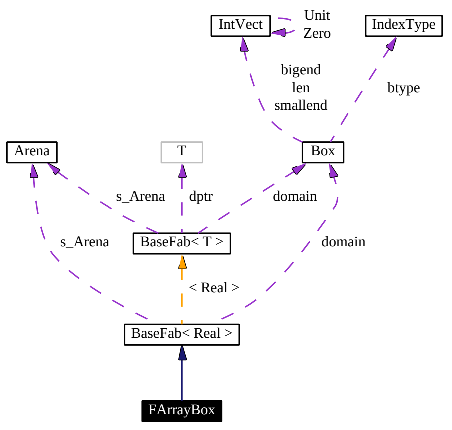
6.56 FArrayBox Class Reference

#include <FArrayBox.H>

Inheritance diagram for FArrayBox:



Collaboration diagram for FArrayBox:



Public Methods

- **FArrayBox** ()

  {\bf constructors and destructor}

- **FArrayBox** (const **Box** &b, int ncomp)
- void **define** (const **Box** &b, int ncomp)
- **FArrayBox** (std::istream &is)
- virtual ~**FArrayBox** ()
- virtual **Real norm** (int p=2, int comp=0, int numcomp=1) const

  {\bf norm functions}

- 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 min** (int comp=0) const

  {\bf min and max functions}

- **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

  {*bf mask functions*}

- 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** ()

  {*bf Arithmetic functions*}

- 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 & **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 Methods

- virtual void **performCopy** (const **BaseFab**< **Real** > &src, const **Box** &srcbox, int srccomp, const **Box** &destbox, int destcomp, int numcomp)

6.56.1 Constructor & Destructor Documentation

6.56.1.1 FArrayBox::FArrayBox ()

{\bf constructors and destructor}

Constructs an invalid FArrayBox with no memory.

6.56.1.2 FArrayBox::FArrayBox (const Box & *b*, int *ncomp*)

6.56.1.3 FArrayBox::FArrayBox (std::istream & *is*) `[explicit]`

Constructs FArrayBox by reading it from istream.

6.56.1.4 virtual FArrayBox::~FArrayBox () `[virtual]`

The (virtual) destructor.

6.56.2 Member Function Documentation

6.56.2.1 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.

6.56.2.2 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).

6.56.2.3 void FArrayBox::abs ()

{\bf Arithmetic functions}

Modifies this FArrayBox by replacing each value with its absolute value.

6.56.2.4 void FArrayBox::define (const Box & *b*, int *ncomp*) `[inline, virtual]`

Reimplemented from **BaseFab< Real >** (p. 86).

6.56.2.5 FArrayBox& FArrayBox::divide (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.6 FArrayBox& FArrayBox::divide (const FArrayBox & *src*, const Box & *subbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.7 FArrayBox& FArrayBox::divide (const FArrayBox & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.8 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.

6.56.2.9 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.

6.56.2.10 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.

6.56.2.11 FArrayBox& FArrayBox::divide (Real *r*)

Modifies this FArrayBox by dividing all values by the scalar Real r.

6.56.2.12 Real FArrayBox::get (const IntVect & *iv*, int *comp*) const `[inline]`

6.56.2.13 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.

6.56.2.14 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.

6.56.2.15 FArrayBox& FArrayBox::invert (Real *v*)

Modifies this FArrayBox by replacing each value x with v/x.

6.56.2.16 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.

6.56.2.17 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.

6.56.2.18 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.

6.56.2.19 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.

6.56.2.20 int FArrayBox::maskLT (BaseFab< int > & *mask*, Real *val*, int *comp* = 0) const

{\bf mask functions}

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.

6.56.2.21 Real FArrayBox::max (const Box & *subbox*, int *comp* = 0) const

Returns the maximum value of given component of this FArrayBox in given subbox.

6.56.2.22 Real FArrayBox::max (int *comp* = 0) const

Returns the maximum value of given component of this FArrayBox.

6.56.2.23 IntVect FArrayBox::maxIndex (const Box & *subbox*, int *comp* = 0) const

6.56.2.24 IntVect FArrayBox::maxIndex (int *comp* = 0) const

Returns location of maximum value in given component of this FArrayBox.

6.56.2.25 Real FArrayBox::min (const Box & *subbox*, int *comp* = 0) const

Returns the minimum value of given component of this FArrayBox in given subbox.

6.56.2.26 Real FArrayBox::min (int *comp* = 0) const

{\bf min and max functions}

Returns the minimum value of given component of this FArrayBox.

6.56.2.27 IntVect FArrayBox::minIndex (const Box & *subbox*, int *comp* = 0) const

6.56.2.28 IntVect FArrayBox::minIndex (int *comp* = 0) const

Finds location of minimum value in given component of this FArrayBox.

6.56.2.29 FArrayBox& FArrayBox::minus (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.30 FArrayBox& FArrayBox::minus (const FArrayBox & *src*, const Box & *subbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.31 FArrayBox& FArrayBox::minus (const FArrayBox & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.32 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.

6.56.2.33 FArrayBox& FArrayBox::mult (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.34 FArrayBox& FArrayBox::mult (const FArrayBox & *src*, const Box & *subbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.35 FArrayBox& FArrayBox::mult (const FArrayBox & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.36 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.

6.56.2.37 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.

6.56.2.38 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.

6.56.2.39 FArrayBox& FArrayBox::mult (Real *r*)

Modifies this FArrayBox by multiplying all values by the scalar Real r.

6.56.2.40 FArrayBox& FArrayBox::negate ()

Modifies this FArrayBox by replacing each value with its additive inverse.

6.56.2.41 FArrayBox& FArrayBox::negate (int *comp*, int *numcomp* = 1)

Modifies this FArrayBox by replacing each value with its additive inverse. For given range of components.

6.56.2.42 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.

6.56.2.43 virtual Real FArrayBox::norm (const Box & *subbox*, int *p* = 2, int *comp* = 0, int *numcomp* = 1) const `[virtual]`

6.56.2.44 virtual Real FArrayBox::norm (int *p* = 2, int *comp* = 0, int *numcomp* = 1) const `[virtual]`

{\bf norm functions}

6.56.2.45 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.

6.56.2.46 FArrayBox& FArrayBox::operator *= (Real *r*)

Modifies this FArrayBox by multiplying all values by the scalar Real r.

6.56.2.47 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.

6.56.2.48 FArrayBox& FArrayBox::operator+= (Real *r*)

Modifies this FArrayBox by adding the scalar Real r to all values.

6.56.2.49 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.

6.56.2.50 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.

6.56.2.51 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.

6.56.2.52 FArrayBox& FArrayBox::operator/= (Real *r*)

Modifies this FArrayBox by dividing all values by the scalar Real r.

6.56.2.53 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. 88).

6.56.2.54 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, const Box & *srcbox*, const Box & *destbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.55 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, const Box & *subbox*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.56 FArrayBox& FArrayBox::plus (const FArrayBox & *src*, int *srccomp*, int *destcomp*, int *numcomp* = 1)

6.56.2.57 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.

6.56.2.58 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.

6.56.2.59 FArrayBox& FArrayBox::plus (Real *r*)

Modifies this FArrayBox by adding the scalar Real r to all values.

6.56.2.60 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.

6.56.2.61 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.

6.56.2.62 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.

6.56.2.63 FArrayBox& FArrayBox::plus_real (Real *r*) `[inline]`

6.56.2.64 void FArrayBox::set (const IntVect & *iv*, int *comp*, Real *val*) `[inline]`

6.56.2.65 Real FArrayBox::sum (const Box & *subbox*, int *comp*, int *numcomp* = 1) const

6.56.2.66 Real FArrayBox::sum (int *comp*, int *numcomp* = 1) const

Returns sum of given component of FArrayBox.

6.56.2.67 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:

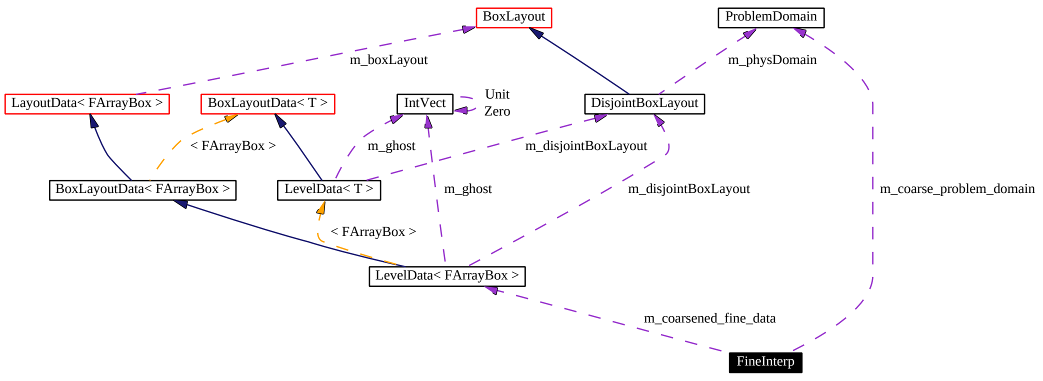
- **FArrayBox.H**

6.57 FineInterp Class Reference

replaces fine level data with interpolation of coarse level data.

`#include <FineInterp.H>`

Collaboration diagram for FineInterp:



Public Methods

- **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 Methods

- 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**

6.57.1 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.

6.57.2 Constructor & Destructor Documentation

6.57.2.1 FineInterp::FineInterp ()

Default constructor. User must subsequently call **define**() (p. 241).

6.57.2.2 FineInterp::~FineInterp ()

Destructor.

6.57.2.3 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**() (p. 241).

{\bf 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.\

6.57.2.4 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**() (p. 241).

{\bf 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.\

6.57.3 Member Function Documentation

6.57.3.1 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.

{\bf 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.\

{\bf This:}\ —This object is modified.—

6.57.3.2 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.

{\bf 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.\

{\bf This:}\ —This object is modified.—

6.57.3.3 void FineInterp::interpGridData (BaseFab< Real > & *a_fine*, const BaseFab< Real > & *a_coarse*, const Box & *a_coarsened_fine_box*, int *a_ref_ratio*) const `[protected]`

6.57.3.4 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**() (p. 241). The domain of a_fine_data should be the same as the fine domain specified in the most recent call to **define**() (p. 241). 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**() (p. 241).

{\bf Arguments:}\ a_fine_data (modified): fine data. \ a_coarse_data (not modified): coarse data. \

{\bf 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.

6.57.3.5 bool FineInterp::isDefined () const

Returns true if this object was created with the defining constructor or if **define**() (p. 241) has been called.

{\bf This:}\ This object is not modified.

6.57.4 Member Data Documentation

6.57.4.1 bool FineInterp::is_defined `[protected]`

6.57.4.2 ProblemDomain FineInterp::m_coarse_problem_domain `[protected]`

6.57.4.3 LevelData<FArrayBox> FineInterp::m_coarsened_fine_data `[protected]`

6.57.4.4 int FineInterp::m_ref_ratio `[protected]`

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

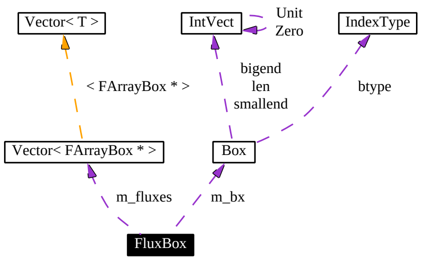
- **FineInterp.H**

6.58 FluxBox Class Reference

A **FArrayBox** (p. 229)-like container for edge-centered fluxes.

`#include <FluxBox.H>`

Collaboration diagram for FluxBox:



Public Methods

- **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**() (p. 88)
- void **define** (const **Box** &bx, int n=1)
- void **clear** ()

  this function returns the FluxBox to the undefined state
- int **nComp** () const

  {\bf access functions} number of components
- const **Box** & **box** () const

  returns cell-centered box which defines fluxBox
- **FArrayBox** & **getFlux** (const int dir)

  returns edge-centered flux in direction dir
- const **FArrayBox** & **getFlux** (const int dir) const
- **FArrayBox** & **operator**[ ] (const int dir)

  returns **FArrayBox** (p. 229) *in direction dir*
- const **FArrayBox** & **operator**[ ] (const int dir) const

  constant version

- void **setVal** (const **Real** 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 edges surrounding cell-centered box bx

- void **setVal** (const **Real** val, const **Box** &bx, const int dir, const int startComp, const int nComp)
- 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)
- void **copy** (const **Box** &srcbox, const **Interval** &srccomps, const **Box** &destbox, const FluxBox &src, const **Interval** &destcomps)
- FluxBox & **shift** (const **IntVect** &v)
- int **size** (const **Box** &bx, const **Interval** &comps) const

  {\bf *Linearization Functions*}

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

Static Public Methods

- int **preAllocatable** ()

Protected Attributes

- **Box m_bx**
- int **m_nvar**
- **Vector< FArrayBox * > m_fluxes**

  BL_SPACEDIM FArrayBoxes which hold fluxes.

6.58.1 Detailed Description

A **FArrayBox** (p. 229)-like container for edge-centered fluxes.

This is a class to contain edge-centered fluxes on a box.

6.58.2 Constructor & Destructor Documentation

6.58.2.1 FluxBox::FluxBox ()

default constructor

6.58.2.2 FluxBox::FluxBox (const Box & *bx*, int *n* = 1)

constructs FluxBox on cell-centered box with n components

6.58.2.3 FluxBox::~FluxBox ()

destructor

6.58.3 Member Function Documentation

6.58.3.1 const Box& FluxBox::box () const

returns cell-centered box which defines fluxBox

6.58.3.2 void FluxBox::clear ()

this function returns the FluxBox to the undefined state

6.58.3.3 void FluxBox::copy (const Box & *srcbox*, const Interval & *srccomps*, const Box & *destbox*, const FluxBox & *src*, const Interval & *destcomps*)

6.58.3.4 void FluxBox::copy (const Box & *R*, const Interval & *Cdest*, const FluxBox & *src*, const Interval & *Csrc*)

6.58.3.5 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

6.58.3.6 void FluxBox::copy (const FluxBox & *src*, const int *srcComp*, const int *destComp*, const int *numComp*)

copy on overlap, for all directions

6.58.3.7 void FluxBox::copy (const FluxBox & *src*)

copy from src to this FluxBox – sizes must be identical

6.58.3.8 void FluxBox::define (const Box & *bx*, int *n* = 1)

6.58.3.9 const FArrayBox& FluxBox::getFlux (const int *dir*) const

6.58.3.10 FArrayBox& FluxBox::getFlux (const int *dir*)

returns edge-centered flux in direction dir

6.58.3.11 void FluxBox::linearIn (void ∗ *buf*, const Box & *R*, const Interval & *comps*)

6.58.3.12 void FluxBox::linearIn (void ∗ *buf*)

6.58.3.13 void FluxBox::linearOut (void ∗ *buf*, const Box & *R*, const Interval & *comps*) const

6.58.3.14 int FluxBox::nComp () const

{\bf access functions} number of components

6.58.3.15 const FArrayBox& FluxBox::operator[] (const int *dir*) const

constant version

6.58.3.16 FArrayBox& FluxBox::operator[] (const int *dir*)

returns **FArrayBox** (p. 229) in direction dir

6.58.3.17 int FluxBox::preAllocatable () `[inline, static]`

6.58.3.18 void FluxBox::resize (const Box & *bx*, int *n* = 1)

resize FluxBox similar to **BaseFab::resize**() (p. 88)

6.58.3.19 void FluxBox::setVal (const Real *val*, const Box & *bx*, const int *dir*, const int *startComp*, const int *nComp*)

6.58.3.20 void FluxBox::setVal (const Real *val*, const Box & *bx*)

sets fluxes on edges surrounding cell-centered box bx

6.58.3.21 void FluxBox::setVal (const Real *val*, const int *dir*, const int *startComp*, const int *nComp*)

more specific setVal

6.58.3.22 void FluxBox::setVal (const Real *val*, const int *dir*)

set fluxes in direction dir to val

6.58.3.23 void FluxBox::setVal (const Real *val*)

{\bf data modification functions} set all fluxes to val

6.58.3.24 FluxBox& FluxBox::shift (const IntVect & *v*)

Modifies the domain of this FluxBox by shifting.

6.58.3.25 int FluxBox::size (const Box & *bx*, const Interval & *comps*) const

{\bf Linearization Functions}

6.58.4 Member Data Documentation

6.58.4.1 Box FluxBox::m_bx `[protected]`

6.58.4.2 Vector<FArrayBox*> FluxBox::m_fluxes `[protected]`

BL_SPACEDIM FArrayBoxes which hold fluxes.

6.58.4.3 int FluxBox::m_nvar `[protected]`

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

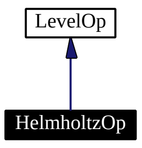
- **FluxBox.H**

6.59 HelmholtzOp Class Reference

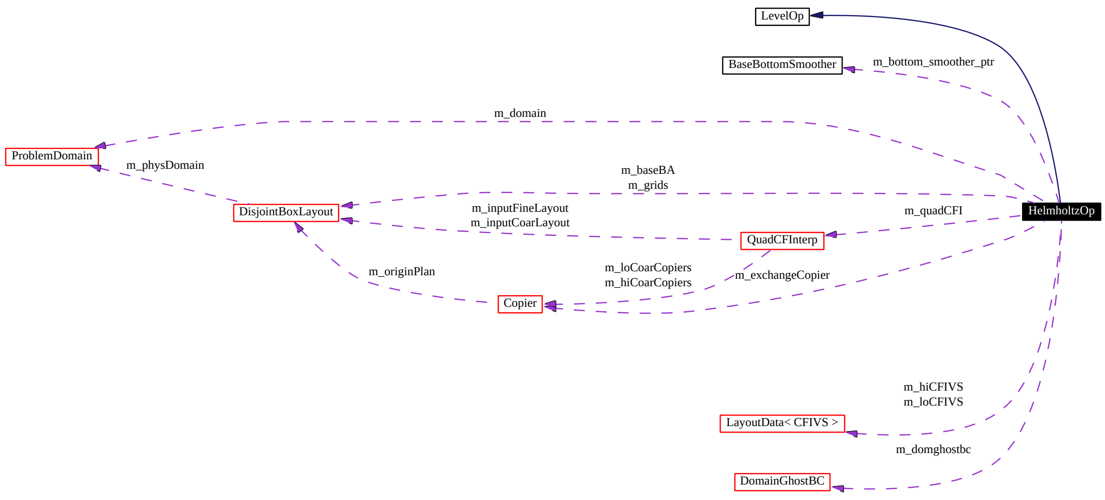
Helmholtz operator class.

`#include <HelmholtzOp.H>`

Inheritance diagram for HelmholtzOp:



Collaboration diagram for HelmholtzOp:



Public Methods

- void **setHelmCoeff** (**Real** helmcoeff_a)
- **LevelOp** ∗ **new_levelop** () const
- void **define** (const **DisjointBoxLayout** &Ba, const **DisjointBoxLayout** ∗base_ba, **Real** DxLevel, int refratio, const **Box** &domf, bool a_homogeneousOnly=false, int a_ncomp=1)
- void **define** (const **DisjointBoxLayout** &Ba, const **DisjointBoxLayout** ∗base_ba, **Real** DxLevel, int refratio, const **ProblemDomain** &domf, bool a_homogeneousOnly=false, int a_ncomp=1)
- void **define** (const **LevelOp** ∗opfine, int refratio)
- **HelmholtzOp** ()
- ∼**HelmholtzOp** ()
- void **smooth** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > &rhs)
- virtual void **levelPreconditioner** (**LevelData**< **FArrayBox** > &a_phihat, const **LevelData**< **FArrayBox** > &a_rhshat)
- void **applyOpI** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > ∗phiCoarse, **LevelData**< **FArrayBox** > &LOfPhi)

- void **applyOpIcfHphys** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > *phiCoarse, **LevelData**< **FArrayBox** > &LOfPhi)
- void **applyOpH** (**LevelData**< **FArrayBox** > &phi, **LevelData**< **FArrayBox** > &LOfPhi)
- void **applyOpHcfIphys** (**LevelData**< **FArrayBox** > &phi, **LevelData**< **FArrayBox** > &LOfPhi)
- void **CFInterp** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > &phiCoarse)
- void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &phi)
- void **setDomainGhostBC** (const **DomainGhostBC** &a_dombcin)
- void **setBottomSmoother** (const **BaseBottomSmoother** &a_bottomSmoother)
- void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &phif, const **DataIndex** &a_datIndex, int idir, **Side::LoHiSide** hiorlo)
- void **bottomSmoother** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > &rhs)
- bool **isDefined** () const
- void **getFlux** (**FArrayBox** &flux, const **FArrayBox** &data, const **DataIndex** &a_datInd, int dir)

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_bottom_smoother_ptr**
- 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**

6.59.1 Detailed Description

Helmholtz operator class.

operator to solve equation (ie (I+alpha*laplacian)phi = rho)

6.59.2 Constructor & Destructor Documentation

6.59.2.1 HelmholtzOp::HelmholtzOp ()

6.59.2.2 HelmholtzOp::~HelmholtzOp ()

6.59.3 Member Function Documentation

6.59.3.1 void HelmholtzOp::applyOpH (LevelData< FArrayBox > & *phi*, LevelData< FArrayBox > & *LOfPhi*) `[virtual]`

Evaluate Operator, homogeneous C/F boundary conditions and phys boundary conditions.

Implements **LevelOp** (p. 316).

6.59.3.2 void HelmholtzOp::applyOpHcfIphys (LevelData< FArrayBox > & *phi*, LevelData< FArrayBox > & *LOfPhi*) `[virtual]`

Evaluate Operator, homogeneous C/F boundary conditions. inhomogeneous domain boundary conditions

Implements **LevelOp** (p. 316).

6.59.3.3 void HelmholtzOp::applyOpI (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > * *phiCoarse*, LevelData< FArrayBox > & *LOfPhi*) `[virtual]`

Evaluate Operator, inhomogeneous C/F boundary conditions and phys boundary conditions.

Implements **LevelOp** (p. 316).

6.59.3.4 void HelmholtzOp::applyOpIcfHphys (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > * *phiCoarse*, LevelData< FArrayBox > & *LOfPhi*) `[virtual]`

Evaluate Operator, inhomogeneous C/F boundary conditions. homogeneous domain boundary conditions.

Implements **LevelOp** (p. 316).

6.59.3.5 void HelmholtzOp::bottomSmoother (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > & *rhs*) `[virtual]`

This does a GSRB Pre/Conditioned BiCGStab on a level for the bottom solver.

Implements **LevelOp** (p. 316).

6.59.3.6 void HelmholtzOp::CFInterp (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > & *phiCoarse*) `[virtual]`

Coarse / Fine interpolation operator. fills ghost cells of phi.

Implements **LevelOp** (p. 317).

6.59.3.7 void HelmholtzOp::define (const LevelOp * *opfine*, int *refratio*) [virtual]

Full define function. define from finer levelop.

Implements **LevelOp** (p. 317).

6.59.3.8 void HelmholtzOp::define (const DisjointBoxLayout & *Ba*, const DisjointBoxLayout * *base_ba*, Real *DxLevel*, int *refratio*, const ProblemDomain & *domf*, bool *a_homogeneousOnly* = false, int *a_ncomp* = 1) [virtual]

Full define function. makes all coarse-fine information and sets internal variables

Implements **LevelOp** (p. 317).

6.59.3.9 void HelmholtzOp::define (const DisjointBoxLayout & *Ba*, const DisjointBoxLayout * *base_ba*, Real *DxLevel*, int *refratio*, const Box & *domf*, bool *a_homogeneousOnly* = false, int *a_ncomp* = 1) [virtual]

Full define function. makes all coarse-fine information and sets internal variables

Implements **LevelOp** (p. 317).

6.59.3.10 void HelmholtzOp::getFlux (FArrayBox & *flux*, const FArrayBox & *data*, const DataIndex & *a_datInd*, int *dir*) [virtual]

get flux(== 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. 317).

6.59.3.11 void HelmholtzOp::homogeneousCFInterp (LevelData< FArrayBox > & *phif*, const DataIndex & *a_datIndex*, int *idir*, Side::LoHiSide *hiorlo*)

does homogeneous coarse/fine interpolation

6.59.3.12 void HelmholtzOp::homogeneousCFInterp (LevelData< FArrayBox > & *phi*) [virtual]

homogeneous coarse/fine interpolation operator

Implements **LevelOp** (p. 317).

6.59.3.13 bool HelmholtzOp::isDefined () const [virtual]

Implements **LevelOp** (p. 318).

6.59.3.14 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. 318).

6.59.3.15 LevelOp* HelmholtzOp::new_levelop () const [virtual]

virtual constructor workaround coefficient and bcs must be defined before called.

Implements **LevelOp** (p. 318).

6.59.3.16 void HelmholtzOp::setBottomSmoother (const BaseBottomSmoother & *a_bottomSmoother*)

6.59.3.17 void HelmholtzOp::setDomainGhostBC (const DomainGhostBC & *a_dombcin*)

6.59.3.18 void HelmholtzOp::setHelmCoeff (Real *helmcoeff_a*)

Set helmholtz coefficient alpha in I+alpha*laplacian)phi = rho.

6.59.3.19 void HelmholtzOp::smooth (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > & *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. 318).

6.59.4 Member Data Documentation

6.59.4.1 DisjointBoxLayout HelmholtzOp::m_baseBA [protected]

6.59.4.2 BaseBottomSmoother* HelmholtzOp::m_bottom_smoother_ptr [protected]

6.59.4.3 ProblemDomain HelmholtzOp::m_domain [protected]

6.59.4.4 DomainGhostBC HelmholtzOp::m_domghostbc [protected]

6.59.4.5 Real HelmholtzOp::m_dxCrse [protected]

6.59.4.6 Real HelmholtzOp::m_dxLevel [protected]

6.59.4.7 Copier HelmholtzOp::m_exchangeCopier [protected]

6.59.4.8 DisjointBoxLayout HelmholtzOp::m_grids [protected]

6.59.4.9 bool HelmholtzOp::m_hcoeffDefined [protected]

6.59.4.10 Real HelmholtzOp::m_helmCoeff [protected]

6.59.4.11 LayoutData<CFIVS> HelmholtzOp::m_hiCFIVS[SpaceDim] [protected]

6.59.4.12 bool HelmholtzOp::m_ihcfiEnabled [protected]

6.59.4.13 bool HelmholtzOp::m_isBCDefined [protected]

6.59.4.14 bool HelmholtzOp::m_isDefined [protected]

6.59.4.15 LayoutData<CFIVS> HelmholtzOp::m_loCFIVS[SpaceDim] [protected]

6.59.4.16 int HelmholtzOp::m_ncomp [protected]

6.59.4.17 QuadCFInterp HelmholtzOp::m_quadCFI [protected]

6.59.4.18 int HelmholtzOp::m_refRatio [protected]

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

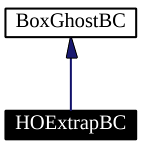
- **HelmholtzOp.H**

6.60 HOExtrapBC Class Reference

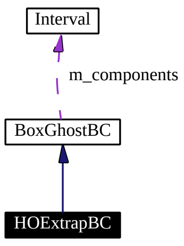
class to do extrapolation BC's

`#include <HOExtrapBC.H>`

Inheritance diagram for HOExtrapBC:



Collaboration diagram for HOExtrapBC:



Public Methods

- **HOExtrapBC** ()
- ~**HOExtrapBC** ()
- **HOExtrapBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- **HOExtrapBC** (int dir, **Side::LoHiSide** sd)

Protected Methods

- 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

  virtual constructor workaround.

Friends

- class **DomainGhostBC**

6.60.1 Detailed Description

class to do extrapolation BC's

6.60.2 Constructor & Destructor Documentation

6.60.2.1 HOExtrapBC::HOExtrapBC ()

6.60.2.2 HOExtrapBC::~HOExtrapBC ()

6.60.2.3 HOExtrapBC::HOExtrapBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.60.2.4 HOExtrapBC::HOExtrapBC (int *dir*, Side::LoHiSide *sd*)

6.60.3 Member Function Documentation

6.60.3.1 virtual void HOExtrapBC::applyHOExtrapBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

6.60.3.2 virtual void HOExtrapBC::applyHOExtrapBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

6.60.3.3 virtual void HOExtrapBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface

Reimplemented from **BoxGhostBC** (p. 134).

6.60.3.4 virtual void HOExtrapBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Reimplemented from **BoxGhostBC** (p. 134).

6.60.3.5 virtual void HOExtrapBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const [protected, virtual]

deprecated interface

Reimplemented from **BoxGhostBC** (p. 134).

6.60.3.6 virtual void HOExtrapBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

Reimplemented from **BoxGhostBC** (p. 134).

6.60.3.7 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. 135).

6.60.3.8 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. 135).

6.60.3.9 BoxGhostBC* HOExtrapBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround.

Implements **BoxGhostBC** (p. 135).

6.60.4 Friends And Related Function Documentation

6.60.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 135).

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

- **HOExtrapBC.H**

6.61 IndexType Class Reference

`#include <Box.H>`

Public Types

- enum **CellIndex** { **CELL** = 0, **NODE** = 1 }

Public Methods

- **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 Methods

- IndexType **TheCellType** ()
- IndexType **TheNodeType** ()

Friends

- std::ostream & **operator**<< (std::ostream &os, const IndexType &itype)
- std::istream & **operator**>> (std::istream &is, IndexType &itype)

6.61.1 Member Enumeration Documentation

6.61.1.1 enum IndexType::CellIndex

The cell index type: one of CELL or NODE.

Enumeration values:
  CELL
  NODE

6.61.2 Constructor & Destructor Documentation

6.61.2.1 IndexType::IndexType () [inline]

The default constructor.

6.61.2.2 IndexType::IndexType (const IndexType & *rhs*) [inline]

The copy constructor.

6.61.2.3 IndexType::IndexType (const IntVect & *iv*) [inline, explicit]

Constructs an IndexType identical to an **IntVect** (p. 264).

6.61.2.4 IndexType::IndexType (D_DECL(CellIndex i, CellIndex j, CellIndex k)) [inline]

Constructs an IndexType given an explicit CellIndex for each direction. D_DECL is a macro that sets the constructor to take BL_SPACEDIM arguments.

6.61.3 Member Function Documentation

6.61.3.1 bool IndexType::any () const [inline]

True if this IndexType is NODE based in any direction.

6.61.3.2 bool IndexType::cellCentered () const [inline]

True if this IndexType is CELL based in all directions.

6.61.3.3 void IndexType::clear () [inline]

Set this IndexType to be CELL based in all directions.

6.61.3.4 void IndexType::flip (int *i*) [inline]

Change this IndexType from CELL to NODE or NODE to CELL in direction dir.

6.61.3.5 IntVect IndexType::ixType () const [inline]

Convert this IndexType to an **IntVect** (p. 264) via component-wise conversion of enum CellIndex to int.

6.61.3.6 IndexType::CellIndex IndexType::ixType (int *dir*) const [inline]

Returns the CellIndex of this IndexType in direction dir.

6.61.3.7 bool IndexType::nodeCentered () const [inline]

True if this IndexType is NODE based in all directions.

6.61.3.8 bool IndexType::ok () const [inline]

True if this IndexType is valid.

6.61.3.9 bool IndexType::operator!= (const IndexType & *t*) const [inline]

True if IndexTypes are not identical.

6.61.3.10 IndexType & IndexType::operator= (const IndexType & *rhs*) [inline]

The assignment operator.

6.61.3.11 bool IndexType::operator== (const IndexType & *t*) const [inline]

True if IndexTypes are identical.

6.61.3.12 int IndexType::operator[] (int *dir*) const [inline]

Return an integer representing this IndexType in direction dir.

6.61.3.13 void IndexType::set (int *dir*) [inline]

Sets this IndexType to be NODE based in direction dir.

6.61.3.14 void IndexType::setall () [inline]

Set this IndexType to be NODE based in all directions.

6.61.3.15 void IndexType::setType (int *dir*, CellIndex *t*) [inline]

Set this IndexType to CellIndex type t in direction dir.

6.61.3.16 bool IndexType::test (int *dir*) const [inline]

True if this IndexType is NODE based in direction dir.

6.61.3.17 IndexType IndexType::TheCellType () [static]

This static member function returns an IndexType object all of whose components are of value **IndexType::CELL** (p. 256). It is provided as a convenience to our users when defining a **Box** (p. 106) all of whose faces should be of type **IndexType::CELL** (p. 256).

6.61.3.18 IndexType IndexType::TheNodeType () [static]

This static member function returns an IndexType object all of whose components are of value **IndexType::NODE** (p. 256). It is provided as a convenience to our users when defining a **Box** (p. 106) all of whose faces should be of type **IndexType::NODE** (p. 256).

6.61.3.19 void IndexType::unset (int *dir*) [inline]

Sets this IndexType to be CELL based in direction dir.

6.61.4 Friends And Related Function Documentation

6.61.4.1 std::ostream& operator<< (std::ostream & *os*, const IndexType & *itype*) [friend]

Write an IndexType to an ostream in ASCII.

6.61.4.2 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:

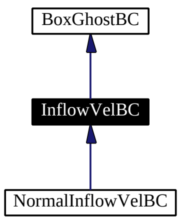
- **Box.H**

6.62 InflowVelBC Class Reference

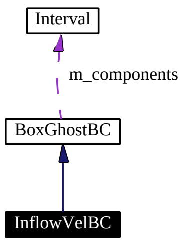
does inflow

`#include <basicVelBC.H>`

Inheritance diagram for InflowVelBC:



Collaboration diagram for InflowVelBC:



Public Methods

- **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 Methods

- 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

  virtual constructor workaround.

Friends

- class **DomainGhostBC**

6.62.1 Detailed Description

does inflow

6.62.2 Constructor & Destructor Documentation

6.62.2.1 InflowVelBC::InflowVelBC ()

6.62.2.2 virtual InflowVelBC::~InflowVelBC () `[virtual]`

6.62.2.3 InflowVelBC::InflowVelBC (int *dir*, Side::LoHiSide *sd*)

6.62.2.4 InflowVelBC::InflowVelBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.62.3 Member Function Documentation

6.62.3.1 Real InflowVelBC::BCVal () const

6.62.3.2 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. 135).

Reimplemented in **NormalInflowVelBC** (p. 356).

6.62.3.3 virtual void InflowVelBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dirfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const `[protected, virtual]`

Implements **BoxGhostBC** (p. 135).

6.62.3.4 bool InflowVelBC::isHomogeneous () const

6.62.3.5 void InflowVelBC::isHomogeneous (bool *a_isHomogeneous*)

is the BC homogeneous?

6.62.3.6 virtual BoxGhostBC* InflowVelBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround.

Implements **BoxGhostBC** (p. 135).

Reimplemented in **NormalInflowVelBC** (p. 356).

6.62.3.7 void InflowVelBC::setBCVal (const Real *a_bcVal*)

for single-valued inflow...

6.62.3.8 Real InflowVelBC::Time () const

6.62.3.9 void InflowVelBC::Time (const Real *m_time*)

set time (for time-varying inflow

6.62.4 Friends And Related Function Documentation

6.62.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 135).

Reimplemented in **NormalInflowVelBC** (p. 356).

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

- **basicVelBC.H**

6.63 Interval Struct Reference

Structure for passing component ranges in code.

`#include <Interval.H>`

Public Methods

- **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

6.63.1 Detailed Description

Structure for passing component ranges in code.

A class to encapsulate component ranges

6.63.2 Constructor & Destructor Documentation

6.63.2.1 Interval::Interval () `[inline]`

6.63.2.2 Interval::Interval (int *a_firstComp*, int *a_lastComp*) `[inline]`

initialize with first and last component numbers

6.63.3 Member Function Documentation

6.63.3.1 int Interval::begin () const `[inline]`

return first component number

6.63.3.2 void Interval::define (int *a_firstComp*, int *a_lastComp*) `[inline]`

6.63.3.3 int Interval::end () const `[inline]`

return last component number

6.63.3.4 int Interval::size () const `[inline]`

return last - first + 1

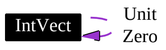
The documentation for this struct was generated from the following file:

- **Interval.H**

6.64 IntVect Class Reference

`#include <IntVect.H>`

Collaboration diagram for IntVect:



Public Methods

- **IntVect** ()

  {\bf Constructors and Accessors}

- **~IntVect** ()
- **IntVect** (D_DECL(int i, int j, int k))
- **IntVect** (const int *a)
- const int * **dataPtr** () const
- **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)
- const int * **getVect** () const
- bool **operator==** (const IntVect &p) const

  {\bf 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 **lexLT** (const IntVect &s) const
- bool **lexGT** (const IntVect &s) const
- IntVect **operator+** () const

  {\bf Arithmetic Operators}

- IntVect **operator-** () const
- IntVect & **operator+=** (int s)

  addition

- IntVect & **operator+=** (const IntVect &p)
- IntVect **operator+** (const IntVect &p) const
- IntVect **operator+** (int s) const
- IntVect & **operator-=** (int s)

  subtraction

- IntVect & **operator-=** (const IntVect &p)

- IntVect **operator-** (const IntVect &p) const
- IntVect **operator-** (int s) const
- IntVect & **operator** ∗= (int s)

  multiplication

- IntVect & **operator** ∗= (const IntVect &p)
- IntVect **operator** ∗ (const IntVect &p) const
- IntVect **operator** ∗ (int s) const
- IntVect & **operator/=** (int s)

  division

- IntVect & **operator/=** (const IntVect &p)
- IntVect **operator/** (const IntVect &p) const
- IntVect **operator/** (int s) const
- IntVect & **min** (const IntVect &p)

  other

- 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)
- void **printOn** (std::ostream &os) const

  {\bf I/O Functions}

- void **p** () const
- void **dumpOn** (std::ostream &os) const

Static Public Methods

- const IntVect **TheZeroVector** ()
- const IntVect **TheUnitVector** ()

Static Public Attributes

- const IntVect **Zero**
- const IntVect **Unit**

Protected Attributes

- int **vect** [**SpaceDim**]

Static Protected Attributes

- const size_t **IntVectSize**

Friends

- class **Box**
- class **HDF5Handle**
- class **VolIndex**
- class **FaceIndex**
- IntVect **operator+** (int s, const IntVect &p)
- IntVect **operator-** (int s, const IntVect &p)
- IntVect **operator** * (int s, const IntVect &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)
- std::ostream & **operator<<** (std::ostream &os, const IntVect &iv)
- std::istream & **operator>>** (std::istream &os, IntVect &iv)
- IntVect **BASISV** (int dir)

  {*bf IntVect Constants*}

6.64.1 Constructor & Destructor Documentation

6.64.1.1 IntVect::IntVect () `[inline]`

{\bf Constructors and Accessors}

Construct an IntVect whose components are uninitialized.

6.64.1.2 IntVect::~IntVect () `[inline]`

Destructor

6.64.1.3 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.

6.64.1.4 IntVect::IntVect (const int * *a*) `[inline, explicit]`

Construct an IntVect setting the coordinates to the corresponding values in the integer array a.

6.64.1.5 IntVect::IntVect (const IntVect & *rhs*) `[inline]`

The copy constructor.

6.64.2 Member Function Documentation

6.64.2.1 IntVect& IntVect::coarsen (int *p*)

Modify IntVect by component-wise integer projection.

6.64.2.2 IntVect& IntVect::coarsen (const IntVect & *p*)

Modify IntVect by component-wise integer projection.

6.64.2.3 IntVect IntVect::copy () const `[inline]`

6.64.2.4 const int * IntVect::dataPtr () const `[inline]`

only for sending to fortran

6.64.2.5 IntVect & IntVect::diagShift (int *s*) `[inline]`

Modifies this IntVect by adding a scalar s to each component.

6.64.2.6 void IntVect::dumpOn (std::ostream & *os*) const

Print an IntVect to the ostream a bit more verbosely.

6.64.2.7 const int * IntVect::getVect () const `[inline]`

Returns a const pointer to an array of coordinates of the IntVect. Useful for arguments to FORTRAN calls.

6.64.2.8 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])); \

int 3-D: \ (iv1[0] > iv2[0]) || (iv1[0]==iv2[0] && ((iv1[1] > iv2[1] || ((iv1[1] == iv2[1]) && (iv1[2] > iv2[2]))))); \

6.64.2.9 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]))))); \

6.64.2.10 IntVect & IntVect::max (const IntVect & *p*) [inline]

Modifies this IntVect by taking component-wise max with //IntVect argument.

6.64.2.11 IntVect & IntVect::min (const IntVect & *p*) [inline]

other

Modifies this IntVect by taking component-wise min with IntVect argument.

6.64.2.12 IntVect IntVect::operator * (int *s*) const [inline]

Returns an IntVect that is this IntVect with each component multiplied by a scalar.

6.64.2.13 IntVect IntVect::operator * (const IntVect & *p*) const [inline]

Returns component-wise product of this IntVect and argument.

6.64.2.14 IntVect & IntVect::operator *= (const IntVect & *p*) [inline]

Modifies this IntVect by component-wise multiplication with argument.

6.64.2.15 IntVect & IntVect::operator *= (int *s*) [inline]

multiplication

Modifies this IntVect by multiplication of a scalar to each component.

6.64.2.16 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.

6.64.2.17 IntVect IntVect::operator+ (int *s*) const [inline]

Return an IntVect that is this IntVect with a scalar added to each component.

6.64.2.18 IntVect IntVect::operator+ (const IntVect & *p*) const [inline]

Returns component-wise sum of this IntVect and argument.

6.64.2.19 IntVect IntVect::operator+ () const [inline]

{\bf Arithmetic Operators}

Unary plus – for completeness.

6.64.2.20 IntVect & IntVect::operator+= (const IntVect & *p*) [inline]

Modifies this IntVect by component-wise addition with argument.

6.64.2.21 IntVect & IntVect::operator+= (int *s*) [inline]

addition

Modifies this IntVect by addition of a scalar to each component.

6.64.2.22 IntVect IntVect::operator- (int *s*) const [inline]

Returns an IntVect that is this IntVect with a scalar s subtracted from each component.

6.64.2.23 IntVect IntVect::operator- (const IntVect & *p*) const [inline]

Returns component-wise difference of this IntVect and argument.

6.64.2.24 IntVect IntVect::operator- () const [inline]

Unary Minus – negates all components of this IntVect.

6.64.2.25 IntVect & IntVect::operator-= (const IntVect & *p*) [inline]

Modifies this IntVect by component-wise subtraction with argument.

6.64.2.26 IntVect & IntVect::operator-= (int *s*) [inline]

subtraction

Modifies this IntVect by subtraction of a scalar from each component.

6.64.2.27 IntVect IntVect::operator/ (int *s*) const [inline]

Returns component-wise division of this IntVect by scalar s.

6.64.2.28 IntVect IntVect::operator/ (const IntVect & *p*) const [inline]

Returns component-wise division of this IntVect by IntVect argument.

6.64.2.29 IntVect & IntVect::operator/= (const IntVect & *p*) [inline]

Modifies this IntVect by component-wise division with IntVect argument.

6.64.2.30 IntVect & IntVect::operator/= (int *s*) [inline]

division

Modifies this IntVect by division by a scalar of each component.

6.64.2.31 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 componentwise, it is possible for an IntVect to be neither greater than, less than, nor equal to another.

6.64.2.32 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 componentwise, it is possible for an IntVect to be neither greater than or equal to, less than or equal to, nor equal to another.

6.64.2.33 IntVect & IntVect::operator= (const IntVect & *rhs*) [inline]

The assignment operator.

6.64.2.34 bool IntVect::operator== (const IntVect & *p*) const [inline]

{\bf Comparison Operators}

Returns true if this IntVect is equivalent to argument IntVect. All comparisons between analogous components must be satisfied.

6.64.2.35 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 componentwise, it is possible for an IntVect to be neither greater than, less than, nor equal to another.

6.64.2.36 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 componentwise, it is possible for an IntVect to be neither greater than or equal to, less than or equal to, nor equal to another.

6.64.2.37 int IntVect::operator[] (int *i*) const [inline]

Returns the i'th coordinate of the IntVect.

6.64.2.38 int & IntVect::operator[] (int *i*) [inline]

Returns a modifiable lvalue reference to the i'th coordinate of the IntVect.

6.64.2.39 void IntVect::p () const

Print an IntVect to the **pout**() (p. 664).

6.64.2.40 void IntVect::printOn (std::ostream & *os*) const

{\bf I/O Functions}

Print an IntVect to the ostream.

6.64.2.41 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 zero-based.

6.64.2.42 IntVect & IntVect::scale (int *s*) [inline]

Modifies this IntVect by multiplying each component by a scalar.

6.64.2.43 void IntVect::setVal (int *i*, int *val*) [inline]

Set i'th coordinate of IntVect to val.

6.64.2.44 IntVect & IntVect::shift (const IntVect & *iv*) [inline]

Modifies this IntVect by component-wise addition with IntVect argument.

6.64.2.45 IntVect & IntVect::shift (int *coord*, int *s*) [inline]

Modifies this IntVect by adding s to component in given coordinate direction.

6.64.2.46 const IntVect IntVect::TheUnitVector () [inline, static]

6.64.2.47 const IntVect IntVect::TheZeroVector () [inline, static]

6.64.3 Friends And Related Function Documentation

6.64.3.1 IntVect BASISV (int *dir*) [friend]

{\bf IntVect Constants}

Returns a basis vector in the given coordinate direction; eg. IntVect3d BASISV(1) == (0,1,0). Note that the coordinate directions are zero based.

6.64.3.2 friend class Box [friend]

6.64.3.3 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.

6.64.3.4 IntVect coarsen (const IntVect & *p*, int *s*) [friend]

Returns an IntVect that is the component-wise integer projection of p by s.

6.64.3.5 IntVect diagShift (const IntVect & *p*, int *s*) [friend]

Returns IntVect obtained by adding a scalar to each of the components of the given IntVect.

6.64.3.6 friend class FaceIndex [friend]

6.64.3.7 friend class HDF5Handle [friend]

6.64.3.8 IntVect max (const IntVect & *p1*, const IntVect & *p2*) [friend]

Returns the IntVect that is the component-wise minimum of two argument IntVects.

6.64.3.9 IntVect min (const IntVect & *p1*, const IntVect & *p2*) [friend]

Returns the IntVect that is the component-wise minimum of two argument IntVects.

6.64.3.10 IntVect operator * (int *s*, const IntVect & *p*) [friend]

Returns and IntVect that is an IntVect p with each component multiplied by a scalar s.

6.64.3.11 IntVect operator+ (int *s*, const IntVect & *p*) [friend]

Returns an IntVect that is an IntVect p with a scalar s added to each component.

6.64.3.12 IntVect operator- (int *s*, const IntVect & *p*) [friend]

Returns -p + s.

6.64.3.13 std::ostream& operator<< (std::ostream & *os*, const IntVect & *iv*) [friend]

Print the IntVect to given output stream in ASCII.

6.64.3.14 std::istream& operator>> (std::istream & *os*, IntVect & *iv*) [friend]

Read next IntVect from given input stream.

6.64.3.15 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.

6.64.3.16 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.

6.64.3.17 friend class VolIndex [friend]

6.64.4 Member Data Documentation

6.64.4.1 const size_t IntVect::IntVectSize [static, protected]

6.64.4.2 const IntVect IntVect::Unit [static]

This is an IntVect all of whose components are equal to one.

6.64.4.3 int IntVect::vect[SpaceDim] [protected]

6.64.4.4 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:

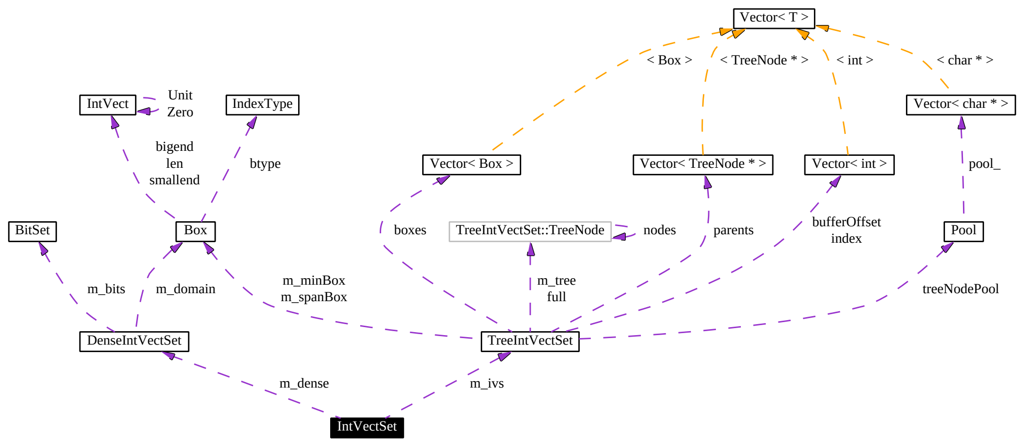
- **IntVect.H**

6.65 IntVectSet Class Reference

An irregular domain on an integer lattice.

`#include <IntVectSet.H>`

Collaboration diagram for IntVectSet:



Public Methods

- ~**IntVectSet** ()

  {\bf Constructors, Destructor, and defines}

- **IntVectSet** ()
- void **define** ()
- **IntVectSet** (const IntVectSet &ige_in)
- void **define** (const IntVectSet &ige_in)
- IntVectSet **copy** () const
- void **define_intvectset** (const IntVectSet &ige_in)
- **IntVectSet** (const **DenseIntVectSet** &a_dense)
- void **define** (const **DenseIntVectSet** &a_dense)
- **IntVectSet** (const **TreeIntVectSet** &a_tree)
- void **define** (const **TreeIntVectSet** &a_tree)
- **IntVectSet** (const **IntVect** &iv_in)
- void **define** (const **IntVect** &iv_in)
- void **define_intvect** (const **IntVect** &iv_in)
- **IntVectSet** (const **Box** &b)
- void **define** (const **Box** &b)
- void **define_box** (const **Box** &b)
- IntVectSet & **operator**\|= (const IntVectSet &ivs)
- void **or_intvectset** (const IntVectSet &ivs)
- IntVectSet & **operator**\|= (const **IntVect** &iv)
- void **or_intvect** (const **IntVect** &ivs)
- IntVectSet & **operator**\|= (const **Box** &b)
- void **or_box** (const **Box** &b)
- IntVectSet **operator-** (const IntVectSet &ivs) const

{*bf complement operators*}

- IntVectSet **operator-** (const **Box** &b) const
- IntVectSet **operator-** (const **IntVect** &iv) const
- IntVectSet & **operator-=** (const IntVectSet &ivs)
- void **minus** (const IntVectSet &ivs)
- IntVectSet & **operator-=** (const **Box** &b)
- void **minus_box** (const **Box** &b)
- IntVectSet & **operator-=** (const **IntVect** &iv)
- void **minus_intvect** (const **IntVect** &iv)
- IntVectSet & **operator &=** (const IntVectSet &ivs)
- void **and_intvectset** (const IntVectSet &ivs)
- IntVectSet & **operator &=** (const **Box** &b)
- void **and_box** (const **Box** &b)
- IntVectSet & **operator &=** (const **ProblemDomain** &domain)
- void **and_domain** (const **ProblemDomain** &d)
- void **grow** (int igrow)

{*bf modification functions*}

- IntVectSet & **grow** (int idir, int igrow)
- void **grow_dir** (int idir, int igrow)
- IntVectSet & **refine** (int iref=2)
- IntVectSet & **coarsen** (int iref=2)
- void **shift** (const **IntVect** &iv)
- void **nestingRegion** (int radius, const **Box** &domain)
- void **nestingRegion** (int radius, const **ProblemDomain** &domain)
- void **nestingRegion_prob** (int radius, const **ProblemDomain** &domain)
- void **makeEmpty** ()
- IntVectSet **chop** (int dir, int chop_pnt)
- int **numPts** () const

{*bf accessor and inquiry functions*}

- const **Box** & **minBox** () const
- bool **isEmpty** () const
- bool **isDense** () const
- bool **contains** (const **IntVect** &iv) const
- bool **contains** (const IntVectSet &iv) const
- bool **contains_intvectset** (const IntVectSet &iv) const
- bool **contains** (const **Box** &box) const
- bool **contains_box** (const **Box** &b) const
- **Vector< Box > boxes** () const
- void **compact** () const
- void **printBoxes** (std::ostream &a_ostrm) const
- void **p** () const
- void **convert** () const

Static Public Attributes

- long int **count**
- long int **peakcount**

Friends

- class **IVSIterator**
- IntVectSet **operator|** (const IntVectSet &ivs1, const IntVectSet &ivs2)

  {\bf *union operators*}

- IntVectSet **operator|** (const IntVectSet &ivs, const **IntVect** &iv)
- IntVectSet **operator|** (const **IntVect** &iv, const IntVectSet &ivs)
- IntVectSet **operator|** (const IntVectSet &ivs, const **Box** &b)
- IntVectSet **operator|** (const **Box** &b, const IntVectSet &ivs)
- IntVectSet **operator &** (const IntVectSet &ivs1, const IntVectSet &ivs2)

  {\bf *Intersection operators*}

- IntVectSet **operator &** (const IntVectSet &ivs, const **Box** &b)
- IntVectSet **operator &** (const **Box** &b, const IntVectSet &ivs)
- IntVectSet **grow** (const IntVectSet &ivs, int igrow)
- IntVectSet **refine** (const IntVectSet &ivs, int iref=2)
- IntVectSet **coarsen** (const IntVectSet &ivs, int iref=2)
- std::ostream & **operator<<** (std::ostream &os, const IntVectSet &ivs)
- void **dumpmemoryatexit** ()

6.65.1 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** (p. 264) or cell-centered **Box** (p. 106) 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** (p. 264) or an IntVectSet and a **Box** (p. 106)). The minimum **Box** (p. 106) of an IntVectSet is defined as the smallest **Box** (p. 106) that contains every **IntVect** (p. 264) in the IntVectSet.

The IntVects in an IntVectSet do not have a canonical ordering.

6.65.2 Constructor & Destructor Documentation

6.65.2.1 IntVectSet::~IntVectSet ()

{\bf Constructors, Destructor, and defines}

6.65.2.2 IntVectSet::IntVectSet () `[inline]`

default constructor: defines an empty IntVectSet.

6.65.2.3 IntVectSet::IntVectSet (const IntVectSet & *ige_in*) `[inline]`

copy constructor: defines this to be a copy of ige_in.

6.65.2.4 IntVectSet::IntVectSet (const DenseIntVectSet & *a_dense*) [explicit]

conversion constructor.

6.65.2.5 IntVectSet::IntVectSet (const TreeIntVectSet & *a_tree*) [explicit]

conversion constructor.

6.65.2.6 IntVectSet::IntVectSet (const IntVect & *iv_in*) [explicit]

IntVect (p. 264) constructor: defines this to be an IntVectSet with just one **IntVect** (p. 264).

6.65.2.7 IntVectSet::IntVectSet (const Box & *b*) [explicit]

Box (p. 106) constructor: defines this to be an IntVectSet with all the cells in a **Box** (p. 106).

6.65.3 Member Function Documentation

6.65.3.1 void IntVectSet::and_box (const Box & *b*) [inline]

6.65.3.2 void IntVectSet::and_domain (const ProblemDomain & *d*) [inline]

6.65.3.3 void IntVectSet::and_intvectset (const IntVectSet & *ivs*) [inline]

6.65.3.4 Vector<Box> IntVectSet::boxes () const

Returns **Vector** (p. 471)<**Box** (p. 106)> representation of this IntVectSet.

6.65.3.5 IntVectSet IntVectSet::chop (int *dir*, int *chop_pnt*)

Chop the IntVectSet at the chop_pnt in the dir direction \ Returns one IntVectSet and modifies the object IntVectSet. \ The union of the two is the original IntVectSet. \ The modified IntVectSet is the low end, the returned IntvectSet \ is the high end.

6.65.3.6 IntVectSet& IntVectSet::coarsen (int *iref* = 2)

Modifies this IntVectSet by coarsening it by a specified factor iref. It is an error if iref <= 0. Definition of coarsening: For each **IntVect** (p. 264) iv in the original IntVectSet, the refined IntVectSet will contain the **IntVect** (p. 264) coarsen(iv,iref).\ {\bf This:} \ —This object is modified.—

6.65.3.7 void IntVectSet::compact () const

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

6.65.3.8 bool IntVectSet::contains (const Box & *box*) const

is every **IntVect** (p. 264) in this 'box' contained in the IntvectSet. note: this is not performed in a crude point-wise fashion and is faster than a user doing it pointwise themselves

6.65.3.9 bool IntVectSet::contains (const IntVectSet & *iv*) const

Does this IntVectSet contain the argument IntVectSet?

6.65.3.10 bool IntVectSet::contains (const IntVect & *iv*) const

Does this IntVectSet contain the argument **IntVect** (p. 264)?

6.65.3.11 bool IntVectSet::contains_box (const Box & *b*) const `[inline]`

6.65.3.12 bool IntVectSet::contains_intvectset (const IntVectSet & *iv*) const `[inline]`

6.65.3.13 void IntVectSet::convert () const

6.65.3.14 IntVectSet IntVectSet::copy () const `[inline]`

6.65.3.15 void IntVectSet::define (const Box & *b*)

6.65.3.16 void IntVectSet::define (const IntVect & *iv_in*)

6.65.3.17 void IntVectSet::define (const TreeIntVectSet & *a_tree*)

6.65.3.18 void IntVectSet::define (const DenseIntVectSet & *a_dense*)

6.65.3.19 void IntVectSet::define (const IntVectSet & *ige_in*) `[inline]`

6.65.3.20 void IntVectSet::define ()

6.65.3.21 void IntVectSet::define_box (const Box & *b*) `[inline]`

6.65.3.22 void IntVectSet::define_intvect (const IntVect & *iv_in*) `[inline]`

6.65.3.23 void IntVectSet::define_intvectset (const IntVectSet & *ige_in*) `[inline]`

6.65.3.24 IntVectSet& IntVectSet::grow (int *idir*, int *igrow*)

Modifies this IntVectSet by growing it by the specified number of cells in the specified coordinate direction. Directions are zero-based.\ {\bf Arguments:} \ idir (not modified): zero-based coordinate direction to grow in.\ igrow (not modified): number of cells to grow.\ {\bf Returns:} \ This IntVectSet grown by igrow the direction idir.\ {\bf This:} \ —This object is modified.—

6.65.3.25 void IntVectSet::grow (int *igrow*)

{\bf modification functions}

Modifies this IntVectSet by growing it by the specified number of cells in all directions, including diagonal. It is an error if igrow < 0. \ {\bf Arguments:} \ igrow (not modified): number of cells to grow.\ {\bf Returns:} \ This IntVectSet grown by igrow in all directions.\ {\bf This:} \ —This object is modified.—

6.65.3.26 void IntVectSet::grow_dir (int *idir*, int *igrow*) `[inline]`

6.65.3.27 bool IntVectSet::isDense () const `[inline]`

Returns true if this IntVectSet is currently being represented in a dense fashion

6.65.3.28 bool IntVectSet::isEmpty () const

Returns true if no cells are in this IntVectSet.

6.65.3.29 void IntVectSet::makeEmpty ()

sets this IntVectSet to be empty

6.65.3.30 const Box& IntVectSet::minBox () const

Returns minimum enclosing box of this IntVectSet.

6.65.3.31 void IntVectSet::minus (const IntVectSet & *ivs*) `[inline]`

6.65.3.32 void IntVectSet::minus_box (const Box & *b*) `[inline]`

6.65.3.33 void IntVectSet::minus_intvect (const IntVect & *iv*) `[inline]`

6.65.3.34 void IntVectSet::nestingRegion (int *radius*, const ProblemDomain & *domain*)

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.

6.65.3.35 void IntVectSet::nestingRegion (int *radius*, const Box & *domain*)

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

6.65.3.36 void IntVectSet::nestingRegion_prob (int *radius*, const ProblemDomain & *domain*) `[inline]`

6.65.3.37 int IntVectSet::numPts () const

{\bf accessor and inquiry functions}

Returns number of IntVects in this IntVectSet.

6.65.3.38 IntVectSet& IntVectSet::operator &= (const ProblemDomain & *domain*)

Modifies this IntVectSet to be its intersection with a **ProblemDomain** (p. 418). \ {\bf Arguments:} \ domain (not modified): \ {\bf Returns:} \ The intersection of the this IntVectSet and the **ProblemDomain** (p. 418).\ {\bf This:} \ —This object is modified.—

6.65.3.39 IntVectSet& IntVectSet::operator &= (const Box & *b*)

Modifies this IntVectSet to be its intersection with a **Box** (p. 106). \ {\bf Arguments:} \ box (not modified): \ {\bf Returns:} \ The intersection of the this IntVectSet and the **Box** (p. 106).\ {\bf This:} \ —This object is modified.—

6.65.3.40 IntVectSet& IntVectSet::operator &= (const IntVectSet & *ivs*)

Sets this IntVectSet to its intersection with another IntVectSet. \ {\bf Arguments:} \ ivs (not modified): \ {\bf Returns:} \ The intersection of the two IntVectSets.\ {\bf This:} \ —This object is modified.—

6.65.3.41 IntVectSet IntVectSet::operator- (const IntVect & *iv*) const

Return the complement of the argument **IntVect** (p. 264) within this IntVectSet.

6.65.3.42 IntVectSet IntVectSet::operator- (const Box & *b*) const

Return the complement of the argument **Box** (p. 106) within this IntVectSet.

6.65.3.43 IntVectSet IntVectSet::operator- (const IntVectSet & *ivs*) const

{\bf complement operators}

Return the complement of the argument IntVectSet within this IntVectSet.

6.65.3.44 IntVectSet& IntVectSet::operator-= (const IntVect & *iv*)

Modifies this IntVectSet to be the complement of the argument **IntVect** (p. 264) within this IntVectSet.\ {\bf This:} \ —This object is modified.—

6.65.3.45 IntVectSet& IntVectSet::operator-= (const Box & *b*)

Modifies this IntVectSet to be the complement of the argument **Box** (p. 106) within this IntVectSet. \ {\bf This:} \ —This object is modified.—

6.65.3.46 IntVectSet& IntVectSet::operator-= (const IntVectSet & *ivs*)

Modifies this IntVectSet to be the complement of the argument IntVectSet within this IntVectSet. \ {\bf This:} \ —This object is modified.—

6.65.3.47 IntVectSet& IntVectSet::operator\|= (const Box & *b*)

Sets this IntVectSet to its union with a **Box** (p. 106). \ {\bf Arguments:} \ box (not modified): \ {\bf Returns:} \ The union of this **IntVect** (p. 264) and the **Box** (p. 106).\ {\bf This:} \ —The object is modified—

6.65.3.48 IntVectSet& IntVectSet::operator\|= (const IntVect & *iv*)

Sets this IntVectSet to its union with an **IntVect** (p. 264). \ {\bf Arguments:} \ iv (not modified): \ {\bf Returns:} \ The union of this IntVectSet and the **IntVect** (p. 264).\ {\bf This:} \ —The object is modified—

6.65.3.49 IntVectSet& IntVectSet::operator\|= (const IntVectSet & *ivs*)

Sets this IntVectSet to its union with another IntVectSet. \ {\bf Arguments:} \ ivs (not modified): \ {\bf Returns:} \ The union of the two IntVectSets.\ {\bf This:} \ —The object is modified—

6.65.3.50 void IntVectSet::or_box (const Box & *b*) `[inline]`

6.65.3.51 void IntVectSet::or_intvect (const IntVect & *ivs*) `[inline]`

6.65.3.52 void IntVectSet::or_intvectset (const IntVectSet & *ivs*) `[inline]`

6.65.3.53 void IntVectSet::p () const `[inline]`

6.65.3.54 void IntVectSet::printBoxes (std::ostream & *a_ostrm*) const

Write **Vector** (p. 471)<**Box** (p. 106)> representation to an output stream.

6.65.3.55 IntVectSet& IntVectSet::refine (int *iref* = 2)

Modifies this IntVectSet by refining it by a specified factor iref. It is an error if iref <= 0. Definition of refinement: For each **IntVect** (p. 264) iv in the original IntVectSet, the refined IntVectSet will contain the **Box** (p. 106) refine(Box(iv,iv),iref).\ {\bf This:} \ —This object is modified.—

6.65.3.56 void IntVectSet::shift (const IntVect & *iv*)

Increment all the IntVects in this IntVectSet by the given iv.\ –this IS changed–

6.65.4 Friends And Related Function Documentation

6.65.4.1 IntVectSet coarsen (const IntVectSet & *ivs*, int *iref* = 2) `[friend]`

Returns the IntVectSet that is the argument IntVectSet coarsened by a specified factor iref. It is an error if iref <= 0. Definition of coarsening: For each **IntVect** (p. 264) iv in the original IntVectSet, the refined IntVectSet will contain the **IntVect** (p. 264) coarsen(iv,iref).

6.65.4.2 void dumpmemoryatexit () [friend]

6.65.4.3 IntVectSet grow (const IntVectSet & *ivs*, int *igrow*) [friend]

Returns the argument IntVectSet grown it by the specified number of cells in all directions, including diagonal. \ {\bf Arguments:} \ ivs (not modified): \ igrow (not midified): number of cells to grow.\ {\bf Returns:} \ The argument IntVectSet grown by igrow in all directions.\

6.65.4.4 friend class IVSIterator [friend]

6.65.4.5 IntVectSet operator & (const Box & *b*, const IntVectSet & *ivs*) [friend]

Returns the intersection of an IntVectSet and a **Box** (p. 106). \ {\bf Arguments:} \ ivs (not modified): \ {\bf Returns:} \ The intersection of the IntVectSet and the **Box** (p. 106).\

6.65.4.6 IntVectSet operator & (const IntVectSet & *ivs*, const Box & *b*) [friend]

Returns the intersection of an IntVectSet and a **Box** (p. 106). \ {\bf Arguments:} \ ivs (not modified): \ {\bf Returns:} \ The intersection of the IntVectSet and the **Box** (p. 106).\

6.65.4.7 IntVectSet operator & (const IntVectSet & *ivs1*, const IntVectSet & *ivs2*) [friend]

{\bf Intersection operators}

Returns the intersection of two IntVectSets. The result may be empty.\ {\bf Arguments:} \ ivs1, ivs2 (not modified): \ {\bf Returns:} \ The intersection of the two IntVectSets ivs1 and ivs2.\

6.65.4.8 std::ostream& operator<< (std::ostream & *os*, const IntVectSet & *ivs*) [friend]

6.65.4.9 IntVectSet operator| (const Box & *b*, const IntVectSet & *ivs*) [friend]

Returns the union of an IntVectSet and a **Box** (p. 106). \ {\bf Arguments:} \ ivs (not midified): \ b (not modified): \ {\bf Returns:} \ The union of the IntVectSet ivs and the **Box** (p. 106) b.\

6.65.4.10 IntVectSet operator| (const IntVectSet & *ivs*, const Box & *b*) [friend]

Returns the union of an IntVectSet and a **Box** (p. 106). \ {\bf Arguments:} \ ivs (not midified): \ b (not modified): \ {\bf Returns:} \ The union of the IntVectSet ivs and the **Box** (p. 106) b.\

6.65.4.11 IntVectSet operator| (const IntVect & *iv*, const IntVectSet & *ivs*) [friend]

Returns the union of an IntVectSet and an **IntVect** (p. 264). \ {\bf Arguments:} \ ivs (not modified) \ iv (not modified): \ {\bf Returns:} \ The union of the IntVectSet ivs and the **IntVect** (p. 264) iv.\

6.65.4.12 IntVectSet operator| (const IntVectSet & *ivs*, const IntVect & *iv*) [friend]

Returns the union of an IntVectSet and an **IntVect** (p. 264). \ {\bf Arguments:} \ ivs (not modified) \ iv (not modified): \ {\bf Returns:} \ The union of the IntVectSet ivs and the **IntVect** (p. 264) iv.\

6.65.4.13 IntVectSet operator| (const IntVectSet & *ivs1*, const IntVectSet & *ivs2*) [friend]

{\bf union operators}

Returns the union of two IntVectSets. \ {\bf Arguments:} \ ivs1, ivs2 (not modified): \ {\bf Returns:} \ The union of the two IntVectSets ivs1 and ivs2.\

6.65.4.14 IntVectSet refine (const IntVectSet & *ivs*, int *iref* = 2) [friend]

Returns the IntVectSet that is the argument IntVectSet refined by a specified factor iref. It is an error if iref <= 0. Definition of refinement: For each **IntVect** (p. 264) iv in the original IntVectSet, the refined IntVectSet will contain the **Box** (p. 106) refine(Box(iv,iv),iref).

6.65.5 Member Data Documentation

6.65.5.1 long int IntVectSet::count [static]

6.65.5.2 long int IntVectSet::peakcount [static]

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

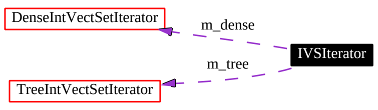
- **IntVectSet.H**

6.66 IVSIterator Class Reference

Iterator for an **IntVectSet** (p. 274).

`#include <IntVectSet.H>`

Collaboration diagram for IVSIterator:



Public Methods

- **IVSIterator** ()
- **IVSIterator** (const **IntVectSet** &ivs)
- ~**IVSIterator** ()
- void **define** (const **IntVectSet** &ivs)
- const **IntVect** & **operator()** () const
- const **IntVect** & **iv** () const
- bool **ok** () const
- void **operator++** ()
- void **incr** ()
- void **begin** ()
- void **reset** ()
- void **end** ()

6.66.1 Detailed Description

Iterator for an **IntVectSet** (p. 274).

IVSIterator iterates over every point (**IntVect** (p. 264)) in an **IntVectSet** (p. 274). It has exactly the same syntax and sematic as **BoxIterator** (p. 136). Typical usage:\

\begin{verbatim} **IntVectSet** (p. 274) ivs; ... IVSIterator ivsit (ivs); for (ivsit.begin(); ivsit.ok(); ++ivsit) { **IntVect** (p. 264) iv = ivsit(); (do operations involving iv) } \end{verbatim}

6.66.2 Constructor & Destructor Documentation

6.66.2.1 IVSIterator::IVSIterator () [inline]

6.66.2.2 IVSIterator::IVSIterator (const IntVectSet & *ivs*)

6.66.2.3 IVSIterator::~IVSIterator () [inline]

6.66.3 Member Function Documentation

6.66.3.1 void IVSIterator::begin () [inline]

6.66.3.2 void IVSIterator::define (const IntVectSet & *ivs*)

6.66.3.3 void IVSIterator::end () [inline]

6.66.3.4 void IVSIterator::incr () [inline]

6.66.3.5 const IntVect& IVSIterator::iv () const [inline]

6.66.3.6 bool IVSIterator::ok () const [inline]

6.66.3.7 const IntVect & IVSIterator::operator() () const [inline]

6.66.3.8 void IVSIterator::operator++ () [inline]

6.66.3.9 void IVSIterator::reset () [inline]

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

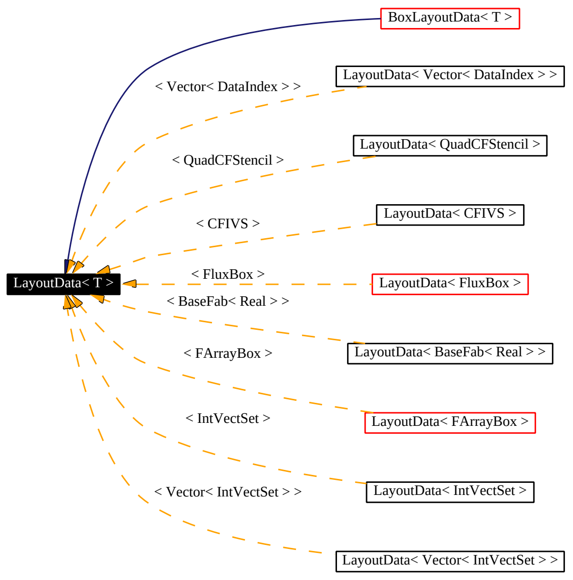
- **IntVectSet.H**

6.67 LayoutData< T > Class Template Reference

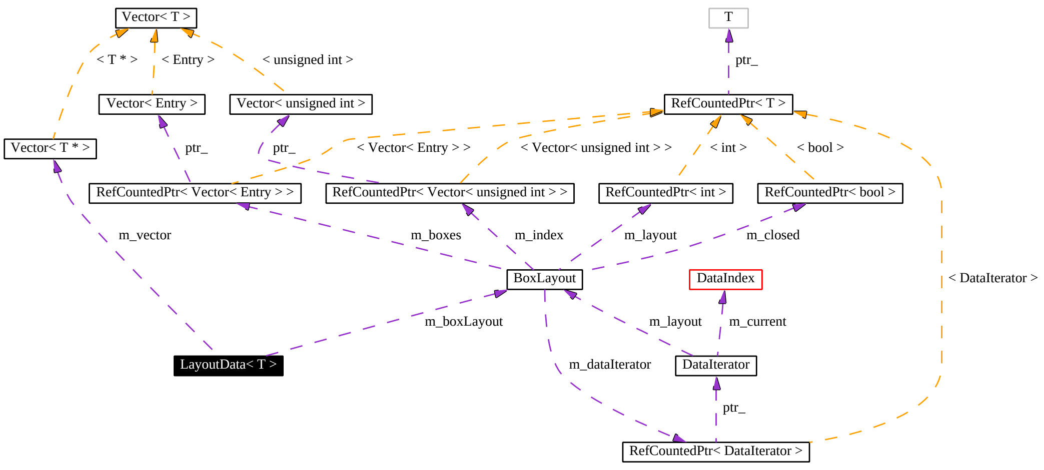
Data that maintains a one-to-one mapping to the boxes in a **BoxLayout** (p. 139).

`#include <LayoutData.H>`

Inheritance diagram for LayoutData< T >:



Collaboration diagram for LayoutData< T >:



Public Methods

- **LayoutData** ()
- **LayoutData** (const **BoxLayout** &dp)
- virtual ∼**LayoutData** ()
- virtual void **define** (const **BoxLayout** &dp)

- **DataIterator dataIterator** () const
- const T & **operator**[ ] (const **DataIndex** &index) const
- T & **operator**[ ] (const **DataIndex** &index)
- **Box box** (const **DataIndex** &index) const
- const **BoxLayout** & **boxLayout** () const

Protected Attributes

- **BoxLayout m_boxLayout**
- **Vector**< T * > **m_vector**

Friends

- class **ArrayViewData**

6.67.1 Detailed Description

template<class T> class LayoutData< T >

Data that maintains a one-to-one mapping to the boxes in a **BoxLayout** (p. 139).

A collection of **Box** (p. 106)-oriented objects. The arrangement of the T objects is given by the underlying **BoxLayout** (p. 139) 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** (p. 139). 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** (p. 196) 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: \begin{verbatim} { T() } \end{verbatim}

This class is useful for processor-local data that needs to be indexable along with a **BoxLayout-Data** (p. 147). Auxillary data objects, etc.

6.67.2 Constructor & Destructor Documentation

6.67.2.1 template<class T> LayoutData< T >::LayoutData () `[inline]`

6.67.2.2 template<class T> LayoutData< T >::LayoutData (const BoxLayout & *dp*) `[inline]`

constructor. allocates a T object for every box in the **BoxLayout** (p. 139) dp using the T null constructor. dp must be closed.

6.67.2.3 template<class T> LayoutData< T >::~LayoutData () [inline, virtual]

6.67.3 Member Function Documentation

6.67.3.1 template<class T> Box LayoutData< T >::box (const DataIndex & *index*) const [inline]

6.67.3.2 template<class T> const BoxLayout& LayoutData< T >::boxLayout () const [inline]

6.67.3.3 template<class T> DataIterator LayoutData< T >::dataIterator () const [inline]

6.67.3.4 template<class T> void LayoutData< T >::define (const BoxLayout & *dp*) [inline, virtual]

constructor. allocates a T object for every box in the **BoxLayout** (p. 139) dp using the T null constructor. dp must be closed. previously stored T objects are removed

Reimplemented in **BoxLayoutData< T >** (p. 149), **LevelData< T >** (p. 298), **BoxLayoutData< FluxBox >** (p. 149), **BoxLayoutData< FArrayBox >** (p. 149), **LevelData< FluxBox >** (p. 298), and **LevelData< FArrayBox >** (p. 298).

6.67.3.5 template<class T> T & LayoutData< T >::operator[] (const DataIndex & *index*) [inline]

6.67.3.6 template<class T> const T & LayoutData< T >::operator[] (const DataIndex & *index*) const [inline]

6.67.4 Friends And Related Function Documentation

6.67.4.1 template<class T> friend class ArrayViewData [friend]

6.67.5 Member Data Documentation

6.67.5.1 template<class T> BoxLayout LayoutData< T >::m_boxLayout [protected]

6.67.5.2 template<class T> Vector<T*> LayoutData< T >::m_vector [protected]

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

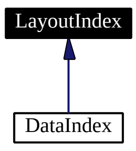
- **LayoutData.H**
- **LayoutDataI.H**

6.68 LayoutIndex Class Reference

An index for **LayoutIterator** (p. 291).

#include <DataIndex.H>

Inheritance diagram for LayoutIndex:



Public Methods

- **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** (p. 194) *has been null constructed.*

- int **intCode** () const

Friends

- class **DataIterator**
- class **LayoutIterator**
- class **BoxLayout**
- class **DataIndex**

6.68.1 Detailed Description

An index for **LayoutIterator** (p. 291).

6.68.2 Constructor & Destructor Documentation

6.68.2.1 LayoutIndex::LayoutIndex () [inline]

6.68.2.2 LayoutIndex::~LayoutIndex () [inline]

6.68.3 Member Function Documentation

6.68.3.1 bool LayoutIndex::eq (const LayoutIndex & *rhs*) const [inline]

6.68.3.2 int LayoutIndex::intCode () const [inline]

6.68.3.3 bool LayoutIndex::isNull () const [inline]

returns 'true' if this **DataIndex** (p. 194) has been null constructed.

6.68.3.4 bool LayoutIndex::operator!= (const LayoutIndex & *rhs*) const [inline]

6.68.3.5 bool LayoutIndex::operator== (const LayoutIndex & *rhs*) const [inline]

6.68.4 Friends And Related Function Documentation

6.68.4.1 friend class BoxLayout [friend]

Reimplemented in **DataIndex** (p. 194).

6.68.4.2 friend class DataIndex [friend]

6.68.4.3 friend class DataIterator [friend]

Reimplemented in **DataIndex** (p. 195).

6.68.4.4 friend class LayoutIterator [friend]

Reimplemented in **DataIndex** (p. 195).

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

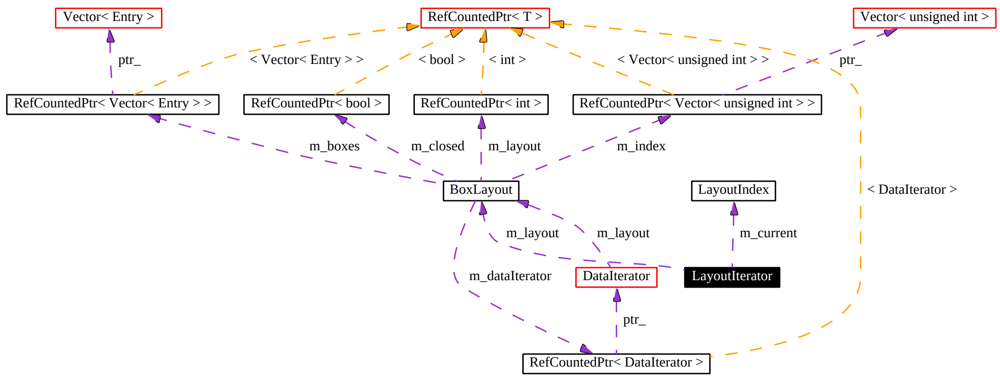
- **DataIndex.H**

6.69 LayoutIterator Class Reference

An Iterator based on a **BoxLayout** (p. 139) object.

`#include <LayoutIterator.H>`

Collaboration diagram for LayoutIterator:



Public Methods

- **~LayoutIterator** ()
- const **LayoutIndex** & **operator()** () const
- **LayoutIndex i** () const
- void **operator++** ()
- void **incr** ()
- bool **ok** () const
- void **reset** ()
- void **begin** ()
- void **end** ()

Friends

- class **BoxLayout**

6.69.1 Detailed Description

An Iterator based on a **BoxLayout** (p. 139) object.

An Iterator based on a **BoxLayout** (p. 139) object. It does not support a dereferencing operation(1), since it is intended to work with all of BoxLayouts, DisjointBoxLayouts, BoxLayoutDatas **LevelData** (p. 295)'s, and any object that is built on top of a **BoxLayout** (p. 139) object. LayoutIterator access the data in a **BoxLayout** (p. 139)-based object in a NON-data-parallel manner.

BoxLayout (p. 139)-based objects 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.

6.69.2 Constructor & Destructor Documentation

6.69.2.1 LayoutIterator::~LayoutIterator () [inline]

6.69.3 Member Function Documentation

6.69.3.1 void LayoutIterator::begin () [inline]

6.69.3.2 void LayoutIterator::end ()

skip the iterator to the end.

6.69.3.3 LayoutIndex LayoutIterator::i () const [inline]

6.69.3.4 void LayoutIterator::incr () [inline]

6.69.3.5 bool LayoutIterator::ok () const [inline]

6.69.3.6 const LayoutIndex & LayoutIterator::operator() () const [inline]

6.69.3.7 void LayoutIterator::operator++ () [inline]

6.69.3.8 void LayoutIterator::reset ()

6.69.4 Friends And Related Function Documentation

6.69.4.1 friend class BoxLayout [friend]

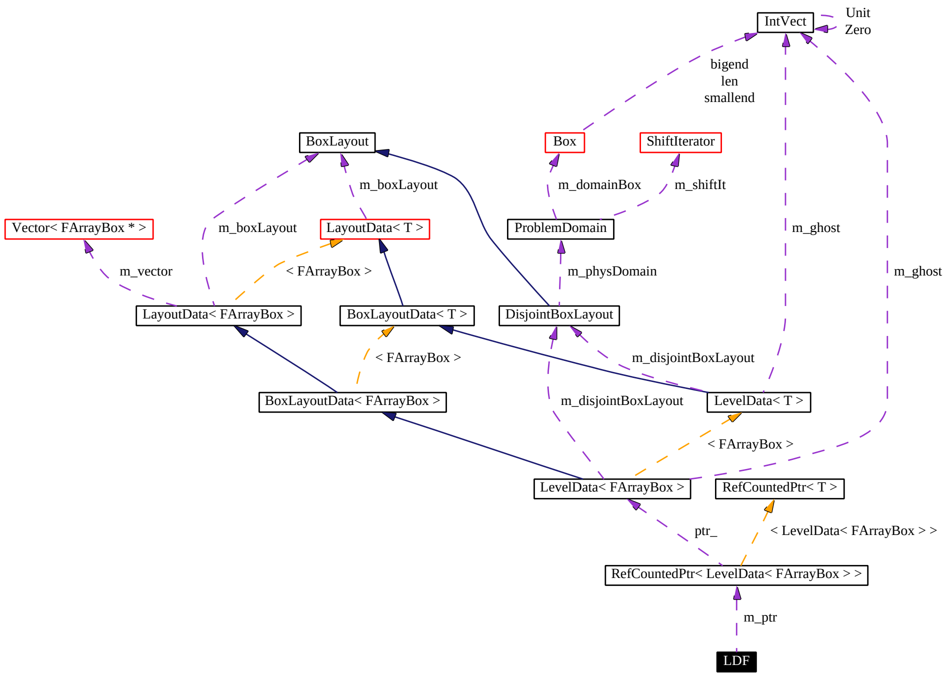
The documentation for this class was generated from the following file:

- **LayoutIterator.H**

6.70 LDF Class Reference

`#include <LDF.H>`

Collaboration diagram for LDF:



Public Methods

- **LDF** ()
- **LDF** (**LevelData**< **FArrayBox** > ∗newPtr)
- void **define** (const **DisjointBoxLayout** &dp, int comps, const **IntVect** &ghost)
- ∼**LDF** ()
- void **exchange** (const **Interval** &comps)
- void **copyTo** (const **Interval** &srcComps, LDF &dest, const **Interval** &destComps) const
- **IntVect ghostVect** () const
- const **DisjointBoxLayout** & **getBoxes** () const
- const **DisjointBoxLayout** & **disjoiuntBoxLayout** () const
- **Interval interval** () const
- **DataIterator dataIterator** () const
- **Box box** (const **DataIndex** &index) const
- **FArrayBox** & **get** (const **DataIndex** &index)
- const **LevelData**< **FArrayBox** > ∗ **dataPtr** () const
- void **operator=** (**LevelData**< **FArrayBox** > ∗newPtr)

6.70.1 Constructor & Destructor Documentation

6.70.1.1 LDF::LDF () [inline]

6.70.1.2 LDF::LDF (LevelData< FArrayBox > * *newPtr*) [inline]

6.70.1.3 LDF::~LDF () [inline]

6.70.2 Member Function Documentation

6.70.2.1 Box LDF::box (const DataIndex & *index*) const [inline]

6.70.2.2 void LDF::copyTo (const Interval & *srcComps*, LDF & *dest*, const Interval & *destComps*) const [inline]

6.70.2.3 DataIterator LDF::dataIterator () const [inline]

6.70.2.4 const LevelData<FArrayBox>* LDF::dataPtr () const [inline]

6.70.2.5 void LDF::define (const DisjointBoxLayout & *dp*, int *comps*, const IntVect & *ghost*) [inline]

6.70.2.6 const DisjointBoxLayout& LDF::disjoiuntBoxLayout () const [inline]

6.70.2.7 void LDF::exchange (const Interval & *comps*) [inline]

6.70.2.8 FArrayBox& LDF::get (const DataIndex & *index*) [inline]

6.70.2.9 const DisjointBoxLayout& LDF::getBoxes () const [inline]

6.70.2.10 IntVect LDF::ghostVect () const [inline]

6.70.2.11 Interval LDF::interval () const [inline]

6.70.2.12 void LDF::operator= (LevelData< FArrayBox > * *newPtr*) [inline]

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

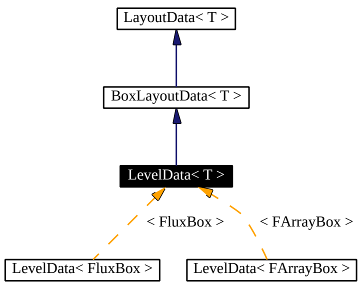
- **LDF.H**

6.71 LevelData< T > Class Template Reference

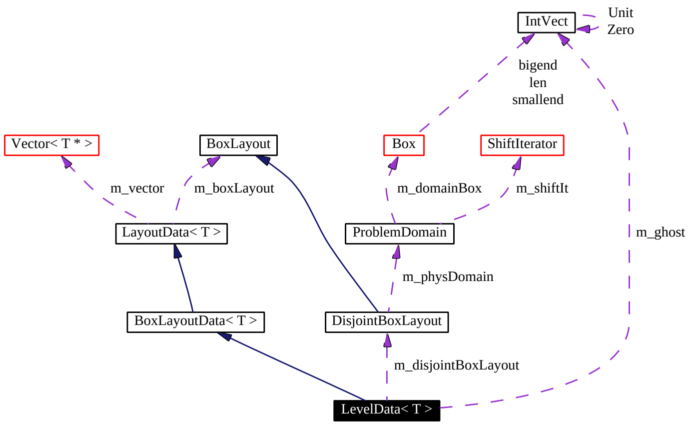
Data over a disjoint union of rectangles.

`#include <LevelData.H>`

Inheritance diagram for LevelData< T >:



Collaboration diagram for LevelData< T >:



Public Methods

- **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
- virtual void **exchange** (const **Interval** &comps)
- virtual void **exchange** (const **Interval** &comps, const **Copier** &copier)
- const **IntVect** & **ghostVect** () const
- 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.

- const **DisjointBoxLayout** & **getBoxes** () const
- const **DisjointBoxLayout** & **disjointBoxLayout** () const

Protected Methods

- void **makeItSo** (const **Interval** &a_srcComps, const LevelData< T > &a_src, **BoxLayoutData**< T > &a_dest, const **Interval** &a_destComps, const **Copier** &a_copier) 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
- void **writeSendDataFromMeIntoBuffers** (const LevelData< T > &a_src, const **Interval** &a_srcComps) const
- void **postSendsFromMe** () const
- void **postReceivesToMe** () const
- void **unpackReceivesToMe** (**BoxLayoutData**< T > &a_dest, const **Interval** &a_destComps) const

Protected Attributes

- **DisjointBoxLayout m_disjointBoxLayout**
- **IntVect m_ghost**
- void ∗ **m_sendbuffer**
- size_t **m_sendcapacity**
- void ∗ **m_recbuffer**
- size_t **m_reccapacity**

6.71.1 Detailed Description

template<class T> class LevelData< T >

Data over a disjoint union of rectangles.

LevelData is-a **BoxLayoutData** (p. 147). It is built from a **DisjointBoxLayout** (p. 210) instead though, thus making the data in a **DisjointBoxLayout** (p. 210) uniquely defined for all spatial indices defined by the underlying **DisjointBoxLayout** (p. 210). It carries within it an inherent concept of ghost cells (which the user can choose to be empty).

Since LevelData is-a **BoxLayoutData** (p. 147), all the methods required of the template class T for **BoxLayoutData** (p. 147) are also required for LevelData. LevelData does not have any extra requirements.

6.71.2 Constructor & Destructor Documentation

6.71.2.1 template<class T> LevelData< T >::LevelData () `[inline]`

6.71.2.2 template<class T> LevelData< T >::LevelData (const DisjointBoxLayout & *dp*, int *comps*, const IntVect & *ghost* = IntVect::TheZeroVector(), const DataFactory< T > & *a_factory* = DefaultDataFactory< T >()) `[inline]`

6.71.2.3 template<class T> LevelData< T >::~LevelData () `[inline, virtual]`

6.71.3 Member Function Documentation

6.71.3.1 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 `[inline, protected]`

6.71.3.2 template<class T> void LevelData< T >::completePendingSends () const `[inline, protected]`

6.71.3.3 template<class T> void LevelData< T >::copyTo (const Interval & *srcComps*, LevelData< T > & *dest*, const Interval & *destComps*, const Copier & *copier*) const `[inline, virtual]`

6.71.3.4 template<class T> void LevelData< T >::copyTo (const Interval & *srcComps*, LevelData< T > & *dest*, const Interval & *destComps*) const `[inline, virtual]`

same as copyTo that takes a **BoxLayoutData** (p. 147), except that it fills the ghost cells of 'dest' with data from 'this' also.

6.71.3.5 template<class T> void LevelData< T >::copyTo (const Interval & *srcComps*, BoxLayoutData< T > & *dest*, const Interval & *destComps*, const Copier & *copier*) const `[inline, virtual]`

6.71.3.6 template<class T> void LevelData< T >::copyTo (const Interval & *srcComps*, BoxLayoutData< T > & *dest*, const Interval & *destComps*) const `[inline, virtual]`

6.71.3.7 template<class T> void LevelData< T >::define (const BoxLayout & *deadFunction*) `[inline, virtual]`

overridden and throws an error.

Reimplemented from **BoxLayoutData< T >** (p. 149).

6.71.3.8 template<class T> void LevelData< T >::define (const BoxLayoutData< T > & *da*, const Interval & *comps*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) `[inline, virtual]`

Define this **BoxLayoutData** (p. 147) to be the subset of the data in da defined by the **Interval** (p. 263) comps.

Reimplemented from **BoxLayoutData< T >** (p. 149).

6.71.3.9 template<class T> void LevelData< T >::define (const BoxLayoutData< T > & *da*, const DataFactory< T > & *factory* = DefaultDataFactory< T >()) `[inline, virtual]`

Reimplemented from **BoxLayoutData< T >** (p. 149).

6.71.3.10 template<class T> void LevelData< T >::define (const BoxLayout & *dp*, int *comps*, const DataFactory< T > & *factory*) `[inline, virtual]`

Reimplemented from **BoxLayoutData< T >** (p. 149).

6.71.3.11 template<class T> void LevelData< T >::define (const LevelData< T > & *da*, const Interval & *comps*, const DataFactory< T > & *a_factory* = DefaultDataFactory< T >()) `[inline, 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()]

6.71.3.12 template<class T> void LevelData< T >::define (const LevelData< T > & *da*, const DataFactory< T > & *a_factory* = DefaultDataFactory< T >()) `[inline, virtual]`

copy definer. this LevelData thrown away and da's LevelData copied

6.71.3.13 template<class T> void LevelData< T >::define (const DisjointBoxLayout & *dp*, int *comps*, const IntVect & *ghost* = IntVect::TheZeroVector(), const DataFactory< T > & *a_factory* = DefaultDataFactory< T >()) [inline, virtual]

6.71.3.14 template<class T> const DisjointBoxLayout& LevelData< T >::disjointBoxLayout () const [inline]

6.71.3.15 template<class T> void LevelData< T >::exchange (const Interval & *comps*, const Copier & *copier*) [inline, virtual]

6.71.3.16 template<class T> void LevelData< T >::exchange (const Interval & *comps*) [inline, virtual]

6.71.3.17 template<class T> const DisjointBoxLayout& LevelData< T >::getBoxes () const [inline]

6.71.3.18 template<class T> const IntVect& LevelData< T >::ghostVect () const [inline]

6.71.3.19 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 [inline, protected]

6.71.3.20 template<class T> void LevelData< T >::postReceivesToMe () const [inline, protected]

6.71.3.21 template<class T> void LevelData< T >::postSendsFromMe () const [inline, protected]

6.71.3.22 template<class T> void LevelData< T >::unpackReceivesToMe (BoxLayoutData< T > & *a_dest*, const Interval & *a_destComps*) const [inline, protected]

6.71.3.23 template<class T> void LevelData< T >::writeSendDataFromMeIntoBuffers (const LevelData< T > & *a_src*, const Interval & *a_srcComps*) const [inline, protected]

6.71.4 Member Data Documentation

6.71.4.1 template<class T> DisjointBoxLayout LevelData< T >::m_disjointBoxLayout [protected]

6.71.4.2 template<class T> IntVect LevelData< T >::m_ghost [protected]

6.71.4.3 template<class T> void* LevelData< T >::m_recbuffer [protected]

6.71.4.4 template<class T> size_t LevelData< T >::m_reccapacity [protected]

6.71.4.5 template<class T> void* LevelData< T >::m_sendbuffer [protected]

6.71.4.6 template<class T> size_t LevelData< T >::m_sendcapacity [protected]

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

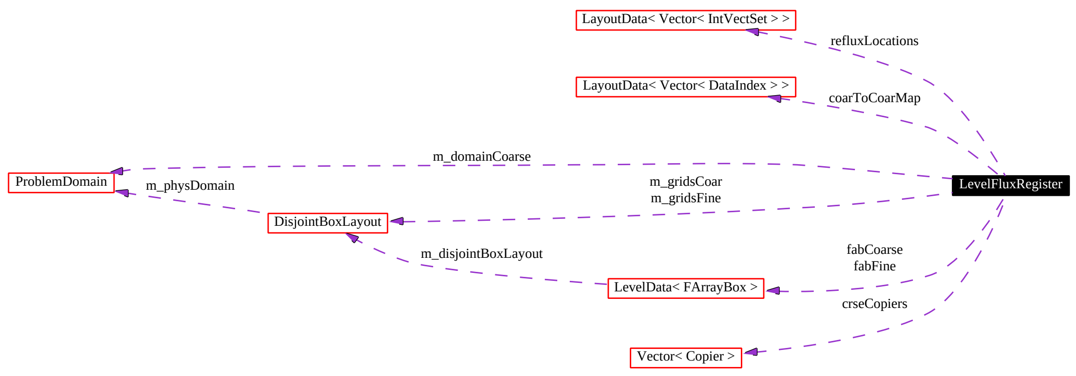
- **LevelData.H**
- **LevelDataI.H**

6.72 LevelFluxRegister Class Reference

LevelFluxRegister-A class to encapsulate a levels worth of flux registers.

`#include <LevelFluxRegister.H>`

Collaboration diagram for LevelFluxRegister:



Public Methods

- **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 **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
- void **dump** ()
- void **dumpLoCoar** (int idir)
- void **dumpHiCoar** (int idir)
- void **dumpLoFine** (int idir)
- void **dumpHiFine** (int idir)

Protected Methods

- void **setDefaultValues** ()

Static Protected Methods

- int **index** (int dir, **Side::LoHiSide** side)

Protected Attributes

- **LevelData**< **FArrayBox** > **fabCoarse** [**SpaceDim** *2]
- **LevelData**< **FArrayBox** > **fabFine** [**SpaceDim** *2]
- **Vector**< **Copier** > **crseCopiers**
- **LayoutData**< **Vector**< **IntVectSet** > > **refluxLocations** [**SpaceDim** *2]
- **LayoutData**< **Vector**< **DataIndex** > > **coarToCoarMap** [**SpaceDim** *2]
- bool **m_isDefined**
- int **m_nComp**

  number of components in register

- int **m_nRefine**

  refinement ratio between levels

- **ProblemDomain m_domainCoarse**

  domain at the coarse grid resolution

- **DisjointBoxLayout m_gridsFine**
- **DisjointBoxLayout m_gridsCoar**

6.72.1 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.

6.72.2 Constructor & Destructor Documentation

6.72.2.1 LevelFluxRegister::LevelFluxRegister ()

Default constructor. Creates an uninitialized levelfluxregister.

6.72.2.2 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

6.72.2.3 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

6.72.2.4 LevelFluxRegister::~LevelFluxRegister ()

6.72.3 Member Function Documentation

6.72.3.1 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**() (p. 304).

6.72.3.2 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**() (p. 304).

6.72.3.3 void LevelFluxRegister::dump ()

6.72.3.4 void LevelFluxRegister::dumpHiCoar (int *idir*)

6.72.3.5 void LevelFluxRegister::dumpHiFine (int *idir*)

6.72.3.6 void LevelFluxRegister::dumpLoCoar (int *idir*)

6.72.3.7 void LevelFluxRegister::dumpLoFine (int *idir*)

6.72.3.8 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.

6.72.3.9 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.

6.72.3.10 int LevelFluxRegister::index (int *dir*, Side::LoHiSide *side*) [static, protected]

6.72.3.11 bool LevelFluxRegister::isDefined () const

has full define function been called? return true if so.

6.72.3.12 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.

6.72.3.13 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.

6.72.3.14 void LevelFluxRegister::setDefaultValues () [protected]

6.72.3.15 void LevelFluxRegister::setToZero ()

Initialize values of registers to zero.

6.72.3.16 void LevelFluxRegister::undefine ()

Modifies this LevelFluxRegister so that it is returned to the uninitialized state. User must now call the full **define**() (p. 303) before using it.

6.72.4 Member Data Documentation

6.72.4.1 LayoutData<Vector<DataIndex> > LevelFluxRegister::coarToCoarMap[SpaceDim*2] [protected]

6.72.4.2 Vector<Copier> LevelFluxRegister::crseCopiers [protected]

6.72.4.3 LevelData<FArrayBox> LevelFluxRegister::fabCoarse[SpaceDim*2] [protected]

6.72.4.4 LevelData<FArrayBox> LevelFluxRegister::fabFine[SpaceDim*2] [protected]

6.72.4.5 ProblemDomain LevelFluxRegister::m_domainCoarse [protected]

domain at the coarse grid resolution

6.72.4.6 DisjointBoxLayout LevelFluxRegister::m_gridsCoar [protected]

6.72.4.7 DisjointBoxLayout LevelFluxRegister::m_gridsFine [protected]

6.72.4.8 bool LevelFluxRegister::m_isDefined [protected]

6.72.4.9 int LevelFluxRegister::m_nComp [protected]

number of components in register

6.72.4.10 int LevelFluxRegister::m_nRefine [protected]

refinement ratio between levels

6.72.4.11 LayoutData<Vector<IntVectSet> > LevelFluxRegister::refluxLocations[SpaceDim*2] [protected]

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

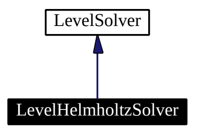
- **LevelFluxRegister.H**

6.73 LevelHelmholtzSolver Class Reference

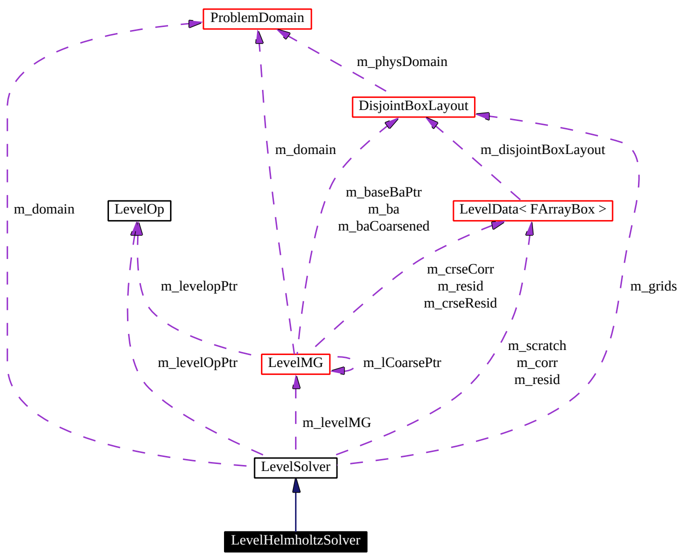
special case **LevelSolver** (p. 319) to solve Helmholtz eqn

`#include <LevelHelmholtzSolver.H>`

Inheritance diagram for LevelHelmholtzSolver:



Collaboration diagram for LevelHelmholtzSolver:



Public Methods

- **LevelHelmholtzSolver** ()

  Default constructor.

- virtual ∼**LevelHelmholtzSolver** ()
- **LevelHelmholtzSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse)

  constructor – note that there is no need to pass in a **LevelOp** (p. 315)*,*

- **LevelHelmholtzSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse)

  (deprecated) constructor – note there is no need to pass in a **LevelOp** (p. 315)*,*

- **LevelHelmholtzSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** ∗a_baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **HelmholtzOp** ∗const a_opin)

- **LevelHelmholtzSolver** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse, const **HelmholtzOp** *const a_opin)

  deprecated interface using **Box** (p. 106) *instead of* **ProblemDomain** (p. 418)

- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **ProblemDomain** &a_domain, **Real** a_dxLevel, int a_nRefCrse)
- void **define** (const **DisjointBoxLayout** &a_grids, const **DisjointBoxLayout** *a_baseGrids, const **Box** &a_domain, **Real** a_dxLevel, int a_nRefCrse)

  deprecated interface using **Box** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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)
- 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)

  (deprecated) interface using **Box** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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)

  this function hides the **LevelSolver** (p. 319) *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)

  this function hides the **LevelSolver** (p. 319) *version – should never be called*

- void **setBeta** (**Real** a_beta)
- void **setBC** (const **DomainGhostBC** &a_BC)

6.73.1 Detailed Description

special case **LevelSolver** (p. 319) to solve Helmholtz eqn

LevelHelmholtzSolver is a special case of **LevelSolver** (p. 319) for solving the Helmholtz equation on a level. It allows the coefficients to be changed without allocating an entire new **LevelSolver** (p. 319).

6.73.2 Constructor & Destructor Documentation

6.73.2.1 LevelHelmholtzSolver::LevelHelmholtzSolver ()

Default constructor.

6.73.2.2 virtual LevelHelmholtzSolver::~LevelHelmholtzSolver () [virtual]

6.73.2.3 LevelHelmholtzSolver::LevelHelmholtzSolver (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*)

constructor – note that there is no need to pass in a **LevelOp** (p. 315),

6.73.2.4 LevelHelmholtzSolver::LevelHelmholtzSolver (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*)

(deprecated) constructor – note there is no need to pass in a **LevelOp** (p. 315),

6.73.2.5 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*)

6.73.2.6 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*)

deprecated interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

6.73.3 Member Function Documentation

6.73.3.1 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*) [virtual]

this function hides the **LevelSolver** (p. 319) version – should never be called

6.73.3.2 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*) [virtual]

this function hides the **LevelSolver** (p. 319) version – should never be called

6.73.3.3 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*) [virtual]

(deprecated) interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

6.73.3.4 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*) `[virtual]`

6.73.3.5 void LevelHelmholtzSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const Box & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*)

deprecated interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

6.73.3.6 void LevelHelmholtzSolver::define (const DisjointBoxLayout & *a_grids*, const DisjointBoxLayout ∗ *a_baseGrids*, const ProblemDomain & *a_domain*, Real *a_dxLevel*, int *a_nRefCrse*)

6.73.3.7 void LevelHelmholtzSolver::setBC (const DomainGhostBC & *a_BC*)

6.73.3.8 void LevelHelmholtzSolver::setBeta (Real *a_beta*)

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

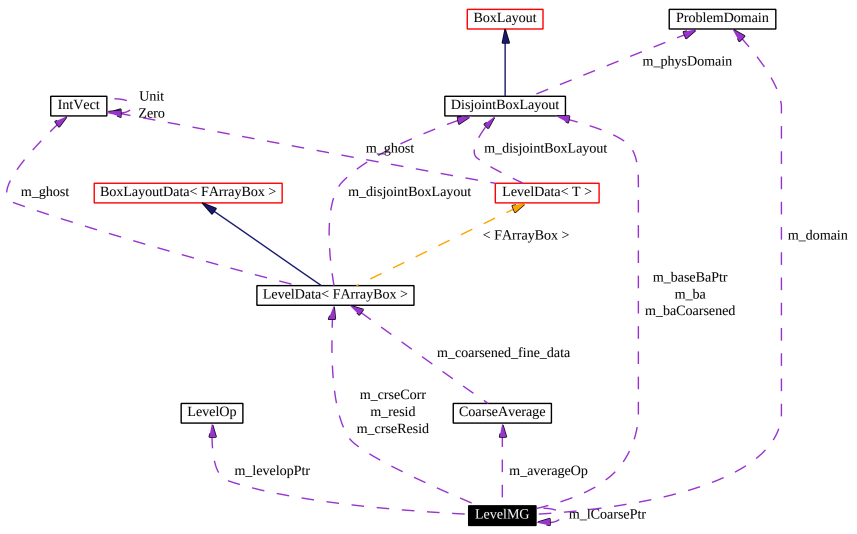
- **LevelHelmholtzSolver.H**

6.74 LevelMG Class Reference

Multigrid solver on a level.

`#include <LevelMG.H>`

Collaboration diagram for LevelMG:



Public Methods

- bool **isDefined** () const
- **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)
- **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)
- 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)
- 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)
- void **define** (const LevelMG &L, int refCoarse, const **LevelOp** ∗const opin_a)

  Constructor for coarsened version of object.

- **LevelMG** (const LevelMG &L, int refCoarse, const **LevelOp** ∗opin_a)

  Constructor for coarsened version of object.

- **∼LevelMG** ()
- void **clear** ()
- void **mgRelax** (**LevelData**< **FArrayBox** > &a_soln, const **LevelData**< **FArrayBox** > &a_rhs, bool bottomsolveflag)

- void **setnumBottomGSRB** (int a_numBottomGSRB)
- void **setnumSmoothUp** (int a_numSmoothUp)
- void **setnumSmoothDown** (int a_numSmoothDown)
- **LevelOp** * **levelOpPtr** ()
- LevelMG * **lCoarsePtr** ()

Protected Methods

- 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**

6.74.1 Detailed Description

Multigrid solver on a level.

Multigrid solver on a level. This class is to be considered internal to **AMRSolver** (p. 69) and not a part of the Chombo API.

6.74.2 Constructor & Destructor Documentation

6.74.2.1 LevelMG::LevelMG ()

6.74.2.2 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)

6.74.2.3 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)

6.74.2.4 LevelMG::LevelMG (const LevelMG & *L*, int *refCoarse*, const LevelOp * *opin_a*)

Constructor for coarsened version of object.

6.74.2.5 LevelMG::~LevelMG ()

6.74.3 Member Function Documentation

6.74.3.1 void LevelMG::clear ()

6.74.3.2 void LevelMG::clearMemory () `[protected]`

6.74.3.3 void LevelMG::define (const LevelMG & *L*, int *refCoarse*, const LevelOp *const *opin_a*)

Constructor for coarsened version of object.

6.74.3.4 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*)

6.74.3.5 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)

6.74.3.6 bool LevelMG::isDefined () const

6.74.3.7 LevelMG* LevelMG::lCoarsePtr ()

6.74.3.8 LevelOp* LevelMG::levelOpPtr ()

6.74.3.9 void LevelMG::mgRelax (LevelData< FArrayBox > & *a_soln*, const LevelData< FArrayBox > & *a_rhs*, bool *bottomsolveflag*)

: Invoke 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.

6.74.3.10 void LevelMG::setDefaultValues () [protected]

6.74.3.11 void LevelMG::setnumBottomGSRB (int *a_numBottomGSRB*) [inline]

6.74.3.12 void LevelMG::setnumSmoothDown (int *a_numSmoothDown*) [inline]

6.74.3.13 void LevelMG::setnumSmoothUp (int *a_numSmoothUp*) [inline]

6.74.4 Member Data Documentation

6.74.4.1 CoarseAverage LevelMG::m_averageOp [protected]

6.74.4.2 DisjointBoxLayout LevelMG::m_ba [protected]

6.74.4.3 DisjointBoxLayout LevelMG::m_baCoarsened [protected]

6.74.4.4 const DisjointBoxLayout* LevelMG::m_baseBaPtr [protected]

6.74.4.5 LevelData<FArrayBox> LevelMG::m_crseCorr [protected]

6.74.4.6 LevelData<FArrayBox> LevelMG::m_crseResid [protected]

6.74.4.7 ProblemDomain LevelMG::m_domain [protected]

6.74.4.8 Real LevelMG::m_dxLevel [protected]

6.74.4.9 bool LevelMG::m_isDefined [protected]

6.74.4.10 LevelMG* LevelMG::m_lCoarsePtr [protected]

6.74.4.11 LevelOp* LevelMG::m_levelopPtr [protected]

6.74.4.12 int LevelMG::m_nCoarserLevels [protected]

6.74.4.13 int LevelMG::m_numBottomGSRB [protected]

6.74.4.14 int LevelMG::m_numSmoothDown [protected]

6.74.4.15 int LevelMG::m_numSmoothUp [protected]

6.74.4.16 int LevelMG::m_refToCoar [protected]

6.74.4.17 LevelData<FArrayBox> LevelMG::m_resid [protected]

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

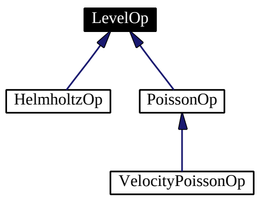
- **LevelMG.H**

6.75 LevelOp Class Reference

Pure base class to encapsulate level operations API for elliptic solvers.

`#include <LevelOp.H>`

Inheritance diagram for LevelOp:



Public Methods

- virtual void **CFInterp** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_phiCoarse)=0

    constructors, destructors, defines

- virtual void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &a_phi)=0
- **LevelOp** ()
- virtual LevelOp ∗ **new_levelop** () const=0
- virtual bool **isDefined** () const=0
- virtual void **define** (const **DisjointBoxLayout** &Ba, const **DisjointBoxLayout** ∗base_ba, **Real** DxLevel, int refratio, const **Box** &domf, bool homogeneousOnly, int ncomp=1)=0
- virtual void **define** (const **DisjointBoxLayout** &Ba, const **DisjointBoxLayout** ∗base_ba, **Real** DxLevel, int refratio, const **ProblemDomain** &domf, bool homogeneousOnly, int ncomp=1)=0
- virtual void **define** (const LevelOp ∗opfine, int reftofine)=0
- virtual ∼**LevelOp** ()
- virtual void **smooth** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > &rhs)=0
- virtual void **levelPreconditioner** (**LevelData**< **FArrayBox** > &a_phihat, const **LevelData**< **FArrayBox** > &a_rhshat)=0
- virtual void **applyOpI** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > ∗phiCoarse, **LevelData**< **FArrayBox** > &LOfPhi)=0
- virtual void **applyOpIcfHphys** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > ∗phiCoarse, **LevelData**< **FArrayBox** > &LOfPhi)=0
- virtual void **applyOpHcfIphys** (**LevelData**< **FArrayBox** > &phi, **LevelData**< **FArrayBox** > &LOfPhi)=0
- virtual void **applyOpH** (**LevelData**< **FArrayBox** > &phi, **LevelData**< **FArrayBox** > &LOfPhi)=0
- virtual void **bottomSmoother** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > &rhs)=0
- virtual void **getFlux** (**FArrayBox** &flux, const **FArrayBox** &data, const **DataIndex** &a_datInd, int dir)=0

6.75.1 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** (p. 69) inherit the LevelOp interface.

6.75.2 Constructor & Destructor Documentation

6.75.2.1 LevelOp::LevelOp () `[inline]`

6.75.2.2 virtual LevelOp::~LevelOp () `[inline, virtual]`

6.75.3 Member Function Documentation

6.75.3.1 virtual void LevelOp::applyOpH (LevelData< FArrayBox > & *phi*, LevelData< FArrayBox > & *LOfPhi*) `[pure virtual]`

Evaluate Operator, homogeneous C/F boundary conditions. homogeneous phys boundary conditions.

Implemented in **HelmholtzOp** (p. 249), and **PoissonOp** (p. 385).

6.75.3.2 virtual void LevelOp::applyOpHcfIphys (LevelData< FArrayBox > & *phi*, LevelData< FArrayBox > & *LOfPhi*) `[pure virtual]`

Evaluate Operator, homogeneous C/F boundary conditions. inhomogeneous phys boundary conditions.

Implemented in **HelmholtzOp** (p. 249), and **PoissonOp** (p. 385).

6.75.3.3 virtual void LevelOp::applyOpI (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > * *phiCoarse*, LevelData< FArrayBox > & *LOfPhi*) `[pure virtual]`

Evaluate Operator, inhomogeneous C/F boundary conditions. inhomogeneous phys boundary conditions.

Implemented in **HelmholtzOp** (p. 249), and **PoissonOp** (p. 385).

6.75.3.4 virtual void LevelOp::applyOpIcfHphys (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > * *phiCoarse*, LevelData< FArrayBox > & *LOfPhi*) `[pure virtual]`

Evaluate Operator, inhomogeneous C/F boundary conditions. homogeneous phys boundary conditions.

Implemented in **HelmholtzOp** (p. 249), and **PoissonOp** (p. 385).

6.75.3.5 virtual void LevelOp::bottomSmoother (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > & *rhs*) `[pure virtual]`

smoother at bottom level

Implemented in **HelmholtzOp** (p. 249), and **PoissonOp** (p. 385).

6.75.3.6 virtual void LevelOp::CFInterp (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_phiCoarse*) [pure virtual]

constructors, destructors, defines

Coarse / Fine interpolation operator. Fill ghost cells of phi

Implemented in **HelmholtzOp** (p. 249), and **PoissonOp** (p. 385).

6.75.3.7 virtual void LevelOp::define (const LevelOp * *opfine*, int *reftofine*) [pure virtual]

Full define function. define from finer levelop. reftofine is the refinement ratio to get from *this opfine

Implemented in **HelmholtzOp** (p. 250), and **PoissonOp** (p. 386).

6.75.3.8 virtual void LevelOp::define (const DisjointBoxLayout & *Ba*, const DisjointBoxLayout * *base_ba*, Real *DxLevel*, int *refratio*, const ProblemDomain & *domf*, bool *homogeneousOnly*, int *ncomp* = 1) [pure virtual]

Full define function. makes all coarse-fine information and sets internal variables

Implemented in **HelmholtzOp** (p. 250), and **PoissonOp** (p. 386).

6.75.3.9 virtual void LevelOp::define (const DisjointBoxLayout & *Ba*, const DisjointBoxLayout * *base_ba*, Real *DxLevel*, int *refratio*, const Box & *domf*, bool *homogeneousOnly*, int *ncomp* = 1) [pure virtual]

Full define function. makes all coarse-fine information and sets internal variables

Implemented in **HelmholtzOp** (p. 250), and **PoissonOp** (p. 386).

6.75.3.10 virtual void LevelOp::getFlux (FArrayBox & *flux*, const FArrayBox & *data*, const DataIndex & *a_datInd*, int *dir*) [pure virtual]

get flux(== 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.

Implemented in **HelmholtzOp** (p. 250), and **PoissonOp** (p. 386).

6.75.3.11 virtual void LevelOp::homogeneousCFInterp (LevelData< FArrayBox > & *a_phi*) [pure virtual]

homogeneous coarse/fine interpolation operator

Implemented in **HelmholtzOp** (p. 250), and **PoissonOp** (p. 386).

6.75.3.12 virtual bool LevelOp::isDefined () const [pure virtual]

Implemented in **VelocityPoissonOp** (p. 476), **HelmholtzOp** (p. 250), and **PoissonOp** (p. 386).

6.75.3.13 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. 250), and **PoissonOp** (p. 387).

6.75.3.14 virtual LevelOp* LevelOp::new_levelop () const [pure virtual]

this gets around the "no virtual constructor" rule

Implemented in **VelocityPoissonOp** (p. 477), **HelmholtzOp** (p. 251), and **PoissonOp** (p. 387).

6.75.3.15 virtual void LevelOp::smooth (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > & *rhs*) [pure virtual]

Smoother. Assumes that problem has already been put in residual correction form, so that C/F boundary conditions are homogeneous.

Implemented in **HelmholtzOp** (p. 251), and **PoissonOp** (p. 387).

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

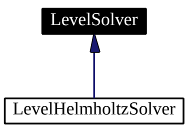
- **LevelOp.H**

6.76 LevelSolver Class Reference

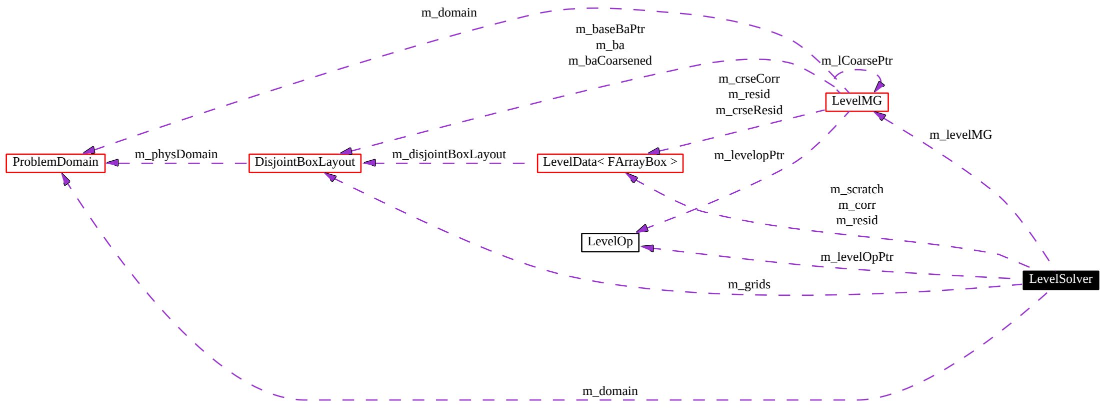
Solves elliptic equations on a level.

`#include <LevelSolver.H>`

Inheritance diagram for LevelSolver:



Collaboration diagram for LevelSolver:



Public Methods

- **LevelSolver** ()
- virtual ~**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)
- **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)
- 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)

  Full define function.

- 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)

Full define function.

- bool **isDefined** () const
- void **clearMemory** ()
- void **setDefaultValues** ()
- void **levelSolveH** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > &rhs, bool initializePhiToZero=true)
- void **levelSolve** (**LevelData**< **FArrayBox** > &phi, const **LevelData**< **FArrayBox** > *phic, const **LevelData**< **FArrayBox** > &rhs, bool initializePhiToZero=true)
- void **setTolerance** (**Real** a_tolerance)

  set solution tolerance

- void **setVerbose** (bool a_verbose)
- void **setMaxIter** (int a_maxIter)

  set max number multigrid vcycles to use in solution

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**
- **Real m_tolerance**
- bool **m_verbose**

6.76.1 Detailed Description

Solves elliptic equations on a level.

LevelSolver Solves elliptic equations on a level using multigrid.

6.76.2 Constructor & Destructor Documentation

6.76.2.1 LevelSolver::LevelSolver ()

Default constructor leaves LevelSolver undefined.

6.76.2.2 virtual LevelSolver::~LevelSolver () [virtual]

6.76.2.3 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)

Full constructor. Calls full define function.\ {\bf 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

6.76.2.4 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*)

Full constructor. Calls full define function.\ {\bf 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

6.76.3 Member Function Documentation

6.76.3.1 void LevelSolver::clearMemory ()

6.76.3.2 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) [virtual]

Full define function.

Full define function.\ {\bf 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

6.76.3.3 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) [virtual]

Full define function.

Full define function.\ {\bf 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

6.76.3.4 bool LevelSolver::isDefined () const

6.76.3.5 void LevelSolver::levelSolve (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > ∗ *phic*, const LevelData< FArrayBox > & *rhs*, bool *initializePhiToZero* = true)

do a level solve on this level using multigrid uses inhomogeneous bcs at interfaces\ {\bf Inputs:} \ phi: initial guess at solution. phic: solution at next coarser level. should be == NULL if there is no coarser level. rhs: right-hand side. \ {\bf Outputs:} \ phi: solution.

6.76.3.6 void LevelSolver::levelSolveH (LevelData< FArrayBox > & *phi*, const LevelData< FArrayBox > & *rhs*, bool *initializePhiToZero* = true)

do a level solve on this level using multigrid uses homogeneous bcs at coarse/fine interfaces \ {\bf Inputs:} \ phi: initial guess at solution. rhs: right-hand side. \ {\bf Outputs:} \ phi: solution.

6.76.3.7 void LevelSolver::setDefaultValues ()

6.76.3.8 void LevelSolver::setMaxIter (int *a_maxIter*)

set max number multigrid vcycles to use in solution

6.76.3.9 void LevelSolver::setTolerance (Real *a_tolerance*)

set solution tolerance

6.76.3.10 void LevelSolver::setVerbose (bool *a_verbose*)

6.76.4 Member Data Documentation

6.76.4.1 LevelData<FArrayBox> LevelSolver::m_corr [protected]

6.76.4.2 ProblemDomain LevelSolver::m_domain [protected]

6.76.4.3 Real LevelSolver::m_dxLevel [protected]

6.76.4.4 DisjointBoxLayout LevelSolver::m_grids [protected]

6.76.4.5 bool LevelSolver::m_isDefined [protected]

6.76.4.6 LevelMG LevelSolver::m_levelMG [protected]

6.76.4.7 LevelOp* LevelSolver::m_levelOpPtr [protected]

6.76.4.8 int LevelSolver::m_maxIter [protected]

6.76.4.9 int LevelSolver::m_nRefCrse [protected]

6.76.4.10 LevelData<FArrayBox> LevelSolver::m_resid [protected]

6.76.4.11 LevelData<FArrayBox> LevelSolver::m_scratch [protected]

6.76.4.12 Real LevelSolver::m_tolerance [protected]

6.76.4.13 bool LevelSolver::m_verbose [protected]

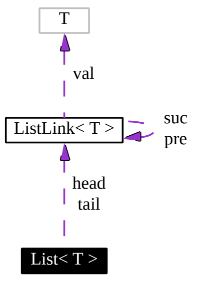
The documentation for this class was generated from the following file:

- **LevelSolver.H**

6.77 List< T > Class Template Reference

`#include <List.H>`

Collaboration diagram for List< T >:



Public Methods

- **List** ()
- **List** (const List< T > &rhs)
- List< T > & **operator=** (const List< T > &rhs)
- ~**List** ()
- void **prepend** (const T &value)
- void **append** (const T &value)
- void **add** (const T &value)
- void **join** (const List< T > &src)
- void **catenate** (List< T > &src)
- void **clear** ()
- List< T > * **copy** () const
- T & **firstElement** () const
- T & **lastElement** () const
- bool **includes** (const T &value) const
- bool **operator==** (const List< T > &rhs) const
- bool **operator!=** (const List< T > &rhs) const
- bool **isEmpty** () const
- bool **isNotEmpty** () const
- int **length** () const
- void **removeFirst** ()
- void **removeLast** ()
- const T & **operator**[ ] (const **ListIterator**< T > &li) const
- T & **operator**[ ] (const **ListIterator**< T > &li)
- void **remove** (const T &value)
- void **remove** (const List< T > &**lst**)
- void **remove** (**ListIterator**< T > &lit)
- void **replace** (**ListIterator**< T > &li, const T &val)
- void **addAfter** (**ListIterator**< T > &lit, const T &val)
- void **addBefore** (**ListIterator**< T > &lit, const T &val)
- **ListIterator**< T > **first** () const
- **ListIterator**< T > **last** () const

Protected Methods

- 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** >

template<class T> class List< T >

6.77.1 Constructor & Destructor Documentation

6.77.1.1 template<class T> List< T >::List () `[inline]`

6.77.1.2 template<class T> List< T >::List (const List< T > & *rhs*)

6.77.1.3 template<class T> List< T >::~List () `[inline]`

6.77.2 Member Function Documentation

6.77.2.1 template<class T> void List< T >::add (const T & *value*)

6.77.2.2 template<class T> ListLink< T > ∗ List< T >::addAfter (ListLink< T > ∗ *ln*, const T & *val*) `[protected]`

6.77.2.3 template<class T> void List< T >::addAfter (ListIterator< T > & *lit*, const T & *val*) `[inline]`

6.77.2.4 template<class T> ListLink< T > ∗ List< T >::addBefore (ListLink< T > ∗ *ln*, const T & *val*) `[protected]`

6.77.2.5 template<class T> void List< T >::addBefore (ListIterator< T > & *lit*, const T & *val*) `[inline]`

6.77.2.6 template<class T> void List< T >::append (const T & *value*) `[inline]`

6.77.2.7 template<class T> void List< T >::catenate (List< T > & *src*)

6.77.2.8 template<class T> void List< T >::clear ()

6.77.2.9 template<class T> List< T > ∗ List< T >::copy () const `[inline]`

6.77.2.10 template<class T> ListIterator< T > List< T >::first () const `[inline]`

6.77.2.11 template<class T> T & List< T >::firstElement () const `[inline]`

6.77.2.12 template<class T> bool List< T >::includes (const T & *value*) const

6.77.2.13 template<class T> bool List< T >::isEmpty () const `[inline]`

6.77.2.14 template<class T> bool List< T >::isNotEmpty () const `[inline]`

6.77.2.15 template<class T> void List< T >::join (const List< T > & *src*)

6.77.2.16 template<class T> ListIterator< T > List< T >::last () const `[inline]`

6.77.2.17 template<class T> T & List< T >::lastElement () const `[inline]`

6.77.2.18 template<class T> int List< T >::length () const

6.77.2.19 template<class T> bool List< T >::operator!= (const List< T > & *rhs*) const

6.77.2.20 template<class T> List< T > & List< T >::operator= (const List< T > & *rhs*)

6.77.2.21 template<class T> bool List< T >::operator== (const List< T > & *rhs*) const

6.77.2.22 template<class T> T & List< T >::operator[] (const ListIterator< T > & *li*) `[inline]`

6.77.2.23 template<class T> const T & List< T >::operator[] (const ListIterator< T > & *li*) const `[inline]`

6.77.2.24 template<class T> void List< T >::prepend (const T & *value*) `[inline]`

6.77.2.25 template<class T> void List< T >::remove (ListLink< T > ∗ *ln*) `[protected]`

6.77.2.26 template<class T> void List< T >::remove (ListIterator< T > & *lit*)

6.77.2.27 template<class T> void List< T >::remove (const List< T > & *lst*)

6.77.2.28 template<class T> void List< T >::remove (const T & *value*)

6.77.2.29 template<class T> void List< T >::removeFirst () `[inline]`

6.77.2.30 template<class T> void List< T >::removeLast () `[inline]`

6.77.2.31 template<class T> void List< T >::replace (ListIterator< T > & *li*, const T & *val*) `[inline]`

6.77.3 Friends And Related Function Documentation

6.77.3.1 template<class T> friend class ListIterator< T > `[friend]`

6.77.4 Member Data Documentation

6.77.4.1 template<class T> ListLink<T>∗ List< T >::head `[protected]`

6.77.4.2 template<class T> ListLink<T>∗ List< T >::tail `[protected]`

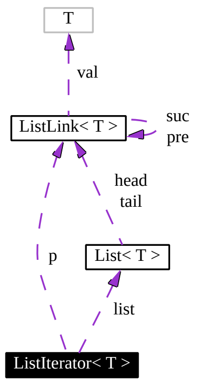
The documentation for this class was generated from the following files:

- **List.H**
- **ListImplem.H**

6.78 ListIterator< T > Class Template Reference

#include <List.H>

Collaboration diagram for ListIterator< T >:



Public Methods

- **ListIterator** (const **List**< T > &aList)
- **ListIterator** (const ListIterator< T > &rhs)
- void **rewind** ()
- const T & **operator()** () const
- const T & **operator** ∗ () const
- **operator bool** () const
- bool **operator!** () const
- const T & **value** () const
- ListIterator< T > & **operator++** ()
- ListIterator< T > & **operator–** ()
- ListIterator< T > **operator–** (int)
- ListIterator< T > **operator++** (int)
- bool **operator==** (const ListIterator< T > &) const
- bool **operator!=** (const ListIterator< T > &) const

Protected Methods

- **ListIterator** (const **List**< T > &_list, **ListLink**< T > ∗_p)

Protected Attributes

- const **List**< T > & **list**
- **ListLink**< T > ∗ **p**

Friends

- class **List**< **T** >

template<class T> class ListIterator< T >

6.78.1 Constructor & Destructor Documentation

6.78.1.1 template<class T> ListIterator< T >::ListIterator (const List< T > & *aList*) `[inline]`

6.78.1.2 template<class T> ListIterator< T >::ListIterator (const ListIterator< T > & *rhs*) `[inline]`

6.78.1.3 template<class T> ListIterator< T >::ListIterator (const List< T > & *_list*, ListLink< T > * *_p*) `[inline, protected]`

6.78.2 Member Function Documentation

6.78.2.1 template<class T> const T & ListIterator< T >::operator * () const `[inline]`

6.78.2.2 template<class T> ListIterator< T >::operator bool () const `[inline]`

6.78.2.3 template<class T> bool ListIterator< T >::operator! () const `[inline]`

6.78.2.4 template<class T> bool ListIterator< T >::operator!= (const ListIterator< T > &) const `[inline]`

6.78.2.5 template<class T> const T & ListIterator< T >::operator() () const `[inline]`

6.78.2.6 template<class T> ListIterator< T > ListIterator< T >::operator++ (int) `[inline]`

6.78.2.7 template<class T> ListIterator< T > & ListIterator< T >::operator++ () `[inline]`

6.78.2.8 template<class T> ListIterator< T > ListIterator< T >::operator-- (int) `[inline]`

6.78.2.9 template<class T> ListIterator< T > & ListIterator< T >::operator-- () `[inline]`

6.78.2.10 template<class T> bool ListIterator< T >::operator== (const ListIterator< T > &) const `[inline]`

6.78.2.11 template<class T> void ListIterator< T >::rewind () `[inline]`

6.78.2.12 template<class T> const T & ListIterator< T >::value () const `[inline]`

6.78.3 Friends And Related Function Documentation

6.78.3.1 template<class T> friend class List< T > `[friend]`

6.78.4 Member Data Documentation

6.78.4.1 template<class T> const List<T>& ListIterator< T >::list `[protected]`

6.78.4.2 template<class T> ListLink<T>* ListIterator< T >::p `[protected]`

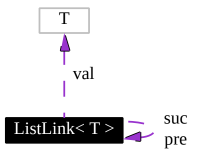
The documentation for this class was generated from the following file:

- **List.H**

6.79 ListLink< T > Class Template Reference

`#include <List.H>`

Collaboration diagram for ListLink< T >:



Friends

- class **List< T >**
- class **ListIterator< T >**

template<class T> class ListLink< T >

6.79.1 Friends And Related Function Documentation

6.79.1.1 template<class T> friend class List< T > `[friend]`

6.79.1.2 template<class T> friend class ListIterator< T > `[friend]`

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

- **List.H**

6.80 Mask Class Reference

`#include <Mask.H>`

Public Types

- enum **cellType** { **maskCovered** = -2, **maskPhysical** = -1, **maskCopy** = 0, **maskCoarse** = 1 }

Public Methods

- 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)

6.80.1 Member Enumeration Documentation

6.80.1.1 enum Mask::cellType

Enumeration values:

maskCovered

maskPhysical

maskCopy

maskCoarse

6.80.2 Member Function Documentation

6.80.2.1 void Mask::buildMask (BaseFab< int > & *a_mask*, const Box & *a_dProblem*, const BoxLayout & *a_grids*, const BoxLayout * *a_fineGridsPtr* = NULL, int *a_nRefFine* = -1)

6.80.2.2 void Mask::buildMask (BaseFab< int > & *a_mask*, const ProblemDomain & *a_dProblem*, const BoxLayout & *a_grids*, const BoxLayout * *a_fineGridsPtr* = NULL, int *a_nRefFine* = -1)

6.80.2.3 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)

6.80.2.4 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**

6.81 MayDay Class Reference

`#include <MayDay.H>`

Static Public Methods

- void **Error** (const char *msg=**nullString**)
- void **Abort** (const char *msg=**nullString**)
- void **Warning** (const char *msg=**nullString**)

Static Public Attributes

- const char *const **version**

Static Protected Attributes

- const char * **nullString**

6.81.1 Member Function Documentation

6.81.1.1 void MayDay::Abort (const char * *msg* = nullString) `[static]`

Print out message to cerr and exit via abort().

6.81.1.2 void MayDay::Error (const char * *msg* = nullString) `[static]`

Print out message to cerr and exit via **MayDay::Abort**() (p. 334).

6.81.1.3 void MayDay::Warning (const char * *msg* = nullString) `[static]`

Issues warning to stderr, but does not halt program execution.

6.81.2 Member Data Documentation

6.81.2.1 const char* MayDay::nullString `[static, protected]`

6.81.2.2 const char* const MayDay::version `[static]`

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

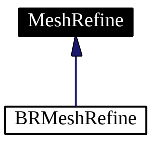
- **MayDay.H**

6.82 MeshRefine Class Reference

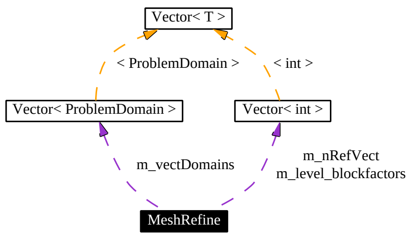
class which manages grid generation

`#include <MeshRefine.H>`

Inheritance diagram for MeshRefine:



Collaboration diagram for MeshRefine:



Public Methods

- **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?

- 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 Methods

- 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

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**

6.82.1 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** (p. 153) class).

There are two ways grids can be defined based on tagged cells. one takes a single **IntVectSet** (p. 274) of tags defined on the BaseLevel mesh and uses that set of tags for every level to be refined; the other takes a **Vector** (p. 471)<**IntVectSet** (p. 274)> of tags defined on all the mesh levels to be refined and uses those. \ \

{\bf 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** (p. 274) of tags is passed (instead of a **Vector** (p. 471)<**IntVectSet** (p. 274)>) 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.

6.82.2 Constructor & Destructor Documentation

6.82.2.1 MeshRefine::MeshRefine ()

default constructor – leaves object in an unusable state

6.82.2.2 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.

6.82.2.3 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.

6.82.2.4 virtual MeshRefine::~MeshRefine () [virtual]

destructor

6.82.3 Member Function Documentation

6.82.3.1 void MeshRefine::blockFactor (const int *a_block_factor*)

sets blocking factor

6.82.3.2 int MeshRefine::blockFactor () const

returns blocking factor

6.82.3.3 void MeshRefine::bufferSize (const int *a_buffer_size*)

sets proper nesting buffer size

6.82.3.4 int MeshRefine::bufferSize () const

returns proper nesting buffer size

6.82.3.5 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.

6.82.3.6 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. 156).

6.82.3.7 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. 156).

6.82.3.8 void MeshRefine::fillRatio (const Real *a_fill_ratio*)

sets fillRatio

6.82.3.9 Real MeshRefine::fillRatio () const

returns fillRatio

6.82.3.10 bool MeshRefine::isDefined () const

has this object been defined properly?

6.82.3.11 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. 157).

6.82.3.12 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]`

Parameters:

a_pnds output: proper nesting domains at each level

a_baseLevel input: index of highest **AMR** (p. 17) level not to be refined

a_topLevel input: index of highest **AMR** (p. 17) level in output

a_domains input: (same as in \func{meshRefine})

a_baseMesh input: boxes at mesh level \var{BaseLevel}

a_bufferSize input: (similar to \func{meshRefine; but with level-dependent coarsening factors})

6.82.3.13 void MeshRefine::maxSize (const int *a_max_size*)

sets maximum box size in any dimension – 0 means no limit

6.82.3.14 int MeshRefine::maxSize () const

returns maximum box size in any dimension – 0 means no limit

6.82.3.15 void MeshRefine::refRatios (const Vector< int > & *a_nRefVect*)

sets vector of refinement ratios

6.82.3.16 const Vector<int>& MeshRefine::refRatios () const

returns vector of refinement ratios

6.82.3.17 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)

6.82.3.18 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)

6.82.4 Member Data Documentation

6.82.4.1 int MeshRefine::m_blockFactor `[protected]`

6.82.4.2 int MeshRefine::m_bufferSize `[protected]`

6.82.4.3 Real MeshRefine::m_fillRatio `[protected]`

6.82.4.4 bool MeshRefine::m_isDefined `[protected]`

6.82.4.5 Vector<int> MeshRefine::m_level_blockfactors `[protected]`

6.82.4.6 int MeshRefine::m_maxSize `[protected]`

6.82.4.7 Vector<int> MeshRefine::m_nRefVect `[protected]`

6.82.4.8 Vector<ProblemDomain> MeshRefine::m_vectDomains `[protected]`

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

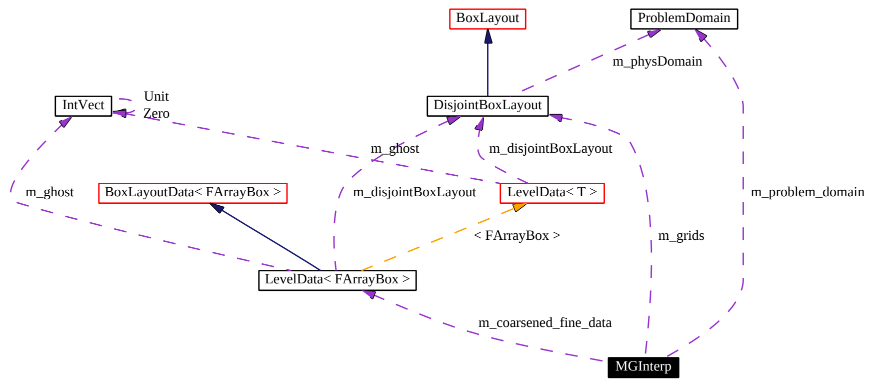
- **MeshRefine.H**

6.83 MGInterp Class Reference

class to interpolate coarse level data to a fine level

`#include <MGInterp.H>`

Collaboration diagram for MGInterp:



Public Methods

- **MGInterp** ()
- **~MGInterp** ()
- **MGInterp** (const **DisjointBoxLayout** &a_fine_domain, int a_numcomps, int a_ref_ratio, const **Box** &a_problem_domain)
- **MGInterp** (const **DisjointBoxLayout** &a_fine_domain, int a_numcomps, int a_ref_ratio, const **ProblemDomain** &a_problem_domain)
- void **define** (const **DisjointBoxLayout** &a_fine_domain, int a_numcomps, int a_ref_ratio, const **Box** &a_problem_domain)
- void **define** (const **DisjointBoxLayout** &a_fine_domain, int a_numcomps, int a_ref_ratio, const **ProblemDomain** &a_problem_domain)
- void **interpToFine** (**LevelData**< **FArrayBox** > &a_fine_data, const **LevelData**< **FArrayBox** > &a_coarse_data)
- bool **isDefined** () const

Protected Attributes

- bool **is_defined**
- int **m_ref_ratio**
- **LevelData**< **FArrayBox** > **m_coarsened_fine_data**
- **DisjointBoxLayout** **m_grids**
- **ProblemDomain** **m_problem_domain**

6.83.1 Detailed Description

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** (p. 69) and not a part of the Chombo API.

6.83.2 Constructor & Destructor Documentation

6.83.2.1 MGInterp::MGInterp ()

6.83.2.2 MGInterp::~MGInterp ()

6.83.2.3 MGInterp::MGInterp (const DisjointBoxLayout & *a_fine_domain*, int *a_numcomps*, int *a_ref_ratio*, const Box & *a_problem_domain*)

6.83.2.4 MGInterp::MGInterp (const DisjointBoxLayout & *a_fine_domain*, int *a_numcomps*, int *a_ref_ratio*, const ProblemDomain & *a_problem_domain*)

6.83.3 Member Function Documentation

6.83.3.1 void MGInterp::define (const DisjointBoxLayout & *a_fine_domain*, int *a_numcomps*, int *a_ref_ratio*, const ProblemDomain & *a_problem_domain*)

specify the fine level domain and refinement ratio.

6.83.3.2 void MGInterp::define (const DisjointBoxLayout & *a_fine_domain*, int *a_numcomps*, int *a_ref_ratio*, const Box & *a_problem_domain*)

specify the fine level domain and refinement ratio.

6.83.3.3 void MGInterp::interpToFine (LevelData< FArrayBox > & *a_fine_data*, const LevelData< FArrayBox > & *a_coarse_data*)

interpolate coarse level data to fine level

6.83.3.4 bool MGInterp::isDefined () const

6.83.4 Member Data Documentation

6.83.4.1 bool MGInterp::is_defined `[protected]`

6.83.4.2 LevelData<FArrayBox> MGInterp::m_coarsened_fine_data `[protected]`

6.83.4.3 DisjointBoxLayout MGInterp::m_grids `[protected]`

6.83.4.4 ProblemDomain MGInterp::m_problem_domain `[protected]`

6.83.4.5 int MGInterp::m_ref_ratio `[protected]`

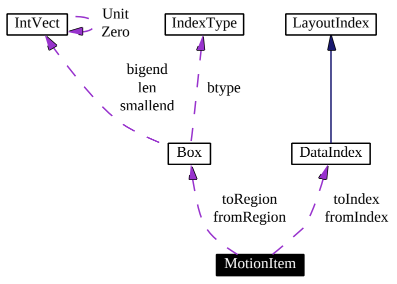
The documentation for this class was generated from the following file:

- **MGInterp.H**

6.84 MotionItem Class Reference

`#include <Copier.H>`

Collaboration diagram for MotionItem:



Public Methods

- **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)

Public Attributes

- **DataIndex fromIndex**
- **DataIndex toIndex**
- **Box fromRegion**
- **Box toRegion**
- int **procID**

6.84.1 Constructor & Destructor Documentation

6.84.1.1 MotionItem::MotionItem (const DataIndex & *a_from*, const DataIndex & *a_to*, const Box & *a_region*) `[inline]`

6.84.1.2 MotionItem::MotionItem (const DataIndex & *a_from*, const DataIndex & *a_to*, const Box & *a_fromRegion*, const Box & *a_toRegion*) `[inline]`

6.84.2 Member Data Documentation

6.84.2.1 DataIndex MotionItem::fromIndex

6.84.2.2 Box MotionItem::fromRegion

6.84.2.3 int MotionItem::procID

6.84.2.4 DataIndex MotionItem::toIndex

6.84.2.5 Box MotionItem::toRegion

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

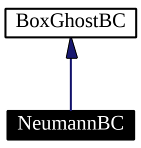
- **Copier.H**

6.85 NeumannBC Class Reference

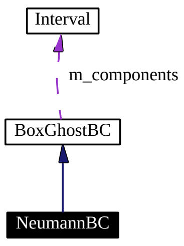
Class to enforce homogeneous neumann boundary conditions.

`#include <PoissonBC.H>`

Inheritance diagram for NeumannBC:



Collaboration diagram for NeumannBC:



Public Methods

- **~NeumannBC** ()
- **NeumannBC** ()
- **NeumannBC** (int dir, **Side::LoHiSide** sd)
- **NeumannBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)

Protected Methods

- 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

  virtual constructor workaround.

Friends

- class **DomainGhostBC**

6.85.1 Detailed Description

Class to enforce homogeneous neumann boundary conditions.

BoxGhostBC (p. 132) class implentation to enforce homogeneous neumann boundary conditions Recall that **BoxGhostBC** (p. 132) 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** (p. 215)

6.85.2 Constructor & Destructor Documentation

6.85.2.1 NeumannBC::~NeumannBC ()

6.85.2.2 NeumannBC::NeumannBC ()

6.85.2.3 NeumannBC::NeumannBC (int *dir*, Side::LoHiSide *sd*)

6.85.2.4 NeumannBC::NeumannBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.85.3 Member Function Documentation

6.85.3.1 virtual void NeumannBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const `[protected, virtual]`

fills neumann factor with 1.0 and other two fields with zero.

Implements **BoxGhostBC** (p. 135).

6.85.3.2 virtual void NeumannBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const `[protected, virtual]`

fills neumann factor with 1.0 and other two fields with zero.

Implements **BoxGhostBC** (p. 135).

6.85.3.3 BoxGhostBC* NeumannBC::new_boxghostbc () const `[protected, virtual]`

virtual constructor workaround.

Implements **BoxGhostBC** (p. 135).

6.85.4 Friends And Related Function Documentation

6.85.4.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 135).

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

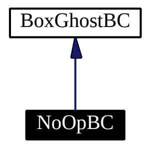
- **PoissonBC.H**

6.86 NoOpBC Class Reference

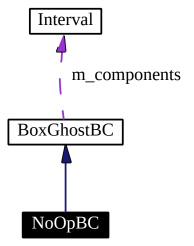
class to do nothing on an edge

`#include <edgeGhostBC.H>`

Inheritance diagram for NoOpBC:



Collaboration diagram for NoOpBC:



Public Methods

- **NoOpBC** ()
- **NoOpBC** (int a_dir, **Side::LoHiSide** a_sd)
- **NoOpBC** (int a_dir, **Side::LoHiSide** a_sd, const **Interval** &a_comps)
- virtual ~**NoOpBC** ()

Protected Methods

- 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** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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** (p. 106) *instead of* **ProblemDomain** (p. 418)

- 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

6.86.1 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!

6.86.2 Constructor & Destructor Documentation

6.86.2.1 NoOpBC::NoOpBC ()

6.86.2.2 NoOpBC::NoOpBC (int *a_dir*, Side::LoHiSide *a_sd*)

6.86.2.3 NoOpBC::NoOpBC (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*)

6.86.2.4 virtual NoOpBC::~NoOpBC () `[virtual]`

6.86.3 Member Function Documentation

6.86.3.1 virtual void NoOpBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

Reimplemented from **BoxGhostBC** (p. 134).

6.86.3.2 virtual void NoOpBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.86.3.3 virtual void NoOpBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

deprecated interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

Reimplemented from **BoxGhostBC** (p. 134).

6.86.3.4 virtual void NoOpBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.86.3.5 void NoOpBC::define (int *a_dir*, Side::LoHiSide *a_sd*, const Interval & *a_comps*) `[protected]`

Reimplemented from **BoxGhostBC** (p. 134).

6.86.3.6 void NoOpBC::define (int *a_dir*, Side::LoHiSide *a_sd*) `[protected]`

sets interval to be (0,0)

Reimplemented from **BoxGhostBC** (p. 134).

6.86.3.7 virtual void NoOpBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const Box & *a_domain*) const `[protected, virtual]`

Implements **BoxGhostBC** (p. 135).

6.86.3.8 virtual void NoOpBC::fillBCValues (FArrayBox & *a_neumfac*, FArrayBox & *a_dircfac*, FArrayBox & *a_inhmval*, Real *a_dx*, const ProblemDomain & *a_domain*) const `[protected, virtual]`

Implements **BoxGhostBC** (p. 135).

6.86.3.9 virtual BoxGhostBC* NoOpBC::new_boxghostbc () const `[protected, virtual]`

virtual constructor workaround

Implements **BoxGhostBC** (p. 135).

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

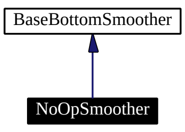
- **edgeGhostBC.H**

6.87 NoOpSmoother Class Reference

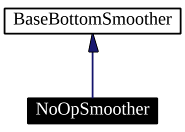
does nothing as a bottom smoother.

`#include <NoOpSmoother.H>`

Inheritance diagram for NoOpSmoother:



Collaboration diagram for NoOpSmoother:



Public Methods

- **NoOpSmoother** ()

  constructor, destructor default constructor

- virtual ~**NoOpSmoother** ()

  destructor

- virtual **BaseBottomSmoother** ∗ **new_bottomSmoother** () const
- virtual void **doBottomSmooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs, **LevelOp** ∗a_levelop_ptr)

  do (no) smoothing

6.87.1 Detailed Description

does nothing as a bottom smoother.

The NoOpSmoother class is a bottom smoother for **LevelOp** (p. 315)-derived classes which does nothing

6.87.2 Constructor & Destructor Documentation

6.87.2.1 NoOpSmoother::NoOpSmoother ()

constructor, destructor default constructor

6.87.2.2 virtual NoOpSmoother::~NoOpSmoother () [virtual]

destructor

6.87.3 Member Function Documentation

6.87.3.1 virtual void NoOpSmoother::doBottomSmooth (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*, LevelOp * *a_levelop_ptr*) [virtual]

do (no) smoothing

true to its name, this class does nothing in this functino

Implements **BaseBottomSmoother** (p. 82).

6.87.3.2 virtual BaseBottomSmoother* NoOpSmoother::new_bottomSmoother () const [virtual]

this gets around the "no virtual constructor" rule

Implements **BaseBottomSmoother** (p. 82).

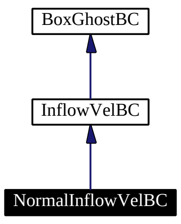
The documentation for this class was generated from the following file:

- **NoOpSmoother.H**

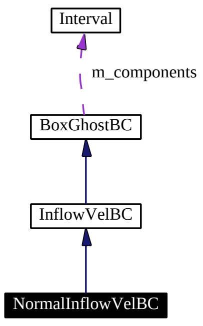
6.88 NormalInflowVelBC Class Reference

`#include <channelBC.H>`

Inheritance diagram for NormalInflowVelBC:



Collaboration diagram for NormalInflowVelBC:



Public Methods

- **NormalInflowVelBC** ()
- virtual ∼**NormalInflowVelBC** ()
- **NormalInflowVelBC** (int dir, **Side::LoHiSide** sd)
- **NormalInflowVelBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)

Protected Methods

- 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

  virtual constructor workaround.

Friends

- class **DomainGhostBC**

6.88.1 Constructor & Destructor Documentation

6.88.1.1 NormalInflowVelBC::NormalInflowVelBC ()

6.88.1.2 virtual NormalInflowVelBC::~NormalInflowVelBC () [virtual]

6.88.1.3 NormalInflowVelBC::NormalInflowVelBC (int *dir*, Side::LoHiSide *sd*)

6.88.1.4 NormalInflowVelBC::NormalInflowVelBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.88.2 Member Function Documentation

6.88.2.1 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. 261).

6.88.2.2 virtual BoxGhostBC* NormalInflowVelBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround.

Reimplemented from **InflowVelBC** (p. 262).

6.88.3 Friends And Related Function Documentation

6.88.3.1 friend class DomainGhostBC [friend]

Reimplemented from **InflowVelBC** (p. 262).

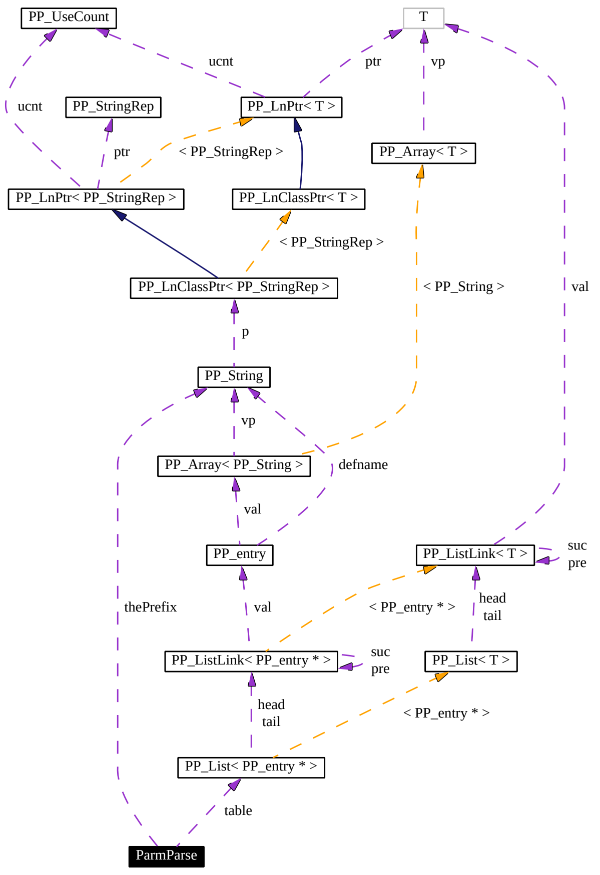
The documentation for this class was generated from the following file:

- **channelBC.H**

6.89 ParmParse Class Reference

`#include <ParmParse.H>`

Collaboration diagram for ParmParse:



Public Types

- enum **PPType** { **ppDefn**, **ppOption**, **ppInt**, **ppFloat**, **ppDouble**, **ppString**, **ppEQ_-sign**, **ppEOF** }

Public Methods

- **ParmParse** (int argc, char **argv, const char *prefix=0, const char *parfile=0)

  {\bf *constructors, destructor*}

- **ParmParse** (const char *prefix=0)
- **~ParmParse** ()
- bool **contains** (const char *name)

  {\bf *inquiry functions*}

- 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)

  {\bf access functions} 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)
- int **query** (const char *name, std::string &ref, int ival=0)
- 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)
- int **queryarr** (const char *name, **Vector**< int > &ref, int start_ix, int num_val)
- int **queryarr** (const char *name, std::vector< int > &ref, int start_ix, int num_val)
- void **getarr** (const char *name, **Vector**< float > &ref, int start_ix, int num_val)
- void **getarr** (const char *name, std::vector< float > &ref, int start_ix, int num_val)
- 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 Methods

- 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**

6.89.1 Member Enumeration Documentation

6.89.1.1 enum ParmParse::PPType

Enumeration values:

ppDefn

ppOption

ppInt

ppFloat

ppDouble

ppString

ppEQ_sign

ppEOF

6.89.2 Constructor & Destructor Documentation

6.89.2.1 ParmParse::ParmParse (int *argc*, char ** *argv*, const char * *prefix* = 0, const char * *parfile* = 0)

{\bf constructors, destructor}

6.89.2.2 ParmParse::ParmParse (const char * *prefix* = 0)

6.89.2.3 ParmParse::~ParmParse ()

6.89.3 Member Function Documentation

6.89.3.1 void ParmParse::addDefn (PP_String & *def*, PP_List< PP_String > & *val*, PP_List< PP_entry * > & *tab*) `[protected]`

6.89.3.2 void ParmParse::bldTable (const char * *str*, int *lenstr*, PP_List< PP_entry * > & *tab*) `[protected]`

6.89.3.3 bool ParmParse::contains (const std::string & *name*)

Returns true if name is in table.

6.89.3.4 bool ParmParse::contains (const char * *name*)

{\bf inquiry functions}

Returns true if name is in table.

6.89.3.5 int ParmParse::countname (const std::string & *name*) `[inline]`

6.89.3.6 int ParmParse::countname (const char * *name*)

6.89.3.7 int ParmParse::countval (const char * *name*, int *n* = -1) `[inline]`

6.89.3.8 void ParmParse::dumpTable (std::ostream & *os*)

6.89.3.9 void ParmParse::get (const char * *name*, std::string & *ref*, int *ival* = 0) `[inline]`

6.89.3.10 void ParmParse::get (const char * *name*, double & *ref*, int *ival* = 0) `[inline]`

6.89.3.11 void ParmParse::get (const char * *name*, float & *ref*, int *ival* = 0) `[inline]`

6.89.3.12 void ParmParse::get (const char * *name*, int & *ref*, int *ival* = 0) `[inline]`

{\bf access functions} access single object

6.89.3.13 void ParmParse::getarr (const char * *name*, const PPType *type*, void * *ptr*, int *start_ix*, int *num_val*, int *k* = -1) `[protected]`

6.89.3.14 void ParmParse::getarr (const char * *name*, std::vector< std::string > & *ref*, int *start_ix*, int *num_val*) `[inline]`

6.89.3.15 void ParmParse::getarr (const char * *name*, Vector< std::string > & *ref*, int *start_ix*, int *num_val*) `[inline]`

6.89.3.16 void ParmParse::getarr (const char * *name*, std::vector< double > & *ref*, int *start_ix*, int *num_val*) `[inline]`

6.89.3.17 void ParmParse::getarr (const char * *name*, Vector< double > & *ref*, int *start_ix*, int *num_val*) `[inline]`

6.89.3.18 void ParmParse::getarr (const char * *name*, std::vector< float > & *ref*, int *start_ix*, int *num_val*) `[inline]`

6.89.3.19 void ParmParse::getarr (const char * *name*, Vector< float > & *ref*, int *start_ix*, int *num_val*) `[inline]`

6.89.3.20 void ParmParse::getarr (const char * *name*, std::vector< int > & *ref*, int *start_ix*, int *num_val*) `[inline]`

6.89.3.21 void ParmParse::getarr (const char * *name*, Vector< int > & *ref*, int *start_ix*, int *num_val*) `[inline]`

access an array of objects

6.89.3.22 PPType ParmParse::getToken (const char *, int &, int, char *) [protected]

6.89.3.23 void ParmParse::getval (const char * *name*, const PPType *type*, void * *ptr*, int *ival*, int *k* = -1) [protected]

6.89.3.24 int ParmParse::isDouble (const PP_String & *str*, double & *val*) [inline, protected]

6.89.3.25 bool ParmParse::isInteger (const PP_String & *str*, int & *val*) [inline, protected]

6.89.3.26 const PP_entry* ParmParse::ppindex (int *n*, const char * *name*) const [protected]

6.89.3.27 void ParmParse::ppinit (const char * *parfile*) [protected]

6.89.3.28 int ParmParse::query (const char * *name*, std::string & *ref*, int *ival* = 0) [inline]

6.89.3.29 int ParmParse::query (const char * *name*, double & *ref*, int *ival* = 0) [inline]

6.89.3.30 int ParmParse::query (const char * *name*, float & *ref*, int *ival* = 0) [inline]

6.89.3.31 int ParmParse::query (const char * *name*, int & *ref*, int *ival* = 0) [inline]

6.89.3.32 int ParmParse::queryarr (const char * *name*, const PPType *type*, void * *ptr*, int *start_ix*, int *num_val*, int *k* = -1) [protected]

6.89.3.33 int ParmParse::queryarr (const char * *name*, std::vector< std::string > & *ref*, int *start_ix*, int *num_val*) [inline]

6.89.3.34 int ParmParse::queryarr (const char * *name*, Vector< std::string > & *ref*, int *start_ix*, int *num_val*) [inline]

6.89.3.35 int ParmParse::queryarr (const char * *name*, std::vector< double > & *ref*, int *start_ix*, int *num_val*) [inline]

6.89.3.36 int ParmParse::queryarr (const char * *name*, Vector< double > & *ref*, int *start_ix*, int *num_val*) [inline]

6.89.3.37 int ParmParse::queryarr (const char * *name*, std::vector< float > & *ref*, int *start_ix*, int *num_val*) [inline]

6.89.3.38 int ParmParse::queryarr (const char * *name*, Vector< float > & *ref*, int *start_ix*, int *num_val*) [inline]

6.89.3.39 int ParmParse::queryarr (const char * *name*, std::vector< int > & *ref*, int *start_ix*, int *num_val*) [inline]

6.89.3.40 int ParmParse::queryarr (const char * *name*, Vector< int > & *ref*, int *start_ix*, int *num_val*) [inline]

6.89.3.41 int ParmParse::queryval (const char * *name*, const PPType *type*, void * *ptr*, int *ival*, int *k* = -1) [protected]

6.89.3.42 void ParmParse::read_file (const char * *fname*, PP_List< PP_entry * > & *tab*) [protected]

6.89.3.43 void ParmParse::rmTable () [protected]

6.89.4 Friends And Related Function Documentation

6.89.4.1 friend class PP_entry [friend]

6.89.5 Member Data Documentation

6.89.5.1 int ParmParse::num_obj [static, protected]

6.89.5.2 PP_List<PP_entry*> ParmParse::table [static, protected]

6.89.5.3 PP_String ParmParse::thePrefix [protected]

6.89.5.4 int ParmParse::xargc [static, protected]

6.89.5.5 char** ParmParse::xargv [static, protected]

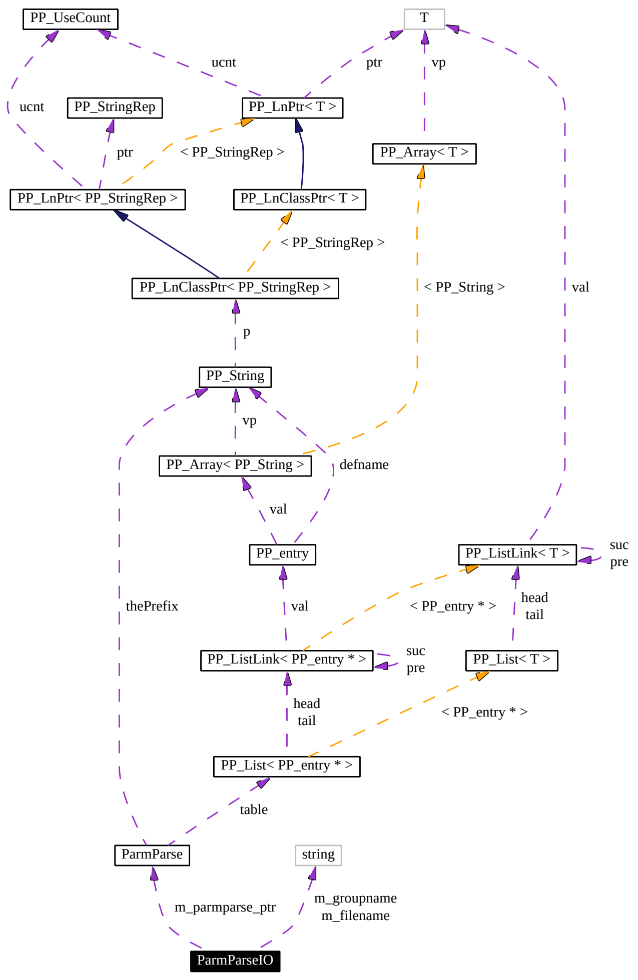
The documentation for this class was generated from the following file:

- **ParmParse.H**

6.90 ParmParseIO Class Reference

`#include <ParmParseIO.H>`

Collaboration diagram for ParmParseIO:



Public Methods

- **ParmParseIO** ()
- ~**ParmParseIO** ()
- void **openForRead** (const std::string &a_filename)
- void **openForWrite** (const std::string &a_filename)
- void **close** ()
- void **setGroup** (const std::string &a_groupname)
- const std::string **getGroup** () const
- const std::string & **filename** () const
- const std::string & **groupname** () const
- void **set** (const std::string &a_name, const **Real** &a_value)
- void **set** (const std::string &a_name, const int &a_value)

- void **set** (const std::string &a_name, const std::string &a_value)
- void **set** (const std::string &a_name, const **IntVect** &a_value)
- void **set** (const std::string &a_name, const **Box** &a_value)
- void **get** (const std::string &a_name, **Real** &a_value) const
- void **get** (const std::string &a_name, int &a_value) const
- void **get** (const std::string &a_name, std::string &a_value) const
- void **get** (const std::string &a_name, **IntVect** &a_value) const
- void **get** (const std::string &a_name, **Box** &a_value) const
- bool **found** (const std::string &a_name) const

Protected Attributes

- **ParmParse** * **m_parmparse_ptr**
- ofstream **m_ostrm**
- std::string **m_groupname**
- std::string **m_filename**

Friends

- void **write** (ParmParseIO &a_ppio, const **BoxLayout** &a_layout)
- int **read** (ParmParseIO &a_ppio, **Vector**< **Box** > &a_boxes)

6.90.1 Constructor & Destructor Documentation

6.90.1.1 ParmParseIO::ParmParseIO ()

6.90.1.2 ParmParseIO::~ParmParseIO ()

6.90.2 Member Function Documentation

6.90.2.1 void ParmParseIO::close ()

6.90.2.2 const std::string& ParmParseIO::filename () const

6.90.2.3 bool ParmParseIO::found (const std::string & *a_name*) const

6.90.2.4 void ParmParseIO::get (const std::string & *a_name*, Box & *a_value*) const

6.90.2.5 void ParmParseIO::get (const std::string & *a_name*, IntVect & *a_value*) const

6.90.2.6 void ParmParseIO::get (const std::string & *a_name*, std::string & *a_value*) const

6.90.2.7 void ParmParseIO::get (const std::string & *a_name*, int & *a_value*) const

6.90.2.8 void ParmParseIO::get (const std::string & *a_name*, Real & *a_value*) const

6.90.2.9 const std::string ParmParseIO::getGroup () const

6.90.2.10 const std::string& ParmParseIO::groupname () const

6.90.2.11 void ParmParseIO::openForRead (const std::string & *a_filename*)

6.90.2.12 void ParmParseIO::openForWrite (const std::string & *a_filename*)

6.90.2.13 void ParmParseIO::set (const std::string & *a_name*, const Box & *a_value*)

6.90.2.14 void ParmParseIO::set (const std::string & *a_name*, const IntVect & *a_value*)

6.90.2.15 void ParmParseIO::set (const std::string & *a_name*, const std::string & *a_value*)

6.90.2.16 void ParmParseIO::set (const std::string & *a_name*, const int & *a_value*)

6.90.2.17 void ParmParseIO::set (const std::string & *a_name*, const Real & *a_value*)

6.90.2.18 void ParmParseIO::setGroup (const std::string & *a_groupname*)

6.90.3 Friends And Related Function Documentation

6.90.3.1 int read (ParmParseIO & *a_ppio*, Vector< Box > & *a_boxes*) `[friend]`

6.90.3.2 void write (ParmParseIO & *a_ppio*, const BoxLayout & *a_layout*) `[friend]`

6.90.4 Member Data Documentation

6.90.4.1 std::string ParmParseIO::m_filename `[protected]`

6.90.4.2 std::string ParmParseIO::m_groupname `[protected]`

6.90.4.3 ofstream ParmParseIO::m_ostrm `[protected]`

6.90.4.4 ParmParse∗ ParmParseIO::m_parmparse_ptr `[protected]`

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

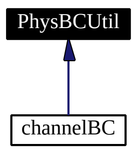
- **ParmParseIO.H**

6.91 PhysBCUtil Class Reference

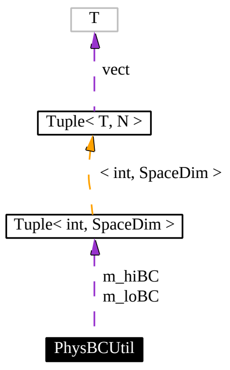
Encapsulates all physical BC's for IAMR solves.

`#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 Methods

- **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 sungle-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 Methods

- 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** (p. 357) *to set physical BC types*

Protected Attributes

- **Tuple**< int, **SpaceDim** > **m_loBC**
- **Tuple**< int, **SpaceDim** > **m_hiBC**

6.91.1 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

6.91.2 Member Enumeration Documentation

6.91.2.1 enum PhysBCUtil::BCType

Enumeration values:

bogusBC

SolidWall

Inflow

Outflow

Symmetry

noShear

Custom

NUM_PHYS_BC_TYPES

6.91.3 Constructor & Destructor Documentation

6.91.3.1 PhysBCUtil::PhysBCUtil ()

6.91.3.2 virtual PhysBCUtil::~PhysBCUtil () `[virtual]`

6.91.4 Member Function Documentation

6.91.4.1 virtual basicEdgeVelBC* PhysBCUtil::advectionVelBC () const `[virtual]`

returns edge-centered vel BC for edge-ctrd vel (single component)

6.91.4.2 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. 178).

6.91.4.3 virtual basicVelBC* PhysBCUtil::basicCCVelBC (bool *a_isHomogeneous*, bool *a_isViscous*) const [protected, virtual]

basic physical BC on cell-centered velocity

6.91.4.4 virtual basicEdgeVelBC* PhysBCUtil::basicECVelBC (bool *a_isHomogeneous*, bool *a_isViscous*, bool *a_isSingleComponent*) const [protected, virtual]

Reimplemented in **channelBC** (p. 179).

6.91.4.5 virtual DomainGhostBC* PhysBCUtil::basicGradPressureBC () const [protected, virtual]

sets basic physical BC on pressure before taking gradients

6.91.4.6 virtual DomainGhostBC* PhysBCUtil::basicPressureBC (bool *a_isHomogeneous*) const [protected, virtual]

sets basic physical BC on pressure (called by public BC functions)

6.91.4.7 virtual DomainGhostBC* PhysBCUtil::basicScalBC (bool *a_isHomogeneous*, int *a_scalType*) const [protected, virtual]

Reimplemented in **channelBC** (p. 179).

6.91.4.8 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. 179).

6.91.4.9 virtual DomainGhostBC* PhysBCUtil::diffusiveBC (int *a_scalarType*) const [virtual]

Scalar BC's.

Reimplemented in **channelBC** (p. 179).

6.91.4.10 virtual basicVelBC* PhysBCUtil::diffusiveSrcBC (int *scalarType*) const [virtual]

BC on diffusive source term itself.

6.91.4.11 Real PhysBCUtil::Dx () const

returns cell spacing

6.91.4.12 void PhysBCUtil::Dx (const Real *a_dx*)

miscellaneous utility functions sets cell spacing

6.91.4.13 virtual basicEdgeVelBC* PhysBCUtil::edgeVelBC () const [virtual]

returns edge-centered velocity BC for edge-centered vel (multicomponent)

6.91.4.14 virtual DomainGhostBC* PhysBCUtil::FreestreamCorrBC () const [virtual]

returns BC for freestream preservation correction

6.91.4.15 int PhysBCUtil::getBC (int *a_dir*, const Side::LoHiSide & *a_side*) const

6.91.4.16 virtual DomainGhostBC* PhysBCUtil::gradELambdaBC () const [virtual]

BC for grad eLambda.

Reimplemented in **channelBC** (p. 179).

6.91.4.17 virtual DomainGhostBC* PhysBCUtil::gradESyncBC () const [virtual]

returns BC applied to sync correction before computing gradient

6.91.4.18 virtual DomainGhostBC* PhysBCUtil::gradInitialProjCorrBC () const [virtual]

BC for gradient of correction for initial projection.

6.91.4.19 virtual DomainGhostBC* PhysBCUtil::gradMacPressureBC () const [virtual]

returns BC applied to MAC pressure before computing gradient

Reimplemented in **channelBC** (p. 179).

6.91.4.20 virtual DomainGhostBC* PhysBCUtil::gradPiBC () const [virtual]

returns BC applied to Pi before computing gradient

6.91.4.21 int PhysBCUtil::hiBC (int *a_dir*) const

6.91.4.22 virtual DomainGhostBC* PhysBCUtil::InitialProjBC () const [virtual]

returns pressure BC for initial velocity projection

6.91.4.23 virtual DomainGhostBC* PhysBCUtil::lambdaBC () const [virtual]

volume discrepancy BC's

6.91.4.24 virtual DomainGhostBC* PhysBCUtil::LevelPressureBC () const [virtual]

returns pressure BC for level projection

6.91.4.25 int PhysBCUtil::loBC (int *a_dir*) const

6.91.4.26 virtual DomainGhostBC* PhysBCUtil::MacPressureBC () const [virtual]

Pressure BCs returns pressure BC for MAC projection.

6.91.4.27 virtual PhysBCUtil* PhysBCUtil::newPhysBCUtil () const [virtual]

"virtual" constructor

Reimplemented in **channelBC** (p. 179).

6.91.4.28 virtual DomainGhostBC* PhysBCUtil::scalarDiffusionSolveBC (int *a_scalarType*) const [virtual]

6.91.4.29 virtual DomainGhostBC* PhysBCUtil::scalarRefluxSolveBC (int *a_scalarType*) const [virtual]

6.91.4.30 virtual DomainGhostBC* PhysBCUtil::scalarTraceBC (int *a_scalarType*) const [virtual]

6.91.4.31 virtual void PhysBCUtil::setBCs () [protected, virtual]

interact with **ParmParse** (p. 357) to set physical BC types

Reimplemented in **channelBC** (p. 179).

6.91.4.32 virtual DomainGhostBC* PhysBCUtil::streamFunctionBC (const Interval & *a_comps*) const [virtual]

6.91.4.33 virtual DomainGhostBC* PhysBCUtil::SyncProjBC () const [virtual]

returns pressure BC for sync projection

6.91.4.34 Real PhysBCUtil::Time () const

returns current time

6.91.4.35 void PhysBCUtil::Time (const Real *a_time*)

sets time

6.91.4.36 virtual basicVelBC* PhysBCUtil::tracingVelBC () const [virtual]

returns velocity BC for tracing (inviscid)

6.91.4.37 virtual basicVelBC* PhysBCUtil::uDelUBC (bool *a_isViscous*) const [virtual]

returns BC on cell-centered u-dot-del-u term

6.91.4.38 virtual basicVelBC* PhysBCUtil::uStarBC (bool *a_isViscous* = true) const [virtual]

returns pre-projection velocity BC

6.91.4.39 virtual DomainGhostBC* PhysBCUtil::viscousRefluxBC (int *a_dir*) const [virtual]

returns sungle-component BC for viscous refluxing solves

6.91.4.40 virtual DomainGhostBC* PhysBCUtil::viscousSolveBC (int *a_dir*) const [virtual]

returns single-component BC for viscous solves

6.91.4.41 virtual basicVelBC* PhysBCUtil::viscousSrcBC () const [virtual]

Miscellaneous BC's BC on viscous source term itself.

6.91.4.42 virtual basicVelBC* PhysBCUtil::viscousVelBC () const [virtual]

returns velocity BC for viscous source

6.91.5 Member Data Documentation

6.91.5.1 Tuple<int, SpaceDim> PhysBCUtil::m_hiBC [protected]

6.91.5.2 Tuple<int, SpaceDim> PhysBCUtil::m_loBC [protected]

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

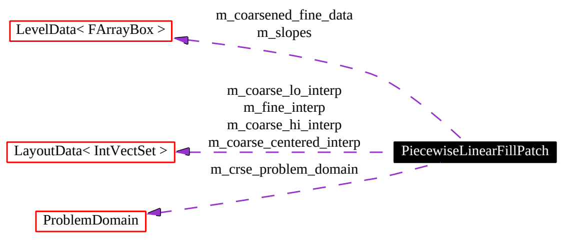
- **PhysBCUtil.H**

6.92 PiecewiseLinearFillPatch Class Reference

Fills ghost cells by linear interpolation in space and time.

`#include <PiecewiseLinearFillPatch.H>`

Collaboration diagram for PiecewiseLinearFillPatch:



Public Methods

- **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 Methods

- 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**

6.92.1 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.

\begin{verbatim} +======================================+ | | | | | | | | | | + - - - - - - - - - - - - - - - | + | 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 | +======================================+ \end{verbatim}

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.

\begin{verbatim} +——————–+===============+ | | coarse grid 1 | | | | | | | +=======================+ - - - - - - - | - + | \ / \ / \ / \ /|\ / \ / \ / \ /| | | V V V V | X X X < | . | | / \ / \ / \ / \|/ \ / \ / \ / \| | | +————————————+ | | \ /| | | | | X | | | . | | / \| | | | | | | | | | | | \ /| | | | | X | | | . | | / \| | | | +——-+—+—————+——-+ | | | |\ /

\ /|\ / \ /| | | | | ^ ^ | X X | X X | ^ ^ | . | | | |/ \ / \|/ \ / \| | | | + - - - | - - - - - - - +——-+ - + | | | | | | | | | | | | | | | | | +———–+ +===============+ | | | | | | | coarse grid 0 | | +======================+————–+ \end{verbatim}

6.92.2 Constructor & Destructor Documentation

6.92.2.1 PiecewiseLinearFillPatch::PiecewiseLinearFillPatch ()

Default constructor. User must subsequently call **define**() (p. 380).

6.92.2.2 PiecewiseLinearFillPatch::~PiecewiseLinearFillPatch ()

Destructor.

6.92.2.3 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.

{\bf 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. \

6.92.2.4 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.

{\bf 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. \

6.92.3 Member Function Documentation

6.92.3.1 void PiecewiseLinearFillPatch::computeSlopes (int *a_src_comp*, int *a_num_comp*) [protected]

6.92.3.2 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**() (p. 380) once and call **fillInterp**() (p. 380) multiple times with different valid data sets.

{\bf 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. \

{\bf This:}\ —This object is modified.—

6.92.3.3 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**() (p. 380) once and call **fillInterp**() (p. 380) multiple times with different valid data sets.

{\bf 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. \

{\bf This:}\ —This object is modified.—

6.92.3.4 void PiecewiseLinearFillPatch::fillConstantInterp (LevelData< FArrayBox > & *a_fine_data*, int *a_src_comp*, int *a_dest_comp*, int *a_num_comp*) const [protected]

6.92.3.5 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**() (p. 381). 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**() (p. 380). It is expected that the coarse and fine level's domains are properly nested.

{\bf 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.

{\bf 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.

6.92.3.6 void PiecewiseLinearFillPatch::incrementLinearInterp (LevelData< FArrayBox > & *a_fine_data*, int *a_src_comp*, int *a_dest_comp*, int *a_num_comp*) const `[protected]`

6.92.3.7 bool PiecewiseLinearFillPatch::isDefined () const

Returns true if this object was created with the defining constructor or if **define**() (p. 380) has been called.

{\bf This:}\ This object is not modified.

6.92.3.8 void PiecewiseLinearFillPatch::printIntVectSets () const

6.92.3.9 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]

6.92.4 Member Data Documentation

6.92.4.1 LayoutData<IntVectSet> PiecewiseLinearFillPatch::m_coarse_centered_interp[SpaceDim] [protected]

6.92.4.2 LayoutData<IntVectSet> PiecewiseLinearFillPatch::m_coarse_hi_interp[SpaceDim] [protected]

6.92.4.3 LayoutData<IntVectSet> PiecewiseLinearFillPatch::m_coarse_lo_interp[SpaceDim] [protected]

6.92.4.4 LevelData<FArrayBox> PiecewiseLinearFillPatch::m_coarsened_fine_data [protected]

6.92.4.5 ProblemDomain PiecewiseLinearFillPatch::m_crse_problem_domain [protected]

6.92.4.6 LayoutData<IntVectSet> PiecewiseLinearFillPatch::m_fine_interp [protected]

6.92.4.7 int PiecewiseLinearFillPatch::m_interp_radius [protected]

6.92.4.8 bool PiecewiseLinearFillPatch::m_is_defined [protected]

6.92.4.9 int PiecewiseLinearFillPatch::m_ref_ratio [protected]

6.92.4.10 LevelData<FArrayBox> PiecewiseLinearFillPatch::m_slopes[3] [protected]

6.92.4.11 const int PiecewiseLinearFillPatch::s_stencil_radius [static, protected]

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

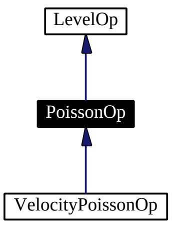
- **PiecewiseLinearFillPatch.H**

6.93 PoissonOp Class Reference

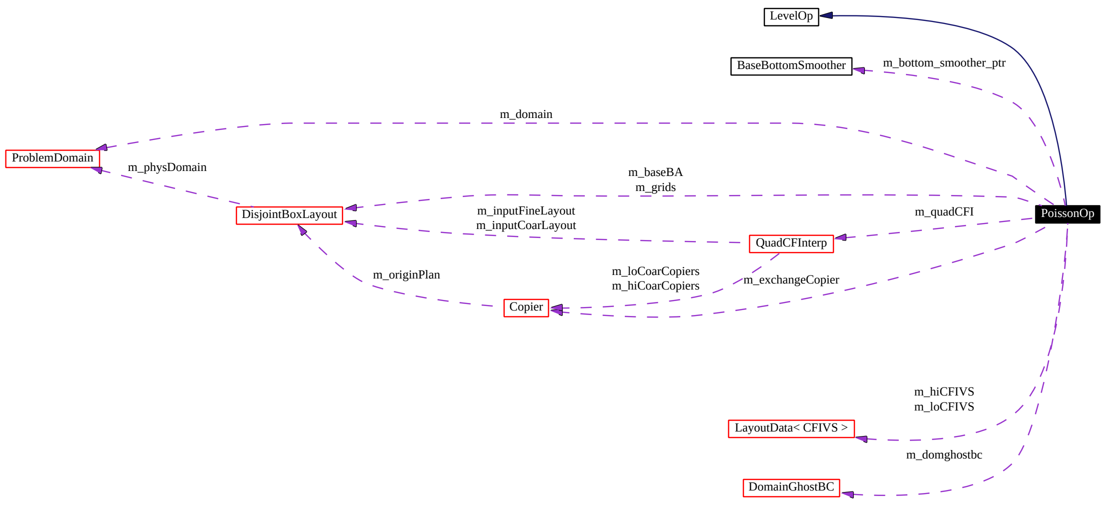
one-sided differencing version of poisson operator

`#include <PoissonOp.H>`

Inheritance diagram for PoissonOp:



Collaboration diagram for PoissonOp:



Public Methods

- virtual **LevelOp** ∗ **new_levelop** () const
- virtual bool **isDefined** () const
- void **define** (const **DisjointBoxLayout** &Ba, const **DisjointBoxLayout** ∗base_ba, **Real** DxLevel, int refratio, const **Box** &domf, bool a_homogeneousOnly=false, int ncomp=1)
- void **define** (const **DisjointBoxLayout** &Ba, const **DisjointBoxLayout** ∗base_ba, **Real** DxLevel, int refratio, const **ProblemDomain** &domf, bool a_homogeneousOnly=false, int ncomp=1)
- void **define** (const **LevelOp** ∗opfine, int refratio)
- **PoissonOp** ()
- **~PoissonOp** ()
- void **smooth** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs)

- virtual void **levelPreconditioner** (**LevelData**< **FArrayBox** > &a_phihat, const **LevelData**< **FArrayBox** > &a_rhshat)
- void **applyOpI** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phiCoarse, **LevelData**< **FArrayBox** > &a_LOfPhi)
- void **applyOpIcfHphys** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > *a_phiCoarse, **LevelData**< **FArrayBox** > &a_LOfPhi)
- void **applyOpH** (**LevelData**< **FArrayBox** > &a_phi, **LevelData**< **FArrayBox** > &a_LOfPhi)
- void **CFInterp** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_phiCoarse)
- void **applyOpHcfIphys** (**LevelData**< **FArrayBox** > &a_phi, **LevelData**< **FArrayBox** > &a_lOfPhi)
- void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &a_phi)
- void **setDomainGhostBC** (const **DomainGhostBC** &a_dombcin)
- void **setBottomSmoother** (const **BaseBottomSmoother** &a_bottomSmoother)
- void **homogeneousCFInterp** (**LevelData**< **FArrayBox** > &a_phif, const **DataIndex** &a_datInd, int a_idir, **Side::LoHiSide** a_hiorlo)
- void **bottomSmoother** (**LevelData**< **FArrayBox** > &a_phi, const **LevelData**< **FArrayBox** > &a_rhs)
- void **getFlux** (**FArrayBox** &flux, const **FArrayBox** &a_data, const **DataIndex** &a_datInd, int a_dir)

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_bottom_smoother_ptr**
- bool **m_isDefined**
- bool **m_isBCDefined**
- bool **m_ihcfiEnabled**
- **LayoutData**< **CFIVS** > **m_loCFIVS** [**SpaceDim**]
- **LayoutData**< **CFIVS** > **m_hiCFIVS** [**SpaceDim**]

6.93.1 Detailed Description

one-sided differencing version of poisson operator

this class contains stuff to solve div (grad phi) = rhs

6.93.2 Constructor & Destructor Documentation

6.93.2.1 PoissonOp::PoissonOp ()

6.93.2.2 PoissonOp::~PoissonOp ()

6.93.3 Member Function Documentation

6.93.3.1 void PoissonOp::applyOpH (LevelData< FArrayBox > & *a_phi*, LevelData< FArrayBox > & *a_LOfPhi*) [virtual]

Evaluate Operator, homogeneous C/F boundary conditions. and homogeneous physical boundary conditions

Implements **LevelOp** (p. 316).

6.93.3.2 void PoissonOp::applyOpHcfIphys (LevelData< FArrayBox > & *a_phi*, LevelData< FArrayBox > & *a_lOfPhi*) [virtual]

Evaluate Operator, homogeneous C/F boundary conditions. with inhomogeneous physical boundary conditions

Implements **LevelOp** (p. 316).

6.93.3.3 void PoissonOp::applyOpI (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > * *a_phiCoarse*, LevelData< FArrayBox > & *a_LOfPhi*) [virtual]

Evaluate Operator, inhomogeneous C/F boundary conditions. and inhomogeneous phys boundary conditions

Implements **LevelOp** (p. 316).

6.93.3.4 void PoissonOp::applyOpIcfHphys (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > * *a_phiCoarse*, LevelData< FArrayBox > & *a_LOfPhi*) [virtual]

Evaluate Operator, inhomogeneous C/F boundary conditions. with homogeneous physical boundary conditions

Implements **LevelOp** (p. 316).

6.93.3.5 void PoissonOp::bottomSmoother (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_rhs*) [virtual]

This does a GSRB Pre/Conditioned BiCGStab on a level for the bottom solver.

Implements **LevelOp** (p. 316).

6.93.3.6 void PoissonOp::CFInterp (LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > & *a_phiCoarse*) [virtual]

Coarse / Fine interpolation operator.

Implements **LevelOp** (p. 317).

6.93.3.7 void PoissonOp::define (const LevelOp * *opfine*, int *refratio*) [virtual]

Full define function. define from finer levelop.

Implements **LevelOp** (p. 317).

6.93.3.8 void PoissonOp::define (const DisjointBoxLayout & *Ba*, const DisjointBoxLayout * *base_ba*, Real *DxLevel*, int *refratio*, const ProblemDomain & *domf*, bool *a_homogeneousOnly* = false, int *ncomp* = 1) [virtual]

Full define function. makes all coarse-fine information and sets internal variables

Implements **LevelOp** (p. 317).

6.93.3.9 void PoissonOp::define (const DisjointBoxLayout & *Ba*, const DisjointBoxLayout * *base_ba*, Real *DxLevel*, int *refratio*, const Box & *domf*, bool *a_homogeneousOnly* = false, int *ncomp* = 1) [virtual]

Full define function. makes all coarse-fine information and sets internal variables

Implements **LevelOp** (p. 317).

6.93.3.10 void PoissonOp::getFlux (FArrayBox & *flux*, const FArrayBox & *a_data*, const DataIndex & *a_datInd*, int *a_dir*) [virtual]

get flux(== 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. 317).

6.93.3.11 void PoissonOp::homogeneousCFInterp (LevelData< FArrayBox > & *a_phif*, const DataIndex & *a_datInd*, int *a_idir*, Side::LoHiSide *a_hiorlo*)

does homogeneous coarse/fine interpolation

6.93.3.12 void PoissonOp::homogeneousCFInterp (LevelData< FArrayBox > & *a_phi*) [virtual]

homogeneous coarse/fine interpolation operator

Implements **LevelOp** (p. 317).

6.93.3.13 virtual bool PoissonOp::isDefined () const [virtual]

Implements **LevelOp** (p. 318).

Reimplemented in **VelocityPoissonOp** (p. 476).

6.93.3.14 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. 318).

6.93.3.15 virtual LevelOp* PoissonOp::new_levelop () const [virtual]

virtual constructor workaround

Implements **LevelOp** (p. 318).

Reimplemented in **VelocityPoissonOp** (p. 477).

6.93.3.16 void PoissonOp::setBottomSmoother (const BaseBottomSmoother & *a_bottomSmoother*)

6.93.3.17 void PoissonOp::setDomainGhostBC (const DomainGhostBC & *a_dombcin*)

6.93.3.18 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. 318).

6.93.4 Member Data Documentation

6.93.4.1 DisjointBoxLayout PoissonOp::m_baseBA [protected]

6.93.4.2 BaseBottomSmoother* PoissonOp::m_bottom_smoother_ptr [protected]

6.93.4.3 ProblemDomain PoissonOp::m_domain [protected]

6.93.4.4 DomainGhostBC PoissonOp::m_domghostbc [protected]

6.93.4.5 Real PoissonOp::m_dxCrse [protected]

6.93.4.6 Real PoissonOp::m_dxLevel [protected]

6.93.4.7 Copier PoissonOp::m_exchangeCopier [protected]

6.93.4.8 DisjointBoxLayout PoissonOp::m_grids [protected]

6.93.4.9 LayoutData<CFIVS> PoissonOp::m_hiCFIVS[SpaceDim] [protected]

6.93.4.10 bool PoissonOp::m_ihcfiEnabled [protected]

6.93.4.11 bool PoissonOp::m_isBCDefined [protected]

6.93.4.12 bool PoissonOp::m_isDefined [protected]

6.93.4.13 LayoutData<CFIVS> PoissonOp::m_loCFIVS[SpaceDim] [protected]

6.93.4.14 int PoissonOp::m_ncomp [protected]

6.93.4.15 QuadCFInterp PoissonOp::m_quadCFI [protected]

6.93.4.16 int PoissonOp::m_refRatio [protected]

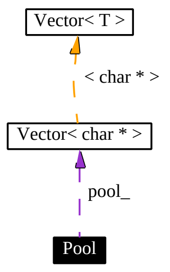
The documentation for this class was generated from the following file:

- **PoissonOp.H**

6.94 Pool Class Reference

`#include <Pool.H>`

Collaboration diagram for Pool:



Public Methods

- **Pool** (int ptrSize, const char *name="unnamed", int poolSize=100, int alignment=sizeof(int))
- **~Pool** ()
- void * **getPtr** ()
- void **returnPtr** (void *ptr)
- long **memUsage** () const
- void **clear** ()

Public Attributes

- char **name_** [64]

Static Public Attributes

- **PoolList** * **poolList_**

6.94.1 Constructor & Destructor Documentation

6.94.1.1 Pool::Pool (int *ptrSize*, const char ∗ *name* = "unnamed", int *poolSize* = 100, int *alignment* = sizeof(int))

6.94.1.2 Pool::∼Pool ()

6.94.2 Member Function Documentation

6.94.2.1 void Pool::clear ()

6.94.2.2 void∗ Pool::getPtr ()

6.94.2.3 long Pool::memUsage () const

6.94.2.4 void Pool::returnPtr (void ∗ *ptr*)

6.94.3 Member Data Documentation

6.94.3.1 char Pool::name_[64]

6.94.3.2 PoolList∗ Pool::poolList_ [static]

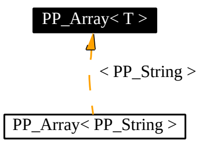
The documentation for this class was generated from the following file:

- **Pool.H**

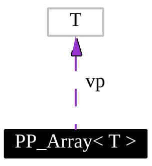
6.95 PP_Array< T > Class Template Reference

`#include <ParmParse.H>`

Inheritance diagram for PP_Array< T >:



Collaboration diagram for PP_Array< T >:



Public Methods

- **PP_Array** ()
- **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)
- PP_Array< T > & **operator=** (const PP_Array< T > &rhs)
- **~PP_Array** ()
- void **clear** ()
- bool **ready** () const
- void **reserve** (long _truesize)
- void **shrinkWrap** ()
- void **resize** (long newlen)
- void **resize** (long newlen, const T &initialvalue)
- long **length** () const
- long **trueSize** () const
- T & **operator**[] (long K)
- const T & **operator**[] (long K) const
- T & **get** (long i)
- const T & **get** (long i) const
- T * **dataPtr** ()
- const T * **dataPtr** () const
- void **set** (long i, const T &elem)
- void **swap** (long i, long j)
- bool **operator==** (const PP_Array< T > &rhs) const
- bool **operator!=** (const PP_Array< T > &rhs) const

Protected Attributes

- long **truesize**

- long **nelem**

- T ∗ **vp**

template<class T> class PP_Array< T >

6.95.1 Constructor & Destructor Documentation

6.95.1.1 template<class T> PP_Array< T >::PP_Array () [inline]

6.95.1.2 template<class T> PP_Array< T >::PP_Array (long *len*) [inline]

6.95.1.3 template<class T> PP_Array< T >::PP_Array (long *len*, const T & *initialvalue*)

6.95.1.4 template<class T> PP_Array< T >::PP_Array (const T * *vec*, long *len*)

6.95.1.5 template<class T> PP_Array< T >::PP_Array (const PP_Array< T > & *rhs*)

6.95.1.6 template<class T> PP_Array< T >::~PP_Array () [inline]

6.95.2 Member Function Documentation

6.95.2.1 template<class T> void PP_Array< T >::clear () [inline]

6.95.2.2 template<class T> const T * PP_Array< T >::dataPtr () const [inline]

6.95.2.3 template<class T> T * PP_Array< T >::dataPtr () [inline]

6.95.2.4 template<class T> const T & PP_Array< T >::get (long *i*) const [inline]

6.95.2.5 template<class T> T & PP_Array< T >::get (long *i*) [inline]

6.95.2.6 template<class T> long PP_Array< T >::length () const [inline]

6.95.2.7 template<class T> bool PP_Array< T >::operator!= (const PP_Array< T > & *rhs*) const [inline]

6.95.2.8 template<class T> PP_Array< T > & PP_Array< T >::operator= (const PP_Array< T > & *rhs*)

6.95.2.9 template<class T> bool PP_Array< T >::operator== (const PP_Array< T > & *rhs*) const

6.95.2.10 template<class T> const T & PP_Array< T >::operator[] (long *K*) const [inline]

6.95.2.11 template<class T> T & PP_Array< T >::operator[] (long *K*) [inline]

6.95.2.12 template<class T> bool PP_Array< T >::ready () const [inline]

6.95.2.13 template<class T> void PP_Array< T >::reserve (long *_truesize*)

6.95.2.14 template<class T> void PP_Array< T >::resize (long *newlen*, const T & *initialvalue*) [inline]

6.95.2.15 template<class T> void PP_Array< T >::resize (long *newlen*) [inline]

6.95.2.16 template<class T> void PP_Array< T >::set (long *i*, const T & *elem*) [inline]

6.95.2.17 template<class T> void PP_Array< T >::shrinkWrap ()

6.95.2.18 template<class T> void PP_Array< T >::swap (long *i*, long *j*) [inline]

6.95.2.19 template<class T> long PP_Array< T >::trueSize () const [inline]

6.95.3 Member Data Documentation

6.95.3.1 template<class T> long PP_Array< T >::nelem [protected]

6.95.3.2 template<class T> long PP_Array< T >::truesize [protected]

6.95.3.3 template<class T> T* PP_Array< T >::vp [protected]

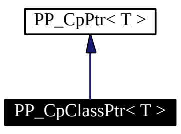
The documentation for this class was generated from the following file:

- **ParmParse.H**

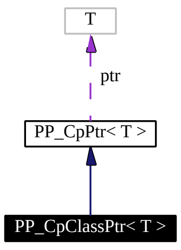
6.96 PP_CpClassPtr< T > Class Template Reference

`#include <ParmParse.H>`

Inheritance diagram for PP_CpClassPtr< T >:



Collaboration diagram for PP_CpClassPtr< T >:



Public Methods

- **PP_CpClassPtr** ()
- **PP_CpClassPtr** (T ∗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

template<class T> class PP_CpClassPtr< T >

6.96.1 Constructor & Destructor Documentation

6.96.1.1 template<class T> PP_CpClassPtr< T >::PP_CpClassPtr () `[inline]`

6.96.1.2 template<class T> PP_CpClassPtr< T >::PP_CpClassPtr (T * *rhs*) `[inline]`

6.96.1.3 template<class T> PP_CpClassPtr< T >::PP_CpClassPtr (const PP_CpClassPtr< T > & *rhs*) `[inline]`

6.96.2 Member Function Documentation

6.96.2.1 template<class T> T * PP_CpClassPtr< T >::operator → () const `[inline]`

6.96.2.2 template<class T> PP_CpClassPtr< T > & PP_CpClassPtr< T >::operator= (const PP_CpClassPtr< T > & *rhs*) `[inline]`

6.96.2.3 template<class T> PP_CpClassPtr< T > & PP_CpClassPtr< T >::operator= (T * *rhs*) `[inline]`

Reimplemented from **PP_CpPtr< T >** (p. 398).

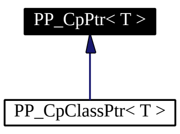
The documentation for this class was generated from the following file:

- **ParmParse.H**

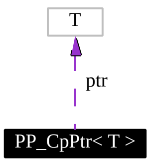
6.97 PP_CpPtr< T > Class Template Reference

```
#include <ParmParse.H>
```

Inheritance diagram for PP_CpPtr< T >:



Collaboration diagram for PP_CpPtr< T >:



Public Methods

- **PP_CpPtr** ()
- **PP_CpPtr** (T ∗rhs)
- ∼**PP_CpPtr** ()
- **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
- T ∗ **release** ()
- bool **operator==** (const PP_CpPtr< T > &rhs) const
- bool **operator!=** (const PP_CpPtr< T > &rhs) const

Protected Attributes

- T ∗ **ptr**

template<class T> class PP_CpPtr< T >

6.97.1 Constructor & Destructor Documentation

6.97.1.1 template<class T> PP_CpPtr< T >::PP_CpPtr () `[inline]`

6.97.1.2 template<class T> PP_CpPtr< T >::PP_CpPtr (T * *rhs*) `[inline]`

6.97.1.3 template<class T> PP_CpPtr< T >::~PP_CpPtr () `[inline]`

6.97.1.4 template<class T> PP_CpPtr< T >::PP_CpPtr (const PP_CpPtr< T > & *rhs*) `[inline]`

6.97.2 Member Function Documentation

6.97.2.1 template<class T> bool PP_CpPtr< T >::isNull () const `[inline]`

6.97.2.2 template<class T> T & PP_CpPtr< T >::operator * () const `[inline]`

6.97.2.3 template<class T> bool PP_CpPtr< T >::operator!= (const PP_CpPtr< T > & *rhs*) const `[inline]`

6.97.2.4 template<class T> PP_CpPtr< T > & PP_CpPtr< T >::operator= (const PP_CpPtr< T > & *rhs*) `[inline]`

6.97.2.5 template<class T> PP_CpPtr< T > & PP_CpPtr< T >::operator= (T * *rhs*) `[inline]`

Reimplemented in **PP_CpClassPtr< T >** (p. 396).

6.97.2.6 template<class T> bool PP_CpPtr< T >::operator== (const PP_CpPtr< T > & *rhs*) const `[inline]`

6.97.2.7 template<class T> T * PP_CpPtr< T >::release () `[inline]`

6.97.3 Member Data Documentation

6.97.3.1 template<class T> T* PP_CpPtr< T >::ptr `[protected]`

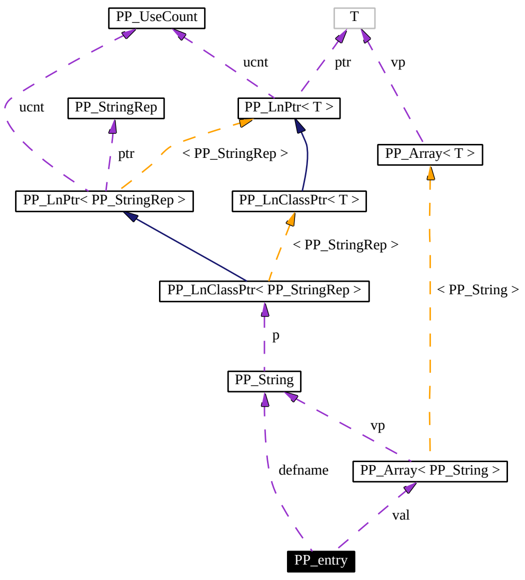
The documentation for this class was generated from the following file:

- **ParmParse.H**

6.98 PP_entry Class Reference

`#include <ParmParse.H>`

Collaboration diagram for PP_entry:



Friends

- class **ParmParse**

6.98.1 Friends And Related Function Documentation

6.98.1.1 friend class ParmParse `[friend]`

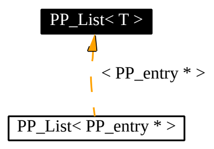
The documentation for this class was generated from the following file:

- **ParmParse.H**

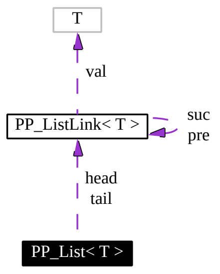
6.99 PP_List< T > Class Template Reference

`#include <ParmParse.H>`

Inheritance diagram for PP_List< T >:



Collaboration diagram for PP_List< T >:



Public Methods

- **PP_List** ()
- **PP_List** (const PP_List< T > &rhs)
- PP_List< T > & **operator=** (const PP_List< T > &rhs)
- **~PP_List** ()
- void **prepend** (const T &value)
- void **append** (const T &value)
- void **add** (const T &value)
- void **join** (const PP_List< T > &src)
- void **catenate** (PP_List< T > &src)
- void **clear** ()
- T & **firstElement** () const
- T & **lastElement** () const
- bool **includes** (const T &value) const
- bool **operator==** (const PP_List< T > &rhs) const
- bool **operator!=** (const PP_List< T > &rhs) const
- bool **isEmpty** () const
- bool **isNotEmpty** () const
- int **length** () const
- void **removeFirst** ()
- void **removeLast** ()
- const T & **operator**[] (const **PP_ListIterator**< T > &li) const
- T & **operator**[] (const **PP_ListIterator**< T > &li)
- void **remove** (const T &value)

- void **remove** (const PP_List< T > &**lst**)
- void **remove** (**PP_ListIterator**< T > &lit)
- void **replace** (**PP_ListIterator**< T > &li, const T &val)
- void **addAfter** (**PP_ListIterator**< T > &lit, const T &val)
- void **addBefore** (**PP_ListIterator**< T > &lit, const T &val)
- **PP_ListIterator**< T > **first** () const
- **PP_ListIterator**< T > **last** () const

Protected Methods

- 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 >**

template<class T> class PP_List< T >

6.99.1 Constructor & Destructor Documentation

6.99.1.1 template<class T> PP_List< T >::PP_List () [inline]

6.99.1.2 template<class T> PP_List< T >::PP_List (const PP_List< T > & *rhs*) [inline]

6.99.1.3 template<class T> PP_List< T >::~PP_List () [inline]

6.99.2 Member Function Documentation

6.99.2.1 template<class T> void PP_List< T >::add (const T & *value*) [inline]

6.99.2.2 template<class T> PP_ListLink< T > * PP_List< T >::addAfter (PP_ListLink< T > * *ln*, const T & *val*) [inline, protected]

6.99.2.3 template<class T> void PP_List< T >::addAfter (PP_ListIterator< T > & *lit*, const T & *val*) [inline]

6.99.2.4 template<class T> PP_ListLink< T > * PP_List< T >::addBefore (PP_ListLink< T > * *ln*, const T & *val*) [inline, protected]

6.99.2.5 template<class T> void PP_List< T >::addBefore (PP_ListIterator< T > & *lit*, const T & *val*) [inline]

6.99.2.6 template<class T> void PP_List< T >::append (const T & *value*) [inline]

6.99.2.7 template<class T> void PP_List< T >::catenate (PP_List< T > & *src*) [inline]

6.99.2.8 template<class T> void PP_List< T >::clear () [inline]

6.99.2.9 template<class T> PP_ListIterator< T > PP_List< T >::first () const [inline]

6.99.2.10 template<class T> T & PP_List< T >::firstElement () const [inline]

6.99.2.11 template<class T> bool PP_List< T >::includes (const T & *value*) const [inline]

6.99.2.12 template<class T> bool PP_List< T >::isEmpty () const [inline]

6.99.2.13 template<class T> bool PP_List< T >::isNotEmpty () const [inline]

6.99.2.14 template<class T> void PP_List< T >::join (const PP_List< T > & *src*) [inline]

6.99.2.15 template<class T> PP_ListIterator< T > PP_List< T >::last () const [inline]

6.99.2.16 template<class T> T & PP_List< T >::lastElement () const [inline]

6.99.2.17 template<class T> int PP_List< T >::length () const [inline]

6.99.2.18 template<class T> bool PP_List< T >::operator!= (const PP_List< T > & *rhs*) const [inline]

6.99.2.19 template<class T> PP_List< T > & PP_List< T >::operator= (const PP_List< T > & *rhs*) [inline]

6.99.2.20 template<class T> bool PP_List< T >::operator== (const PP_List< T > & *rhs*) const [inline]

6.99.2.21 template<class T> T & PP_List< T >::operator[] (const PP_ListIterator< T > & *li*) [inline]

6.99.2.22 template<class T> const T & PP_List< T >::operator[] (const PP_ListIterator< T > & *li*) const [inline]

6.99.2.23 template<class T> void PP_List< T >::prepend (const T & *value*) [inline]

6.99.2.24 template<class T> void PP_List< T >::remove (PP_ListLink< T > * *ln*) [inline, protected]

6.99.2.25 template<class T> void PP_List< T >::remove (PP_ListIterator< T > & *lit*) [inline]

6.99.2.26 template<class T> void PP_List< T >::remove (const PP_List< T > & *lst*) [inline]

6.99.2.27 template<class T> void PP_List< T >::remove (const T & *value*) [inline]

6.99.2.28 template<class T> void PP_List< T >::removeFirst () [inline]

6.99.2.29 template<class T> void PP_List< T >::removeLast () [inline]

6.99.2.30 template<class T> void PP_List< T >::replace (PP_ListIterator< T > & *li*, const T & *val*) [inline]

6.99.3 Friends And Related Function Documentation

6.99.3.1 template<class T> friend class PP_ListIterator< T > [friend]

6.99.4 Member Data Documentation

6.99.4.1 template<class T> PP_ListLink<T>* PP_List< T >::head [protected]

6.99.4.2 template<class T> PP_ListLink<T>* PP_List< T >::tail [protected]

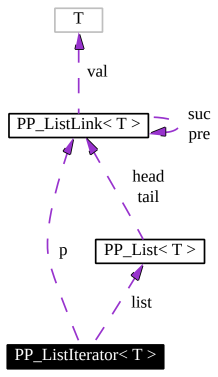
The documentation for this class was generated from the following file:

- **ParmParse.H**

6.100 PP_ListIterator< T > Class Template Reference

`#include <ParmParse.H>`

Collaboration diagram for PP_ListIterator< T >:



Public Methods

- **PP_ListIterator** (const **PP_List**< T > &aList)
- **PP_ListIterator** (const PP_ListIterator< T > &rhs)
- 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

Protected Methods

- **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** >

template<class T> class PP_ListIterator< T >

6.100.1 Constructor & Destructor Documentation

6.100.1.1 template<class T> PP_ListIterator< T >::PP_ListIterator (const PP_List< T > & *aList*) [inline]

6.100.1.2 template<class T> PP_ListIterator< T >::PP_ListIterator (const PP_ListIterator< T > & *rhs*) [inline]

6.100.1.3 template<class T> PP_ListIterator< T >::PP_ListIterator (const PP_List< T > & *_list*, PP_ListLink< T > * *_p*) [inline, protected]

6.100.2 Member Function Documentation

6.100.2.1 template<class T> const T & PP_ListIterator< T >::operator * () const [inline]

6.100.2.2 template<class T> PP_ListIterator< T >::operator void * () [inline]

6.100.2.3 template<class T> bool PP_ListIterator< T >::operator! () const [inline]

6.100.2.4 template<class T> bool PP_ListIterator< T >::operator!= (const PP_ListIterator< T > &) const [inline]

6.100.2.5 template<class T> const T & PP_ListIterator< T >::operator() () const [inline]

6.100.2.6 template<class T> PP_ListIterator< T > PP_ListIterator< T >::operator++ (int) [inline]

6.100.2.7 template<class T> PP_ListIterator< T > & PP_ListIterator< T >::operator++ () [inline]

6.100.2.8 template<class T> PP_ListIterator< T > PP_ListIterator< T >::operator– (int) [inline]

6.100.2.9 template<class T> PP_ListIterator< T > & PP_ListIterator< T >::operator– () [inline]

6.100.2.10 template<class T> bool PP_ListIterator< T >::operator== (const PP_ListIterator< T > &) const [inline]

6.100.2.11 template<class T> void PP_ListIterator< T >::rewind () [inline]

6.100.2.12 template<class T> const T & PP_ListIterator< T >::value () const [inline]

6.100.3 Friends And Related Function Documentation

6.100.3.1 template<class T> friend class PP_List< T > [friend]

6.100.4 Member Data Documentation

6.100.4.1 template<class T> const PP_List<T>& PP_ListIterator< T >::list [protected]

6.100.4.2 template<class T> PP_ListLink<T>* PP_ListIterator< T >::p [protected]

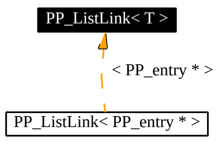
The documentation for this class was generated from the following file:

- **ParmParse.H**

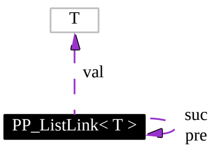
6.101 PP_ListLink< T > Class Template Reference

`#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 >

6.101.1 Friends And Related Function Documentation

6.101.1.1 template<class T> friend class PP_List< T > `[friend]`

6.101.1.2 template<class T> friend class PP_ListIterator< T > `[friend]`

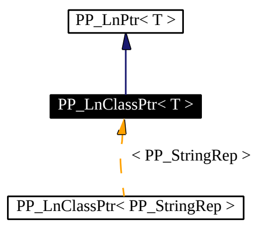
The documentation for this class was generated from the following file:

- **ParmParse.H**

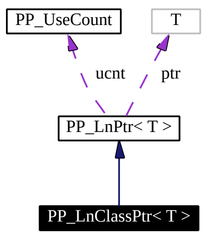
6.102 PP_LnClassPtr< T > Class Template Reference

`#include <ParmParse.H>`

Inheritance diagram for PP_LnClassPtr< T >:



Collaboration diagram for PP_LnClassPtr< T >:



Public Methods

- **PP_LnClassPtr** ()
- **PP_LnClassPtr** (T *rhs)
- PP_LnClassPtr< T > & **operator=** (const PP_LnClassPtr< T > &rhs)
- PP_LnClassPtr< T > & **operator=** (T *rhs)
- T * **operator** → () const

template<class T> class PP_LnClassPtr< T >

6.102.1 Constructor & Destructor Documentation

6.102.1.1 template<class T> PP_LnClassPtr< T >::PP_LnClassPtr () `[inline]`

6.102.1.2 template<class T> PP_LnClassPtr< T >::PP_LnClassPtr (T * *rhs*) `[inline]`

6.102.2 Member Function Documentation

6.102.2.1 template<class T> T * PP_LnClassPtr< T >::operator → () const `[inline]`

6.102.2.2 template<class T> PP_LnClassPtr< T > & PP_LnClassPtr< T >::operator= (T * *rhs*) `[inline]`

Reimplemented from **PP_LnPtr< T >** (p. 411).

6.102.2.3 template<class T> PP_LnClassPtr< T > & PP_LnClassPtr< T >::operator= (const PP_LnClassPtr< T > & *rhs*) `[inline]`

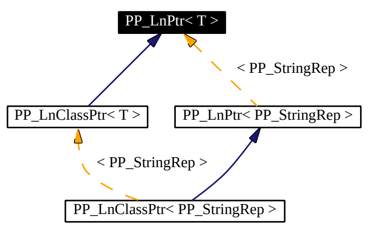
The documentation for this class was generated from the following file:

- **ParmParse.H**

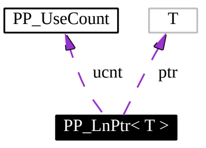
6.103 PP_LnPtr< T > Class Template Reference

`#include <ParmParse.H>`

Inheritance diagram for PP_LnPtr< T >:



Collaboration diagram for PP_LnPtr< T >:



Public Methods

- **PP_LnPtr** ()
- **PP_LnPtr** (T *rhs)
- PP_LnPtr< T > & **operator=** (const PP_LnPtr< T > &rhs)
- PP_LnPtr< T > & **operator=** (T *rhs)
- ~**PP_LnPtr** ()
- bool **unique** () const
- int **linkCount** () const
- T & **operator** * () const
- bool **isNull** () const
- bool **operator==** (const PP_LnPtr< T > &rhs) const
- bool **operator!=** (const PP_LnPtr< T > &rhs) const

Protected Attributes

- T * **ptr**

template<class T> class PP_LnPtr< T >

6.103.1 Constructor & Destructor Documentation

6.103.1.1 template<class T> PP_LnPtr< T >::PP_LnPtr () [inline]

6.103.1.2 template<class T> PP_LnPtr< T >::PP_LnPtr (T * *rhs*) [inline]

6.103.1.3 template<class T> PP_LnPtr< T >::~PP_LnPtr () [inline]

6.103.2 Member Function Documentation

6.103.2.1 template<class T> bool PP_LnPtr< T >::isNull () const [inline]

6.103.2.2 template<class T> int PP_LnPtr< T >::linkCount () const [inline]

6.103.2.3 template<class T> T & PP_LnPtr< T >::operator * () const [inline]

6.103.2.4 template<class T> bool PP_LnPtr< T >::operator!= (const PP_LnPtr< T > & *rhs*) const [inline]

6.103.2.5 template<class T> PP_LnPtr<T>& PP_LnPtr< T >::operator= (T * *rhs*) [inline]

Reimplemented in **PP_LnClassPtr< T >** (p. 409), and **PP_LnClassPtr< PP_StringRep >** (p. 409).

6.103.2.6 template<class T> PP_LnPtr< T > & PP_LnPtr< T >::operator= (const PP_LnPtr< T > & *rhs*) [inline]

6.103.2.7 template<class T> bool PP_LnPtr< T >::operator== (const PP_LnPtr< T > & *rhs*) const [inline]

6.103.2.8 template<class T> bool PP_LnPtr< T >::unique () const [inline]

6.103.3 Member Data Documentation

6.103.3.1 template<class T> T* PP_LnPtr< T >::ptr [protected]

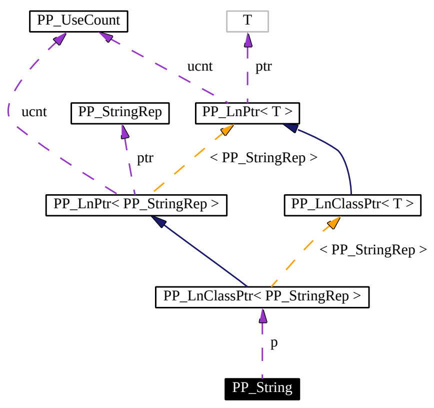
The documentation for this class was generated from the following file:

- **ParmParse.H**

6.104 PP_String Class Reference

`#include <ParmParse.H>`

Collaboration diagram for PP_String:



Public Methods

- **PP_String** ()
- **PP_String** (char c)
- **PP_String** (int len)
- **PP_String** (const char ∗s)
- **PP_String** (const PP_String &rhs)
- PP_String & **operator=** (const PP_String &rhs)
- 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
- char & **operator**[ ] (int k)
- char **operator**[ ] (int k) const
- const char ∗ **c_str** () const
- double **toDouble** () const
- int **toInteger** () const
- long **toLong** () const

Protected Methods

- void **copyModify** ()

Friends

- std::ostream & **operator<<** (std::ostream &os, const PP_String &str)
- 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)

6.104.1 Constructor & Destructor Documentation

6.104.1.1 PP_String::PP_String ()

6.104.1.2 PP_String::PP_String (char *c*)

6.104.1.3 PP_String::PP_String (int *len*)

6.104.1.4 PP_String::PP_String (const char ∗ *s*)

6.104.1.5 PP_String::PP_String (const PP_String & *rhs*)

6.104.2 Member Function Documentation

6.104.2.1 const char ∗ PP_String::c_str () const `[inline]`

6.104.2.2 void PP_String::copyModify () `[protected]`

6.104.2.3 std::istream& PP_String::getline (std::istream & *strm*)

6.104.2.4 bool PP_String::isNull () const `[inline]`

6.104.2.5 int PP_String::length () const `[inline]`

6.104.2.6 PP_String& PP_String::operator+= (char *c*)

6.104.2.7 PP_String& PP_String::operator+= (const char ∗ *right*)

6.104.2.8 PP_String& PP_String::operator+= (const PP_String & *right*)

6.104.2.9 PP_String& PP_String::operator= (const PP_String & *rhs*)

6.104.2.10 char PP_String::operator[] (int *k*) const `[inline]`

6.104.2.11 char& PP_String::operator[] (int *k*)

6.104.2.12 double PP_String::toDouble () const `[inline]`

6.104.2.13 int PP_String::toInteger () const `[inline]`

6.104.2.14 long PP_String::toLong () const `[inline]`

6.104.2.15 PP_String& PP_String::toLower ()

6.104.2.16 PP_String& PP_String::toUpper ()

6.104.3 Friends And Related Function Documentation

6.104.3.1 bool operator!= (const char ∗ *left*, const PP_String & *right*) `[friend]`

6.104.3.2 bool operator!= (const PP_String & *left*, const char ∗ *right*) `[friend]`

6.104.3.3 bool operator!= (const PP_String & *left*, const PP_String & *right*) `[friend]`

6.104.3.4 bool operator< (const char ∗ *left*, const PP_String & *right*) `[friend]`

6.104.3.5 bool operator< (const PP_String & *left*, const char ∗ *right*) `[friend]`

6.104.3.6 bool operator< (const PP_String & *left*, const PP_String & *right*) `[friend]`

6.104.3.7 std::ostream& operator<< (std::ostream & *os*, const PP_String & *str*) `[friend]`

6.104.3.8 bool operator<= (const char ∗ *left*, const PP_String & *right*) `[friend]`

6.104.3.9 bool operator<= (const PP_String & *left*, const char ∗ *right*) `[friend]`

6.104.3.10 bool operator<= (const PP_String & *left*, const PP_String & *right*) `[friend]`

6.104.3.11 bool operator== (const char ∗ *left*, const PP_String & *right*) `[friend]`

6.104.3.12 bool operator== (const PP_String & *left*, const char ∗ *right*) `[friend]`

6.104.3.13 bool operator== (const PP_String & *left*, const PP_String & *right*) `[friend]`

6.104.3.14 bool operator> (const char ∗ *left*, const PP_String & *right*) `[friend]`

6.104.3.15 bool operator> (const PP_String & *left*, const char ∗ *right*) `[friend]`

6.104.3.16 bool operator> (const PP_String & *left*, const PP_String & *right*) `[friend]`

6.104.3.17 bool operator>= (const char ∗ *left*, const PP_String & *right*) `[friend]`

6.104.3.18 bool operator>= (const PP_String & *left*, const char ∗ *right*) `[friend]`

6.104.3.19 bool operator>= (const PP_String & *left*, const PP_String & *right*) `[friend]`

6.104.3.20 std::istream& operator>> (std::istream & *is*, PP_String & *str*) `[friend]`

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

- **ParmParse.H**

6.105 PP_StringRep Class Reference

`#include <ParmParse.H>`

Public Methods

- **PP_StringRep** (int _len=0)
- ~**PP_StringRep** ()
- void **resize** (int n)

Friends

- class **PP_String**

6.105.1 Constructor & Destructor Documentation

6.105.1.1 PP_StringRep::PP_StringRep (int *_len* = 0) [inline]

6.105.1.2 PP_StringRep::~PP_StringRep () [inline]

6.105.2 Member Function Documentation

6.105.2.1 void PP_StringRep::resize (int *n*)

6.105.3 Friends And Related Function Documentation

6.105.3.1 friend class PP_String [friend]

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

- **ParmParse.H**

6.106 PP_UseCount Class Reference

`#include <ParmParse.H>`

Public Methods

- **PP_UseCount** ()
- **PP_UseCount** (const PP_UseCount &rhs)
- PP_UseCount & **operator=** (const PP_UseCount &rhs)
- ~**PP_UseCount** ()
- bool **unique** () const
- int **linkCount** () const

6.106.1 Constructor & Destructor Documentation

6.106.1.1 PP_UseCount::PP_UseCount () `[inline]`

6.106.1.2 PP_UseCount::PP_UseCount (const PP_UseCount & *rhs*) `[inline]`

6.106.1.3 PP_UseCount::~PP_UseCount () `[inline]`

6.106.2 Member Function Documentation

6.106.2.1 int PP_UseCount::linkCount () const `[inline]`

6.106.2.2 PP_UseCount & PP_UseCount::operator= (const PP_UseCount & *rhs*) `[inline]`

6.106.2.3 bool PP_UseCount::unique () const `[inline]`

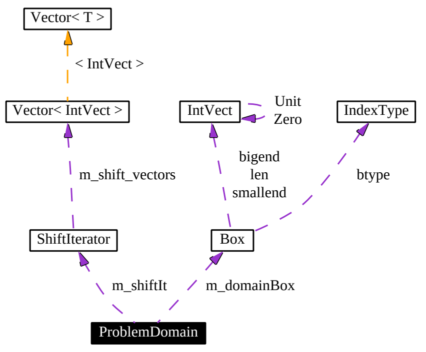
The documentation for this class was generated from the following file:

- **ParmParse.H**

6.107 ProblemDomain Class Reference

`#include <ProblemDomain.H>`

Collaboration diagram for ProblemDomain:



Public Methods

- **ProblemDomain** ()

  {\bf *Constructors*}

- **ProblemDomain** (const **Box** &a_domBox)
- **ProblemDomain** (const **Box** &a_domBox, const bool *a_isPeriodic)
- **ProblemDomain** (const **IntVect** &small, const **IntVect** &big)
- **ProblemDomain** (const **IntVect** &small, const **IntVect** &big, const bool *a_isPeriodic)
- **ProblemDomain** (const **IntVect** &small, const int *vec_len)
- **ProblemDomain** (const **IntVect** &small, const int *vec_len, const bool *a_isPeriodic)
- **ProblemDomain** (const ProblemDomain &a_src)
- const **Box** & **domainBox** () const

  {\bf *Accessors*}

- bool **isPeriodic** (int a_dir) const
- bool **isPeriodic** () const
- **ShiftIterator shiftIterator** () const
- bool **isEmpty** () const
- bool **contains** (const **IntVect** &p) const
- bool **image** (**IntVect** &p) const
- bool **contains** (const **Box** &b) const
- bool **contains_box** (const **Box** &b) const
- bool **intersects** (const **Box** &a_box) const
- bool **intersectsNotEmpty** (const **Box** &a_box) const
- bool **intersects** (const **Box** &box1, const **Box** &box2) const
- ProblemDomain & **operator=** (const ProblemDomain &b)

  {\bf *Modification Functions*}

- void **setPeriodic** (int a_dir, bool a_isPeriodic)
- ProblemDomain & **grow** (int i)
- ProblemDomain & **grow** (const **IntVect** &v)

- ProblemDomain & **grow** (int idir, int n_cell)
- ProblemDomain & **growLo** (int idir, int n_cell=1)
- ProblemDomain & **growHi** (int idir, int n_cell=1)
- **Box operator &** (const **Box** &a_b) const
- ProblemDomain & **refine** (int a_refinement_ratio)

  refinement

- ProblemDomain & **refine** (const **IntVect** &a_refinement_ratio)
- ProblemDomain & **coarsen** (int a_refinement_ratio)
- ProblemDomain & **coarsen** (const **IntVect** &refinement_ratio)
- void **dumpOn** (std::ostream &strm) const

Protected Attributes

- bool **m_isPeriodic** [**SpaceDim**]
- **Box m_domainBox**
- **ShiftIterator m_shiftIt**

Friends

- class **HDF5Handle**
- void **operator &=** (**Box** &a_box, const ProblemDomain &a_probomain)
- **Box operator &** (const **Box** &a_box, const ProblemDomain &a_probdomain)
- ProblemDomain **grow** (const ProblemDomain &pd, int i)
- ProblemDomain **grow** (const ProblemDomain &pd, const **IntVect** &v)
- **Box bdryLo** (const ProblemDomain &a_pd, int a_dir, int a_len=1)
- **Box bdryHi** (const ProblemDomain &a_pd, int a_dir, int a_len=1)
- **Box adjCellLo** (const ProblemDomain &a_pd, int a_dir, int a_len=1)
- **Box adjCellHi** (const ProblemDomain &a_pd, int a_dir, int a_len=1)
- ProblemDomain **refine** (const ProblemDomain &a_probdomain, int a_refinement_ratio)
- ProblemDomain **refine** (const ProblemDomain &a_probdomain, const **IntVect** &a_-refinement_ratio)
- ProblemDomain **coarsen** (const ProblemDomain &a_probdomain, int a_refinement_ratio)
- ProblemDomain **coarsen** (const ProblemDomain &a_probdomain, const **IntVect** &a_-refinement_ratio)
- std::ostream & **operator**<< (std::ostream &os, const ProblemDomain &bx)
- std::istream & **operator**>> (std::istream &is, ProblemDomain &bx)

6.107.1 Constructor & Destructor Documentation

6.107.1.1 ProblemDomain::ProblemDomain () `[inline]`

{\bf Constructors}

The default constructor. The constructed domain box is empty.

6.107.1.2 ProblemDomain::ProblemDomain (const Box & *a_domBox*)

Construct ProblemDomain with a_domBox as computational domain This constructor defaults to non-periodic domain

6.107.1.3 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, true is the physical boundary condition is periodic in the coordinate direction

6.107.1.4 ProblemDomain::ProblemDomain (const IntVect & *small*, const IntVect & *big*)

Construct a ProblemDomain. It is an error if small is greater than big.

6.107.1.5 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, true is the physical boundary condition is periodic in the coordinate direction

6.107.1.6 ProblemDomain::ProblemDomain (const IntVect & *small*, const int * *vec_len*)

Construct ProblemDomain with specified lengths. It is an error if the lengths are negative.

6.107.1.7 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, true is the physical boundary condition is periodic in the coordinate direction

6.107.1.8 ProblemDomain::ProblemDomain (const ProblemDomain & *a_src*) `[inline]`

The copy constructor.

6.107.2 Member Function Documentation

6.107.2.1 ProblemDomain& ProblemDomain::coarsen (const IntVect & *refinement_ratio*)

Modifies this ProblemDomain by coarsening by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function. \

6.107.2.2 ProblemDomain& ProblemDomain::coarsen (int *a_refinement_ratio*)

Modifies this ProblemDomain by coarsening it by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function. \

6.107.2.3 bool ProblemDomain::contains (const Box & *b*) const [inline]

Returns true if argument is contained within this ProblemDomain. It is an error if the **Box** (p. 106) is not cell-centered. An empty ProblemDomain does not contain any **Box** (p. 106).

6.107.2.4 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.

6.107.2.5 bool ProblemDomain::contains_box (const Box & *b*) const [inline]

6.107.2.6 const Box & ProblemDomain::domainBox () const [inline]

{\bf Accessors}

Returns the logical computational domain

6.107.2.7 void ProblemDomain::dumpOn (std::ostream & *strm*) const

Gives more detail than printOn. Useful for exiting due to an error.

6.107.2.8 ProblemDomain & ProblemDomain::grow (int *idir*, int *n_cell*) [inline]

Modifies this ProblemDomain by growing it on the low and high end by n_cell cells in direction idir

6.107.2.9 ProblemDomain & ProblemDomain::grow (const IntVect & *v*) [inline]

modifies this ProblemDomain by growing the domainBox in each direction by the specified amount

6.107.2.10 ProblemDomain & ProblemDomain::grow (int *i*) [inline]

grows (or shrinks) the domain box by i in all directions

6.107.2.11 ProblemDomain & ProblemDomain::growHi (int *idir*, int *n_cell* = 1) [inline]

Modifies this ProblemDomain by growing it on the high end by n_Cell cells in direction idir

6.107.2.12 ProblemDomain & ProblemDomain::growLo (int *idir*, int *n_cell* = 1) [inline]

Modifies this ProblemDomain by growing it on the low end by n_cell cells in direction idir.

6.107.2.13 bool ProblemDomain::image (IntVect & *p*) const [inline]

Returns the periodic image of this **IntVect** (p. 264) inside of the ProblemDomain. Return true if the domain contains this **IntVect** (p. 264), returning the image in 'p', otherwise returns false

6.107.2.14 bool ProblemDomain::intersects (const Box & *box1*, const Box & *box2*) const

returns true if box1 and box2 and any of their periodic images intersect (useful for checking disjointness)

6.107.2.15 bool ProblemDomain::intersects (const Box & *a_box*) const

Returns true if this ProblemDomain and the argument have non-null intersections. It is an error if a_box is not cell-centered. An empty ProblemDomain does not intersect any **Box** (p. 106). 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.

6.107.2.16 bool ProblemDomain::intersectsNotEmpty (const Box & *a_box*) const

Returns true if this ProblemDomain and the argument have non-null intersections. 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.

6.107.2.17 bool ProblemDomain::isEmpty () const [inline]

Returns true if this ProblemDomain is empty or undefined.

6.107.2.18 bool ProblemDomain::isPeriodic () const [inline]

returns true is BC is periodic in _any_ direction

6.107.2.19 bool ProblemDomain::isPeriodic (int *a_dir*) const [inline]

returns true if BC is periodic in direction a_dir

6.107.2.20 Box ProblemDomain::operator & (const Box & *a_b*) const

Returns the **Box** (p. 106) that is intersection of this ProblemDomain and the argument Problem-Domain. The **Box** (p. 106) MUST be cell-centered. The intersection of the Empty ProblemDomain and any **Box** (p. 106) is the Empty **Box** (p. 106). This operator does nothing in periodic directions (since a periodic domain is an infinite domain).

6.107.2.21 ProblemDomain & ProblemDomain::operator= (const ProblemDomain & *b*) [inline]

{\bf Modification Functions}

The assignment operator.

6.107.2.22 ProblemDomain& ProblemDomain::refine (const IntVect & *a_refinement_ratio*)

Modifies this ProblemDomain by refining it by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function. \

6.107.2.23 ProblemDomain& ProblemDomain::refine (int *a_refinement_ratio*)

refinement

Modifies this ProblemDomain by refining it by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function. \

6.107.2.24 void ProblemDomain::setPeriodic (int *a_dir*, bool *a_isPeriodic*)

sets whether BC is periodic in direction a_dir (true == periodic)

6.107.2.25 ShiftIterator ProblemDomain::shiftIterator () const `[inline]`

returns the shiftIterator for this ProblemDomain

6.107.3 Friends And Related Function Documentation

6.107.3.1 Box adjCellHi (const ProblemDomain & *a_pd*, int *a_dir*, int *a_len* = 1) `[friend]`

Returns the cell centered **Box** (p. 106) of the given length adjacent to the argument ProblemDomain on the high end along the given coordinate direction. The return **Box** (p. 106) is identical to the argument ProblemDomain in the other directions. The return **Box** (p. 106) and the argument ProblemDomain have an empty intersection. \

NOTE: len >= 1. \

NOTE: **Box** (p. 106) retval = adjCellHi(b,dir,len) is equivalent to the following set of operations: \

Box (p. 106) 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** (p. 106) of the appropriate type. If dir is a periodic direction, will return an empty **Box** (p. 106).

6.107.3.2 Box adjCellLo (const ProblemDomain & *a_pd*, int *a_dir*, int *a_len* = 1) `[friend]`

Returns the cell centered **Box** (p. 106) of the given length adjacent to the argument ProblemDomain on the low end along the given coordinate direction. The return **Box** (p. 106) is identical to the argument ProblemDomain in the other directions. The return ProblemDomain and the argument ProblemDomain have an empty intersection.\

NOTE: len >= 1. \

NOTE: **Box** (p. 106) retval = adjCellLo(b,dir,len) is equivalent to the following set of operations: \

Box (p. 106) 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** (p. 106) of the appropriate type. If dir is a periodic direction, will return an empty **Box** (p. 106) as well.

6.107.3.3 Box bdryHi (const ProblemDomain & *a_pd*, int *a_dir*, int *a_len* = 1) `[friend]`

Returns the edge-centered **Box** (p. 106) (in direction dir) defining the high side of the argument ProblemDomain. The return **Box** (p. 106) 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** (p. 106) of the appropriate type. If dir is a periodic direction, will return an empty box.

6.107.3.4 Box bdryLo (const ProblemDomain & *a_pd*, int *a_dir*, int *a_len* = 1) `[friend]`

Returns the edge-centered **Box** (p. 106) (in direction dir) defining the low side of the argument ProblemDomain. The output **Box** (p. 106) 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** (p. 106) of the appropriate type. If dir is a periodic direction, will return an empty box.

6.107.3.5 ProblemDomain coarsen (const ProblemDomain & *a_probdomain*, const IntVect & *a_refinement_ratio*) `[friend]`

Returns a ProblemDomain that is the argument ProblemDomain coarsened by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function. \

6.107.3.6 ProblemDomain coarsen (const ProblemDomain & *a_probdomain*, int *a_refinement_ratio*) `[friend]`

Returns a ProblemDomain that is the argument ProblemDomain coarsened by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function. \

6.107.3.7 ProblemDomain grow (const ProblemDomain & *pd*, const IntVect & *v*) `[friend]`

Returns a ProblemDomain that is the argument ProblemDomain with a DomainBox grown by the given amount.

6.107.3.8 ProblemDomain grow (const ProblemDomain & *pd*, int *i*) `[friend]`

returns a ProblemDomain with a domainBox grown by the given amount

6.107.3.9 friend class HDF5Handle `[friend]`

6.107.3.10 Box operator & (const Box & *a_box*, const ProblemDomain & *a_probdomain*) `[friend]`

returns a box which is the interesection of a_box and the problemDomain

6.107.3.11 void operator &= (Box & *a_box*, const ProblemDomain & *a_probomain*) `[friend]`

modifies a_box to be the intersection of a_box and the problemDomain

6.107.3.12 std::ostream& operator<< (std::ostream & *os*, const ProblemDomain & *bx*) `[friend]`

Write an ASCII representation to the ostream.

6.107.3.13 std::istream& operator>> (std::istream & *is*, ProblemDomain & *bx*) `[friend]`

Read from istream.

6.107.3.14 ProblemDomain refine (const ProblemDomain & *a_probdomain*, const IntVect & *a_refinement_ratio*) `[friend]`

Returns a ProblemDomain that is the argument ProblemDomain refined by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function. \

6.107.3.15 ProblemDomain refine (const ProblemDomain & *a_probdomain*, int *a_refinement_ratio*) `[friend]`

Returns a ProblemDomain that is the argument ProblemDomain refined by given (positive) refinement ratio. The Empty ProblemDomain is not modified by this function. \

6.107.4 Member Data Documentation

6.107.4.1 Box ProblemDomain::m_domainBox `[protected]`

6.107.4.2 bool ProblemDomain::m_isPeriodic[SpaceDim] `[protected]`

6.107.4.3 ShiftIterator ProblemDomain::m_shiftIt `[protected]`

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

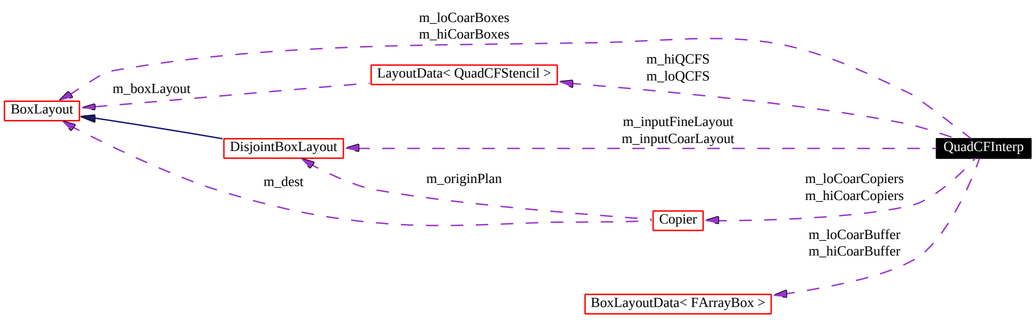
- **ProblemDomain.H**

6.108 QuadCFInterp Class Reference

Quadratic coarse-fine interpolation utility.

`#include <QuadCFInterp.H>`

Collaboration diagram for QuadCFInterp:



Public Methods

- **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 **Box** &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 Methods

- 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?

6.108.1 Detailed Description

Quadratic coarse-fine interpolation utility.

Class to fill ghost cells on coarse-fine interface using quadratic interpolation

6.108.2 Constructor & Destructor Documentation

6.108.2.1 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

6.108.2.2 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

6.108.2.3 QuadCFInterp::QuadCFInterp ()

Default constructor leaves QCFI undefined.

6.108.2.4 QuadCFInterp::∼QuadCFInterp ()

6.108.3 Member Function Documentation

6.108.3.1 void QuadCFInterp::clear ()

return QCFI to undefined state

6.108.3.2 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]`

6.108.3.3 void QuadCFInterp::coarseFineInterp (LevelData< FArrayBox > & *a_phif*, const LevelData< FArrayBox > & *a_phic*)

Coarse / Fine interpolation operator.

6.108.3.4 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

6.108.3.5 void QuadCFInterp::define (const DisjointBoxLayout & *a_fineBoxes*, const DisjointBoxLayout ∗ *a_coarBoxes*, Real *a_dxFine*, int *a_refRatio*, int *a_nComp*, const Box & *a_domf*)

Full define function. makes all coarse-fine information and sets internal variables

6.108.3.6 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]`

6.108.3.7 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]`

6.108.3.8 bool QuadCFInterp::isDefined () const

has full define function been called? return true if so

6.108.4 Member Data Documentation

6.108.4.1 Real QuadCFInterp::m_dxFine [protected]

6.108.4.2 BoxLayout QuadCFInterp::m_hiCoarBoxes[SpaceDim] [protected]

6.108.4.3 BoxLayoutData<FArrayBox> QuadCFInterp::m_hiCoarBuffer[SpaceDim] [protected]

6.108.4.4 Copier QuadCFInterp::m_hiCoarCopiers[SpaceDim] [protected]

6.108.4.5 LayoutData<QuadCFStencil> QuadCFInterp::m_hiQCFS[SpaceDim] [protected]

6.108.4.6 DisjointBoxLayout QuadCFInterp::m_inputCoarLayout [protected]

6.108.4.7 DisjointBoxLayout QuadCFInterp::m_inputFineLayout [protected]

6.108.4.8 bool QuadCFInterp::m_isDefined [protected]

has full define function been called?

6.108.4.9 int QuadCFInterp::m_level [protected]

6.108.4.10 BoxLayout QuadCFInterp::m_loCoarBoxes[SpaceDim] [protected]

6.108.4.11 BoxLayoutData<FArrayBox> QuadCFInterp::m_loCoarBuffer[SpaceDim] [protected]

6.108.4.12 Copier QuadCFInterp::m_loCoarCopiers[SpaceDim] [protected]

6.108.4.13 LayoutData<QuadCFStencil> QuadCFInterp::m_loQCFS[SpaceDim] [protected]

6.108.4.14 int QuadCFInterp::m_nComp [protected]

6.108.4.15 int QuadCFInterp::m_refRatio [protected]

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

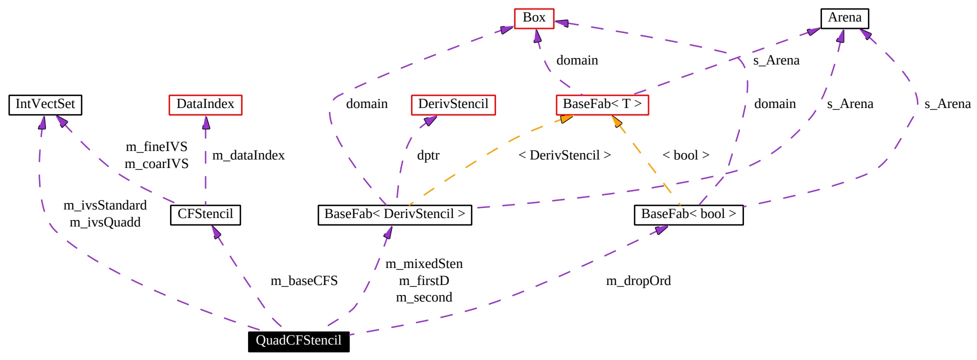
- **QuadCFInterp.H**

6.109 QuadCFStencil Class Reference

class to encapsulate CF info for quadratic interpolation

`#include <QuadCFStencil.H>`

Collaboration diagram for QuadCFStencil:



Public Methods

- **Real computeSecondDerivative** (const **BaseFab**< **Real** > &a_phic, int a_derivdir, int a_ivar, const **IntVect** &a_ivin, **Real** a_dx) const

  {\bf *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

  {\bf *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**

6.109.1 Detailed Description

class to encapsulate CF info for quadratic interpolation

QuadCFStencil is a wrapper around a bunch of things that allow **LevelOp** (p. 315) to do coarse-fine interpolation on a particular face of a grid.

6.109.2 Constructor & Destructor Documentation

6.109.2.1 QuadCFStencil::QuadCFStencil ()

6.109.2.2 QuadCFStencil::~QuadCFStencil ()

6.109.2.3 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*)

6.109.2.4 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*)

6.109.3 Member Function Documentation

6.109.3.1 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

6.109.3.2 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

6.109.3.3 Real QuadCFStencil::computeSecondDerivative (const BaseFab< Real > & *a_phic*, int *a_derivdir*, int *a_ivar*, const IntVect & *a_ivin*, Real *a_dx*) const

{\bf 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

6.109.3.4 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*)

6.109.3.5 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*)

6.109.3.6 const IntVectSet& QuadCFStencil::getCoarIVS () const `[inline]`

get coarse intvects that underly fiinterpivs. This will be empty if **isEmpty**() (p. 432) returns true

6.109.3.7 const IntVectSet& QuadCFStencil::getFineIVS () const `[inline]`

get fine intvects which need to be interpolated \ This will be empty if **isEmpty**() (p. 432) returns true

6.109.3.8 bool QuadCFStencil::isDefined () const

{\bf Access functions}

has full define function been called? return true if so

6.109.3.9 bool QuadCFStencil::isEmpty () const `[inline]`

are there any interpolation points? \ returns false if so.

6.109.4 Member Data Documentation

6.109.4.1 CFStencil QuadCFStencil::m_baseCFS `[protected]`

6.109.4.2 int QuadCFStencil::m_direction `[protected]`

6.109.4.3 BaseFab<bool> QuadCFStencil::m_dropOrd `[protected]`

whether to drop order at any one-sided point

6.109.4.4 BaseFab<DerivStencil> QuadCFStencil::m_firstD[SpaceDim] [protected]

stencil information for unmixed derivatives (first derivatives)

6.109.4.5 bool QuadCFStencil::m_isDefined [protected]

6.109.4.6 IntVectSet QuadCFStencil::m_ivsQuadd [protected]

set of all points which have at least one derivative shifted

6.109.4.7 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.

6.109.4.8 BaseFab<DerivStencil> QuadCFStencil::m_mixedSten [protected]

stencil information for mixed derivatives undefined if SpaceDim != 3

6.109.4.9 BaseFab<DerivStencil> QuadCFStencil::m_second[SpaceDim] [protected]

stencil information for unmixed derivatives (second derivatives)

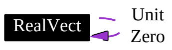
The documentation for this class was generated from the following file:

- **QuadCFStencil.H**

6.110 RealVect Class Reference

```
#include <RealVect.H>
```

Collaboration diagram for RealVect:



Public Methods

- **RealVect** ()

    {\bf Constructors and Accessors}

- **RealVect** (D_DECL(**Real** i, **Real** j, **Real** k))
- **RealVect** (const RealVect &rhs)
- RealVect & **operator=** (const RealVect &rhs)
- **Real** & **operator**[ ] (int i)
- const **Real** & **operator**[ ] (int i) const
- bool **operator==** (const RealVect &p) const

    {\bf Comparison Operators}

- bool **operator!=** (const RealVect &p) const
- RealVect & **operator+=** (**Real** s)
- RealVect & **operator+=** (const RealVect &p)
- 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 *** (const RealVect &p) const
- 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
- const **Real** * **dataPtr** () const

Static Public Attributes

- const RealVect **Zero**
- const RealVect **Unit**
- size_t **io_offset**

Friends

- class **HDF5Handle**
- RealVect **BASISREALV** (int dir)
- 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)

6.110.1 Constructor & Destructor Documentation

6.110.1.1 RealVect::RealVect ()

{\bf Constructors and Accessors}

Construct an RealVect whose components are zero.

6.110.1.2 RealVect::RealVect (D_DECL(Real i, Real j, Real k))

Construct an RealVect given the specific values for its coordinates. D_DECL is a macro that sets the constructor to take CH_SPACEDIM arguments.

6.110.1.3 RealVect::RealVect (const RealVect & *rhs*)

The copy constructor.

6.110.2 Member Function Documentation

6.110.2.1 const Real* RealVect::dataPtr () const

only for sending stuff to fortran

6.110.2.2 RealVect RealVect::operator * (Real *s*) const

Returns an RealVect that is this RealVect with each component multiplied by a scalar.

6.110.2.3 RealVect RealVect::operator * (const RealVect & *p*) const

Returns component-wise product of this RealVect and argument.

6.110.2.4 RealVect& RealVect::operator *= (const RealVect & *p*)

Modifies this RealVect by component-wise multiplication with argument.

6.110.2.5 RealVect& RealVect::operator *= (Real *s*)

Modifies this RealVect by multiplication of a scalar to each component.

6.110.2.6 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.

6.110.2.7 RealVect& RealVect::operator+= (const RealVect & *p*)

Modifies this RealVect by component-wise addition with argument.

6.110.2.8 RealVect& RealVect::operator+= (Real *s*)

Modifies this RealVect by addition of a scalar to each component.

6.110.2.9 RealVect RealVect::operator- (Real *s*) const

Returns an RealVect that is this RealVect with a scalar s subtracted from each component.

6.110.2.10 RealVect RealVect::operator- (const RealVect & *p*) const

Returns component-wise difference of this RealVect and argument.

6.110.2.11 RealVect& RealVect::operator-= (const RealVect & *p*)

Modifies this RealVect by component-wise subtraction with argument.

6.110.2.12 RealVect& RealVect::operator-= (Real *s*)

Modifies this RealVect by subtraction of a scalar from each component.

6.110.2.13 RealVect RealVect::operator/ (Real *s*) const

Returns component-wise division of this RealVect by scalar s.

6.110.2.14 RealVect RealVect::operator/ (const RealVect & *p*) const

Returns component-wise division of this RealVect by RealVect argument.

6.110.2.15 RealVect& RealVect::operator/= (const RealVect & *p*)

Modifies this RealVect by component-wise division with RealVect argument.

6.110.2.16 RealVect& RealVect::operator/= (Real *s*)

Modifies this RealVect by division by a scalar of each component.

6.110.2.17 RealVect& RealVect::operator= (const RealVect & *rhs*)

The assignment operator.

6.110.2.18 bool RealVect::operator== (const RealVect & *p*) const

{\bf Comparison Operators}

Returns true if this RealVect is equivalent to argument RealVect. All comparisons between analogous components must be satisfied.

6.110.2.19 const Real& RealVect::operator[] (int *i*) const

Returns the i'th coordinate of the RealVect.

6.110.2.20 Real& RealVect::operator[] (int *i*)

Returns a modifiable lvalue reference to the i'th coordinate of the RealVect.

6.110.3 Friends And Related Function Documentation

6.110.3.1 RealVect BASISREALV (int *dir*) `[friend]`

Returns a basis vector in the given coordinate direction; eg. RealVect3d BASISV(1) == (0,1,0). Note that the coordinate directions are zero based.

6.110.3.2 friend class HDF5Handle `[friend]`

6.110.3.3 RealVect operator * (const RealVect & *s*, const RealVect & *p*) `[friend]`

Returns and RealVect that is an RealVect p with each component multiplied by a scalar s.

6.110.3.4 RealVect operator * (Real *s*, const RealVect & *p*) `[friend]`

Returns and RealVect that is an RealVect p with each component multiplied by a scalar s.

6.110.3.5 RealVect operator+ (const RealVect & *s*, const RealVect & *p*) `[friend]`

Returns an RealVect that is an RealVect p with a scalar s added to each component.

6.110.3.6 RealVect operator+ (Real *s*, const RealVect & *p*) `[friend]`

Returns an RealVect that is an RealVect p with a scalar s added to each component.

6.110.3.7 RealVect operator- (const RealVect & *s*, const RealVect & *p*) [friend]

Returns -p + s.

6.110.3.8 RealVect operator- (Real *s*, const RealVect & *p*) [friend]

Returns -p + s.

6.110.3.9 RealVect operator/ (const RealVect & *s*, const RealVect & *p*) [friend]

Returns and RealVect that is an RealVect p with each component divided by a scalar s.

6.110.3.10 RealVect operator/ (Real *s*, const RealVect & *p*) [friend]

Returns and RealVect that is an RealVect p with each component divided by a scalar s.

6.110.4 Member Data Documentation

6.110.4.1 size_t RealVect::io_offset [static]

6.110.4.2 const RealVect RealVect::Unit [static]

This is an RealVect all of whose components are equal to one.

6.110.4.3 const RealVect RealVect::Zero [static]

This is an RealVect all of whose components are equal to zero.

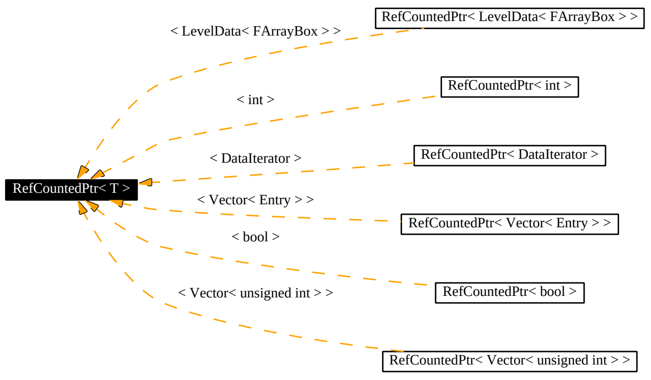
The documentation for this class was generated from the following file:

- **RealVect.H**

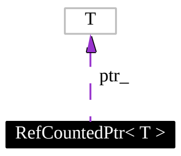
6.111 RefCountedPtr< T > Class Template Reference

`#include <RefCountedPtr.H>`

Inheritance diagram for RefCountedPtr< T >:



Collaboration diagram for RefCountedPtr< T >:



Public Methods

- **RefCountedPtr** (T ∗ptr=0)
- **RefCountedPtr** (const RefCountedPtr< T > &other)
- ∼**RefCountedPtr** ()
- const RefCountedPtr< T > & **operator=** (const RefCountedPtr< T > &rhs)
- T ∗ **operator** → ()
- bool **isNull** () const
- const T ∗ **operator** → () const
- T & **operator** ∗ ()
- const T & **operator** ∗ () const
- **operator const T** ∗ () const
- bool **isNonUnique** () const
- int **refCount** () const

Protected Attributes

- T ∗ **ptr_**
- int ∗ **refCount_**

template<class T> class RefCountedPtr< T >

6.111.1 Constructor & Destructor Documentation

6.111.1.1 template<class T> RefCountedPtr< T >::RefCountedPtr (T * *ptr* = 0) `[inline]`

6.111.1.2 template<class T> RefCountedPtr< T >::RefCountedPtr (const RefCountedPtr< T > & *other*) `[inline]`

6.111.1.3 template<class T> RefCountedPtr< T >::~RefCountedPtr () `[inline]`

6.111.2 Member Function Documentation

6.111.2.1 template<class T> bool RefCountedPtr< T >::isNonUnique () const `[inline]`

6.111.2.2 template<class T> bool RefCountedPtr< T >::isNull () const `[inline]`

6.111.2.3 template<class T> const T & RefCountedPtr< T >::operator * () const `[inline]`

6.111.2.4 template<class T> T & RefCountedPtr< T >::operator * () `[inline]`

6.111.2.5 template<class T> RefCountedPtr< T >::operator const T * () const `[inline]`

6.111.2.6 template<class T> const T * RefCountedPtr< T >::operator → () const `[inline]`

6.111.2.7 template<class T> T * RefCountedPtr< T >::operator → () `[inline]`

6.111.2.8 template<class T> const RefCountedPtr< T > & RefCountedPtr< T >::operator= (const RefCountedPtr< T > & *rhs*) `[inline]`

6.111.2.9 template<class T> int RefCountedPtr< T >::refCount () const `[inline]`

6.111.3 Member Data Documentation

6.111.3.1 template<class T> T* RefCountedPtr< T >::ptr_ `[protected]`

6.111.3.2 template<class T> int* RefCountedPtr< T >::refCount_ `[protected]`

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

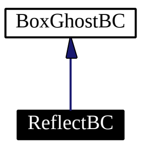
- **RefCountedPtr.H**

6.112 ReflectBC Class Reference

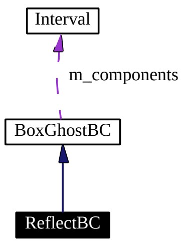
class to do extrapolation BC's

`#include <ReflectBC.H>`

Inheritance diagram for ReflectBC:



Collaboration diagram for ReflectBC:



Public Methods

- **ReflectBC** ()
- ~**ReflectBC** ()
- **ReflectBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- **ReflectBC** (int dir, **Side::LoHiSide** sd)

Protected Methods

- 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

  virtual constructor workaround.

Friends

- class **DomainGhostBC**

6.112.1 Detailed Description

class to do extrapolation BC's

6.112.2 Constructor & Destructor Documentation

6.112.2.1 ReflectBC::ReflectBC ()

6.112.2.2 ReflectBC::~ReflectBC ()

6.112.2.3 ReflectBC::ReflectBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.112.2.4 ReflectBC::ReflectBC (int *dir*, Side::LoHiSide *sd*)

6.112.3 Member Function Documentation

6.112.3.1 virtual void ReflectBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.112.3.2 virtual void ReflectBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.112.3.3 virtual void ReflectBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.112.3.4 virtual void ReflectBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.112.3.5 virtual void ReflectBC::applyReflectBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const [protected, virtual]

6.112.3.6 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. 135).

6.112.3.7 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. 135).

6.112.3.8 BoxGhostBC* ReflectBC::new_boxghostbc () const [protected, virtual]

virtual constructor workaround.

Implements **BoxGhostBC** (p. 135).

6.112.4 Friends And Related Function Documentation

6.112.4.1 friend class DomainGhostBC [friend]

Reimplemented from **BoxGhostBC** (p. 135).

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

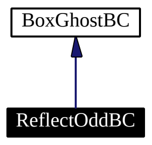
- **ReflectBC.H**

6.113 ReflectOddBC Class Reference

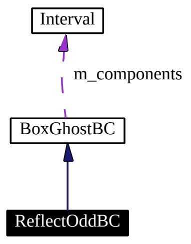
class to do extrapolation BC's

`#include <ReflectOddBC.H>`

Inheritance diagram for ReflectOddBC:



Collaboration diagram for ReflectOddBC:



Public Methods

- **ReflectOddBC** ()
- ~**ReflectOddBC** ()
- **ReflectOddBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- **ReflectOddBC** (int dir, **Side::LoHiSide** sd)

Protected Methods

- 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
  virtual constructor workaround.

Friends

- class **DomainGhostBC**

6.113.1 Detailed Description

class to do extrapolation BC's

6.113.2 Constructor & Destructor Documentation

6.113.2.1 ReflectOddBC::ReflectOddBC ()

6.113.2.2 ReflectOddBC::∼ReflectOddBC ()

6.113.2.3 ReflectOddBC::ReflectOddBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.113.2.4 ReflectOddBC::ReflectOddBC (int *dir*, Side::LoHiSide *sd*)

6.113.3 Member Function Documentation

6.113.3.1 virtual void ReflectOddBC::applyHomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.113.3.2 virtual void ReflectOddBC::applyHomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.113.3.3 virtual void ReflectOddBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.113.3.4 virtual void ReflectOddBC::applyInhomogeneousBCs (FArrayBox & *a_state*, const Box & *a_domain*, Real *a_dx*) const `[protected, virtual]`

Reimplemented from **BoxGhostBC** (p. 134).

6.113.3.5 virtual void ReflectOddBC::applyReflectOddBCs (FArrayBox & *a_state*, const ProblemDomain & *a_domain*, Real *a_dx*) const `[protected, virtual]`

6.113.3.6 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. 135).

6.113.3.7 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. 135).

6.113.3.8 BoxGhostBC* ReflectOddBC::new_boxghostbc () const `[protected, virtual]`

virtual constructor workaround.

Implements **BoxGhostBC** (p. 135).

6.113.4 Friends And Related Function Documentation

6.113.4.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 135).

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

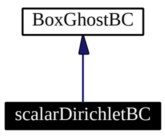
- **ReflectOddBC.H**

6.114 scalarDirichletBC Class Reference

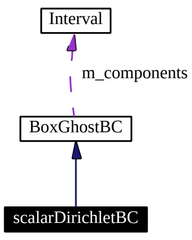
implements dirichlet BCs for scalars

`#include <scalarDirichletBC.H>`

Inheritance diagram for scalarDirichletBC:



Collaboration diagram for scalarDirichletBC:



Public Methods

- **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**

6.114.1 Detailed Description

implements dirichlet BCs for scalars

6.114.2 Constructor & Destructor Documentation

6.114.2.1 scalarDirichletBC::scalarDirichletBC (int *dir*, Side::LoHiSide *sd*)

6.114.2.2 scalarDirichletBC::scalarDirichletBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.114.2.3 virtual scalarDirichletBC::~scalarDirichletBC () `[virtual]`

6.114.2.4 scalarDirichletBC::scalarDirichletBC ()

6.114.3 Member Function Documentation

6.114.3.1 void scalarDirichletBC::setVal (const Real *a_bcVal*)

sets Dirichlet Boundary value

6.114.4 Friends And Related Function Documentation

6.114.4.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 135).

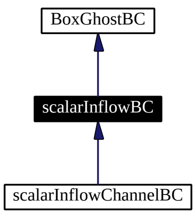
The documentation for this class was generated from the following file:

- **scalarDirichletBC.H**

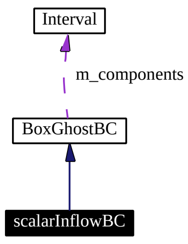
6.115 scalarInflowBC Class Reference

`#include <scalarDirichletBC.H>`

Inheritance diagram for scalarInflowBC:



Collaboration diagram for scalarInflowBC:



Public Methods

- **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**

6.115.1 Constructor & Destructor Documentation

6.115.1.1 scalarInflowBC::scalarInflowBC ()

6.115.1.2 scalarInflowBC::scalarInflowBC (int *dir*, Side::LoHiSide *sd*)

6.115.1.3 scalarInflowBC::scalarInflowBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.115.1.4 virtual scalarInflowBC::~scalarInflowBC () `[virtual]`

6.115.2 Member Function Documentation

6.115.2.1 bool scalarInflowBC::isHomogeneous () const

6.115.2.2 void scalarInflowBC::isHomogeneous (bool *a_isHomogeneous*)

6.115.2.3 int scalarInflowBC::scalarType () const

6.115.2.4 void scalarInflowBC::setScalarType (int *a_scalType*)

set which scalar this is

6.115.3 Friends And Related Function Documentation

6.115.3.1 friend class DomainGhostBC `[friend]`

Reimplemented from **BoxGhostBC** (p. 135).

Reimplemented in **scalarInflowChannelBC** (p. 452).

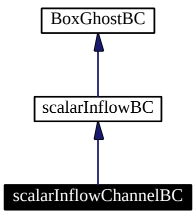
The documentation for this class was generated from the following file:

- **scalarDirichletBC.H**

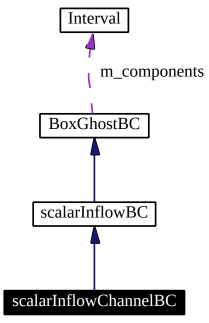
6.116 scalarInflowChannelBC Class Reference

`#include <channelBC.H>`

Inheritance diagram for scalarInflowChannelBC:



Collaboration diagram for scalarInflowChannelBC:



Public Methods

- **scalarInflowChannelBC** ()
- **scalarInflowChannelBC** (int dir, **Side::LoHiSide** sd)
- **scalarInflowChannelBC** (int dir, **Side::LoHiSide** sd, const **Interval** &a_comps)
- virtual ~**scalarInflowChannelBC** ()

Friends

- class **DomainGhostBC**

6.116.1 Constructor & Destructor Documentation

6.116.1.1 scalarInflowChannelBC::scalarInflowChannelBC ()

6.116.1.2 scalarInflowChannelBC::scalarInflowChannelBC (int *dir*, Side::LoHiSide *sd*)

6.116.1.3 scalarInflowChannelBC::scalarInflowChannelBC (int *dir*, Side::LoHiSide *sd*, const Interval & *a_comps*)

6.116.1.4 virtual scalarInflowChannelBC::~scalarInflowChannelBC () `[virtual]`

6.116.2 Friends And Related Function Documentation

6.116.2.1 friend class DomainGhostBC `[friend]`

Reimplemented from **scalarInflowBC** (p. 450).

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

- **channelBC.H**

6.117 SerialTask Class Reference

`#include <SPMD.H>`

Public Types

- enum **task** { **compute** = 0 }

6.117.1 Member Enumeration Documentation

6.117.1.1 enum SerialTask::task

Enumeration values:
compute

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

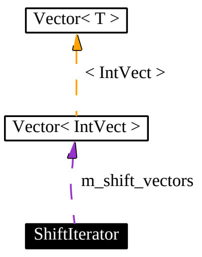
- **SPMD.H**

6.118 ShiftIterator Class Reference

clase to manage box-shifting used to enforce periodic BC's

`#include <ProblemDomain.H>`

Collaboration diagram for ShiftIterator:



Public Methods

- **ShiftIterator** ()

  basic constructor

- **ShiftIterator** (const bool *a_isPeriodic)
- **ShiftIterator** (const ShiftIterator &a_shiftIt)

  copy constructor

- **~ShiftIterator** ()
- ShiftIterator & **operator=** (const ShiftIterator &a_src)
- 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
- void **operator++** ()

  increment to the next shift unit vector

- void **incr** ()
- bool **ok** () const
- void **reset** ()

  reset to first shift unit vector

- void **begin** ()
- void **end** ()

6.118.1 Detailed Description

clase 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** (p. 106) 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.

6.118.2 Constructor & Destructor Documentation

6.118.2.1 ShiftIterator::ShiftIterator () `[inline]`

basic constructor

6.118.2.2 ShiftIterator::ShiftIterator (const bool ∗ *a_isPeriodic*)

6.118.2.3 ShiftIterator::ShiftIterator (const ShiftIterator & *a_shiftIt*) `[inline]`

copy constructor

6.118.2.4 ShiftIterator::~ShiftIterator ()

6.118.3 Member Function Documentation

6.118.3.1 void ShiftIterator::begin () `[inline]`

6.118.3.2 void ShiftIterator::computeShifts (const bool ∗ *a_isPeriodic*)

recompute shift vectors based on periodic directions

6.118.3.3 void ShiftIterator::end () `[inline]`

skip the iterator to the end. **ok**() (p. 455) will return false after this method is called.

6.118.3.4 IntVect ShiftIterator::i () const `[inline]`

6.118.3.5 void ShiftIterator::incr () `[inline]`

6.118.3.6 bool ShiftIterator::ok () const `[inline]`

6.118.3.7 IntVect ShiftIterator::operator() () const `[inline]`

returns the current shift unit vector

6.118.3.8 void ShiftIterator::operator++ () `[inline]`

increment to the next shift unit vector

6.118.3.9 ShiftIterator & ShiftIterator::operator= (const ShiftIterator & *a_src*) [inline]

6.118.3.10 void ShiftIterator::reset () [inline]

reset to first shift unit vector

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

- **ProblemDomain.H**

6.119 Side Class Reference

encapsulation of high or low side

`#include <LoHiSide.H>`

Public Types

- enum **LoHiSide** { **Invalid** = -1, **Lo** = 0, **Hi**, **NUMSIDES** }

Public Methods

- **Side** ()
- **~Side** ()

Static Public Methods

- **Side::LoHiSide flip** (**Side::LoHiSide** side)

Friends

- **Side::LoHiSide flip** (**Side::LoHiSide** side)
- int **sign** (**Side::LoHiSide** side)

6.119.1 Detailed Description

encapsulation of high or low side

Encapsulation of high or low side.

6.119.2 Member Enumeration Documentation

6.119.2.1 enum Side::LoHiSide

Lo is the low side \ Hi is the high side \

Enumeration values:

Invalid

Lo

Hi

NUMSIDES

6.119.3 Constructor & Destructor Documentation

6.119.3.1 Side::Side () [inline]

6.119.3.2 Side::~Side () [inline]

6.119.4 Member Function Documentation

6.119.4.1 Side::LoHiSide Side::flip (Side::LoHiSide *side*) [static]

6.119.5 Friends And Related Function Documentation

6.119.5.1 Side::LoHiSide flip (Side::LoHiSide *side*) [friend]

Returns the other side. I.e. swap Hi for Lo and vice versa.

6.119.5.2 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**

6.120 SideIterator Class Reference

Iterator for low and high side.

`#include <LoHiSide.H>`

Public Methods

- **SideIterator** ()

    {\bf Constructors, destructors}

- **~SideIterator** ()
- void **begin** ()

    {\bf Modification functions}

- void **reset** ()
- void **next** ()
- void **operator++** ()
- **Side::LoHiSide operator()** () const
- bool **ok** () const

6.120.1 Detailed Description

Iterator for low and high side.

SideIterator iterates over the values **Side::Lo** (p. 457) and **Side::Hi** (p. 457). There are no adjustable parameters whatsoever.

6.120.2 Constructor & Destructor Documentation

6.120.2.1 SideIterator::SideIterator ()

{\bf Constructors, destructors}

Default, and only, constructor.

6.120.2.2 SideIterator::~SideIterator () `[inline]`

Default destructor.

6.120.3 Member Function Documentation

6.120.3.1 void SideIterator::begin ()

{\bf Modification functions}

Set the iterator to the first side. \

{\bf Arguments:} none. \ {\bf Returns:} none. \ {\bf This:} \ This object is modified.

6.120.3.2 void SideIterator::next ()

6.120.3.3 bool SideIterator::ok () const

Checks to see if there is a next side. \

{\bf Arguments:} none. \ {\bf Returns:} \ true if there is a next side. \ false otherwise. \ {\bf This:} \ This object is not modified.

6.120.3.4 Side::LoHiSide SideIterator::operator() () const

Returns the value of the side for the current iteration. \

{\bf Arguments:} none. \ {\bf Returns:} \ Returns the value of the side for the current iteration. \ {\bf This:} \ This object is not modified.

6.120.3.5 void SideIterator::operator++ ()

Advances the iterator to the next face. \

{\bf Arguments:} none. \ {\bf Returns:} none. \ {\bf This:} \ This object is modified.

6.120.3.6 void SideIterator::reset () `[inline]`

Set the iterator to the first side. \

{\bf Arguments:} none. \ {\bf Returns:} none. \ {\bf This:} \ This object is modified.

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

- **LoHiSide.H**

6.121 Timer Class Reference

`#include <Timer.H>`

Public Methods

- **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 Methods

- void **TimerSummary** (const int bert, const int ernie)
- void **TimerSummary** (void)

6.121.1 Constructor & Destructor Documentation

6.121.1.1 Timer::Timer () [inline]

6.121.1.2 Timer::Timer (const string &) [inline]

6.121.1.3 Timer::Timer (const string &, Timer &) [inline]

6.121.1.4 Timer::Timer (const string &, Timer &, const *int*) [inline]

6.121.1.5 Timer::~Timer () [inline]

6.121.2 Member Function Documentation

6.121.2.1 void Timer::clear () [inline]

6.121.2.2 long int Timer::Count () [inline]

6.121.2.3 double Timer::getTimeStampWC () [inline]

6.121.2.4 string Timer::Name () [inline]

6.121.2.5 void Timer::setup () [inline]

6.121.2.6 void Timer::start () [inline]

6.121.2.7 void Timer::stop () [inline]

6.121.2.8 void Timer::TimerSummary (void) [inline, static]

6.121.2.9 void Timer::TimerSummary (const int *bert*, const int *ernie*) [inline, static]

6.121.2.10 double Timer::wc_time () [inline]

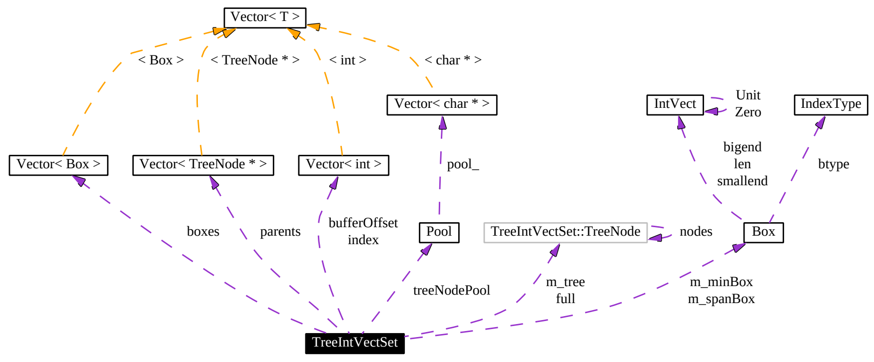
The documentation for this class was generated from the following file:

- **Timer.H**

6.122 TreeIntVectSet Class Reference

`#include <TreeIntVectSet.H>`

Collaboration diagram for TreeIntVectSet:



Public Methods

- **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)
- TreeIntVectSet & **operator &=** (const **ProblemDomain** &a_domain)
- TreeIntVectSet & **operator-=** (const TreeIntVectSet &a_sivs)

    not

- TreeIntVectSet & **operator-=** (const **IntVect** &a_iv)
- TreeIntVectSet & **operator-=** (const **Box** &a_box)
- **Vector**< **Box** > **createBoxes** () const
- void **createBoxes** (**Vector**< **Box** > &boxes, int &size) const
- bool **contains** (const **IntVect** &iv) const
- bool **contains** (const **Box** &box) const
- TreeIntVectSet **chop** (int idir, int chop_pnt)
- void **grow** (int igrow)
- void **grow** (int idir, int igrow)

- void **refine** (int iref=2)
- void **coarsen** (int iref=2)
- void **shift** (const **IntVect** &iv)
- void **clear** ()
- void **nestingRegion** (int a_radius, const **Box** &a_domain)
- void **nestingRegion** (int a_radius, const **ProblemDomain** &a_domain)
- const **Box** & **minBox** () const
- bool **isEmpty** () const
- int **numPts** () const
- void **compact** () const
- void **recalcMinBox** () const
- 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)

6.122.1 Detailed Description

Set calculus class based on trees.

6.122.2 Constructor & Destructor Documentation

6.122.2.1 TreeIntVectSet::TreeIntVectSet () `[inline]`

6.122.2.2 TreeIntVectSet::TreeIntVectSet (const Box &) `[inline]`

6.122.2.3 TreeIntVectSet::TreeIntVectSet (const TreeIntVectSet & *a_sivs*) `[inline]`

6.122.2.4 TreeIntVectSet::~TreeIntVectSet () `[inline]`

6.122.3 Member Function Documentation

6.122.3.1 TreeIntVectSet TreeIntVectSet::chop (int *idir*, int *chop_pnt*)

somewhat expensive, but shouldn't be ;-)

6.122.3.2 void TreeIntVectSet::clear ()

6.122.3.3 void TreeIntVectSet::coarsen (int *iref* = 2)

fast if iref is power of 2

6.122.3.4 void TreeIntVectSet::compact () const

6.122.3.5 bool TreeIntVectSet::contains (const Box & *box*) const

6.122.3.6 bool TreeIntVectSet::contains (const IntVect & *iv*) const

6.122.3.7 void TreeIntVectSet::createBoxes (Vector< Box > & *boxes*, int & *size*) const

6.122.3.8 Vector<Box> TreeIntVectSet::createBoxes () const

Returns **Vector** (p. 471)<**Box** (p. 106)> representation of this **IntVectSet** (p. 274).

6.122.3.9 void TreeIntVectSet::define (const TreeIntVectSet & *a_sivs*)

6.122.3.10 void TreeIntVectSet::define (const Box &)

6.122.3.11 void TreeIntVectSet::grow (int *idir*, int *igrow*)

expensive

6.122.3.12 void TreeIntVectSet::grow (int *igrow*)

expensive

6.122.3.13 bool TreeIntVectSet::isEmpty () const

6.122.3.14 void TreeIntVectSet::linearIn (const void ∗const *inBuf*)

6.122.3.15 void TreeIntVectSet::linearOut (void ∗const *a_outBuf*) const

6.122.3.16 int TreeIntVectSet::linearSize () const

6.122.3.17 const Box & TreeIntVectSet::minBox () const `[inline]`

6.122.3.18 void TreeIntVectSet::nestingRegion (int *a_radius*, const ProblemDomain & *a_domain*)

6.122.3.19 void TreeIntVectSet::nestingRegion (int *a_radius*, const Box & *a_domain*)

6.122.3.20 int TreeIntVectSet::numPts () const

6.122.3.21 TreeIntVectSet& TreeIntVectSet::operator &= (const ProblemDomain & *a_domain*)

6.122.3.22 TreeIntVectSet& TreeIntVectSet::operator &= (const Box & *a_box*)

6.122.3.23 TreeIntVectSet& TreeIntVectSet::operator &= (const TreeIntVectSet & *s_sivs*)

and

6.122.3.24 TreeIntVectSet& TreeIntVectSet::operator-= (const Box & *a_box*)

6.122.3.25 TreeIntVectSet & TreeIntVectSet::operator-= (const IntVect & *a_iv*) [inline]

6.122.3.26 TreeIntVectSet& TreeIntVectSet::operator-= (const TreeIntVectSet & *a_sivs*)

not

6.122.3.27 TreeIntVectSet& TreeIntVectSet::operator= (const TreeIntVectSet & *a_sivs*)

6.122.3.28 TreeIntVectSet& TreeIntVectSet::operator\|= (const Box & *a_box*)

6.122.3.29 TreeIntVectSet & TreeIntVectSet::operator\|= (const IntVect & *a_iv*) [inline]

6.122.3.30 TreeIntVectSet& TreeIntVectSet::operator\|= (const TreeIntVectSet & *a_sivs*)

or

6.122.3.31 void TreeIntVectSet::recalcMinBox () const

6.122.3.32 void TreeIntVectSet::refine (int *iref* = 2)

fast if iref is power of 2

6.122.3.33 void TreeIntVectSet::shift (const IntVect & *iv*)

very very fast

6.122.4 Friends And Related Function Documentation

6.122.4.1 void dumpTree (const TreeIntVectSet * *set*) [friend]

6.122.4.2 friend struct Flag [friend]

6.122.4.3 friend class IntVectSet [friend]

6.122.4.4 friend class TreeIntVectSetIterator [friend]

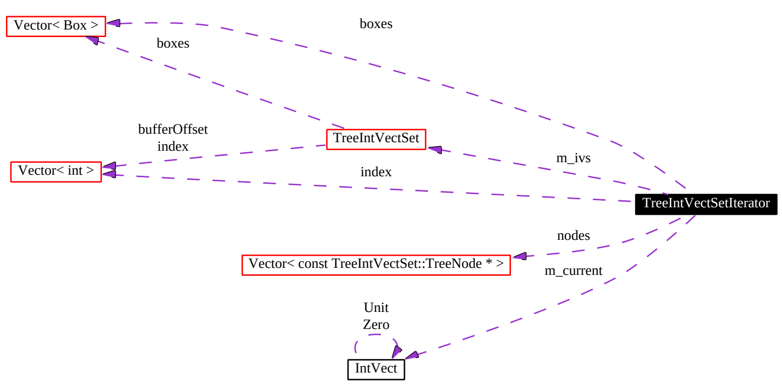
The documentation for this class was generated from the following file:

- **TreeIntVectSet.H**

6.123 TreeIntVectSetIterator Class Reference

`#include <TreeIntVectSet.H>`

Collaboration diagram for TreeIntVectSetIterator:



Public Methods

- **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** ()

6.123.1 Constructor & Destructor Documentation

6.123.1.1 TreeIntVectSetIterator::TreeIntVectSetIterator () [inline]

6.123.1.2 TreeIntVectSetIterator::TreeIntVectSetIterator (const TreeIntVectSet & *ivs*) [inline]

6.123.2 Member Function Documentation

6.123.2.1 void TreeIntVectSetIterator::begin ()

6.123.2.2 void TreeIntVectSetIterator::clear () [inline]

6.123.2.3 void TreeIntVectSetIterator::define (const TreeIntVectSet & *ivs*) [inline]

6.123.2.4 void TreeIntVectSetIterator::end () [inline]

6.123.2.5 bool TreeIntVectSetIterator::ok () const [inline]

6.123.2.6 const IntVect & TreeIntVectSetIterator::operator() () const [inline]

6.123.2.7 void TreeIntVectSetIterator::operator++ () [inline]

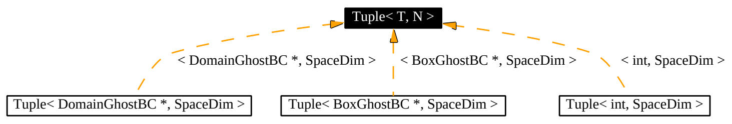
The documentation for this class was generated from the following file:

- **TreeIntVectSet.H**

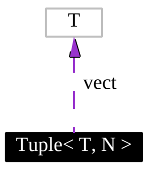
6.124 Tuple< T, N > Class Template Reference

`#include <Tuple.H>`

Inheritance diagram for Tuple< T, N >:



Collaboration diagram for Tuple< T, N >:



Public Methods

- **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]

template<class T, size_t N> class Tuple< T, N >

6.124.1 Constructor & Destructor Documentation

6.124.1.1 template<class T, size_t N> Tuple< T, N >::Tuple () [inline]

6.124.1.2 template<class T, size_t N> Tuple< T, N >::Tuple (const Tuple< T, N > & *rhs*)

6.124.2 Member Function Documentation

6.124.2.1 template<class T, size_t N> Tuple< T, N >::operator const T * () const [inline]

6.124.2.2 template<class T, size_t N> Tuple< T, N > & Tuple< T, N >::operator= (const Tuple< T, N > & *rhs*)

6.124.2.3 template<class T, size_t N> const T & Tuple< T, N >::operator[] (int *i*) const [inline]

6.124.2.4 template<class T, size_t N> T & Tuple< T, N >::operator[] (int *i*) [inline]

6.124.3 Member Data Documentation

6.124.3.1 template<class T, size_t N> T Tuple< T, N >::vect[N] [protected]

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

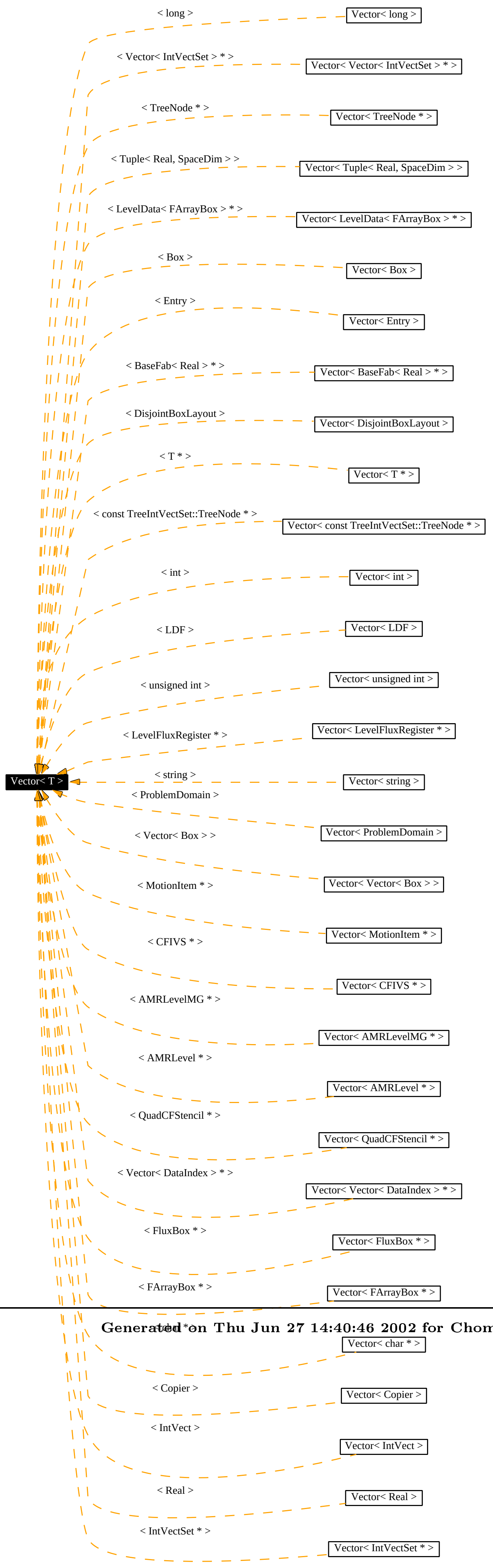
- **Tuple.H**

6.125 Vector< T > Class Template Reference

one dimensional dynamic array

`#include <Vector.H>`

Inheritance diagram for Vector< T >:


Vector< T >
< long >
Vector< long >
< Vector< IntVectSet > * >
Vector< Vector< IntVectSet > * >
< TreeNode * >
Vector< TreeNode * >
< Tuple< Real, SpaceDim > >
Vector< Tuple< Real, SpaceDim > >
< LevelData< FArrayBox > * >
Vector< LevelData< FArrayBox > * >
< Box >
Vector< Box >
< Entry >
Vector< Entry >
< BaseFab< Real > * >
Vector< BaseFab< Real > * >
< DisjointBoxLayout >
Vector< DisjointBoxLayout >
< T * >
Vector< T * >
< const TreeIntVectSet::TreeNode * >
Vector< const TreeIntVectSet::TreeNode * >
< int >
Vector< int >
< LDF >
Vector< LDF >
< unsigned int >
Vector< unsigned int >
< LevelFluxRegister * >
Vector< LevelFluxRegister * >
< string >
Vector< string >
< ProblemDomain >
Vector< ProblemDomain >
< Vector< Box > >
Vector< Vector< Box > >
< MotionItem * >
Vector< MotionItem * >
< CFIVS * >
Vector< CFIVS * >
< AMRLevelMG * >
Vector< AMRLevelMG * >
< AMRLevel * >
Vector< AMRLevel * >
< QuadCFStencil * >
Vector< QuadCFStencil * >
< Vector< DataIndex > * >
Vector< Vector< DataIndex > * >
< FluxBox * >
Vector< FluxBox * >
< FArrayBox * >
Vector< FArrayBox * >
Vector< char * >
< Copier >
Vector< Copier >
< IntVect >
Vector< IntVect >
< Real >
Vector< Real >
< IntVectSet * >
Vector< IntVectSet * >

Public Methods

- **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** (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 **push_back** (const T &in)
- void **append** (const Vector< T > &invec)
- void **resize** (unsigned int isize)
- void **resize** (unsigned int isize, const T &value)
- void **sort** ()

6.125.1 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.

6.125.2 Constructor & Destructor Documentation

6.125.2.1 template<class T> Vector< T >::Vector () `[inline]`

Default constructor. Creates a Vector of zero length with null data.

6.125.2.2 template<class T> virtual Vector< T >::~Vector () `[inline, virtual]`

Destructor.

6.125.2.3 template<class T> Vector< T >::Vector (const Vector< T > & *invec*) `[inline]`

Copy constructor.

6.125.2.4 template<class T> Vector< T >::Vector (const std::vector< T > & *invec*) `[inline]`

conversion constructor

6.125.2.5 template<class T> Vector< T >::Vector (unsigned int *isize*) [inline]

Constructs a Vector with given number of elements.\

{\bf Arguments:} \ size (not modified): number of elements of Vector to construct.\ {\bf This:} \ ——-The object is modified———-

6.125.2.6 template<class T> Vector< T >::Vector (unsigned int *isize*, const T & *value*) [inline]

Constructs a Vector with given number of elements and constant value.\

{\bf Arguments:} \ size (not modified): number of elements of Vector to construct.\ value (not modified): value to set every element to.\ {\bf This:} \ ——-The object is modified———-

6.125.3 Member Function Documentation

6.125.3.1 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**() (p. 475) + invec.size() (where this Vector is considered before the append is performed). The first element of invec will have index this->**size**() (p. 475), the second element will have index this->**size**() (p. 475)+1, etc.\

{\bf Arguments:} \ invec (not modified): Vector whose elements to append to this Vector.\ {\bf This:} \ ——-The object is modified———-

6.125.3.2 template<class T> void Vector< T >::clear () [inline]

6.125.3.3 template<class T> Vector<T>& Vector< T >::operator= (const Vector< T > & *invec*) [inline]

6.125.3.4 template<class T> Vector<T>& Vector< T >::operator= (const std::vector< T > & *invec*) [inline]

6.125.3.5 template<class T> const T& Vector< T >::operator[] (unsigned int *n*) const [inline]

Returns a constant reference to the given element in this Vector.\

{\bf Arguments: } \ n (not modified) index of desired element.\ {\bf Returns:} \ constant reference to value in Vector at index n.\ {\bf This:} \ This object is not modified.

6.125.3.6 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**() (p. 475). \

{\bf Arguments: } \ n (not modified) index of desired element.\ {\bf Returns:} \ modifiable reference to value in Vector at index n.\ {\bf This:} \ —– This object is modified if the returned reference is assigned a new value —–

6.125.3.7 template<class T> void Vector< T >::push_back (const T & *in*) [inline]

6.125.3.8 template<class T> void Vector< T >::resize (unsigned int *isize*, const T & *value*) [inline]

6.125.3.9 template<class T> void Vector< T >::resize (unsigned int *isize*) [inline]

6.125.3.10 template<class T> size_t Vector< T >::size () const [inline]

size function. returns current size of Vector

6.125.3.11 template<class T> void Vector< T >::sort () [inline]

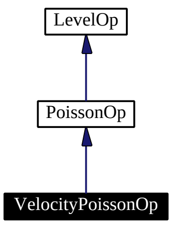
The documentation for this class was generated from the following file:

- **Vector.H**

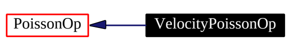
6.126 VelocityPoissonOp Class Reference

`#include <VelocityPoissonOp.H>`

Inheritance diagram for VelocityPoissonOp:



Collaboration diagram for VelocityPoissonOp:



Public Methods

- **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

6.126.1 Constructor & Destructor Documentation

6.126.1.1 VelocityPoissonOp::VelocityPoissonOp ()

6.126.1.2 VelocityPoissonOp::∼VelocityPoissonOp ()

6.126.2 Member Function Documentation

6.126.2.1 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.

6.126.2.2 virtual bool VelocityPoissonOp::isDefined () const `[virtual]`

Reimplemented from **PoissonOp** (p. 386).

6.126.2.3 LevelOp* VelocityPoissonOp::new_levelop () const [virtual]

virtual constructor workaround

Reimplemented from **PoissonOp** (p. 387).

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

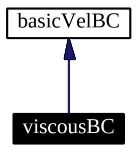
- **VelocityPoissonOp.H**

6.127 viscousBC Class Reference

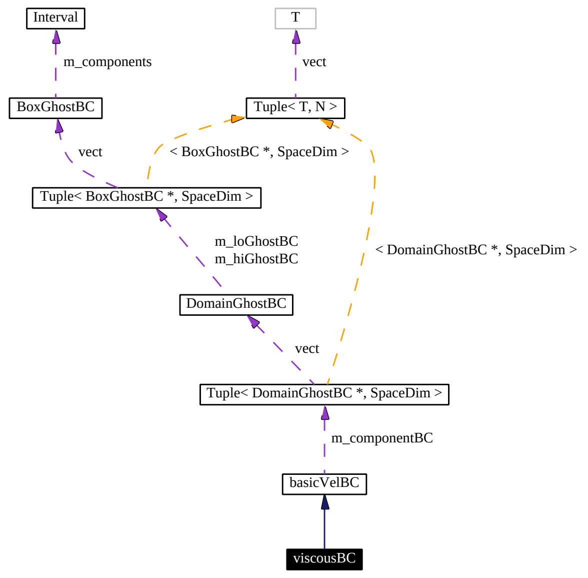
implements boundary conditions on viscous terms themselves

`#include <viscousBC.H>`

Inheritance diagram for viscousBC:



Collaboration diagram for viscousBC:



Public Methods

- **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** (p. 106) *instead of* **ProblemDomain** (p. 418)

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

  THIS HAS NOT BEEN IMPLEMENTED!

6.127.1 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

6.127.2 Constructor & Destructor Documentation

6.127.2.1 viscousBC::viscousBC ()

constructor

6.127.2.2 viscousBC::~viscousBC ()

6.127.3 Member Function Documentation

6.127.3.1 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. 98).

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

old interface using **Box** (p. 106) instead of **ProblemDomain** (p. 418)

Reimplemented from **basicVelBC** (p. 99).

6.127.3.3 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. 99).

6.127.3.4 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. 98).

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

Reimplemented from **basicVelBC** (p. 99).

6.127.3.6 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. 99).

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

- **viscousBC.H**

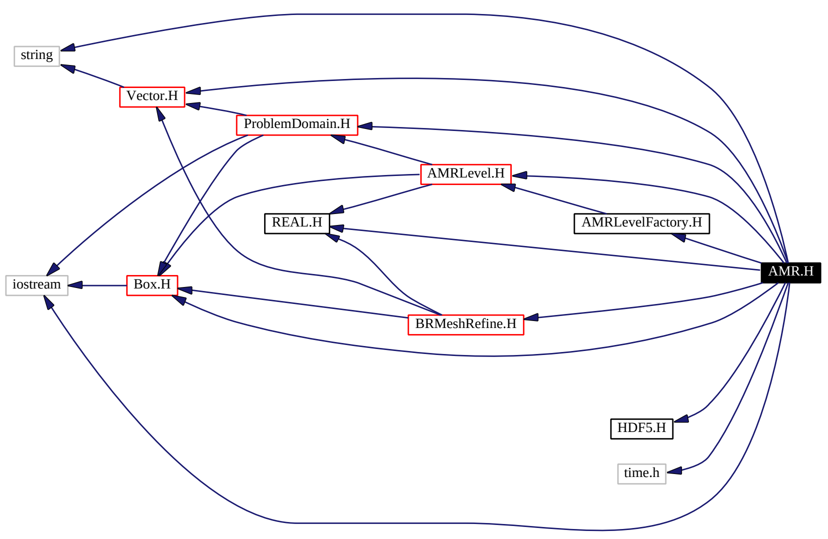
Chapter 7

Chombo&INS File Documentation

7.1 AMR.H File Reference

```
#include <iostream>
#include <string>
#include "REAL.H"
#include "Vector.H"
#include "AMRLevel.H"
#include "AMRLevelFactory.H"
#include "BRMeshRefine.H"
#include "ProblemDomain.H"
#include "Box.H"
#include "HDF5.H"
#include <time.h>
```

Include dependency graph for AMR.H:



Compounds

- class **AMR**

  framework for Berger-Oliger timestepping for AMR

7.2 AMRDerivedClass.H File Reference

`#include "FArrayBox.H"`

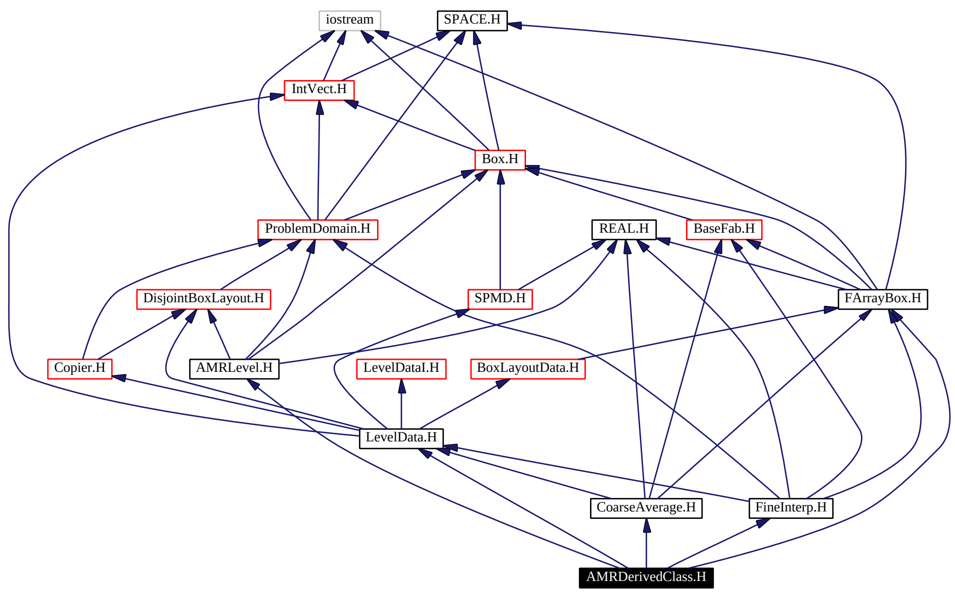
`#include "LevelData.H"`

`#include "AMRLevel.H"`

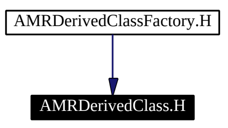
`#include "CoarseAverage.H"`

`#include "FineInterp.H"`

Include dependency graph for AMRDerivedClass.H:



This graph shows which files directly or indirectly include this file:



Compounds

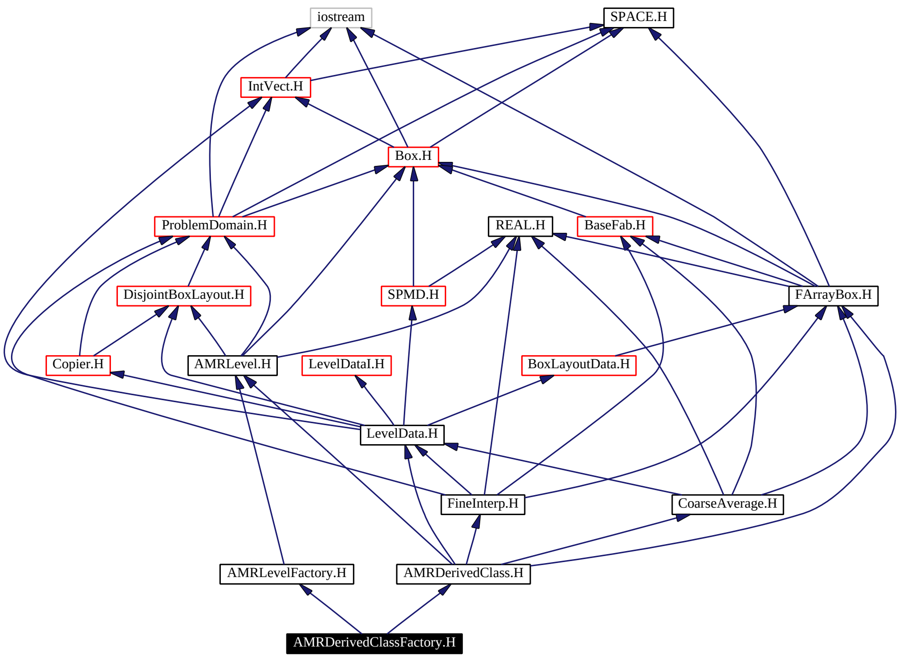
- class **AMRDerivedClass**

7.3 AMRDerivedClassFactory.H File Reference

`#include "AMRDerivedClass.H"`

`#include "AMRLevelFactory.H"`

Include dependency graph for AMRDerivedClassFactory.H:



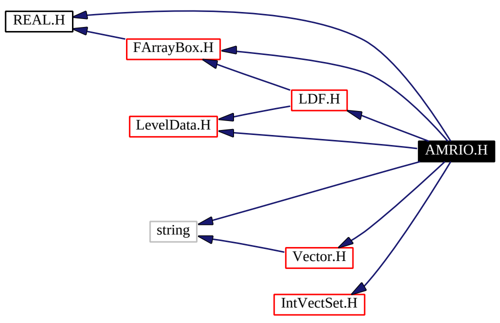
Compounds

- class **AMRDerivedClassFactory**

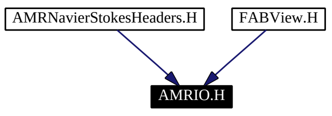
7.4 AMRIO.H File Reference

```
#include "REAL.H"
#include "LevelData.H"
#include "Vector.H"
#include "FArrayBox.H"
#include "IntVectSet.H"
#include <string>
#include "LDF.H"
```

Include dependency graph for AMRIO.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **AMRData**

Functions

- void **WriteAMRHierarchyHDF5** (string filename, const **AMRData** &data)
- void **WriteAMRHierarchyHDF5** (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)
- void **WriteAMRHierarchyHDF5** (HDF5Handle &handle, 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)

- void **WriteAMRHierarchyHDF5** (const string &filename, const **Vector**< **DisjointBoxLayout** > &a_vectGrids, const **Vector**< **LevelData**< **FArrayBox** > * > &a_vectData, const **Box** &a_domain, const **Vector**< int > &a_vectRatio, const int &a_numLevels)
- void **WriteAMRHierarchyHDF5** (HDF5Handle &handle, const **Vector**< **DisjointBoxLayout** > &a_vectGrids, const **Vector**< **LevelData**< **FArrayBox** > * > &a_vectData, const **Box** &a_domain, const **Vector**< int > &a_vectRatio, const int &a_numLevels)
- int **ReadAMRHierarchyHDF5** (string filename, **AMRData** &data)
- int **ReadAMRHierarchyHDF5** (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_refRatio, int &a_numLevels, const **IntVect** &a_ghostVector=IntVect::TheZeroVector())
- int **ReadAMRHierarchyHDF5** (HDF5Handle &handle, **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_refRatio, int &a_numLevels, const **IntVect** &a_ghostVector=IntVect::TheZeroVector())
- int **ReadAMRHierarchyHDF5** (const string &filename, **Vector**< **DisjointBoxLayout** > &a_vectGrids, **Vector**< **LevelData**< **FArrayBox** > * > &a_vectData, **Box** &a_domain, **Vector**< int > &a_refRatio, int &a_numLevels, const **IntVect** &a_ghostVector=IntVect::TheZeroVector())
- int **ReadAMRHierarchyHDF5** (HDF5Handle &handle, **Vector**< **DisjointBoxLayout** > &a_vectGrids, **Vector**< **LevelData**< **FArrayBox** > * > &a_vectData, **Box** &a_domain, **Vector**< int > &a_refRatio, int &a_numLevels, const **IntVect** &a_ghostVector=IntVect::TheZeroVector())
- int **ReadEBAMRASCII** (const string &filename, **Vector**< **DisjointBoxLayout** > &a_vectGrids, **Vector**< **LevelData**< **FArrayBox** > * > &a_vectData, **Vector**< string > &a_vectNames, **Box** &a_domain, **Vector**< **IntVectSet** > &a_coveredCells, **Vector**< **IntVectSet** > &a_multiValuedCells, **Vector**< int > &a_refRatio, int &a_numLevels, const **IntVect** &a_ghostVector=IntVect::TheZeroVector())
- void **writeFAB** (const **FArrayBox** *a_dataPtr)
- void **viewFAB** (const **FArrayBox** *a_dataPtr)
- void **writeFABname** (const **FArrayBox** *a_dataPtr, const char *a_filename)
- void **writeLDF** (const **LDF** *a_dataPtr)
- void **writeLevel** (const **LevelData**< **FArrayBox** > *a_dataPtr)
- void **viewLevel** (const **LevelData**< **FArrayBox** > *a_dataPtr)
- void **writeLDFname** (const **LDF** *a_dataPtr, const string &a_filename)
- void **writeLevelname** (const **LevelData**< **FArrayBox** > *a_dataPtr, const char *a_filename)
- void **WritePartialAMRHierarchyHDF5** (const string &filename, const **Vector**< **DisjointBoxLayout** > &a_vectGrids, const **Vector**< **LevelData**< **FArrayBox** > * > &a_vectData, const **Vector**< string > &a_vectNames, const **Box** &a_baseDomain, const **Real** &a_baseDx, const **Real** &a_dt, const **Real** &a_time, const **Vector**< int > &a_vectRatio, const **Interval** &a_levels)
- 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())

7.4.1 Function Documentation

7.4.1.1 void readAFabASCII (istream & *is*, FArrayBox & *fab*)

7.4.1.2 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.

{\bf 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 .\

{\bf Returns:}\ status code with values:\ 0: success\ -1: failure\

7.4.1.3 int ReadAMRHierarchyHDF5 (HDF5Handle & *handle*, Vector< DisjointBoxLayout > & *a_vectGrids*, Vector< LevelData< FArrayBox > * > & *a_vectData*, Box & *a_domain*, Vector< int > & *a_refRatio*, int & *a_numLevels*, const IntVect & *a_ghostVector* = IntVect::TheZeroVector())

Reads hierarchy of levels in HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation. Returns 0 on success.

{\bf Arguments:}\ handle : handle to input from.\ a_vectGrids : grids at each level.\ a_vectData : data at each level.\ a_domain : domain at coarsest level.\ a_vectRatio : refinement ratio at all levels (ith entry is refinement ratio between levels i and i + 1).\ a_numLevels : number of levels to read.\

{\bf Returns:} \ status code with values:\ 0: success \ -1: bogus number of levels \ -2: bogus number of components \ -3: error in readlevel \ -4: file open failed \ This is not blocking.

7.4.1.4 int ReadAMRHierarchyHDF5 (const string & *filename*, Vector< DisjointBoxLayout > & *a_vectGrids*, Vector< LevelData< FArrayBox > * > & *a_vectData*, Box & *a_domain*, Vector< int > & *a_refRatio*, int & *a_numLevels*, const IntVect & *a_ghostVector* = IntVect::TheZeroVector())

Reads hierarchy of levels in HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation. Returns 0 on success.

{\bf Arguments:}\ filename : file to input from.\ a_vectGrids : grids at each level.\ a_vectData : data at each level.\ a_domain : domain at coarsest level.\ a_vectRatio : refinement ratio at all levels (ith entry is refinement ratio between levels i and i + 1).\ a_numLevels : number of levels to read.\

{\bf Returns:} \ status code with values:\ 0: success \ -1: bogus number of levels \ -2: bogus number of components \ -3: error in readlevel \ -4: file open failed \ This is blocking.

7.4.1.5 int ReadAMRHierarchyHDF5 (HDF5Handle & *handle*, 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_refRatio*, int & *a_numLevels*, const IntVect & *a_ghostVector* = IntVect::TheZeroVector())

Reads hierarchy of levels in HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation. Returns 0 on success.

{\bf Arguments:}\ handle : handle to input from.\ 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 read.\

{\bf Returns:} \ status code with values:\ 0: success \ -1: bogus number of levels \ -2: bogus number of components \ -3: error in readlevel \ -4: file open failed \ This is not blocking

7.4.1.6 int ReadAMRHierarchyHDF5 (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_refRatio*, int & *a_numLevels*, const IntVect & *a_ghostVector* = IntVect::TheZeroVector())

Reads hierarchy of levels in HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation. Returns 0 on success.

{\bf Arguments:}\ filename : file to input from.\ 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 read.\

{\bf Returns:} \ status code with values:\ 0: success \ -1: bogus number of levels \ -2: bogus number of components \ -3: error in readlevel \ -4: file open failed \ This is blocking

7.4.1.7 int ReadAMRHierarchyHDF5 (string *filename*, AMRData & *data*)

Reads hierarchy of levels in HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation. Returns 0 on success.

{\bf Arguments:}\ filename : file to input from.\ 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 read.\

{\bf Returns:} \ status code with values:\ 0: success \ -1: bogus number of levels \ -2: bogus number of components \ -3: error in readlevel \ -4: file open failed \ This is blocking

7.4.1.8 int ReadEBAMRASCII (const string & *filename*, Vector< DisjointBoxLayout > & *a_vectGrids*, Vector< LevelData< FArrayBox > * > & *a_vectData*, Vector< string > & *a_vectNames*, Box & *a_domain*, Vector< IntVectSet > & *a_coveredCells*, Vector< IntVectSet > & *a_multiValuedCells*, Vector< int > & *a_refRatio*, int & *a_numLevels*, const IntVect & *a_ghostVector* = IntVect::TheZeroVector())

Reads hierarchy of EB levels in ASCII format. Returns 0 on success. Data is read in as FArrayBoxes because the data file does not include sufficient information to reconstruct the EBIS. {\bf Arguments:}\ filename : file to input from.\ a_vectGrids : grids at each level.\ a_vectData : data at each level.\ a_vectNames: names of variables.\ a_domain : domain at coarsest level.\ a_vectRatio : refinement ratio at all levels (ith entry is refinement ratio between levels i and i + 1).\ a_multiValuedCells : multi-valued cells at all levels \ (ith entry is refinement ratio between levels i and i + 1).\ a_coveredCells : covered cells at all levels \ (ith entry is refinement ratio between levels i and i + 1).\ a_numLevels : number of levels to read.\ returns 0 if successful.

7.4.1.9 void viewFAB (const FArrayBox * *a_dataPtr*)

View *a_dataPtr by writing it to an HDF5 plotfile (to a temporary file) and then running ChomboVis with a python script which brings up a data browser by default. The file has the same format as WriteAMRHierarchyHDF5, but for a single **FArrayBox** (p. 229). This is useful for debugging.

7.4.1.10 void viewLevel (const LevelData< FArrayBox > * *a_dataPtr*)

View *a_dataPtr by writing it to an HDF5 plotfile (to a temporary file) and then running ChomboVis with a python script which brings up a data browser by default. The file has the same format as WriteAMRHierarchyHDF5, but for a single **LevelData** (p. 295)<**FArrayBox** (p. 229)>. This is useful for debugging.

7.4.1.11 void writeAFabASCII (std::ostream & *os*, const FArrayBox & *fab*)

7.4.1.12 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.

{\bf 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.\

7.4.1.13 void WriteAMRHierarchyHDF5 (HDF5Handle & *handle*, const Vector< DisjointBoxLayout > & *a_vectGrids*, const Vector< LevelData< FArrayBox > * > & *a_vectData*, const Box & *a_domain*, const Vector< int > & *a_vectRatio*, const int & *a_numLevels*)

Writes hierarchy of levels in HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation.

{\bf Arguments:}\ handle : handle to output to.\ a_vectGrids : grids at each level.\ a_vectData : data at each level.\ a_domain : domain at coarsest level.\ 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.\ This is not blocking.

7.4.1.14 void WriteAMRHierarchyHDF5 (const string & *filename*, const Vector< DisjointBoxLayout > & *a_vectGrids*, const Vector< LevelData< FArrayBox > * > & *a_vectData*, const Box & *a_domain*, const Vector< int > & *a_vectRatio*, const int & *a_numLevels*)

Writes hierarchy of levels in HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation.

{\bf Arguments:}\ filename : file to output to.\ a_vectGrids : grids at each level.\ a_vectData : data at each level.\ a_domain : domain at coarsest level.\ 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.\ This is blocking.

7.4.1.15 void WriteAMRHierarchyHDF5 (HDF5Handle & *handle*, 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 HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation.

{\bf Arguments:}\ handle : handle 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.\ This is not blocking.

7.4.1.16 void WriteAMRHierarchyHDF5 (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 HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation.

{\bf 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.\ This is blocking.

7.4.1.17 void WriteAMRHierarchyHDF5 (string *filename*, const AMRData & *data*)

Writes hierarchy of levels in HDF5 format. Only available if the preprocessor macro HDF5 is defined at compilation.

{\bf 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.\ This is blocking.

7.4.1.18 void writeFAB (const FArrayBox ∗ *a_dataPtr*)

Writes a plotfile using the same format as WriteAMRHierarchyHDF5, but for a **FArrayBox** (p. 229). This is useful for debugging. ∗a_dataPtr is written to a file named fab.hdf5

7.4.1.19 void writeFABname (const FArrayBox ∗ *a_dataPtr*, const char ∗ *a_filename*)

Writes a plotfile using the same format as WriteAMRHierarchyHDF5, but for a **FArrayBox** (p. 229). This is useful for debugging. ∗a_dataPtr is written to the file given by a_filename.

7.4.1.20 void writeLDF (const LDF ∗ *a_dataPtr*)

Write a plotfile using the same format as WriteAMRHierarchyHDF5, but for a single **LevelData** (p. 295)<**FArrayBox** (p. 229)>. Useful for debugging. ∗a_dataPtr is written to a file named LDF.hdf5.

7.4.1.21 void writeLDFname (const LDF ∗ *a_dataPtr*, const string & *a_filename*)

Write a plotfile using the same format as WriteAMRHierarchyHDF5, but for a single **LevelData** (p. 295)<**FArrayBox** (p. 229)>. Useful for debugging. ∗a_dataPtr is written to the file given by a_filename.

7.4.1.22 void writeLevel (const LevelData< FArrayBox > ∗ *a_dataPtr*)

Write a plotfile using the same format as WriteAMRHierarchyHDF5, but for a single **LevelData** (p. 295)<**FArrayBox** (p. 229)>. Useful for debugging. ∗a_dataPtr is written to a file named LDF.hdf5.

7.4.1.23 void writeLevelname (const LevelData< FArrayBox > ∗ *a_dataPtr*, const char ∗ *a_filename*)

Write a plotfile using the same format as WriteAMRHierarchyHDF5, but for a single **LevelData** (p. 295)<**FArrayBox** (p. 229)>. Useful for debugging. ∗a_dataPtr is written to the file given by a_filename.

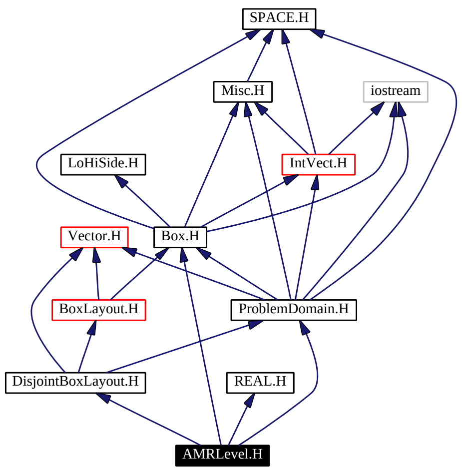
7.4.1.24 void WritePartialAMRHierarchyHDF5 (const string & *filename*, const Vector< DisjointBoxLayout > & *a_vectGrids*, const Vector< LevelData< FArrayBox > * > & *a_vectData*, const Vector< string > & *a_vectNames*, const Box & *a_baseDomain*, const Real & *a_baseDx*, const Real & *a_dt*, const Real & *a_time*, const Vector< int > & *a_vectRatio*, const Interval & *a_levels*)

Write a plotfile with levels that don't necessarily begin with 0 (will translate things so that lBase comes out as level 0, etc) Levels in **Vector** (p. 471) arguments should correspond with actual levels. (and so should be defined for a_levels.begin() through a_levels.end(). {\bf 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 base level (given by a_levels.begin()).\ a_dx : grid spacing at base level.\ a_dt : time step at base 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.\ This is blocking.

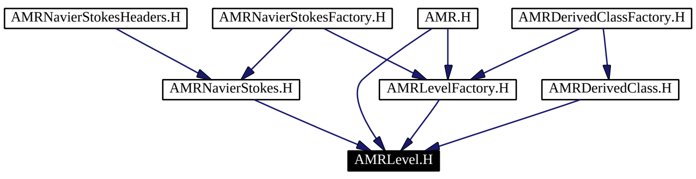
7.5 AMRLevel.H File Reference

```
#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

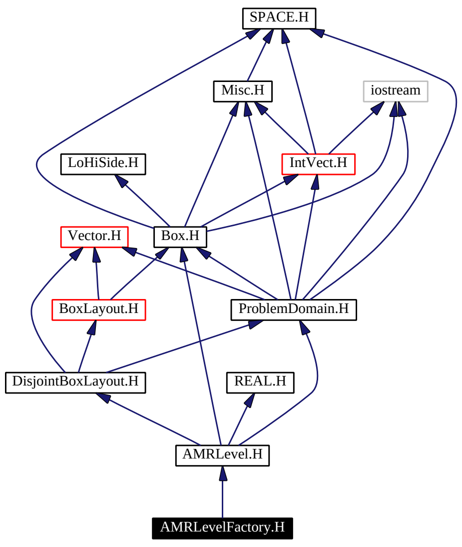
- class **AMRLevel**

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

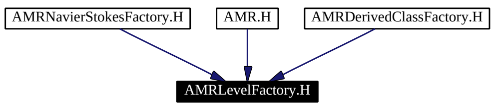
7.6 AMRLevelFactory.H File Reference

`#include "AMRLevel.H"`

Include dependency graph for AMRLevelFactory.H:



This graph shows which files directly or indirectly include this file:



Compounds

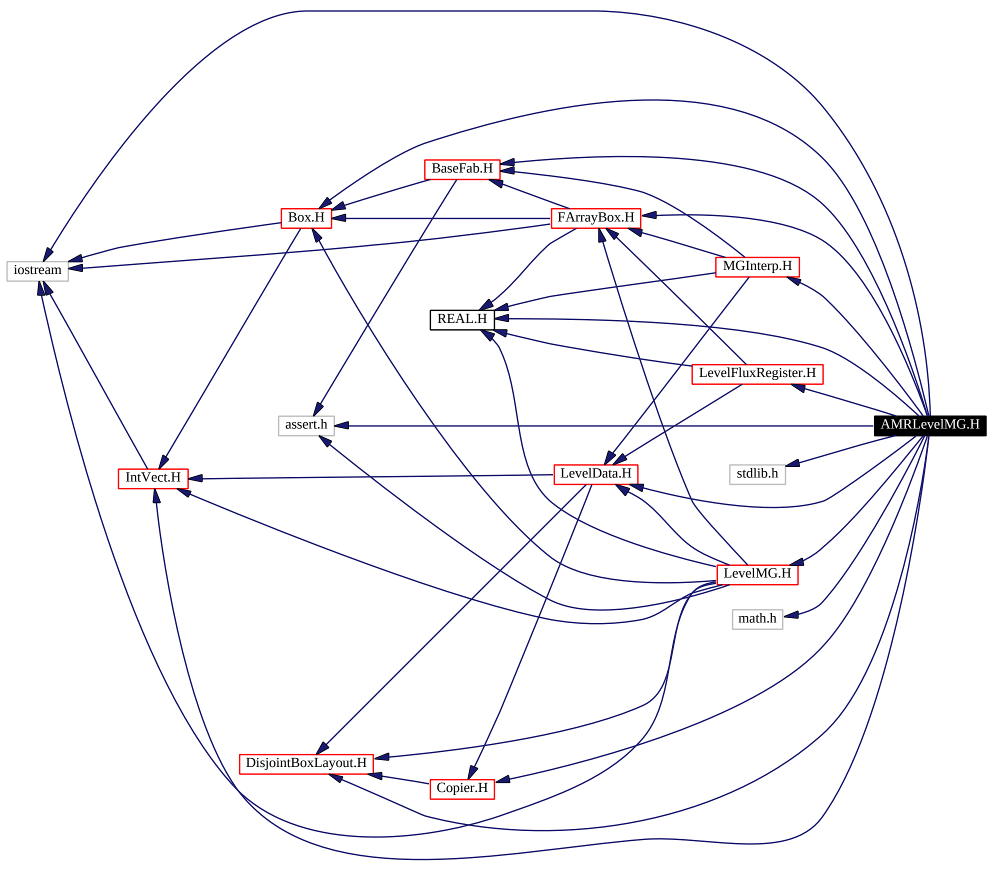
- class **AMRLevelFactory**

  Factory class to produce BaseEBFaceFABs.

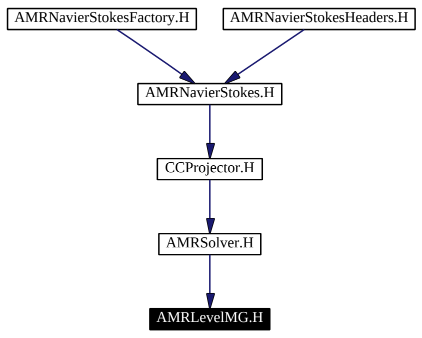
7.7 AMRLevelMG.H File Reference

```
#include <stdlib.h>
#include <iostream>
#include <assert.h>
#include <math.h>
#include "REAL.H"
#include "IntVect.H"
#include "Box.H"
#include "FArrayBox.H"
#include "DisjointBoxLayout.H"
#include "LevelData.H"
#include "LevelFluxRegister.H"
#include "BaseFab.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

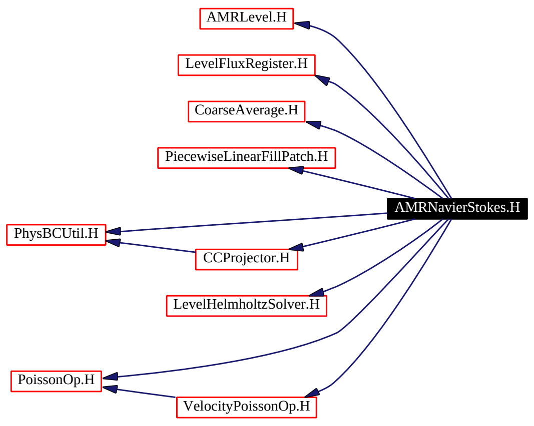
- class **AMRLevelMG**

  class which manages data and operations on a level for **AMRSolver** (p. 69)

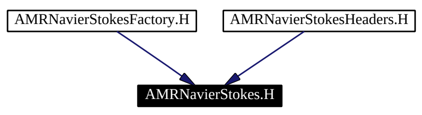
7.8 AMRNavierStokes.H File Reference

```
#include "AMRLevel.H"
#include "LevelFluxRegister.H"
#include "CoarseAverage.H"
#include "PiecewiseLinearFillPatch.H"
#include "CCProjector.H"
#include "LevelHelmholtzSolver.H"
#include "PoissonOp.H"
#include "VelocityPoissonOp.H"
#include "PhysBCUtil.H"
```

Include dependency graph for AMRNavierStokes.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **AMRNavierStokes**

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

7.8.1 Enumeration Type Documentation

7.8.1.1 enum scalTypes

scalar types – defined to faciliate lagrangian tracking

Enumeration values:

- **xMarker**
- **yMarker**
- **NUM_SCAL_TYPES**

7.8.1.2 enum viscousSolverTypes

Enumeration values:

- **backwardEuler**
- **CrankNicolson**
- **TGA**
- **NUM_SOLVER_TYPES**

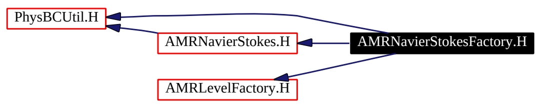
7.9 AMRNavierStokesFactory.H File Reference

`#include "AMRNavierStokes.H"`

`#include "AMRLevelFactory.H"`

`#include "PhysBCUtil.H"`

Include dependency graph for AMRNavierStokesFactory.H:



Compounds

- class **AMRNavierStokesFactory**

7.10 AMRNavierStokesHeaders.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 "AMRNavierStokes.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 "IAMRGodunovF_F.H"

#include "PROB_F_F.H"

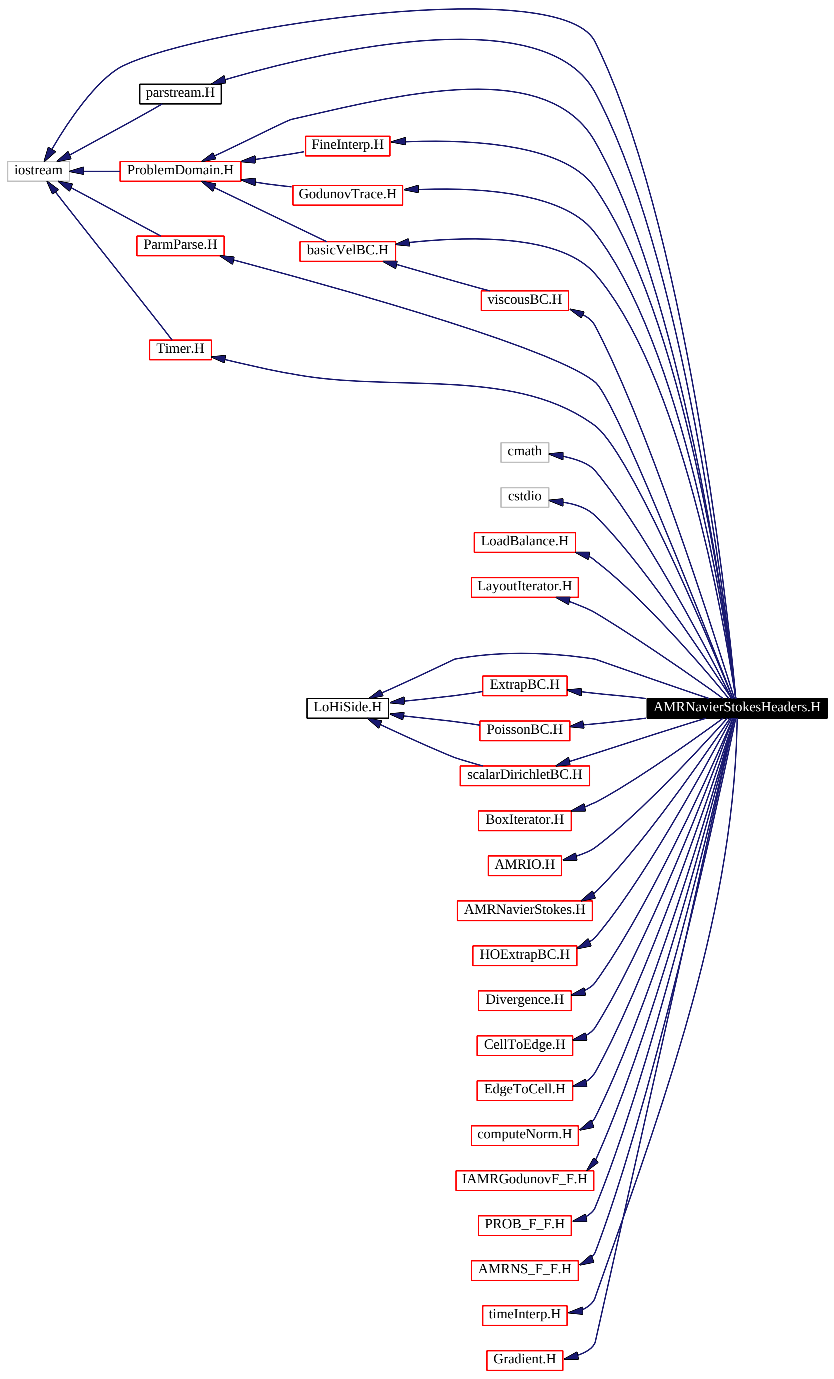
#include "AMRNS_F_F.H"

#include "timeInterp.H"

#include "Gradient.H"

#include "Timer.H"

Include dependency graph for AMRNavierStokesHeaders.H:



Defines

- #define **ADVECT_GROW** 2
- #define **TIME_EPS** 1.0e-5
- #define **TGA_EPS** 1.0e-8

Variables

- **Timer TimeNSadvance**
- **Timer TimeNSadvanceIntro**
- **Timer TimeNSadvanceCompAV**
- **Timer TimeNSadvanceFillLamba**
- **Timer TimeNSadvanceAdvectScalar**
- **Timer TimeNSadvanceAdvectDiffScalar**
- **Timer TimeNSadvancePredictVelocities**
- **Timer TimeNSadvanceCompUStar**
- **Timer TimeNSadvanceCLD**
- **Timer TimeNSadvancePBCE**
- **Timer TimeNSadvanceLevelProject**
- **Timer cs0**
- **Timer cs1**
- **Timer cs2**
- **Timer cs3**
- **Timer cs4**
- **Timer cs5**
- **Timer cs7**
- **Timer cs8**
- **Timer TimeNScompUStarlevelSolve1**
- **Timer cs10**
- **Timer cs11**
- **Timer TimeNScompUStarlevelSolve2**
- **Timer cs13**
- **Timer cs14**
- **Timer cs15**
- **Timer TimeNSpost**
- **Timer TimeNSpostPart1**
- **Timer TimeNSpostPart2**
- **Timer TimeNSpostImplicitReflux**
- **Timer TimeNSpostSyncOperations**
- **Timer TimeNSpostDiagnostic**
- **Timer TimeNSpostCreateBC**
- **Timer TimeNSpostRegrid**
- **Timer TimeinitialVelocityProject**
- **Timer TimedoPostRegridOps**
- **Timer TimeinitializeGlobalPressure**
- **Timer TimecomputeAdvectionVelocities**
- **Timer TimeSetPBC**
- **Timer TimeTraceState**
- **Timer TimelevelMacProject**
- **Timer NST_Everything**
- **Timer TimeSA_diffusionSolver**
- **Timer TimeSA_viscousSolver**
- **Timer TimeSA_initialData**
- **Timer TimeSA_velocity**
- **Timer TimeSA_scalar**
- **Timer TimeLS_advectScalar1**
- **Timer TimeLS_advectScalar2**
- **Timer TimeLS_compUstar1**
- **Timer TimeLS_compUstar2**

7.10.1 Define Documentation

7.10.1.1 #define ADVECT_GROW 2

7.10.1.2 #define TGA_EPS 1.0e-8

7.10.1.3 #define TIME_EPS 1.0e-5

7.10.2 Variable Documentation

7.10.2.1 Timer cs0

7.10.2.2 Timer cs1

7.10.2.3 Timer cs10

7.10.2.4 Timer cs11

7.10.2.5 Timer cs13

7.10.2.6 Timer cs14

7.10.2.7 Timer cs15

7.10.2.8 Timer cs2

7.10.2.9 Timer cs3

7.10.2.10 Timer cs4

7.10.2.11 Timer cs5

7.10.2.12 Timer cs7

7.10.2.13 Timer cs8

7.10.2.14 Timer NST_Everything

7.10.2.15 Timer TimecomputeAdvectionVelocities

7.10.2.16 Timer TimedoPostRegridOps

7.10.2.17 Timer TimeinitializeGlobalPressure

7.10.2.18 Timer TimeinitialVelocityProject

7.10.2.19 Timer TimelevelMacProject

7.10.2.20 Timer TimeLS_advectScalar1

7.10.2.21 Timer TimeLS_advectScalar2

7.10.2.22 Timer TimeLS_compUstar1

7.10.2.23 Timer TimeLS_compUstar2

7.10.2.24 Timer TimeNSadvance

7.10.2.25 Timer TimeNSadvanceAdvectDiffScalar

7.10.2.26 Timer TimeNSadvanceAdvectScalar

7.10.2.27 Timer TimeNSadvanceCLD

7.10.2.28 Timer TimeNSadvanceCompAV

7.10.2.29 Timer TimeNSadvanceCompUStar

7.10.2.30 Timer TimeNSadvanceFillLamba

7.10.2.31 Timer TimeNSadvanceIntro

7.10.2.32 Timer TimeNSadvanceLevelProject

7.10.2.33 Timer TimeNSadvancePBCE

7.10.2.34 Timer TimeNSadvancePredictVelocities

7.10.2.35 Timer TimeNScompUStarlevelSolve1

7.10.2.36 Timer TimeNScompUStarlevelSolve2

7.10.2.37 Timer TimeNSpost

7.10.2.38 Timer TimeNSpostCreateBC

7.10.2.39 Timer TimeNSpostDiagnostic

7.10.2.40 Timer TimeNSpostImplicitReflux

7.10.2.41 Timer TimeNSpostPart1

7.10.2.42 Timer TimeNSpostPart2

7.10.2.43 Timer TimeNSpostRegrid

7.10.2.44 Timer TimeNSpostSyncOperations

7.10.2.45 Timer TimeSA_diffusionSolver

7.10.2.46 Timer TimeSA_initialData

7.10.2.47 Timer TimeSA_scalar

7.10.2.48 Timer TimeSA_velocity

7.10.2.49 Timer TimeSA_viscousSolver

7.10.2.50 Timer TimeSetPBC

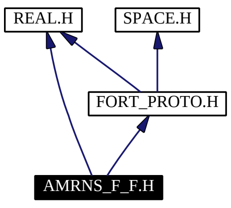
7.10.2.51 Timer TimeTraceState

7.11 AMRNS_F_F.H File Reference

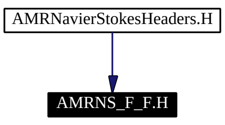
`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for AMRNS_F_F.H:



This graph shows which files directly or indirectly include this file:



Defines

- #define **FORT_COMPUTEVORT** FORTRAN_NAME(COMPUTEVORT ,computevort)
- #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_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))

7.11.1 Define Documentation

7.11.1.1 #define FORT_CENTERED_DERIV FORTRAN_NAME(CENTERED_DERIV ,centered_deriv)

7.11.1.2 #define FORT_COMPUTEVORT FORTRAN_NAME(COMPUTEVORT ,computevort)

7.11.1.3 #define FORT_MAGVECT FORTRAN_NAME(MAGVECT ,magvect)

7.11.2 Function Documentation

7.11.2.1 void FORT_CENTERED_DERIV (CHFp_FRA1(deriv), CHFp_CONST_FRA1(phi), CHFp_BOX(grid), CHFp_CONST_REAL(dx), CHFp_INT(dir))

7.11.2.2 void FORT_COMPUTEVORT (CHFp_FRA1(vort), CHFp_FRA(vel), CHFp_BOX(gridBox), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir))

7.11.2.3 void FORT_MAGVECT (CHFp_FRA1(magVector), CHFp_FRA(vect), CHFp_BOX(gridBox))

7.12 AMRSolver.H File Reference

`#include <cstdlib>`

`#include <cmath>`

`#include <iostream>`

`#include <assert.h>`

`#include "REAL.H"`

`#include "IntVect.H"`

`#include "Box.H"`

`#include "FArrayBox.H"`

`#include "Vector.H"`

`#include "IntVectSet.H"`

`#include "DisjointBoxLayout.H"`

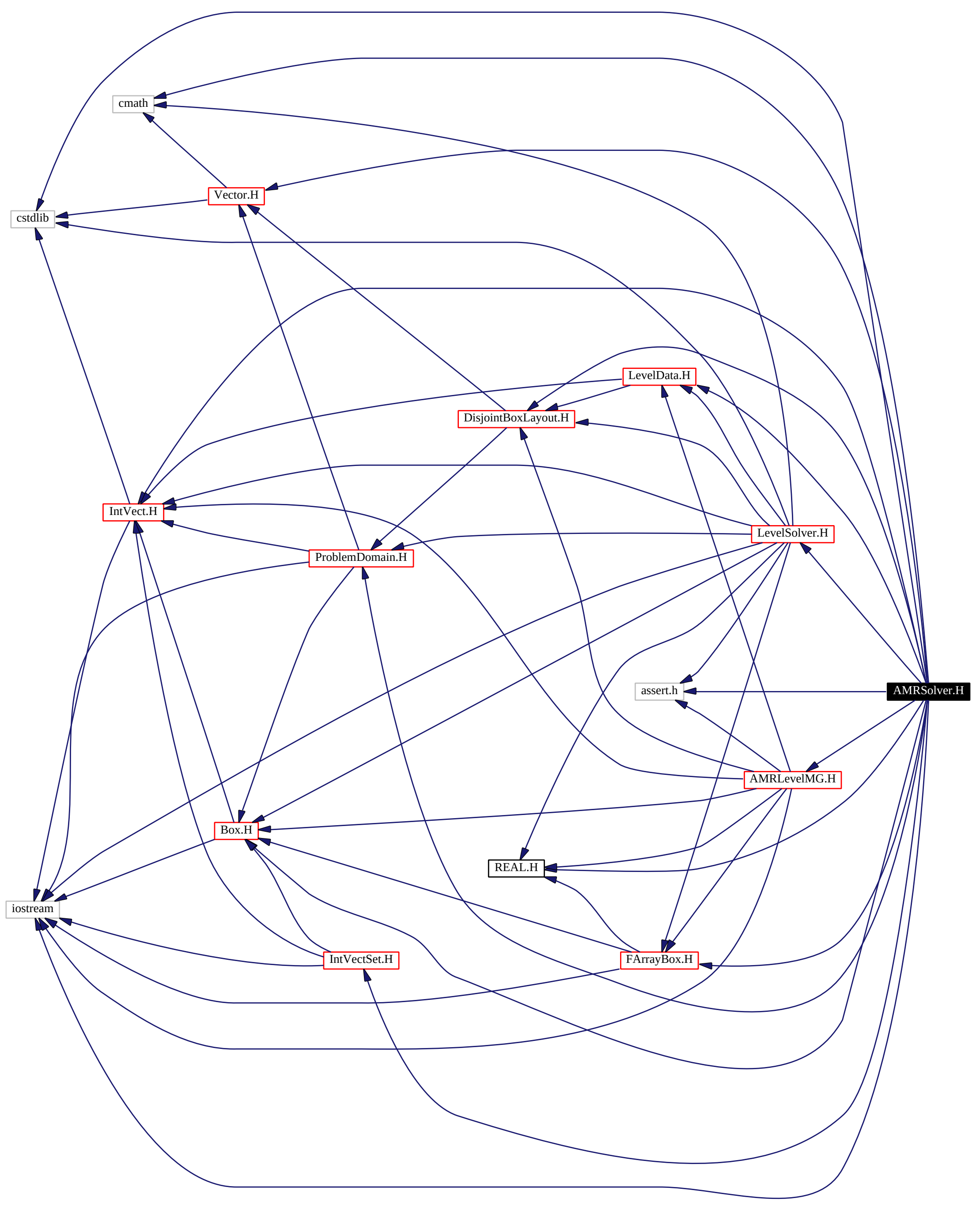
`#include "LevelData.H"`

`#include "AMRLevelMG.H"`

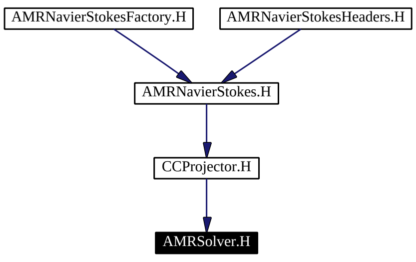
`#include "LevelSolver.H"`

`#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.

7.13 Arena.H File Reference

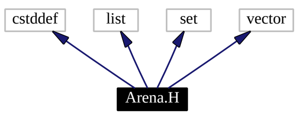
`#include <cstddef>`

`#include <list>`

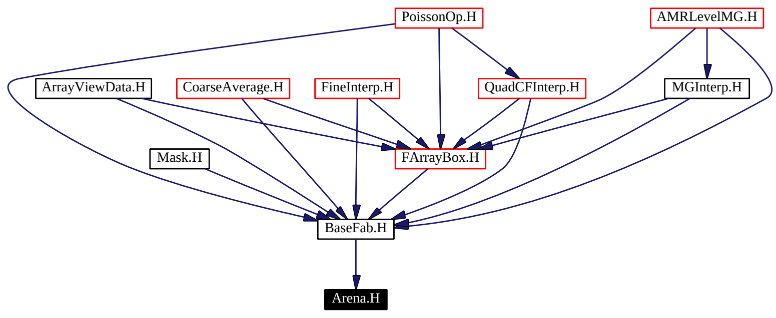
`#include <set>`

`#include <vector>`

Include dependency graph for Arena.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Arena**
- union **Word**
- class **BArena**
- class **CArena**
- class **Node**

Typedefs

- typedef std::list< **Arena** ∗ > **ArenaList**

Variables

- **Arena** ∗ **The_FAB_Arena**

7.13.1 Typedef Documentation

7.13.1.1 typedef std::list<Arena*> ArenaList

7.13.2 Variable Documentation

7.13.2.1 Arena* The_FAB_Arena

7.14 ArrayViewData.H File Reference

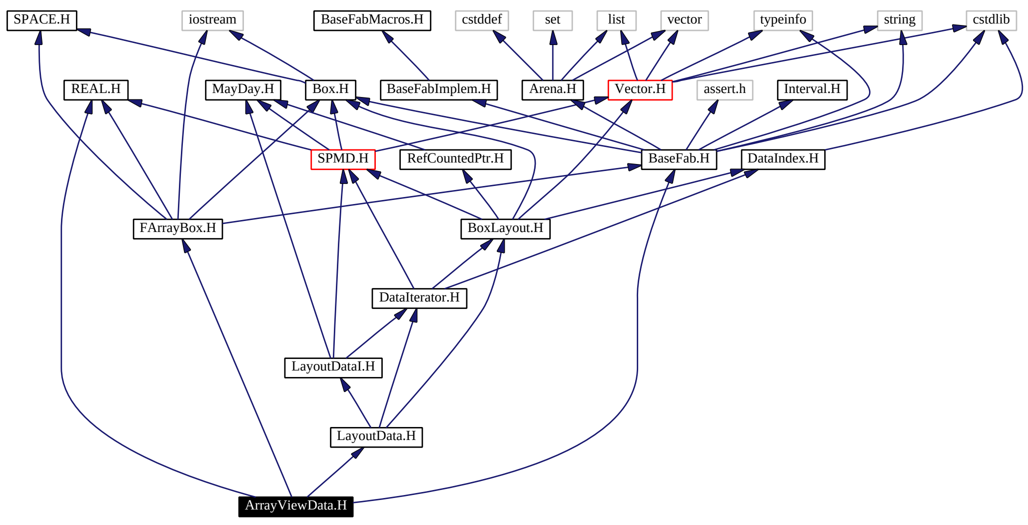
`#include "LayoutData.H"`

`#include "BaseFab.H"`

`#include "REAL.H"`

`#include "FArrayBox.H"`

Include dependency graph for ArrayViewData.H:



Compounds

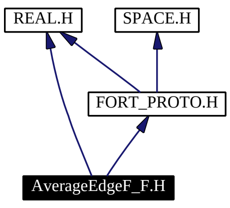
- class **ArrayViewData**

7.15 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))

7.15.1 Define Documentation

7.15.1.1 #define FORT_AVERAGEEDGE FORTRAN_NAME(AVERAGEEDGE ,averageedge)

7.15.2 Function Documentation

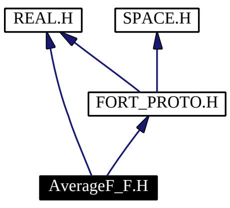
7.15.2.1 void FORT_AVERAGEEDGE (CHFp_FRA(coarse), CHFp_CONST_FRA(fine), CHFp_BOX(crseBox), CHFp_CONST_INT(dir), CHFp_CONST_INT(nRef), CHFp_BOX(refBox))

7.16 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))

7.16.1 Define Documentation

7.16.1.1 #define FORT_AVERAGE FORTRAN_NAME(AVERAGE ,average)

7.16.2 Function Documentation

7.16.2.1 void FORT_AVERAGE (CHFp_FRA(coarse), CHFp_CONST_FRA(fine), CHFp_BOX(b), CHFp_CONST_INT(ref_ratio), CHFp_BOX(bref))

7.17 BaseBottomSmoother.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>

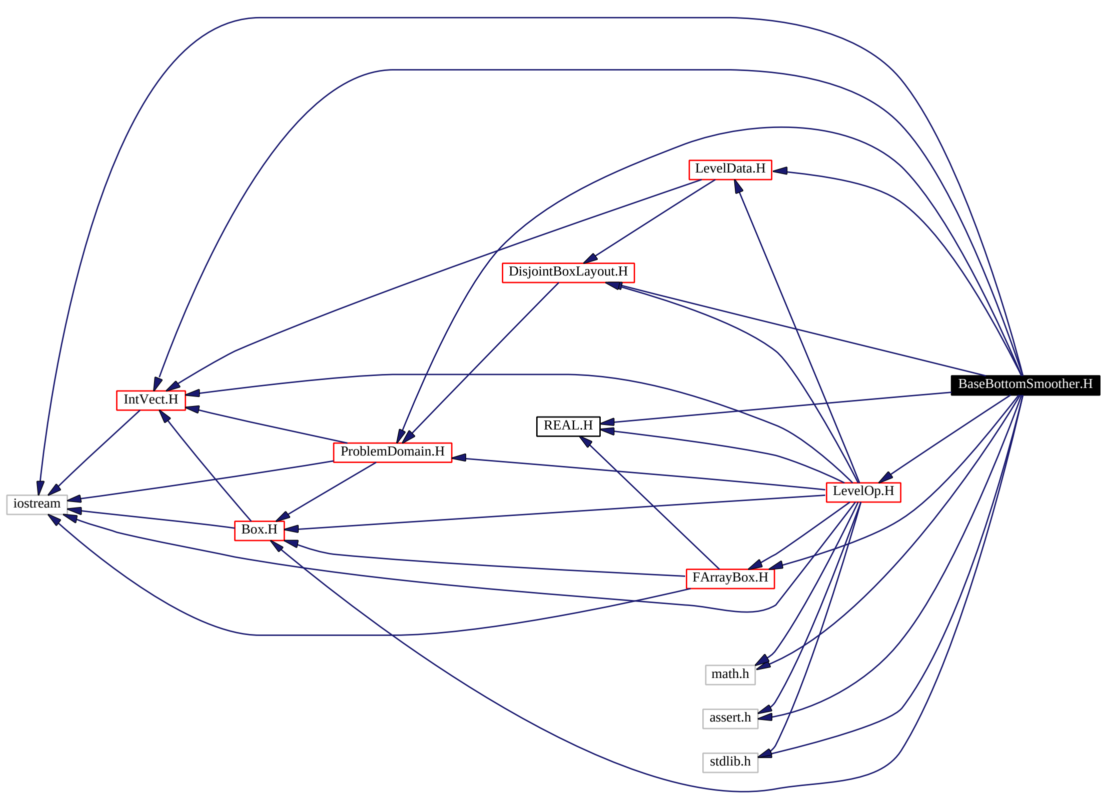
#include <LevelData.H>

#include <FArrayBox.H>

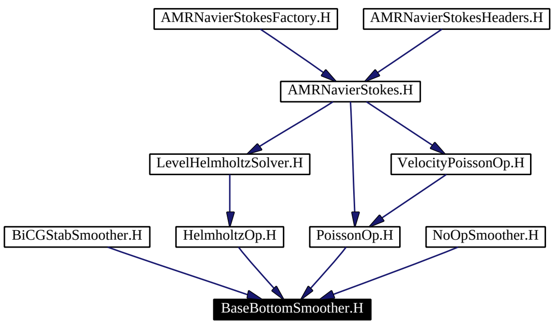
#include <ProblemDomain.H>

#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** (p. 315)*-derived objects.*

7.18 BaseFab.H File Reference

`#include <cstdlib>`

`#include <assert.h>`

`#include <string>`

`#include <typeinfo>`

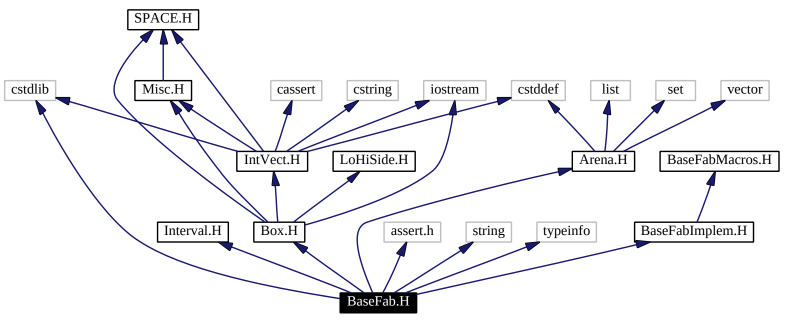
`#include "Box.H"`

`#include "Arena.H"`

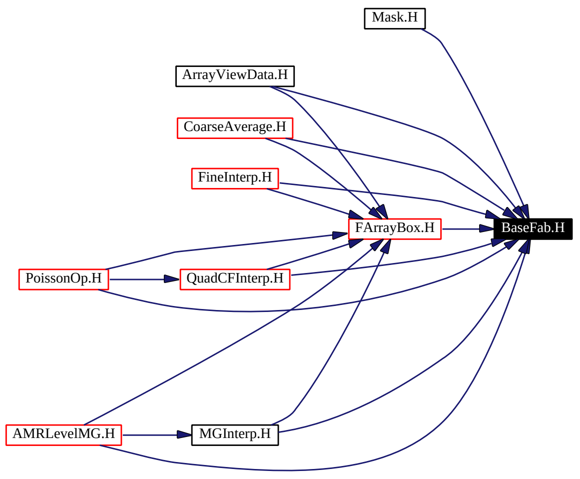
`#include "Interval.H"`

`#include "BaseFabImplem.H"`

Include dependency graph for BaseFab.H:



This graph shows which files directly or indirectly include this file:



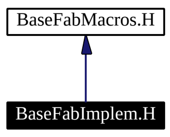
Compounds

- class **BaseFab**

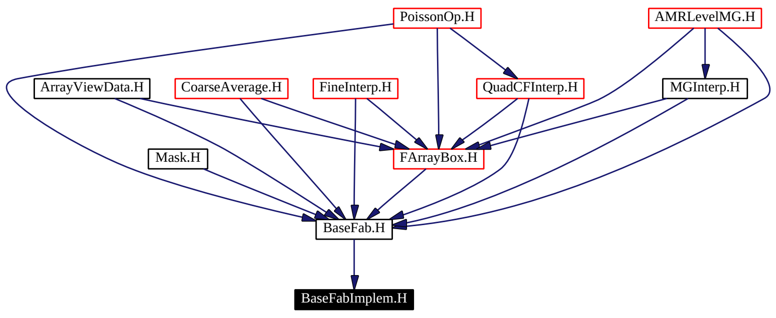
7.19 BaseFabImplem.H File Reference

`#include "BaseFabMacros.H"`

Include dependency graph for BaseFabImplem.H:

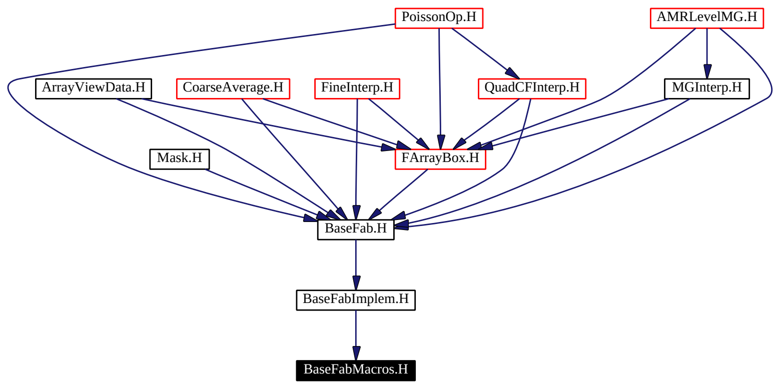


This graph shows which files directly or indirectly include this file:



7.20 BaseFabMacros.H File Reference

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)

7.20.1 Define Documentation

7.20.1.1 #define ForAllThis(T) ForAllThisBNN(T,domain,0,nComp())

7.20.1.2 #define ForAllThisB(T, b) ForAllThisBNN(T,(b),0,nComp())

7.20.1.3 #define ForAllThisC(T) ForAllThisCBNN(T,domain,0,nComp())

7.20.1.4 #define ForAllThisCB(T, b) ForAllThisCBNN(T,(b),0,nComp())

7.20.1.5 #define ForAllThisNN(T, ns, nc) ForAllThisBNN(T,domain,ns,nc)

7.20.1.6 #define ForAllThisXC(T, x) ForAllThisBNNXC(T,domain,0,nComp(),x,0)

7.20.1.7 #define ForAllX(T, x) ForAllXBNN(T,x,((x).box()),0,((x).nComp()))

7.20.1.8 #define ForAllXB(T, x, b) ForAllXBNN(T,x,(b),0,(x).nComp())

7.20.1.9 #define ForAllXBC(T, x, b) ForAllXCBNN(T,x,(b),0,(x).nComp())

7.20.1.10 #define ForAllXC(T, x) ForAllXCBNN(T,x,((x).box()),0,((x).nComp()))

7.20.1.11 #define ForAllXPencil(T, x) ForAllXBPencil(T,x,((x).box()),0,((x).nComp()))

7.21 basicScalBC.H File Reference

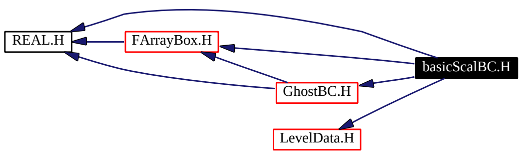
`#include "FArrayBox.H"`

`#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

7.22 basicVelBC.H File Reference

`#include "FArrayBox.H"`

`#include "FluxBox.H"`

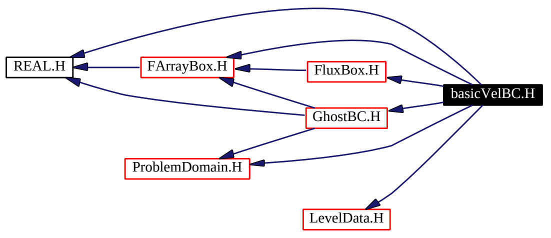
`#include "REAL.H"`

`#include "LevelData.H"`

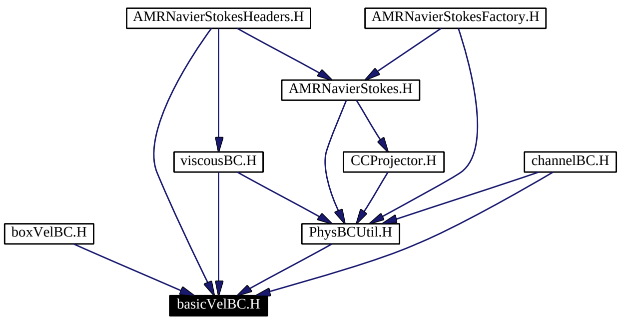
`#include "GhostBC.H"`

`#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**

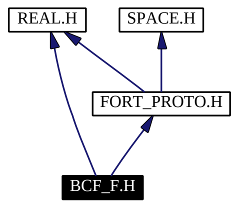
  does inflow

7.23 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))

7.23.1 Define Documentation

7.23.1.1 #define FORT_EXTRAPGHOSTBC FORTRAN_NAME(EXTRAPGHOSTBC ,extrapghostbc)

7.23.1.2 #define FORT_HOEXTRAPGHOSTBC FORTRAN_NAME(HOEXTRAPGHOSTBC ,hoextrapghostbc)

7.23.1.3 #define FORT_REFLECTGHOSTBC FORTRAN_NAME(REFLECTGHOSTBC ,reflectghostbc)

7.23.2 Function Documentation

7.23.2.1 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))

7.23.2.2 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))

7.23.2.3 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))

7.24 BiCGStabSmoother.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>`

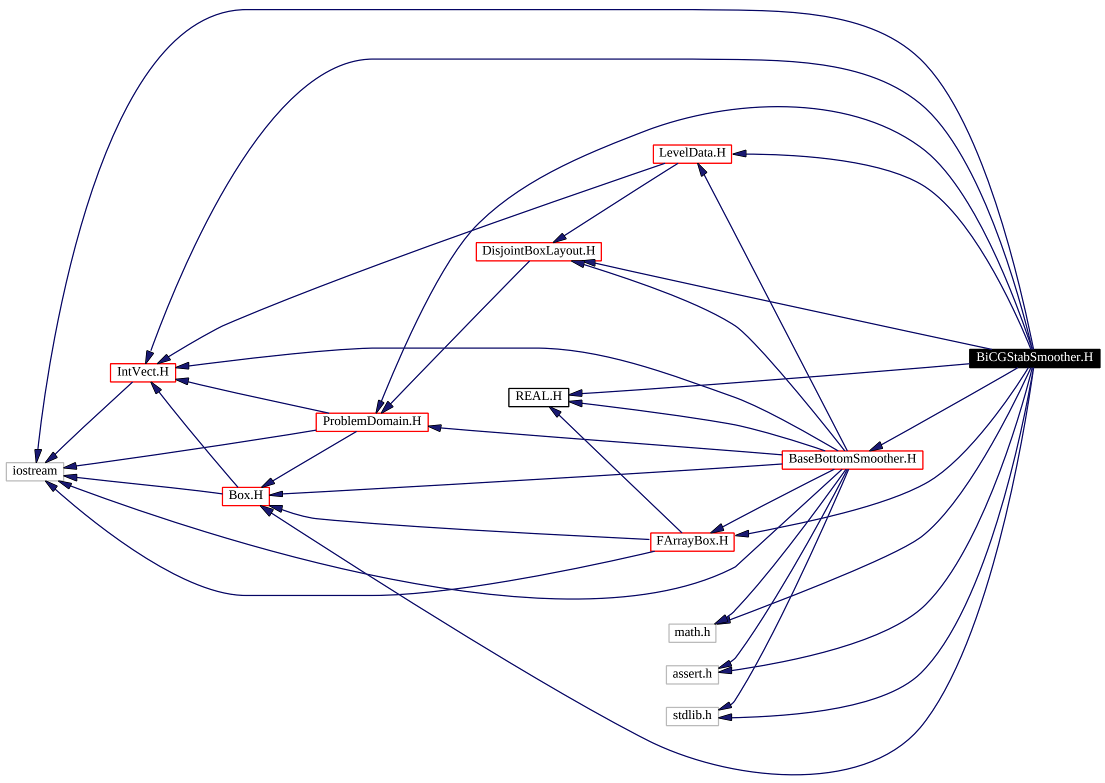
`#include <LevelData.H>`

`#include <FArrayBox.H>`

`#include <ProblemDomain.H>`

`#include "BaseBottomSmoother.H"`

Include dependency graph for BiCGStabSmoother.H:



Compounds

- class **BiCGStabSmoother**

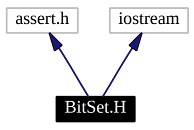
  Implements BiCGStab as a bottom smoother for **LevelOp** (p. 315)*-derived classes.*

7.25 BitSet.H File Reference

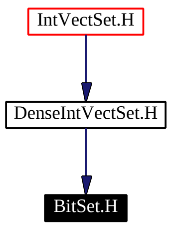
`#include <assert.h>`

`#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 **WORD** unsigned int

Variables

- const int **WORDSIZE** = 8*sizeof(unsigned int)

7.25.1 Define Documentation

7.25.1.1 #define WORD unsigned int

7.25.2 Variable Documentation

7.25.2.1 const int WORDSIZE = 8*sizeof(unsigned int)

7.26 Box.H File Reference

`#include "SPACE.H"`

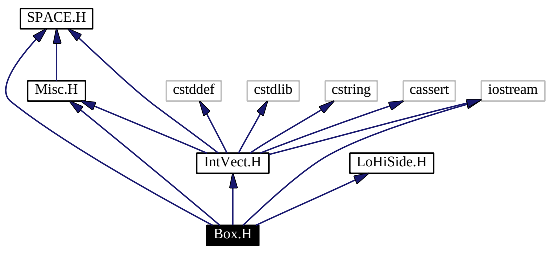
`#include <iostream>`

`#include "IntVect.H"`

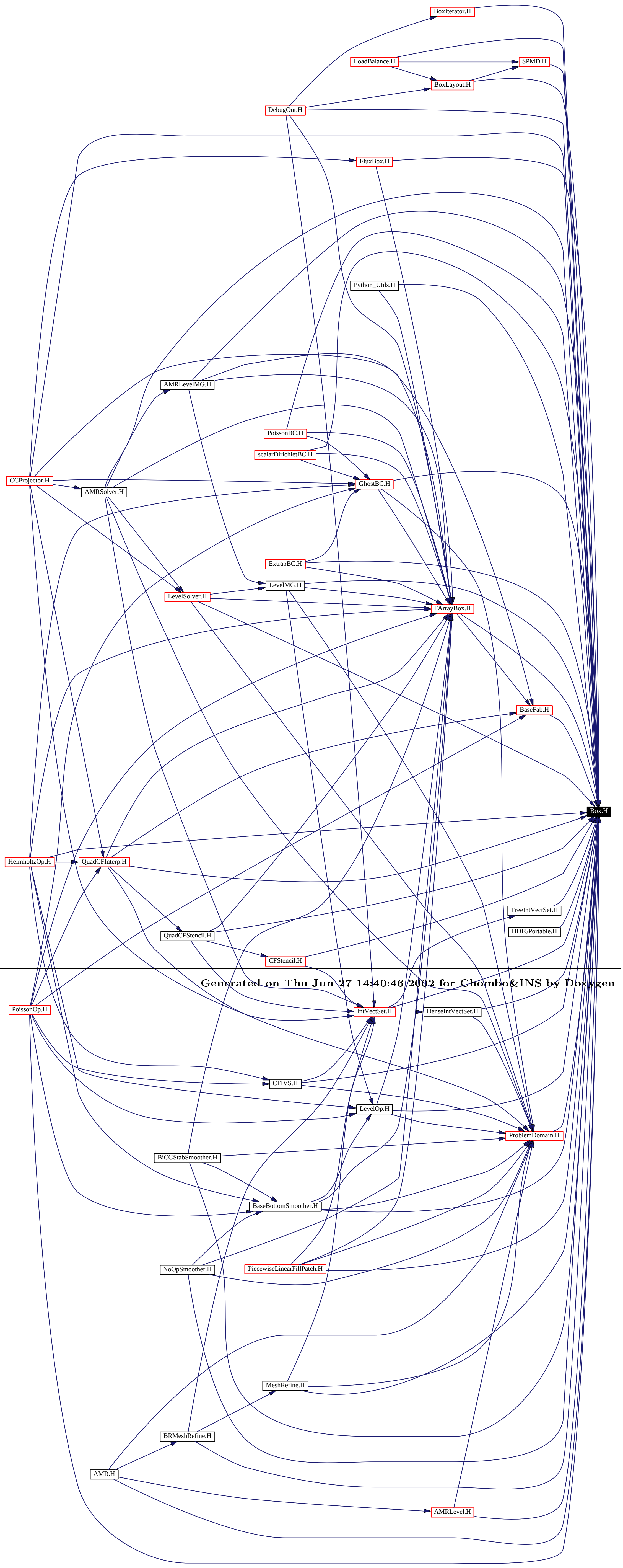
`#include "Misc.H"`

`#include "LoHiSide.H"`

Include dependency graph for Box.H:



This graph shows which files directly or indirectly include this file:



BoxIterator.H
LoadBalance.H
SPMD.H
BoxLayout.H
DebugOut.H
FluxBox.H
Python_Utils.H
AMRLevelMG.H
PoissonBC.H
scalarDirichletBC.H
CCProjector.H
AMRSolver.H
GhostBC.H
ExtrapBC.H
LevelMG.H
LevelSolver.H
FArrayBox.H
BaseFab.H
Box.H
HelmholtzOp.H
QuadCFInterp.H
TreeIntVectSet.H
HDF5Portable.H
QuadCFStencil.H
CFStencil.H
PoissonOp.H
IntVectSet.H
DenseIntVectSet.H
CFIVS.H
LevelOp.H
ProblemDomain.H
BiCGStabSmoother.H
BaseBottomSmoother.H
NoOpSmoother.H
PiecewiseLinearFillPatch.H
MeshRefine.H
BRMeshRefine.H
AMR.H
AMRLevel.H

Compounds

- class **Box**
- class **IndexType**

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)

7.26.1 Function Documentation

7.26.1.1 Box adjCellHi (const Box & *b*, int *dir*, int *len* = 1)

Returns the cell centered **Box** (p. 106) of the given length adjacent to the argument **Box** (p. 106) on the high end along the given coordinate direction. The return **Box** (p. 106) is identical to the argument **Box** (p. 106) in the other directions. The return **Box** (p. 106) and the argument **Box** (p. 106) have an empty intersection. \

NOTE: len >= 1. \

NOTE: **Box** (p. 106) retval = adjCellHi(b,dir,len) is equivalent to the following set of operations: \

Box (p. 106) 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** (p. 106) is an Empty **Box** (p. 106) of the appropriate type.

7.26.1.2 Box adjCellLo (const Box & *b*, int *dir*, int *len* = 1)

Returns the cell centered **Box** (p. 106) of the given length adjacent to the argument **Box** (p. 106) on the low end along the given coordinate direction. The return **Box** (p. 106) is identical to the argument **Box** (p. 106) in the other directions. The return **Box** (p. 106) and the argument **Box** (p. 106) have an empty intersection. \

NOTE: len >= 1. \

NOTE: **Box** (p. 106) retval = adjCellLo(b,dir,len) is equivalent to the following set of operations: \

Box (p. 106) 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** (p. 106) is an Empty **Box** (p. 106) of the appropriate type.

7.26.1.3 Box bdryBox (const Box & *b*, int *dir*, Side::LoHiSide *a_sd*, int *len* = 1)

7.26.1.4 Box bdryHi (const Box & *b*, int *dir*, int *len* = 1)

Returns the edge-centered **Box** (p. 106) (in direction dir) defining the high side of the argument **Box** (p. 106). The return **Box** (p. 106) 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** (p. 106) is an Empty **Box** (p. 106) of the appropriate type.

7.26.1.5 Box bdryLo (const Box & *b*, int *dir*, int *len* = 1)

Returns the edge-centered **Box** (p. 106) (in direction dir) defining the low side of the argument **Box** (p. 106). The output **Box** (p. 106) 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** (p. 106) is an Empty **Box** (p. 106) of the appropriate type.

7.26.1.6 Box enclosedCells (const Box & *b*) `[inline]`

Returns a **Box** (p. 106) with CELL based coordinates in all directions that is enclosed by argument **Box** (p. 106). The upper corner of the return **Box** (p. 106) will be one less in all components than the upper corner of the argument **Box** (p. 106). The Empty **Box** (p. 106) is not modified by this function except for changing of type.

7.26.1.7 Box enclosedCells (const Box & *b*, int *dir*) `[inline]`

Returns a **Box** (p. 106) with CELL based coordinates in direction dir that is enclosed by argument **Box** (p. 106). 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** (p. 106) is not modified by this function except for changing of type.

7.26.1.8 Box grow (const Box & *b*, const IntVect & *v*) `[inline]`

Returns a **Box** (p. 106) that is the argument **Box** (p. 106) grown in each direction by specified amount. The Empty **Box** (p. 106) is not modified by this function. NOTE: components of iv may be negative, which would return a shrunken **Box** (p. 106). If shrinking produces an empty **Box** (p. 106), the **Box** (p. 106) is transformed into the canonical Empty **Box** (p. 106).

7.26.1.9 Box grow (const Box & *b*, int *i*) `[inline]`

Returns a **Box** (p. 106) that is the argument **Box** (p. 106) grown in all directions by given amount. The Empty **Box** (p. 106) is not modified by this function. NOTE: n_cell negative shrinks the **Box** (p. 106) by that number of cells. If shrinking produces an empty **Box** (p. 106), the **Box** (p. 106) is transformed into the canonical Empty **Box** (p. 106).

7.26.1.10 Box minBox (const Box & *b1*, const Box & *b2*)

Returns a **Box** (p. 106) that is the minimum **Box** (p. 106) containing both the argument Boxes. Both Boxes must have identical type.

7.26.1.11 Box surroundingNodes (const Box & *b*) `[inline]`

Returns a **Box** (p. 106) with NODE based coordinates in all directions that encloses argument **Box** (p. 106). The upper corner of the return **Box** (p. 106) will be one more in all components than the upper corner of the argument **Box** (p. 106). The Empty **Box** (p. 106) is not modified by this function except for changing of type.

7.26.1.12 Box surroundingNodes (const Box & *b*, int *dir*) `[inline]`

Returns a **Box** (p. 106) with NODE based coordinates in direction dir that encloses the argument **Box** (p. 106). 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** (p. 106) is not modified by this function except for changing of type.

7.27 BoxIterator.H File Reference

`#include <cstdlib>`

`#include "Box.H"`

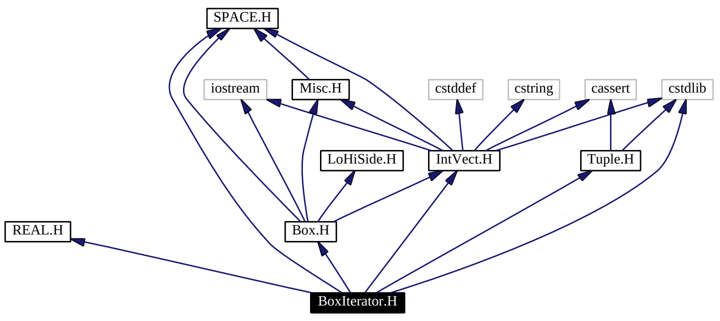
`#include "REAL.H"`

`#include "SPACE.H"`

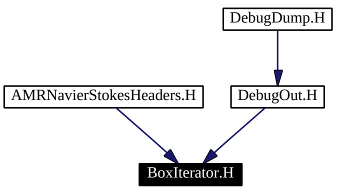
`#include "Tuple.H"`

`#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** (p. 106)

7.28 BoxLayout.H File Reference

`#include "Box.H"`

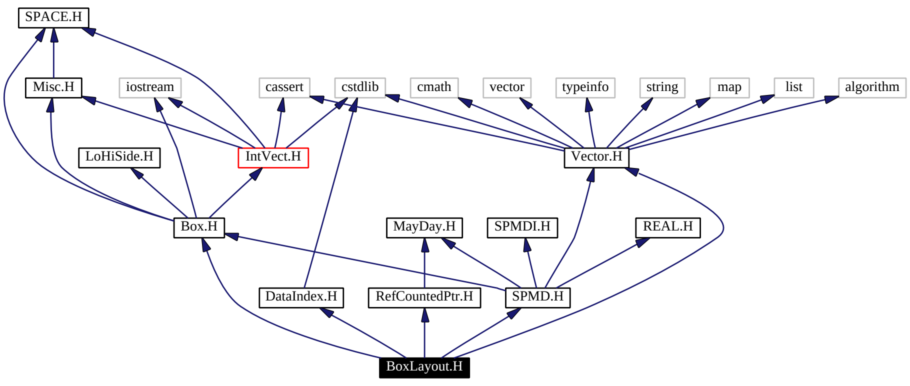
`#include "Vector.H"`

`#include "RefCountedPtr.H"`

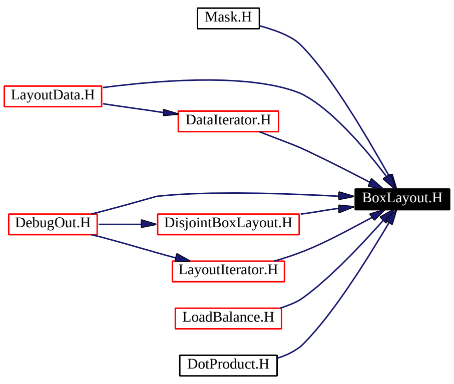
`#include "DataIndex.H"`

`#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)

7.28.1 Function Documentation

7.28.1.1 void coarsen_bl (BoxLayout & *output*, const BoxLayout & *input*, int *refinement*) `[inline]`

7.28.1.2 std::ostream& operator<< (std::ostream & *os*, const BoxLayout & *a_layout*)

7.28.1.3 void refine_bl (BoxLayout & *output*, const BoxLayout & *input*, int *refinement*) `[inline]`

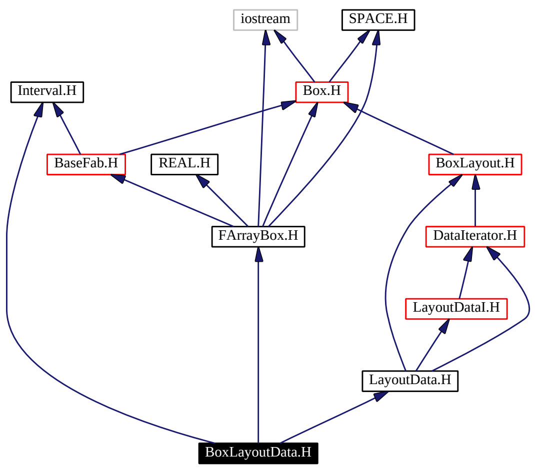
7.29 BoxLayoutData.H File Reference

`#include "LayoutData.H"`

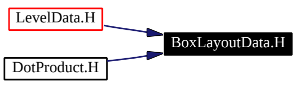
`#include "Interval.H"`

`#include "FArrayBox.H"`

Include dependency graph for BoxLayoutData.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **BoxLayoutData**

  Data on a **BoxLayout** (p. 139).

- class **DataFactory**

  Factory object to data members of a **BoxLayoutData** (p. 147) *container.*

- class **DefaultDataFactory**

  Factory object to data members of a **BoxLayoutData** (p. 147) *container.*

Functions

- **Real norm** (const **BoxLayoutData**< **FArrayBox** > &A, const **Interval** &interval, const int &p=2)

7.29.1 Function Documentation

7.29.1.1 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.

returns: (sum[| A[i][j,interval] |^p)^(1/p) p>=2, for all i in A, for valid regions j in A[i]

global max(A[i][j, interval]) p==0

to turn into an L-p norm, one needs to multiply by dx^(SpaceDim/p)

some people don't like that this norm is not normalized based on number of points in A. Normalization is your problem.

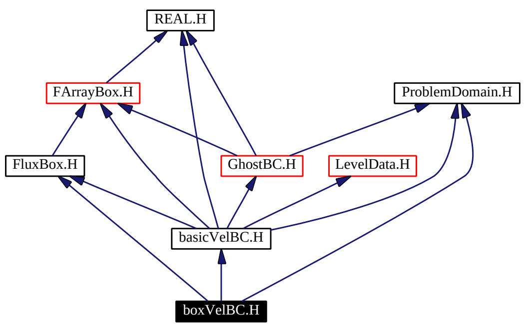
7.30 boxVelBC.H File Reference

`#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!

7.31 BRMeshRefine.H File Reference

`#include <climits>`

`#include "Vector.H"`

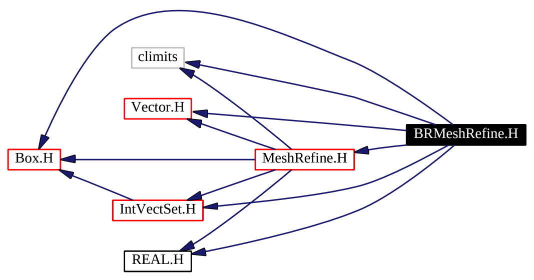
`#include "Box.H"`

`#include "IntVectSet.H"`

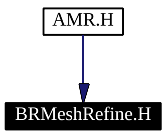
`#include "REAL.H"`

`#include "MeshRefine.H"`

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)
- void **domainSplit** (const **Box** &a_domain, **Vector**< **Box** > &a_vbox, int a_maxSize, int a_blockfactor=1)

7.31.1 Define Documentation

7.31.1.1 #define _BR_MIN_BOX_FILL_RATIO_ (0.75)

7.31.2 Function Documentation

7.31.2.1 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** (p. 153) object.

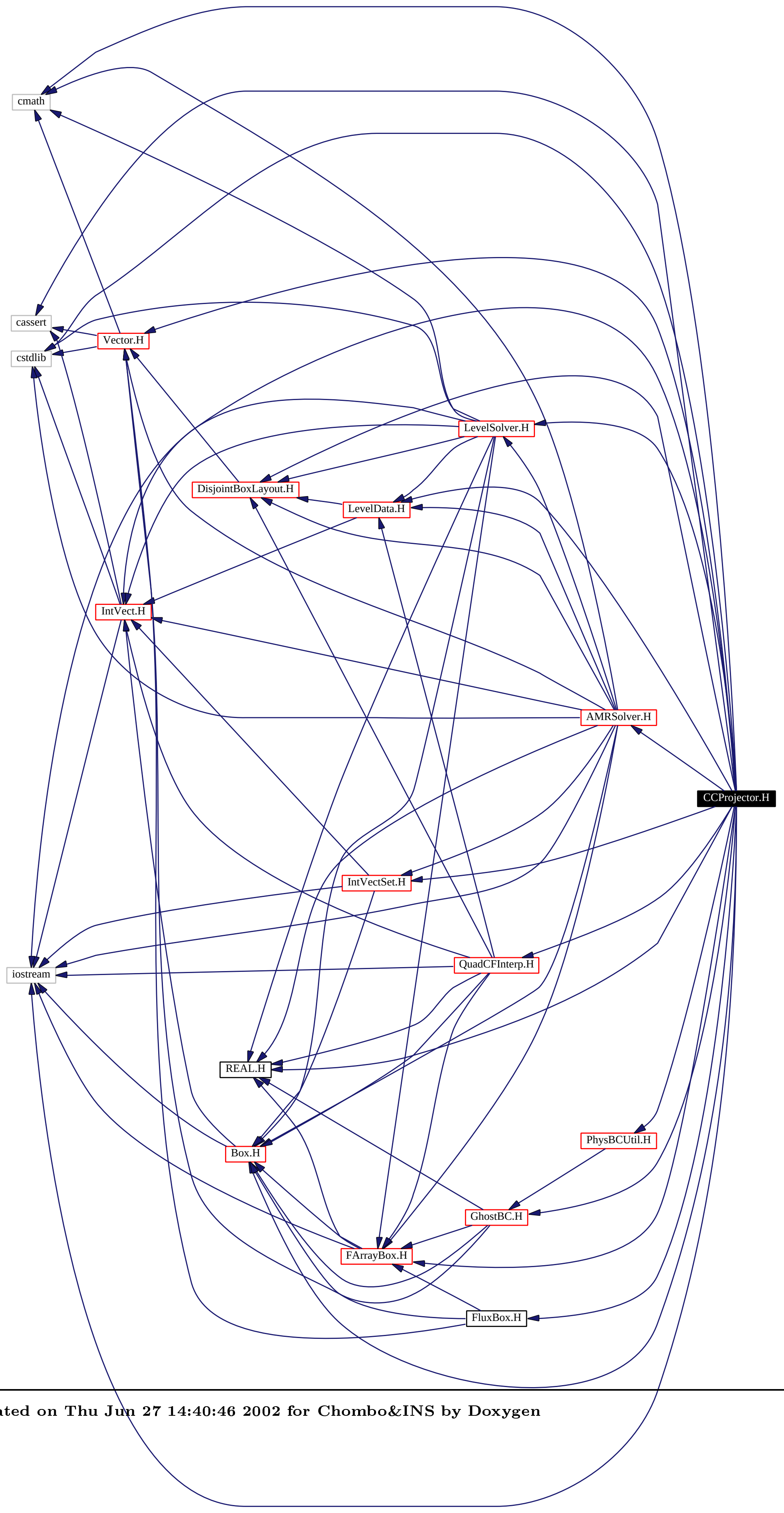
7.31.2.2 void domainSplit (const ProblemDomain & *a_domain*, Vector< Box > & *a_vbox*, int *a_maxSize*, int *a_blockfactor* = 1)

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** (p. 153) object.

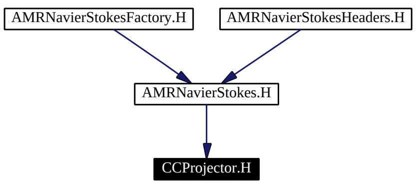
7.32 CCProjector.H File Reference

```
#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
cassert
cstdlib
Vector.H
LevelSolver.H
DisjointBoxLayout.H
LevelData.H
IntVect.H
AMRSolver.H
CCProjector.H
IntVectSet.H
iostream
QuadCFInterp.H
REAL.H
PhysBCUtil.H
Box.H
GhostBC.H
FArrayBox.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

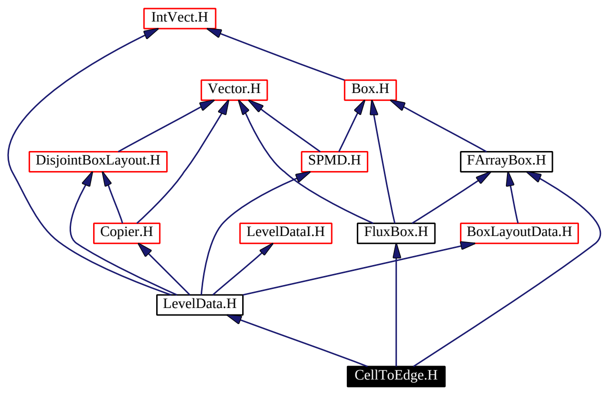
7.33 CellToEdge.H File Reference

`#include "LevelData.H"`

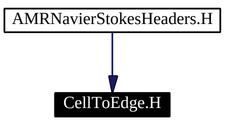
`#include "FArrayBox.H"`

`#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

7.33.1 Function Documentation

7.33.1.1 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

7.33.1.2 void CellToEdge (const FArrayBox & *a_cellData*, FArrayBox & *a_edgeData*, const int *a_dir*)

this one does a single direction

7.33.1.3 void CellToEdge (const FArrayBox & *a_cellData*, FluxBox & *a_edgeData*)

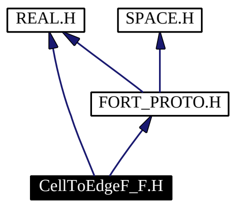
7.33.1.4 void CellToEdge (const LevelData< FArrayBox > & *a_cellData*, LevelData< FluxBox > & *a_edgeData*)

7.34 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))

7.34.1 Define Documentation

7.34.1.1 #define FORT_CELLTOEDGE FORTRAN_NAME(CELLTOEDGE ,celltoedge)

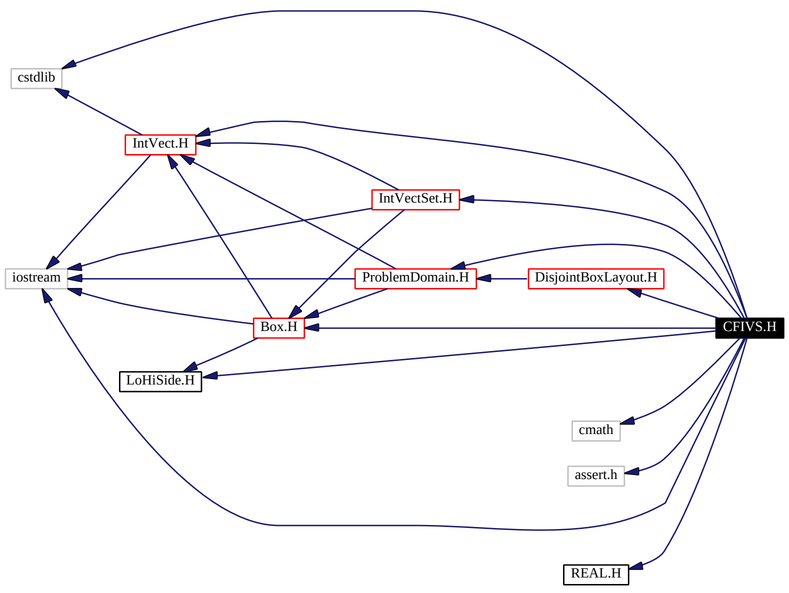
7.34.2 Function Documentation

7.34.2.1 void FORT_CELLTOEDGE (CHFp_CONST_FRA1(cellData), CHFp_FRA1(edgeData), CHFp_BOX(edgeBox), CHFp_CONST_INT(dir))

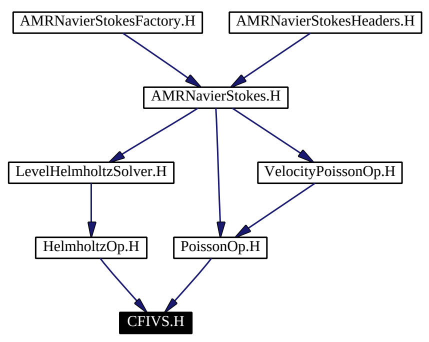
7.35 CFIVS.H File Reference

```
#include <iostream>
#include <cmath>
#include <assert.h>
#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

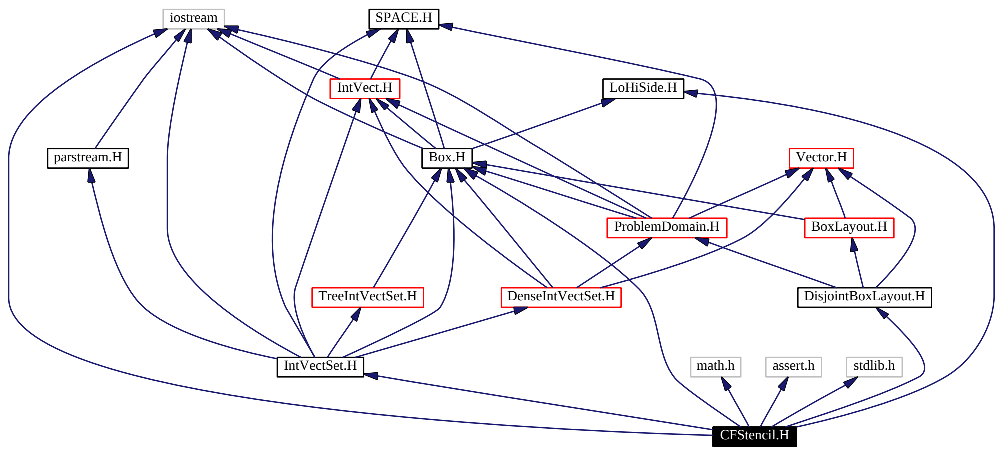
- class **CFIVS**

  Class to get fine intvectset at CF interface.

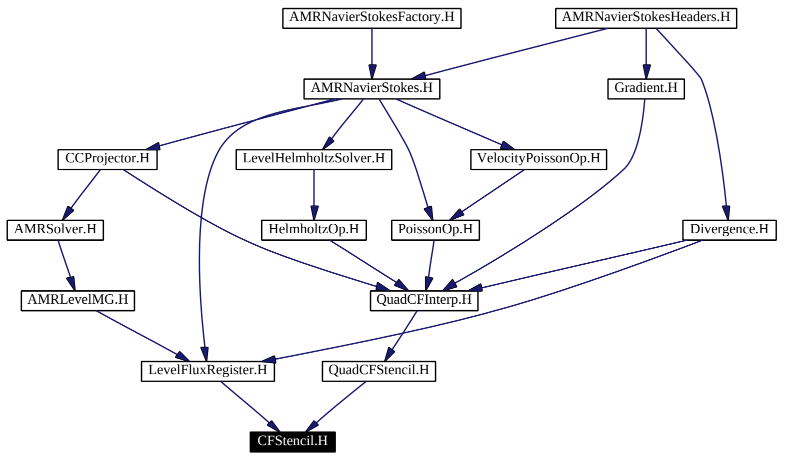
7.36 CFStencil.H File Reference

```
#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** (p. 17) *operations.*

7.37 channelBC.H File Reference

`#include "PhysBCUtil.H"`

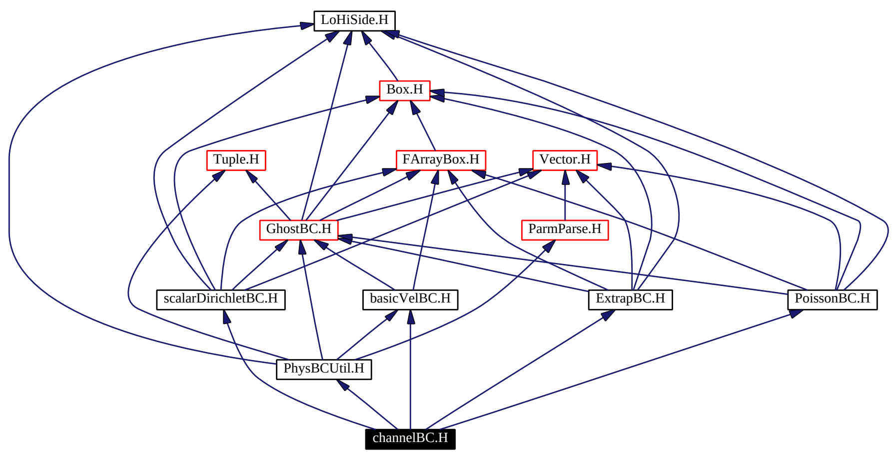
`#include "scalarDirichletBC.H"`

`#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)

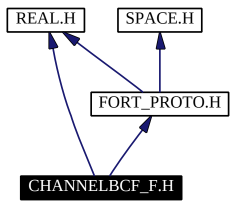
- class **NormalInflowVelBC**
- class **scalarInflowChannelBC**

7.38 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(CHANNELNORMINFLOWBC ,channelnorminflowbc)

Functions

- void **FORT_CHANNELNORMINFLOWBC** (CHFp_FRA1(inflowVel),CHFp_REAL(dx),CHFp_CONST_INT(dir),CHFp_BOX(domain))

7.38.1 Define Documentation

7.38.1.1 #define FORT_CHANNELNORMINFLOWBC FORTRAN_NAME(CHANNELNORMINFLOWBC ,channelnorminflowbc)

7.38.2 Function Documentation

7.38.2.1 void FORT_CHANNELNORMINFLOWBC (CHFp_FRA1(inflowVel), CHFp_REAL(dx), CHFp_CONST_INT(dir), CHFp_BOX(domain))

7.39 CHOMBO_VERSION.H File Reference

Defines

- #define **CHOMBO_VERSION** 1.2
- #define **CHOMBO_VERSION_MAJOR** 1
- #define **CHOMBO_VERSION_MINOR** 2

7.39.1 Define Documentation

7.39.1.1 #define CHOMBO_VERSION 1.2

7.39.1.2 #define CHOMBO_VERSION_MAJOR 1

7.39.1.3 #define CHOMBO_VERSION_MINOR 2

7.40 CoarseAverage.H File Reference

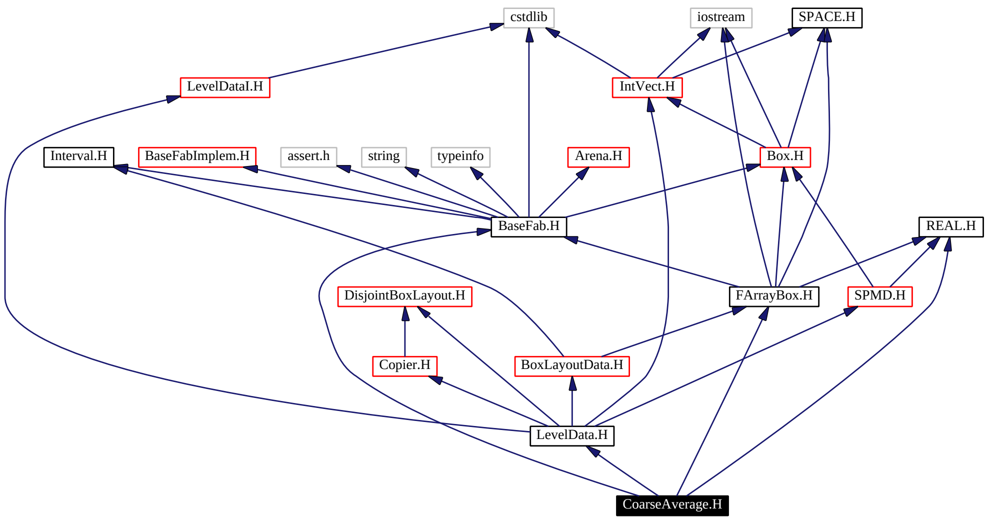
`#include "REAL.H"`

`#include "BaseFab.H"`

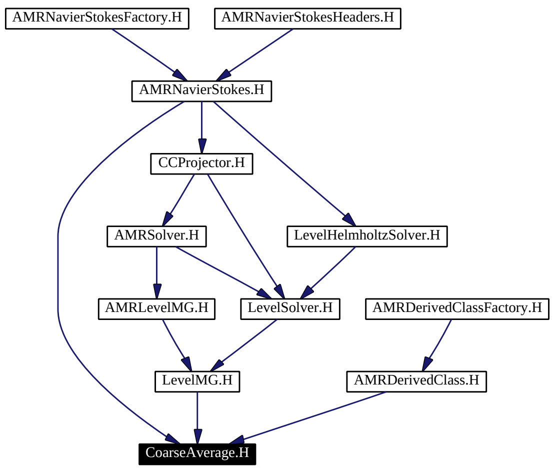
`#include "FArrayBox.H"`

`#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.

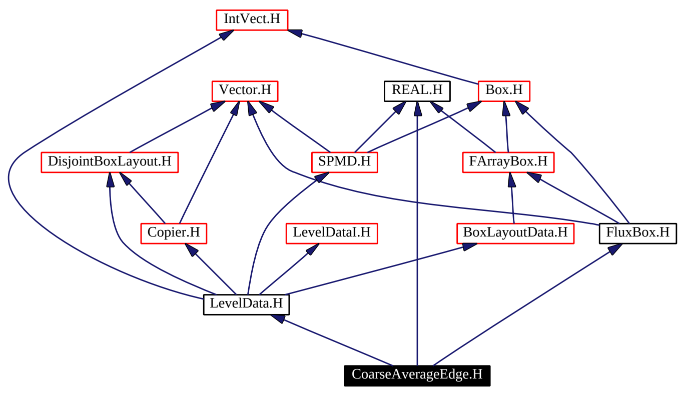
7.41 CoarseAverageEdge.H File Reference

`#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

7.42 computeNorm.H File Reference

`#include "REAL.H"`

`#include "Vector.H"`

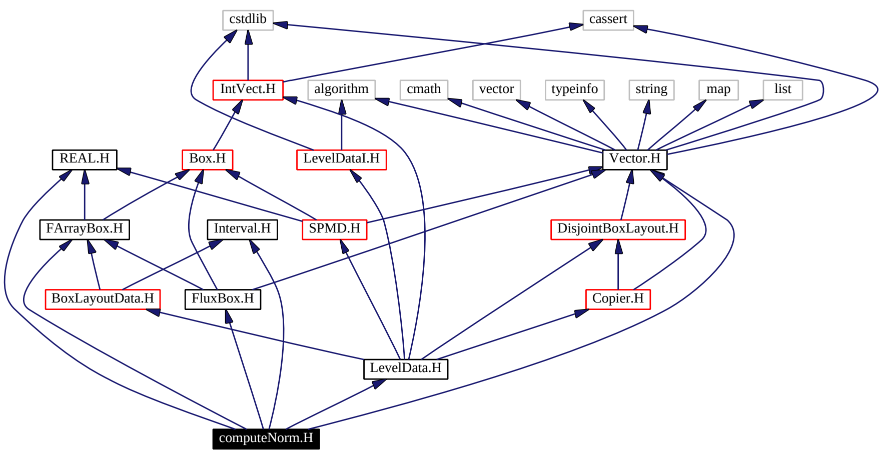
`#include "LevelData.H"`

`#include "FArrayBox.H"`

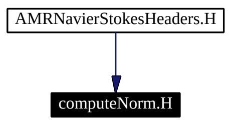
`#include "Interval.H"`

`#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 **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.

- **Real norm** (const **BoxLayoutData**< **FluxBox** > &a_layout, const **Interval** &interval, const int &p)

extension of the norm(...) function for FluxBoxes

7.42.1 Function Documentation

7.42.1.1 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.

7.42.1.2 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.

7.42.1.3 Real norm (const BoxLayoutData< FluxBox > & *a_layout*, const Interval & *interval*, const int & *p*)

extension of the norm(...) function for FluxBoxes

7.43 computeSum.H File Reference

`#include "REAL.H"`

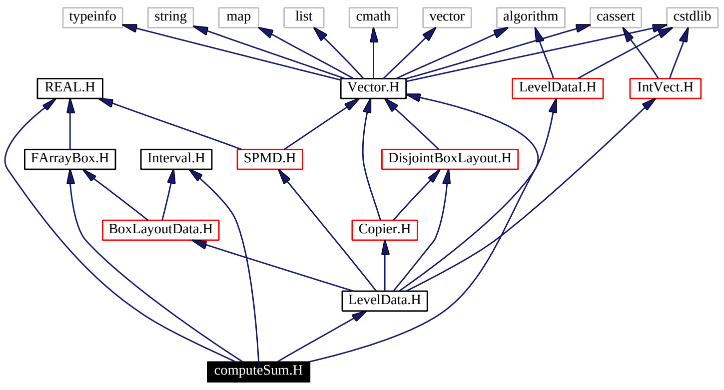
`#include "Vector.H"`

`#include "LevelData.H"`

`#include "FArrayBox.H"`

`#include "Interval.H"`

Include dependency graph for computeSum.H:



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.

7.43.1 Function Documentation

7.43.1.1 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.

7.43.1.2 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.

7.44 CONSTANTS.H File Reference

7.45 Copier.H File Reference

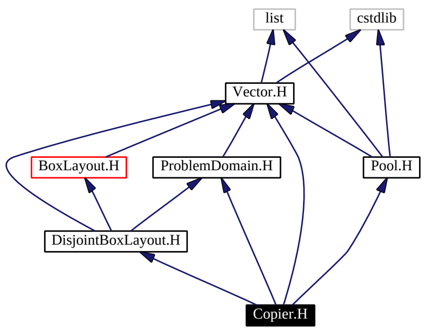
`#include "DisjointBoxLayout.H"`

`#include "Pool.H"`

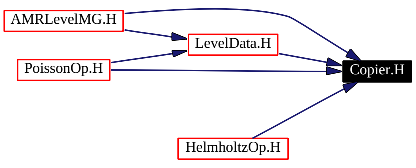
`#include "Vector.H"`

`#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)

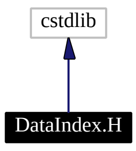
7.45.1 Function Documentation

7.45.1.1 std::ostream& operator<< (std::ostream & *os*, const Copier & *copier*)

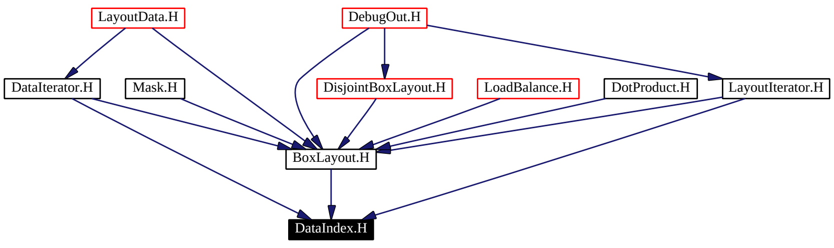
7.46 DataIndex.H File Reference

`#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** (p. 291).

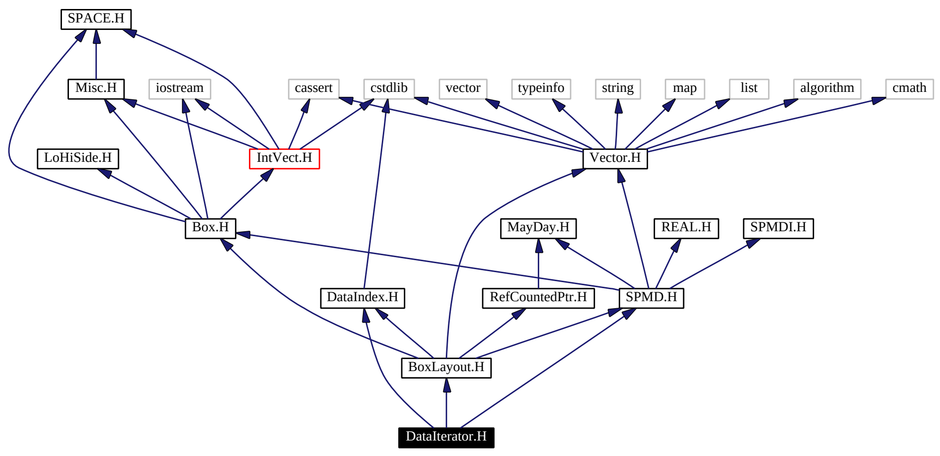
7.47 DataIterator.H File Reference

`#include "DataIndex.H"`

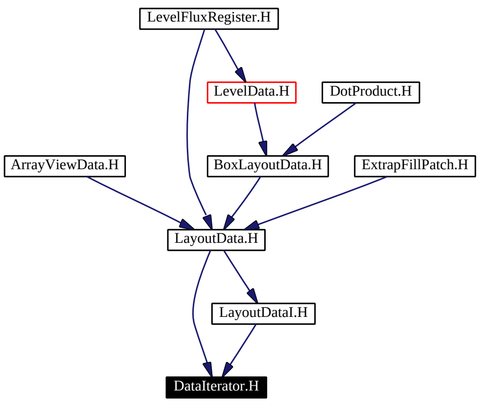
`#include "BoxLayout.H"`

`#include "SPMD.H"`

Include dependency graph for DataIterator.H:



This graph shows which files directly or indirectly include this file:



Compounds

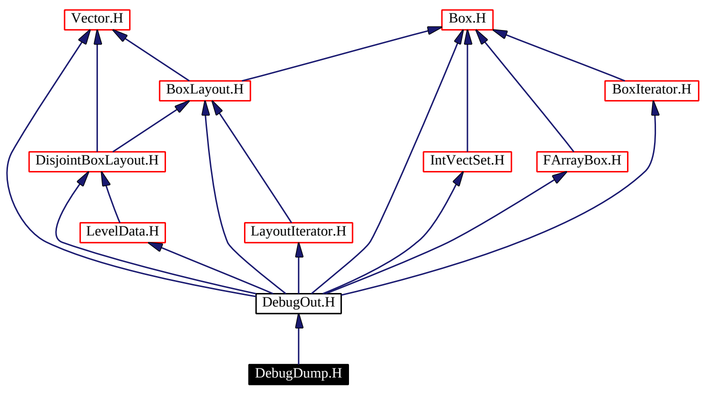
- class **DataIterator**

  An Iterator based on a **BoxLayout** (p. 139) *object.*

7.48 DebugDump.H File Reference

`#include "DebugOut.H"`

Include dependency graph for DebugDump.H:



Functions

- void **DontYouDareCallMe** ()

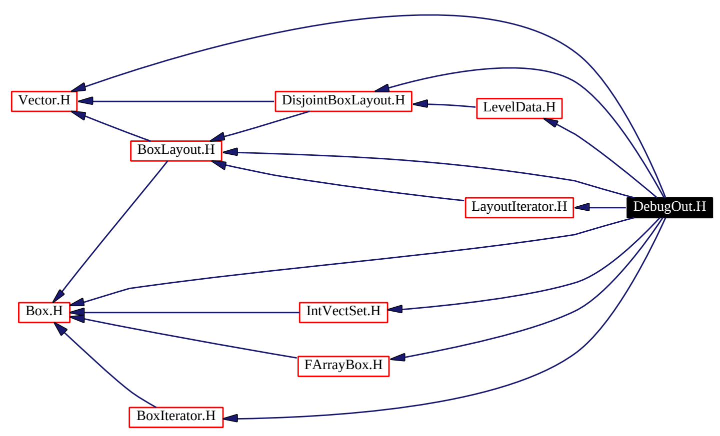
7.48.1 Function Documentation

7.48.1.1 void DontYouDareCallMe ()

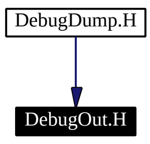
7.49 DebugOut.H File Reference

```
#include "DisjointBoxLayout.H"
#include "BoxLayout.H"
#include "Vector.H"
#include "Box.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 **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)

7.49.1 Function Documentation

7.49.1.1 void dumpBL (const BoxLayout * *a_dblInPtr*)

Dump inputs to standard out. For use inside debuggers.

7.49.1.2 void dumpBox (const Box * *a_boxPtr*)

Dump inputs to standard out. For use inside debuggers.

7.49.1.3 void dumpDBL (const DisjointBoxLayout * *a_dblInPtr*)

Dump inputs to standard out. For use inside debuggers.

7.49.1.4 void dumpIVS (const IntVectSet * *a_ivsInPtr*)

Dump inputs to standard out. For use inside debuggers.

7.49.1.5 void dumpLDF (const LevelData< FArrayBox > * *memLDF*)

Dump boxlayout of inputs to standard out. For use inside debuggers.

7.49.1.6 void dumpVBox (const Vector< Box > * *a_vectPtr*)

Dump inputs to standard out. For use inside debuggers.

7.49.1.7 void dumpVVBox (const Vector< Vector< Box > > * *a_vectPtr*)

Dump inputs to standard out. For use inside debuggers.

7.50 DenseIntVectSet.H File Reference

`#include "BitSet.H"`

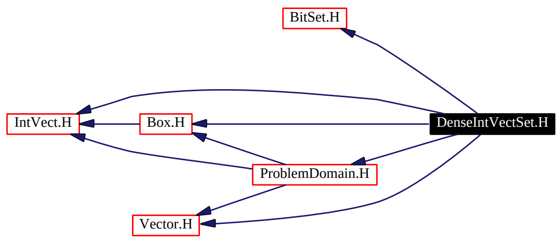
`#include "IntVect.H"`

`#include "Box.H"`

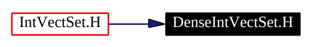
`#include "Vector.H"`

`#include "ProblemDomain.H"`

Include dependency graph for DenseIntVectSet.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DenseIntVectSet**
- class **DenseIntVectSetIterator**

7.51 DerivStencil.H File Reference

`#include <iostream>`

`#include <math.h>`

`#include <assert.h>`

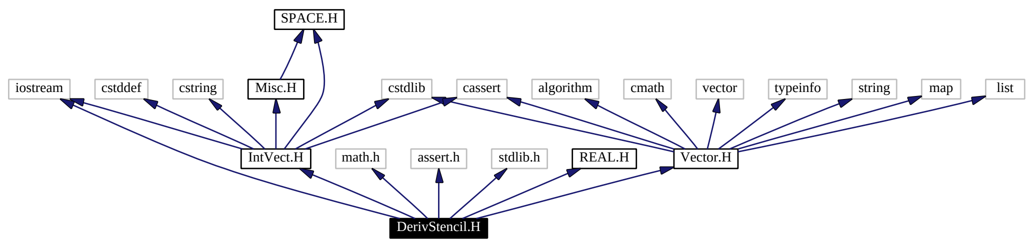
`#include <stdlib.h>`

`#include "REAL.H"`

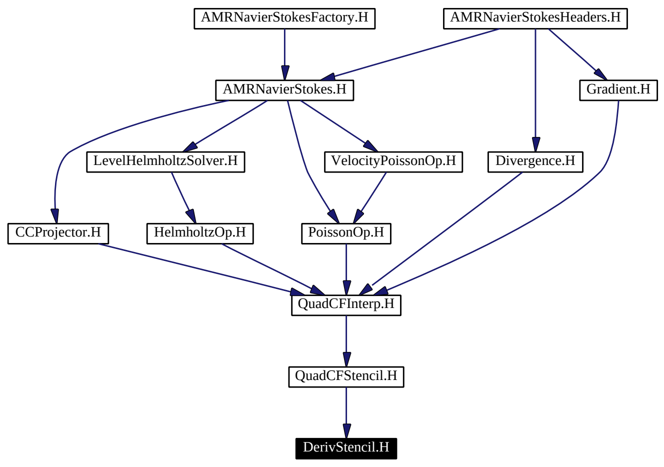
`#include "IntVect.H"`

`#include "Vector.H"`

Include dependency graph for DerivStencil.H:



This graph shows which files directly or indirectly include this file:



Compounds

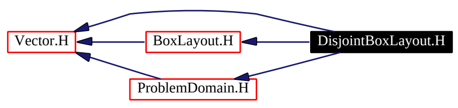
- class **DerivStencil**

  class to encapsulate the operations to create derivs on irreg stencils

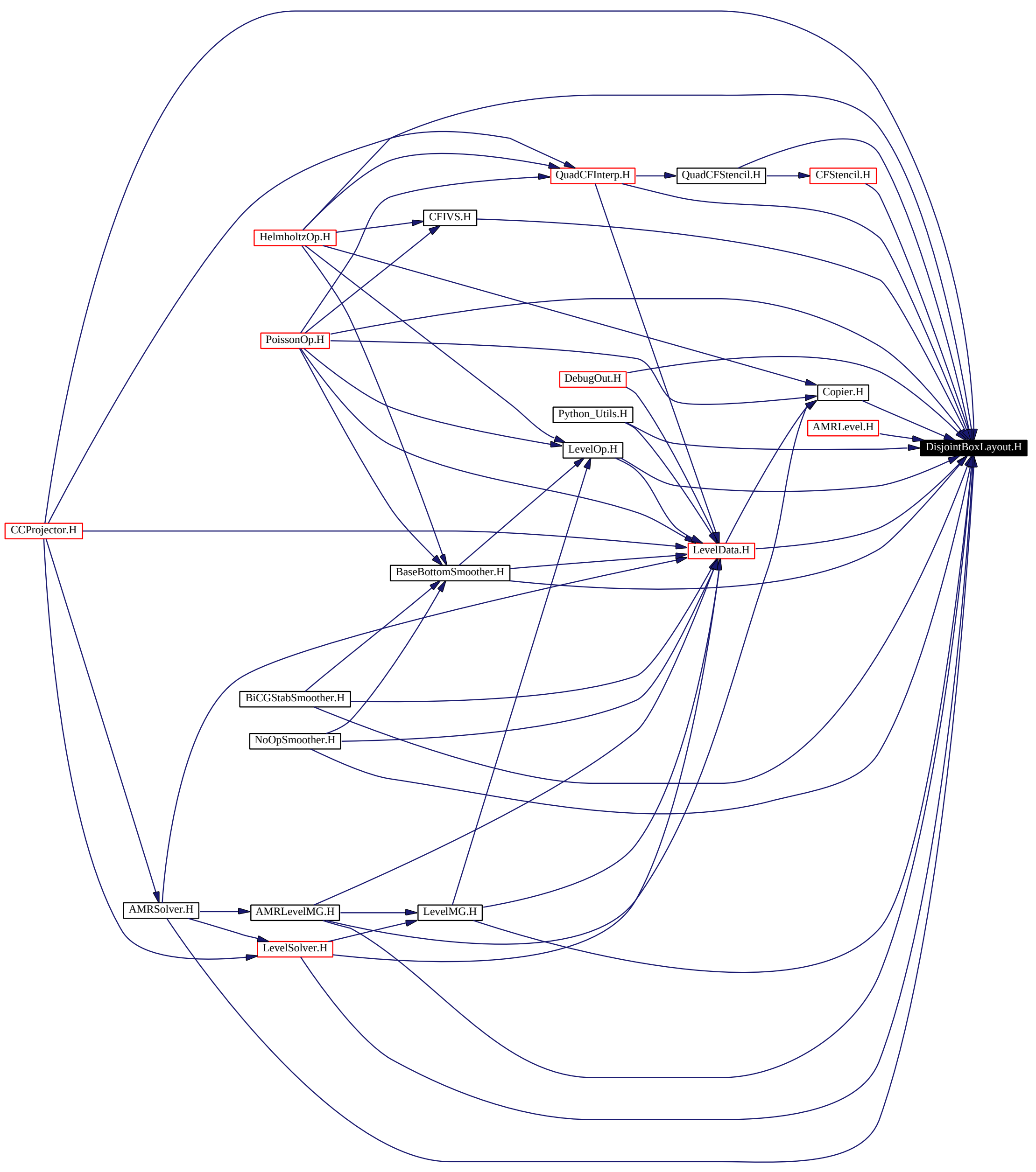
7.52 DisjointBoxLayout.H File Reference

```
#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** (p. 139) *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)

7.52.1 Function Documentation

7.52.1.1 void adjCellHi_dbl (DisjointBoxLayout & *a_output*, const DisjointBoxLayout & *a_input*, int *a_dir*, int *a_len*) `[inline]`

7.52.1.2 void adjCellLo_dbl (DisjointBoxLayout & *a_output*, const DisjointBoxLayout & *a_input*, int *a_dir*, int *a_len*) `[inline]`

7.52.1.3 void coarsen_dbl (DisjointBoxLayout & *output*, const DisjointBoxLayout & *input*, int *refinement*) `[inline]`

7.52.1.4 void refine_dbl (DisjointBoxLayout & *output*, const DisjointBoxLayout & *input*, int *refinement*) `[inline]`

7.53 Divergence.H File Reference

`#include "LevelData.H"`

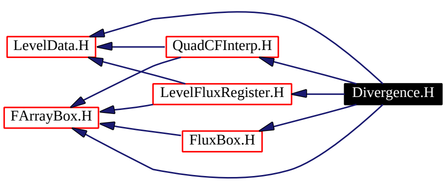
`#include "FArrayBox.H"`

`#include "FluxBox.H"`

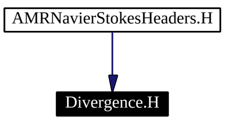
`#include "LevelFluxRegister.H"`

`#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_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)

7.53.1 Function Documentation

7.53.1.1 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*)

7.53.1.2 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*)

7.53.1.3 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*)

7.53.1.4 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*)

7.53.1.5 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*)

7.53.1.6 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*)

7.53.1.7 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*)

7.53.1.8 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*)

7.53.1.9 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*)

7.53.1.10 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*)

7.53.1.11 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*)

7.53.1.12 void levelDivergenceMAC (LevelData< FArrayBox > & *a_div*, const LevelData< FluxBox > & *a_uEdge*, const Real *a_dx*)

7.53.1.13 void simpleDivergenceMAC (FArrayBox & *a_div*, const FluxBox & *a_uEdge*, const Real *a_dx*)

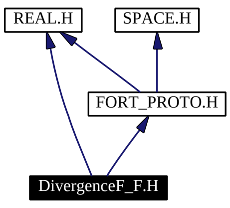
really basic single-fab MAC divergence

7.54 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))

7.54.1 Define Documentation

7.54.1.1 #define FORT_DIVERGENCE FORTRAN_NAME(DIVERGENCE ,divergence)

7.54.2 Function Documentation

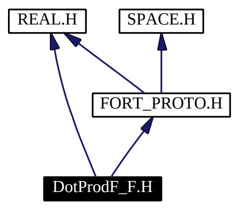
7.54.2.1 void FORT_DIVERGENCE (CHFp_CONST_FRA(uEdge), CHFp_FRA(div), CHFp_BOX(gridInt), CHFp_CONST_REAL(dx), CHFp_INT(dir))

7.55 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))

7.55.1 Define Documentation

7.55.1.1 #define FORT_DOTPRODUCT FORTRAN_NAME(DOTPRODUCT ,dotproduct)

7.55.2 Function Documentation

7.55.2.1 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))

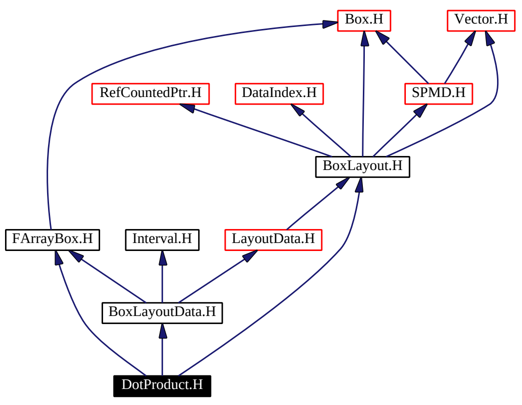
7.56 DotProduct.H File Reference

`#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)

7.56.1 Function Documentation

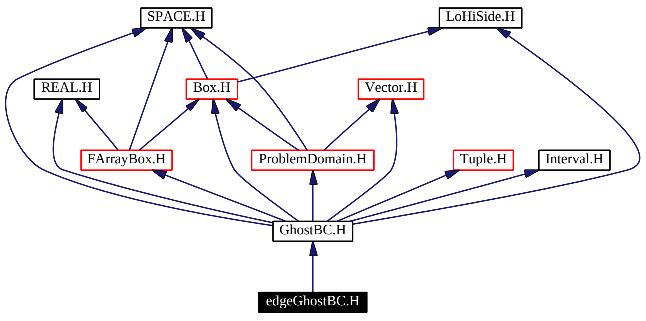
7.56.1.1 Real DotProduct (const BoxLayoutData< FArrayBox > & *a_dataOne*, const BoxLayoutData< FArrayBox > & *a_dataTwo*, const BoxLayout & *a_dblIn*, const Interval & *a_comps*)

7.56.1.2 Real DotProduct (const BoxLayoutData< FArrayBox > & *a_dataOne*, const BoxLayoutData< FArrayBox > & *a_dataTwo*, const BoxLayout & *a_dblIn*)

7.57 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**

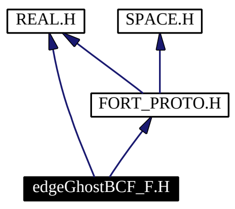
  class to do nothing on an edge

7.58 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))

7.58.1 Define Documentation

7.58.1.1 #define FORT_BOXEDGEBC FORTRAN_NAME(BOXEDGEBC ,boxedgebc)

7.58.2 Function Documentation

7.58.2.1 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))

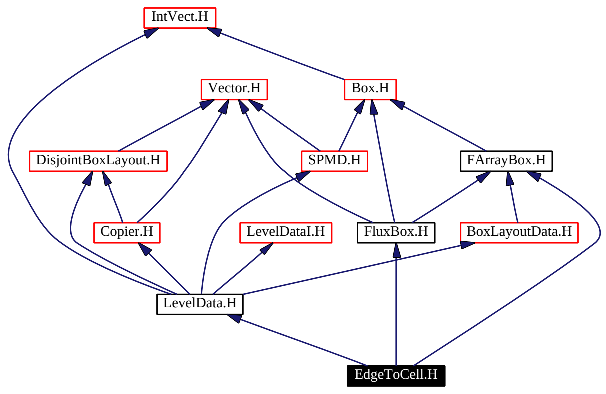
7.59 EdgeToCell.H File Reference

`#include "LevelData.H"`

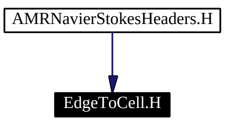
`#include "FArrayBox.H"`

`#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**< **FArrayBox** > &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

7.59.1 Function Documentation

7.59.1.1 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

7.59.1.2 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.

7.59.1.3 void EdgeToCell (const LevelData< FluxBox > & *a_edgeData*, LevelData< FArrayBox > & *a_cellData*)

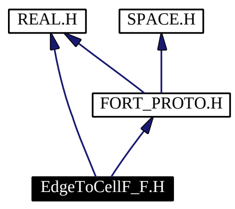
averages cell-centered cellData to edge-centered edgeData

7.60 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))

7.60.1 Define Documentation

7.60.1.1 #define FORT_EDGETOCELL FORTRAN_NAME(EDGETOCELL ,edgetocell)

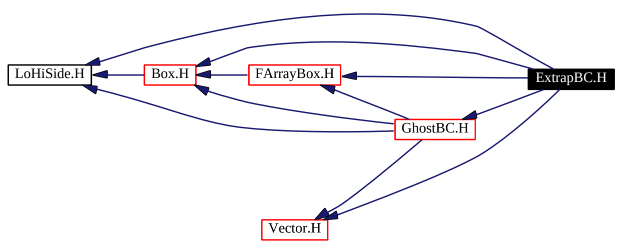
7.60.2 Function Documentation

7.60.2.1 void FORT_EDGETOCELL (CHFp_CONST_FRA1(edgeData), CHFp_FRA1(cellData), CHFp_BOX(cellBox), CHFp_CONST_INT(dir))

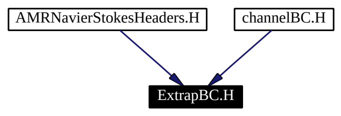
7.61 ExtrapBC.H File Reference

```
#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

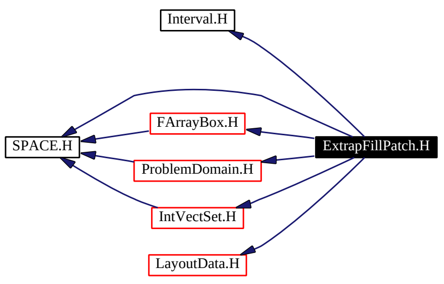
- class **ExtrapBC**

  Class to enforce homogeneous neumann boundary conditions.

7.62 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

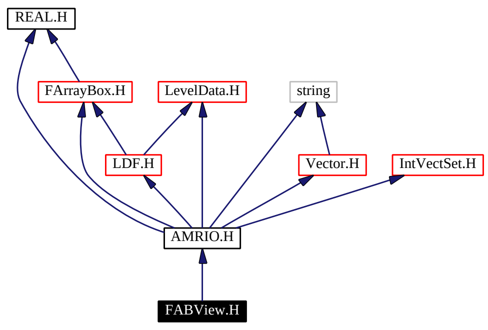
- class **ExtrapFillPatch**

    fills outer ghost cells by copying from inner ghost cells.

7.63 FABView.H File Reference

`#include "AMRIO.H"`

Include dependency graph for FABView.H:



Functions

- int **InitWriteFAB** ()

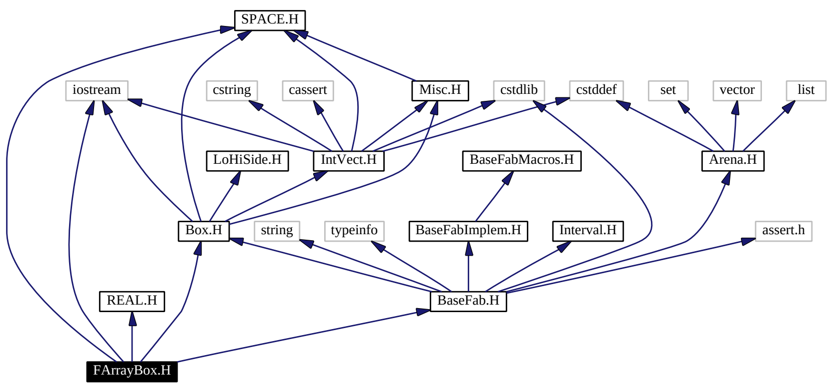
7.63.1 Function Documentation

7.63.1.1 int InitWriteFAB ()

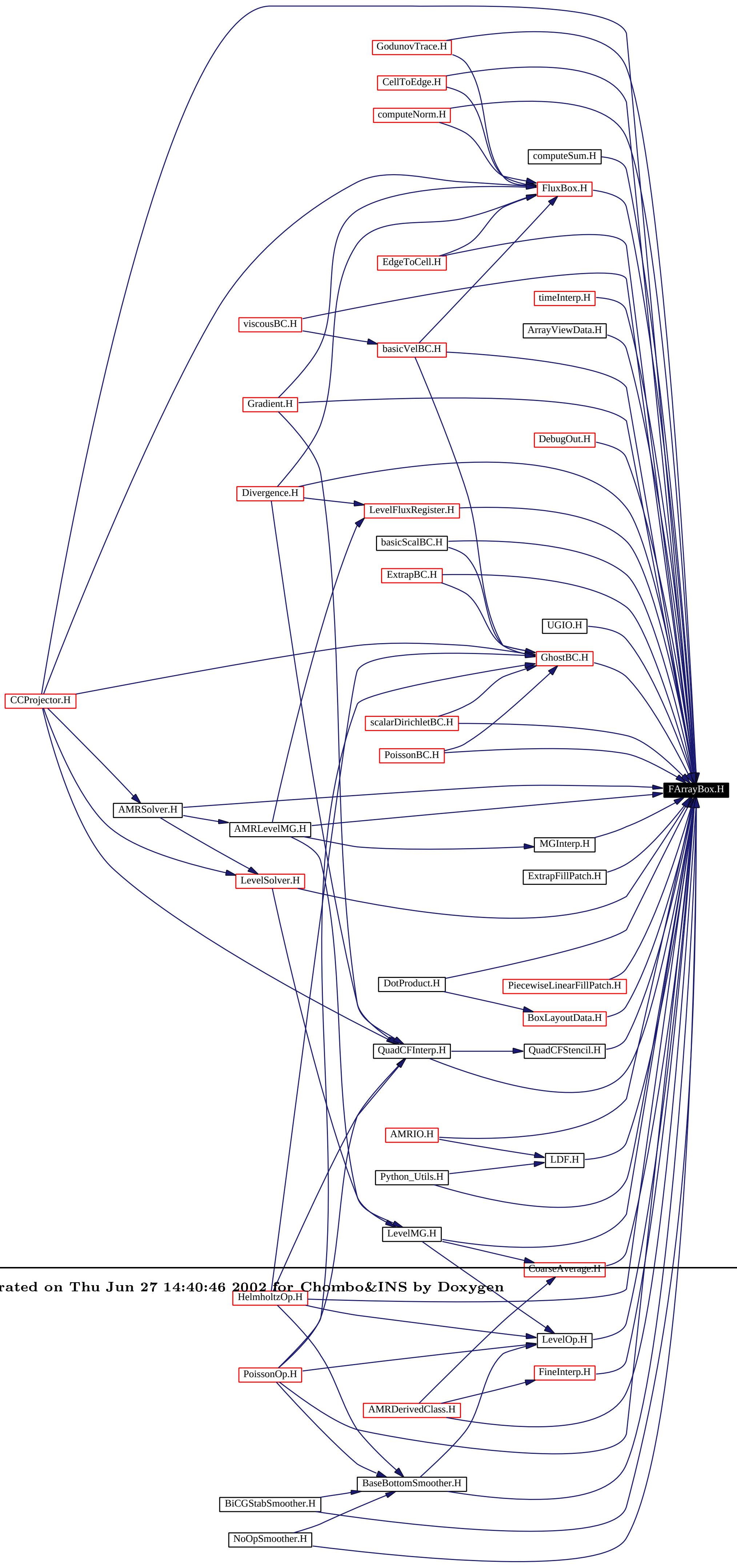
7.64 FArrayBox.H File Reference

```
#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:


GodunovTrace.H
CellToEdge.H
computeNorm.H
computeSum.H
FluxBox.H
EdgeToCell.H
timeInterp.H
viscousBC.H
ArrayViewData.H
basicVelBC.H
Gradient.H
DebugOut.H
Divergence.H
LevelFluxRegister.H
basicScalBC.H
ExtrapBC.H
UGIO.H
GhostBC.H
CCProjector.H
scalarDirichletBC.H
PoissonBC.H
FArrayBox.H
AMRSolver.H
AMRLevelMG.H
MGInterp.H
ExtrapFillPatch.H
LevelSolver.H
DotProduct.H
PiecewiseLinearFillPatch.H
BoxLayoutData.H
QuadCFInterp.H
QuadCFStencil.H
AMRIO.H
LDF.H
Python_Utils.H
LevelMG.H
CoarseAverage.H
HelmholtzOp.H
LevelOp.H
PoissonOp.H
FineInterp.H
AMRDerivedClass.H
BaseBottomSmoother.H
BiCGStabSmoother.H
NoOpSmoother.H

Compounds

- class **FArrayBox**

7.65 FineInterp.H File Reference

`#include "REAL.H"`

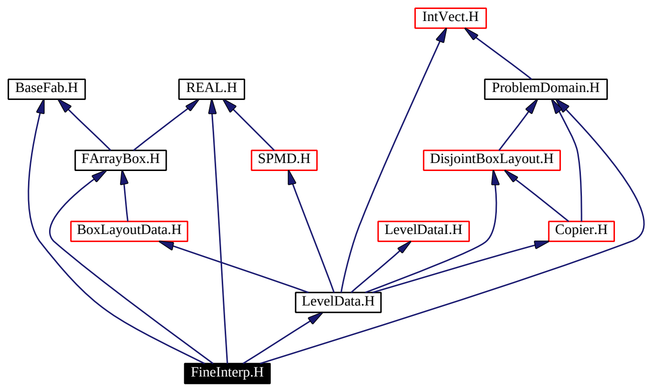
`#include "LevelData.H"`

`#include "BaseFab.H"`

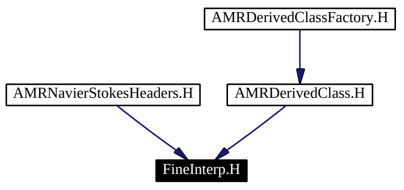
`#include "FArrayBox.H"`

`#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.

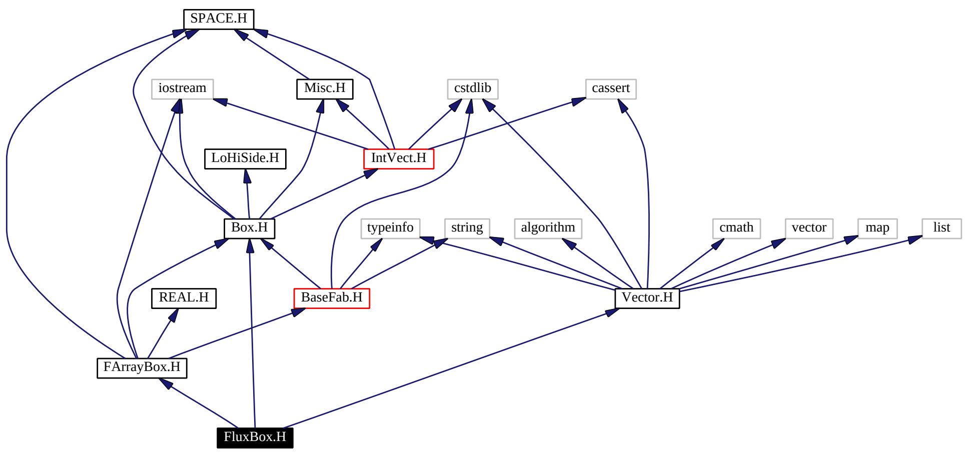
7.66 FluxBox.H File Reference

`#include "Box.H"`

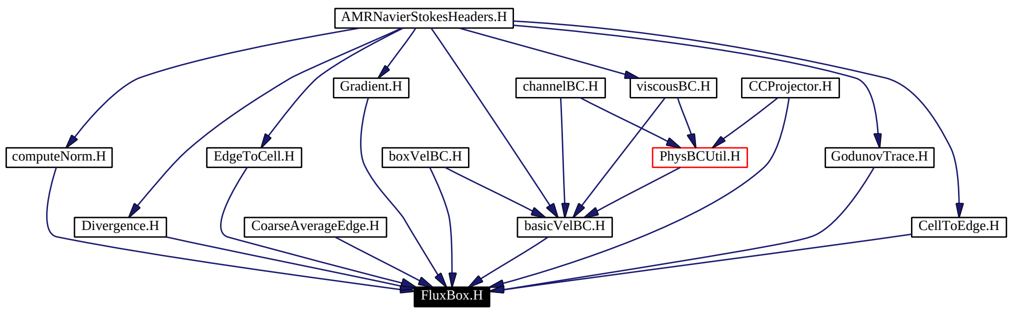
`#include "Vector.H"`

`#include "FArrayBox.H"`

Include dependency graph for FluxBox.H:



This graph shows which files directly or indirectly include this file:



Compounds

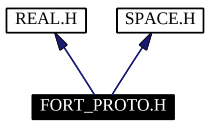
- class **FluxBox**

  A **FArrayBox** (p. 229)*-like container for edge-centered fluxes.*

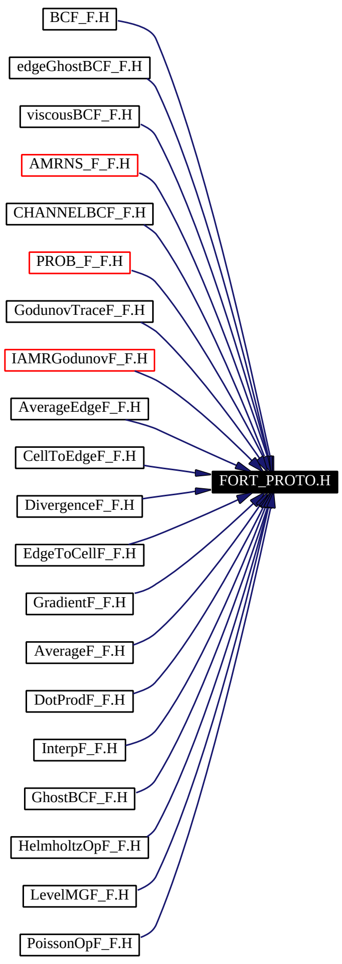
7.67 FORT_PROTO.H File Reference

```
#include "REAL.H"
#include "SPACE.H"
```

Include dependency graph for FORT_PROTO.H:



This graph shows which files directly or indirectly include this file:



Defines

- #define **FORTRAN_NAME**(NAME,name) name
- #define **CHF_INT**(i) &i
- #define **CHF_REAL**(r) &r

- #define **CHF_CONST_INT**(i) &i
- #define **CHF_CONST_REAL**(r) &r
- #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_INTVECT**(a) a.dataPtr( )
- #define **CHF_REALVECT**(a) a.dataPtr( )
- #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_REALVECT**(r) const **Real**∗ const r
- #define **CHFp_INTVECT**(r) const int∗ const r
- #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)

7.67.1 Define Documentation

7.67.1.1 #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] )
```

7.67.1.2 #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()
```

7.67.1.3 #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] )
```

7.67.1.4 #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()
```

7.67.1.5 #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] )
```

7.67.1.6 #define CHF_CONST_INT(i) &i

7.67.1.7 #define CHF_CONST_REAL(r) &r

7.67.1.8 #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()
```

7.67.1.9 #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] )
```

7.67.1.10 #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()
```

7.67.1.11 #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] )
```

7.67.1.12 #define CHF_INT(i) &i

7.67.1.13 #define CHF_INTVECT(a) a.dataPtr( )

7.67.1.14 #define CHF_REAL(r) &r

7.67.1.15 #define CHF_REALVECT(a) a.dataPtr( )

7.67.1.16 #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 )
```

7.67.1.17 #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
```

7.67.1.18 #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 )
```

7.67.1.19 #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
```

7.67.1.20 #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 )
```

7.67.1.21 #define CHFp_CONST_INT(i) const int∗ const i

7.67.1.22 #define CHFp_CONST_REAL(r) const Real∗ const r

7.67.1.23 #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
```

7.67.1.24 #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 )
```

7.67.1.25 #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
```

7.67.1.26 #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 )
```

7.67.1.27 #define CHFp_INT(i) int* const i

7.67.1.28 #define CHFp_INTVECT(r) const int* const r

7.67.1.29 #define CHFp_REAL(r) Real* const r

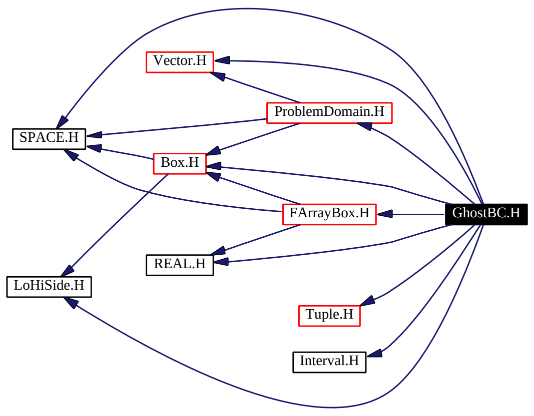
7.67.1.30 #define CHFp_REALVECT(r) const Real* const r

7.67.1.31 #define FORTRAN_NAME(NAME, name) name

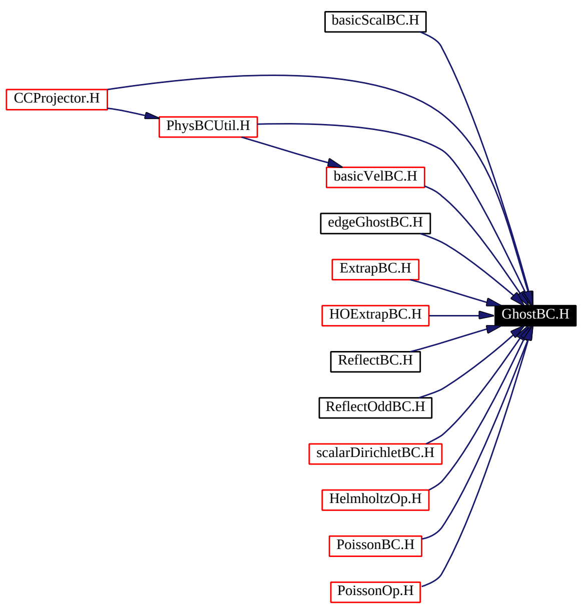
7.68 GhostBC.H File Reference

```
#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**

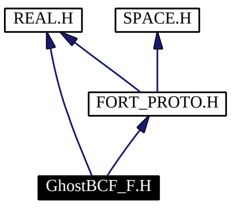
  Class to enforce ghost-cell based boundary conditions.

7.69 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))

7.69.1 Define Documentation

7.69.1.1 #define FORT_BOXGHOSTBC FORTRAN_NAME(BOXGHOSTBC ,boxghostbc)

7.69.2 Function Documentation

7.69.2.1 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))

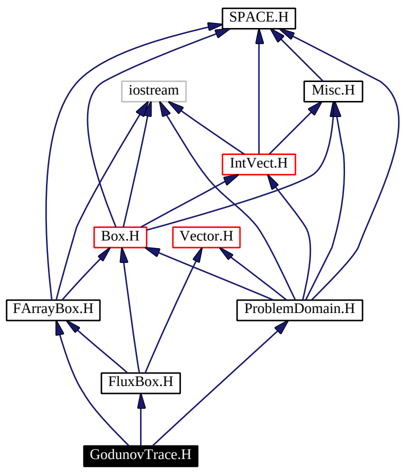
7.70 GodunovTrace.H File Reference

`#include "FluxBox.H"`

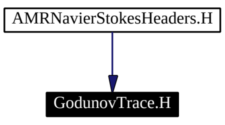
`#include "FArrayBox.H"`

`#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)

7.70.1 Function Documentation

7.70.1.1 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,

7.70.1.2 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,

7.70.1.3 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,

7.70.1.4 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

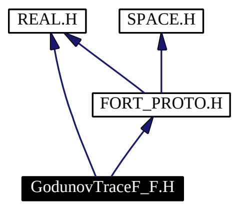
a_destComps where to put traced components in a_stateHalf,

7.71 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),CHFp_FRA1(dUpS))
- 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))

7.71.1 Define Documentation

7.71.1.1 #define FORT_PREDICT FORTRAN_NAME(PREDICT ,predict)

7.71.1.2 #define FORT_SLOPES FORTRAN_NAME(SLOPES ,slopes)

7.71.1.3 #define FORT_TRANSVERSE FORTRAN_NAME(TRANSVERSE ,transverse)

7.71.1.4 #define FORT_TRANSVERSECROSS FORTRAN_NAME(TRANSVERSECROSS ,transversecross)

7.71.1.5 #define FORT_UPWINDCELLTOEDGE FORTRAN_NAME(UPWINDCELLTOEDGE ,upwindcelltoedge)

7.71.2 Function Documentation

7.71.2.1 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))

7.71.2.2 void FORT_SLOPES (CHFp_FRA1(delS), CHFp_CONST_-FRA1(s), CHFp_BOX(gridBox), CHFp_CONST_INT(dir), CHFp_CONST_INT(doLimit))

7.71.2.3 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), CHFp_FRA1(dUpS))

7.71.2.4 void FORT_TRANSVERSECROSS (CHFp_FRA1(sHat), CHFp_CONST_FRA1(s), CHFp_CONST_FRA(vel), CHFp_BOX(grid-Box), CHFp_CONST_REAL(dt), CHFp_CONST_REAL(dx), CHFp_CONST_INT(dir))

7.71.2.5 void FORT_UPWINDCELLTOEDGE (CHFp_FRA1(edgeState), CHFp_CONST_FRA1(cellState), CHFp_CONST_FRA(edgeVel), CHFp_BOX(edgeBox), CHFp_CONST_INT(dir))

7.72 Gradient.H File Reference

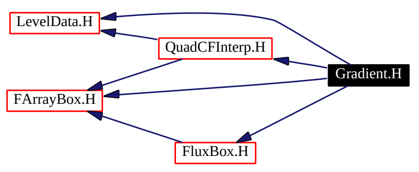
`#include "LevelData.H"`

`#include "FArrayBox.H"`

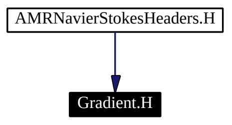
`#include "FluxBox.H"`

`#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)

7.72.1 Function Documentation

7.72.1.1 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*)

7.72.1.2 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*)

7.72.1.3 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*)

7.72.1.4 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*)

7.72.1.5 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*)

7.72.1.6 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*)

7.72.1.7 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*)

7.72.1.8 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*)

7.72.1.9 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*)

7.72.1.10 void levelGradientCC (LevelData< FArrayBox > & *a_grad*, const LevelData< FArrayBox > & *a_phi*, const Real *a_dx*)

7.72.1.11 void levelGradientCC (LevelData< FArrayBox > & *a_grad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const Real *a_dx*, QuadCFInterp & *a_cfInterp*)

7.72.1.12 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*)

7.72.1.13 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*)

7.72.1.14 void levelGradientMAC (LevelData< FluxBox > & *a_edgeGrad*, const LevelData< FArrayBox > & *a_phi*, const Real *a_dx*)

7.72.1.15 void levelGradientMAC (LevelData< FluxBox > & *a_edgeGrad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const Real *a_dx*, QuadCFInterp & *a_cfInterpCrse*)

7.72.1.16 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*)

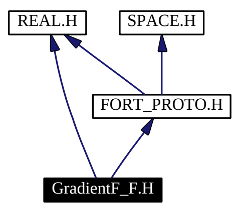
7.72.1.17 void levelGradientMAC (LevelData< FluxBox > & *a_edgeGrad*, LevelData< FArrayBox > & *a_phi*, const LevelData< FArrayBox > ∗ *a_phiCrsePtr*, const Real *a_dx*, const int *a_nRefCrse*, const ProblemDomain & *a_dProblem*)

7.73 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))

7.73.1 Define Documentation

7.73.1.1 #define FORT_CRSEONESIDEGRAD FORTRAN_NAME(CRSEONESIDEGRAD ,crseonesidegrad)

7.73.1.2 #define FORT_GRADCC FORTRAN_NAME(GRADCC ,gradcc)

7.73.1.3 #define FORT_MACGRAD FORTRAN_NAME(MACGRAD ,macgrad)

7.73.2 Function Documentation

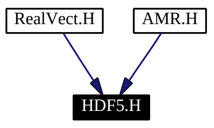
7.73.2.1 void FORT_CRSEONESIDEGRAD (CHFp_FRA1(edgeGrad), CHFp_FIA1(mask), CHFp_BOX(loEdgeBox), CHFp_BOX(hiEdgeBox), CHFp_INT(dir), CHFp_INT(doLo), CHFp_INT(doHi))

7.73.2.2 void FORT_GRADCC (CHFp_FRA1(gradPhi), CHFp_CONST_-FRA1(phi), CHFp_BOX(gridBox), CHFp_CONST_REAL(dx), CHFp_INT(dir))

7.73.2.3 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))

7.74 HDF5.H File Reference

This graph shows which files directly or indirectly include this file:



7.75 HDF5Portable.H File Reference

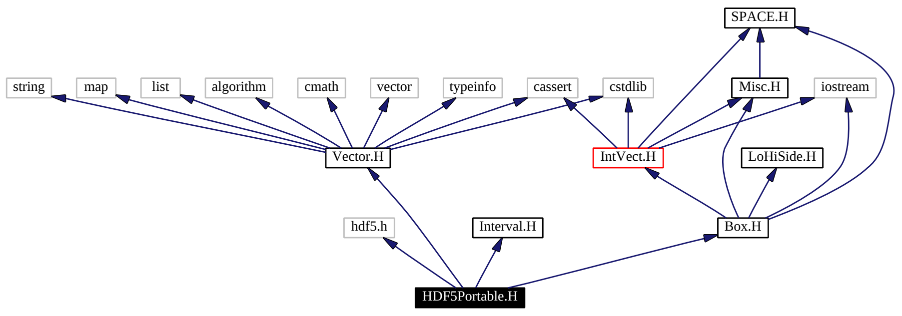
`#include "Vector.H"`

`#include <hdf5.h>`

`#include "Box.H"`

`#include "Interval.H"`

Include dependency graph for HDF5Portable.H:



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)

7.75.1 Function Documentation

7.75.1.1 template<class T> void dataSize (const T & *item*, Vector< int > & *a_sizes*, const Box & *box*, const Interval & *comps*)

7.75.1.2 template<class T> void dataTypes (Vector< hid_t > & *a_types*, const T & *a_dummySpecializationArg*)

7.75.1.3 template<class T> const char* name (const T & *a_dummySpecializationArg*)

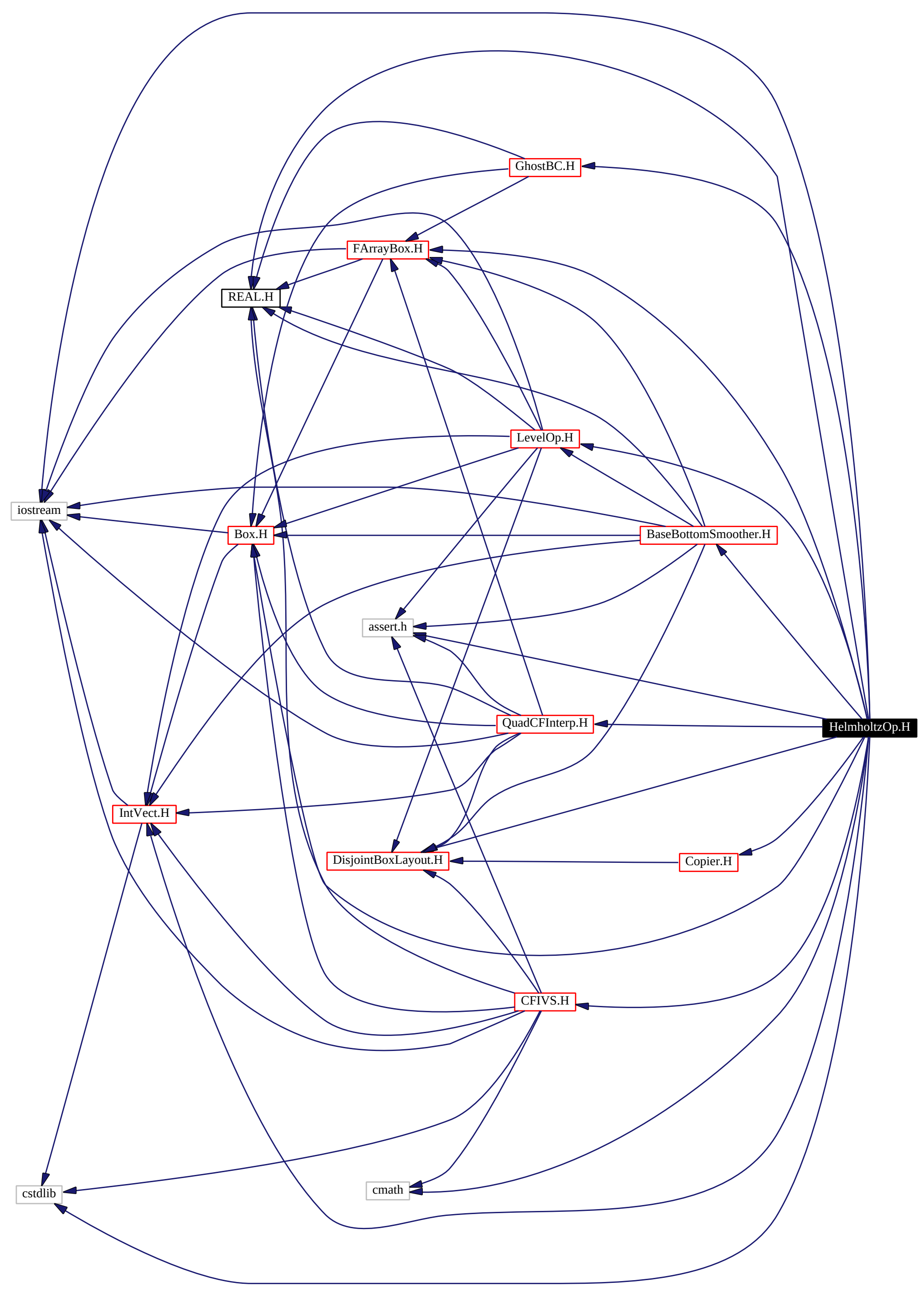
7.75.1.4 template<class T> void read (T & *item*, Vector< void * > & *a_allocatedBuffers*, const Box & *box*, const Interval & *comps*)

7.75.1.5 template<class T> void write (const T & *item*, Vector< void * > & *a_allocatedBuffers*, const Box & *box*, const Interval & *comps*)

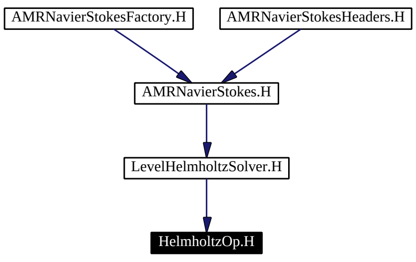
7.76 HelmholtzOp.H File Reference

```
#include <iostream>
#include <cmath>
#include <assert.h>
#include <cstdlib>
#include <REAL.H>
#include <IntVect.H>
#include <Box.H>
#include <DisjointBoxLayout.H>
#include <FArrayBox.H>
#include "GhostBC.H"
#include "LevelOp.H"
#include "QuadCFInterp.H"
#include "CFIVS.H"
#include "Copier.H"
#include "BaseBottomSmoother.H"
```

Include dependency graph for HelmholtzOp.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **HelmholtzOp**

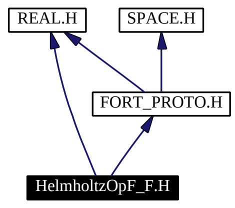
  Helmholtz operator class.

7.77 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_INT(redBlack))
- void **FORT_OPERATORHELM** (CHFp_FRA(lofphi),CHFp_CONST_FRA(phi),CHFp_BOX(region),CHFp_CONST_REAL(dx),CHFp_CONST_REAL(alpha))

7.77.1 Define Documentation

7.77.1.1 #define FORT_GSRBLEVELHELM FORTRAN_NAME(GSRBLEVELHELM ,gsrblevelhelm)

7.77.1.2 #define FORT_OPERATORHELM FORTRAN_NAME(OPERATORHELM ,operatorhelm)

7.77.2 Function Documentation

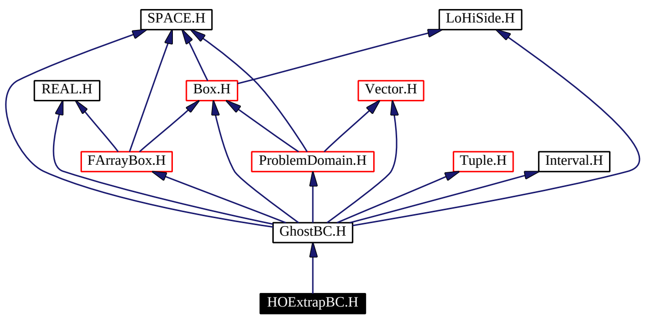
7.77.2.1 void FORT_GSRBLEVELHELM (CHFp_FRA(phi), CHFp_CONST_FRA(rhs), CHFp_BOX(region), CHFp_CONST_REAL(dx), CHFp_CONST_REAL(alpha), CHFp_CONST_INT(redBlack))

7.77.2.2 void FORT_OPERATORHELM (CHFp_FRA(lofphi), CHFp_CONST_FRA(phi), CHFp_BOX(region), CHFp_CONST_REAL(dx), CHFp_CONST_REAL(alpha))

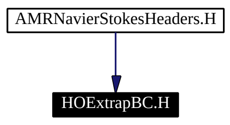
7.78 HOExtrapBC.H File Reference

`#include "GhostBC.H"`

Include dependency graph for HOExtrapBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **HOExtrapBC**

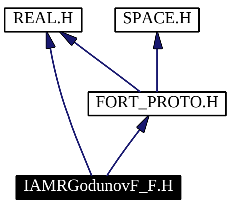
  class to do extrapolation BC's

7.79 IAMRGodunovF_F.H File Reference

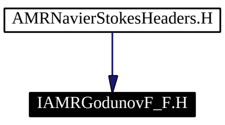
`#include "FORT_PROTO.H"`

`#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))

7.79.1 Define Documentation

7.79.1.1 #define FORT_UDELS FORTRAN_NAME(UDELS ,udels)

7.79.2 Function Documentation

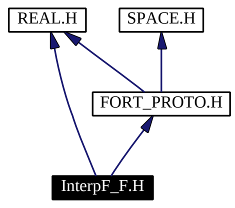
7.79.2.1 void FORT_UDELS (CHFp_FRA(uDotDelS), CHFp_FRA(sEdge), CHFp_FRA(vel), CHFp_BOX(gridBox), CHFp_REAL(dx), CHFp_INT(dir))

7.80 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))

7.80.1 Define Documentation

7.80.1.1 #define FORT_INTERPCENTRALSLOPE FORTRAN_NAME(INTERPCENTRALSLOPE ,interpcentralslope)

7.80.1.2 #define FORT_INTERPCONSTANT FORTRAN_NAME(INTERPCONSTANT ,interpconstant)

7.80.1.3 #define FORT_INTERPHISIDESLOPE FORTRAN_NAME(INTERPHISIDESLOPE ,interphisideslope)

7.80.1.4 #define FORT_INTERPLIMIT FORTRAN_NAME(INTERPLIMIT ,interplimit)

7.80.1.5 #define FORT_INTERPLINEAR FORTRAN_NAME(INTERPLINEAR ,interplinear)

7.80.1.6 #define FORT_INTERPLOSIDESLOPE FORTRAN_NAME(INTERPLOSIDESLOPE ,interplosideslope)

7.80.2 Function Documentation

7.80.2.1 void FORT_INTERPCENTRALSLOPE (CHFp_FRA(slope), CHFp_CONST_FRA(state), CHFp_BOX(b), CHFp_CONST_INT(dir))

7.80.2.2 void FORT_INTERPCONSTANT (CHFp_FRA(fine), CHFp_-CONST_FRA(coarse), CHFp_BOX(b), CHFp_CONST_INT(ref_ratio), CHFp_BOX(bref))

7.80.2.3 void FORT_INTERPHISIDESLOPE (CHFp_FRA(slope), CHFp_CONST_FRA(state), CHFp_BOX(b), CHFp_CONST_INT(dir))

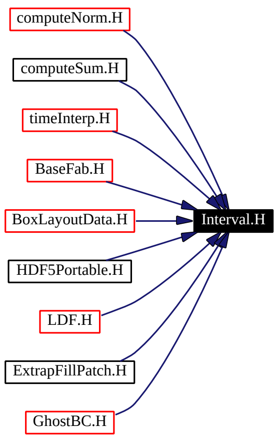
7.80.2.4 void FORT_INTERPLIMIT (CHFp_FRA(islope), CHFp_FRA(jslope), CHFp_FRA(kslope), CHFp_CONST_FRA(state), CHFp_BOX(bcoarse), CHFp_BOX(bn), CHFp_BOX(physdomain))

7.80.2.5 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))

7.80.2.6 void FORT_INTERPLOSIDESLOPE (CHFp_FRA(slope), CHFp_CONST_FRA(state), CHFp_BOX(b), CHFp_CONST_INT(dir))

7.81 Interval.H File Reference

This graph shows which files directly or indirectly include this file:



Compounds

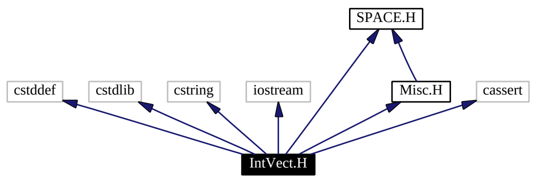
- struct **Interval**

  Structure for passing component ranges in code.

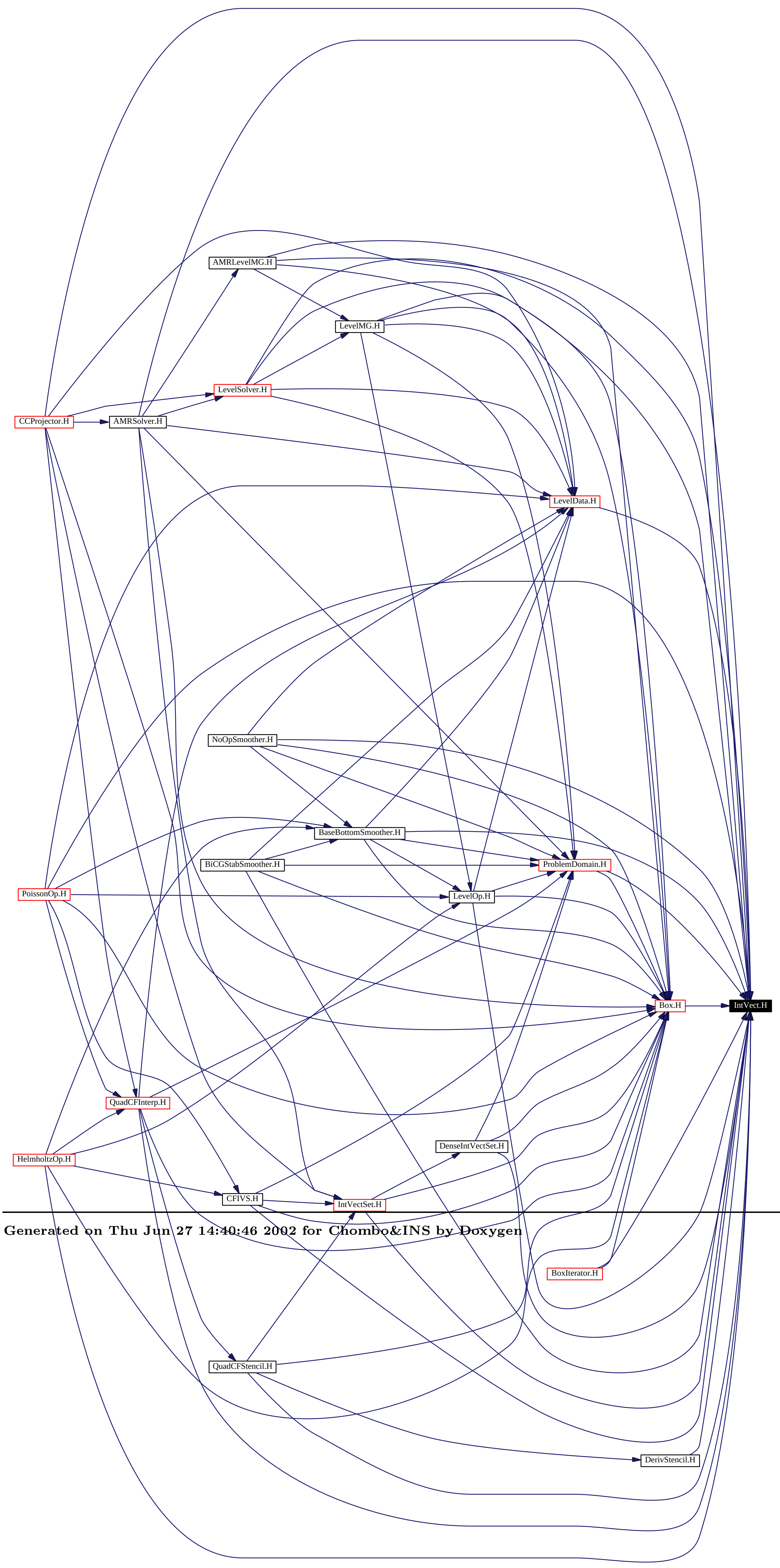
7.82 IntVect.H File Reference

```
#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:


AMRLevelMG.H
LevelMG.H
LevelSolver.H
CCProjector.H
AMRSolver.H
LevelData.H
NoOpSmoother.H
BaseBottomSmoother.H
BiCGStabSmoother.H
ProblemDomain.H
PoissonOp.H
LevelOp.H
Box.H
IntVect.H
QuadCFInterp.H
DenseIntVectSet.H
HelmholtzOp.H
CFIVS.H
IntVectSet.H
BoxIterator.H
QuadCFStencil.H
DerivStencil.H

Compounds

- class **IntVect**

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)

7.82.1 Define Documentation

7.82.1.1 #define LGT0 (vect[0] > s[0])

7.82.1.2 #define LGT1 ((vect[0] == s[0]) && (vect[1] > s[1]))

7.82.1.3 #define LGT2 ((vect[1] == s[1]) && (vect[2] > s[2]))

7.82.1.4 #define LLT0 (vect[0] < s[0])

7.82.1.5 #define LLT1 ((vect[0] == s[0]) && (vect[1] < s[1]))

7.82.1.6 #define LLT2 ((vect[1] == s[1]) && (vect[2] < s[2]))

7.82.2 Function Documentation

7.82.2.1 IntVect diagShift (const IntVect & *p*, int *s*) `[inline]`

Returns **IntVect** (p. 264) obtained by adding a scalar to each of the components of the given **IntVect** (p. 264).

7.82.2.2 IntVect max (const IntVect & *p1*, const IntVect & *p2*) `[inline]`

Returns the **IntVect** (p. 264) that is the component-wise minimum of two argument IntVects.

7.82.2.3 IntVect min (const IntVect & *p1*, const IntVect & *p2*) [inline]

Returns the **IntVect** (p. 264) that is the component-wise minimum of two argument IntVects.

7.82.2.4 IntVect operator * (int *s*, const IntVect & *p*) [inline]

Returns and **IntVect** (p. 264) that is an **IntVect** (p. 264) p with each component multiplied by a scalar s.

7.82.2.5 IntVect operator+ (int *s*, const IntVect & *p*) [inline]

Returns an **IntVect** (p. 264) that is an **IntVect** (p. 264) p with a scalar s added to each component.

7.82.2.6 IntVect operator- (int *s*, const IntVect & *p*) [inline]

Returns -p + s.

7.82.2.7 IntVect scale (const IntVect & *p*, int *s*) [inline]

Returns an **IntVect** (p. 264) obtained by multiplying each of the components of the given **IntVect** (p. 264) by a scalar.

7.82.2.8 IntVect UnitVector ()

7.82.2.9 IntVect ZeroVector ()

7.83 IntVectSet.H File Reference

`#include "SPACE.H"`

`#include <iostream>`

`#include "Box.H"`

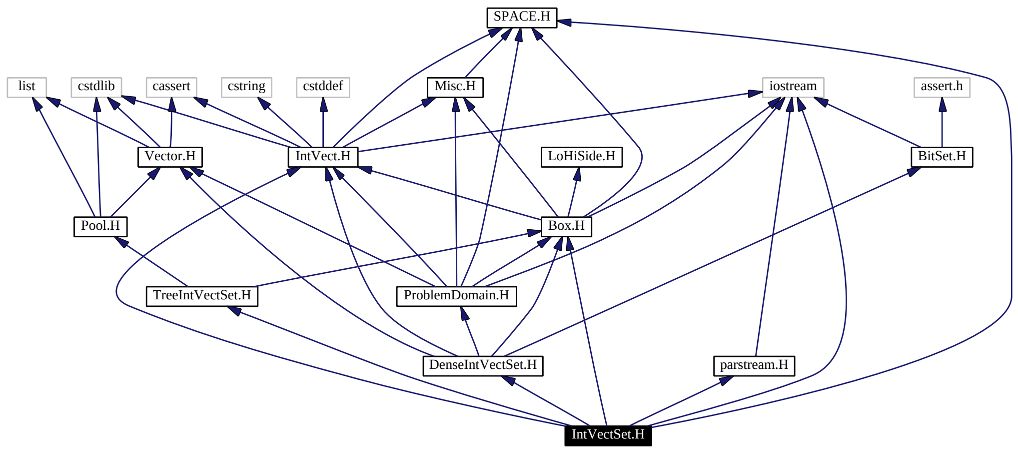
`#include "IntVect.H"`

`#include "TreeIntVectSet.H"`

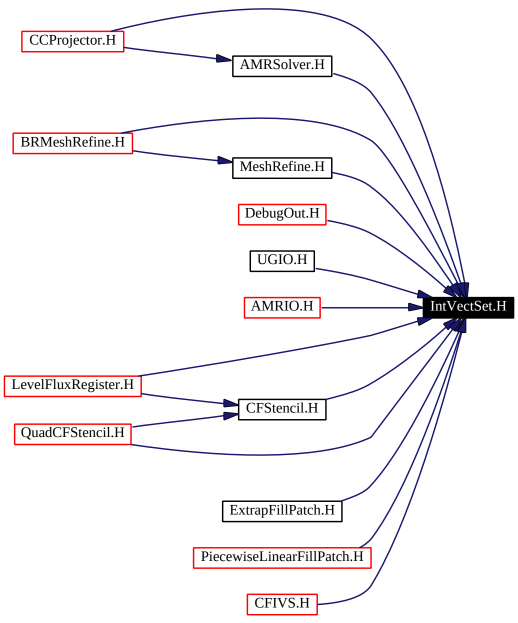
`#include "DenseIntVectSet.H"`

`#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** (p. 274).

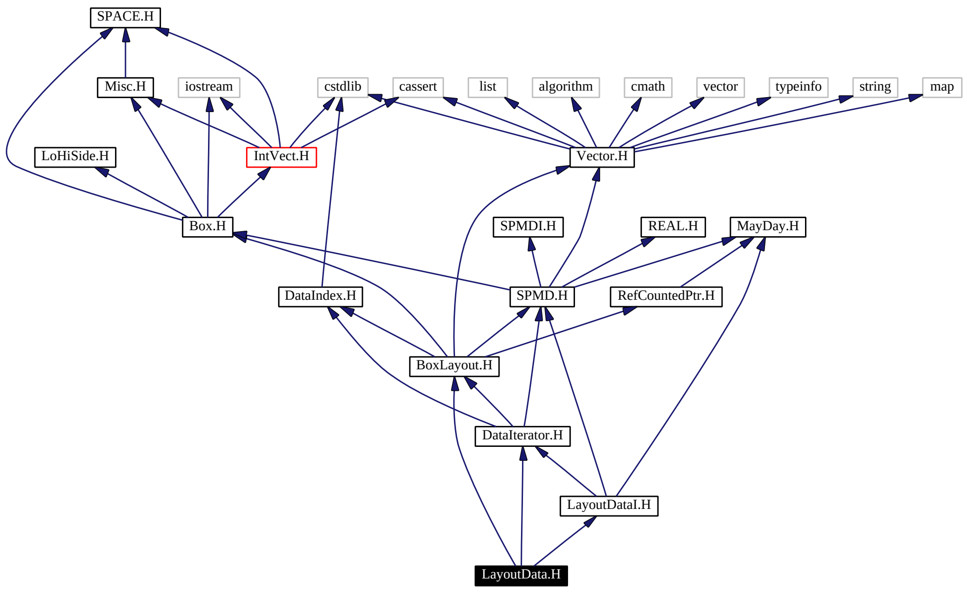
7.84 LayoutData.H File Reference

`#include "BoxLayout.H"`

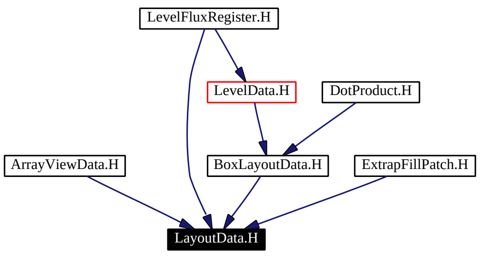
`#include "DataIterator.H"`

`#include "LayoutDataI.H"`

Include dependency graph for LayoutData.H:



This graph shows which files directly or indirectly include this file:



Compounds

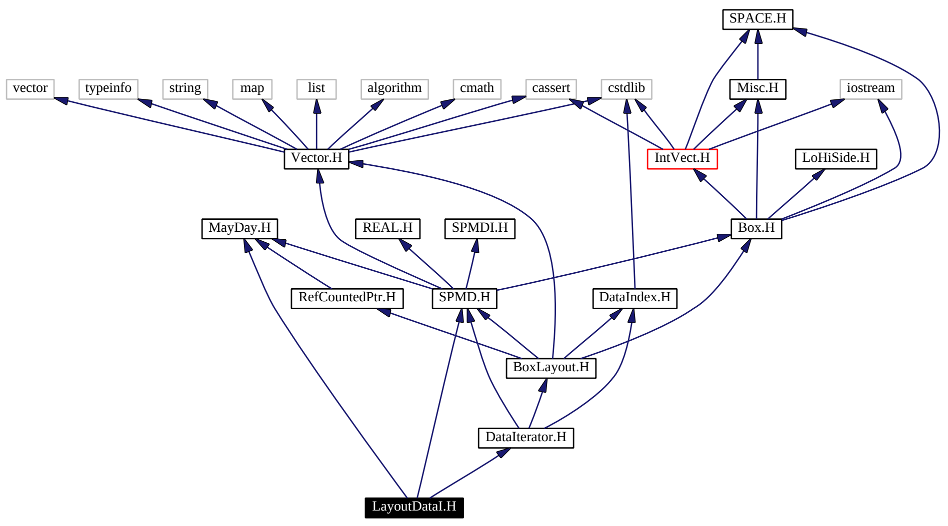
- class **LayoutData**

  Data that maintains a one-to-one mapping to the boxes in a **BoxLayout** (p. 139).

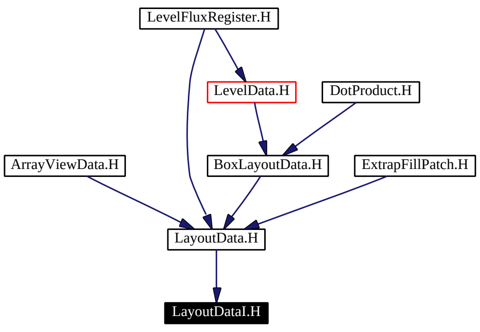
7.85 LayoutDataI.H File Reference

```
#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:

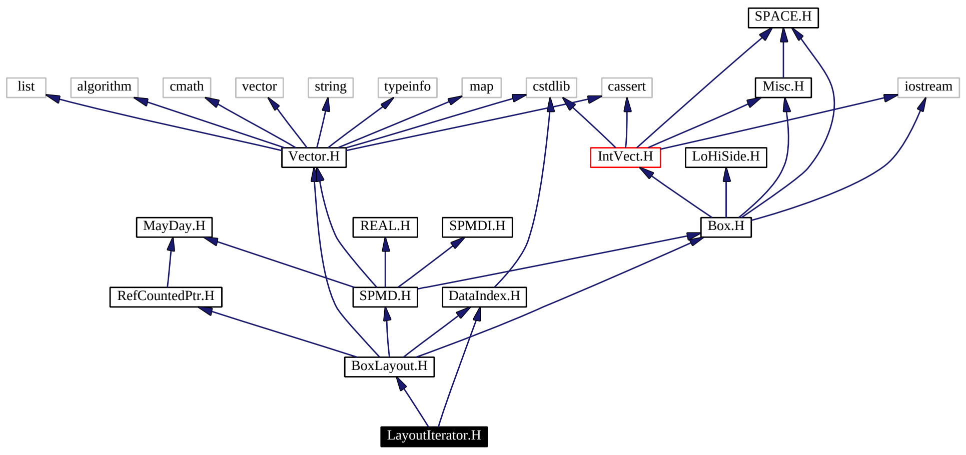


7.86 LayoutIterator.H File Reference

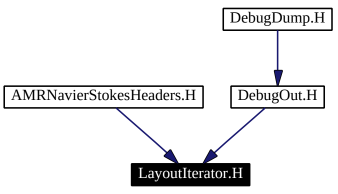
`#include "DataIndex.H"`

`#include "BoxLayout.H"`

Include dependency graph for LayoutIterator.H:



This graph shows which files directly or indirectly include this file:



Compounds

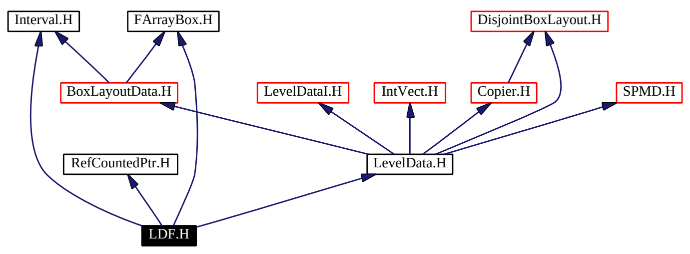
- class **LayoutIterator**

  An Iterator based on a **BoxLayout** (p. 139) *object.*

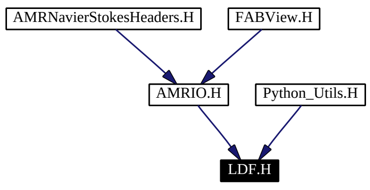
7.87 LDF.H File Reference

```
#include "LevelData.H"
#include "FArrayBox.H"
#include "RefCountedPtr.H"
#include "Interval.H"
```

Include dependency graph for LDF.H:



This graph shows which files directly or indirectly include this file:



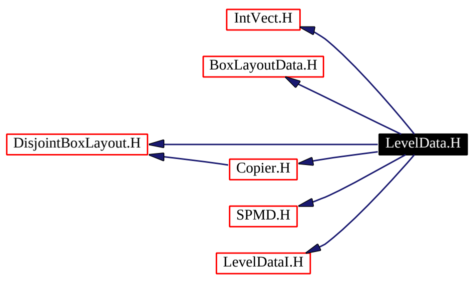
Compounds

- class **LDF**

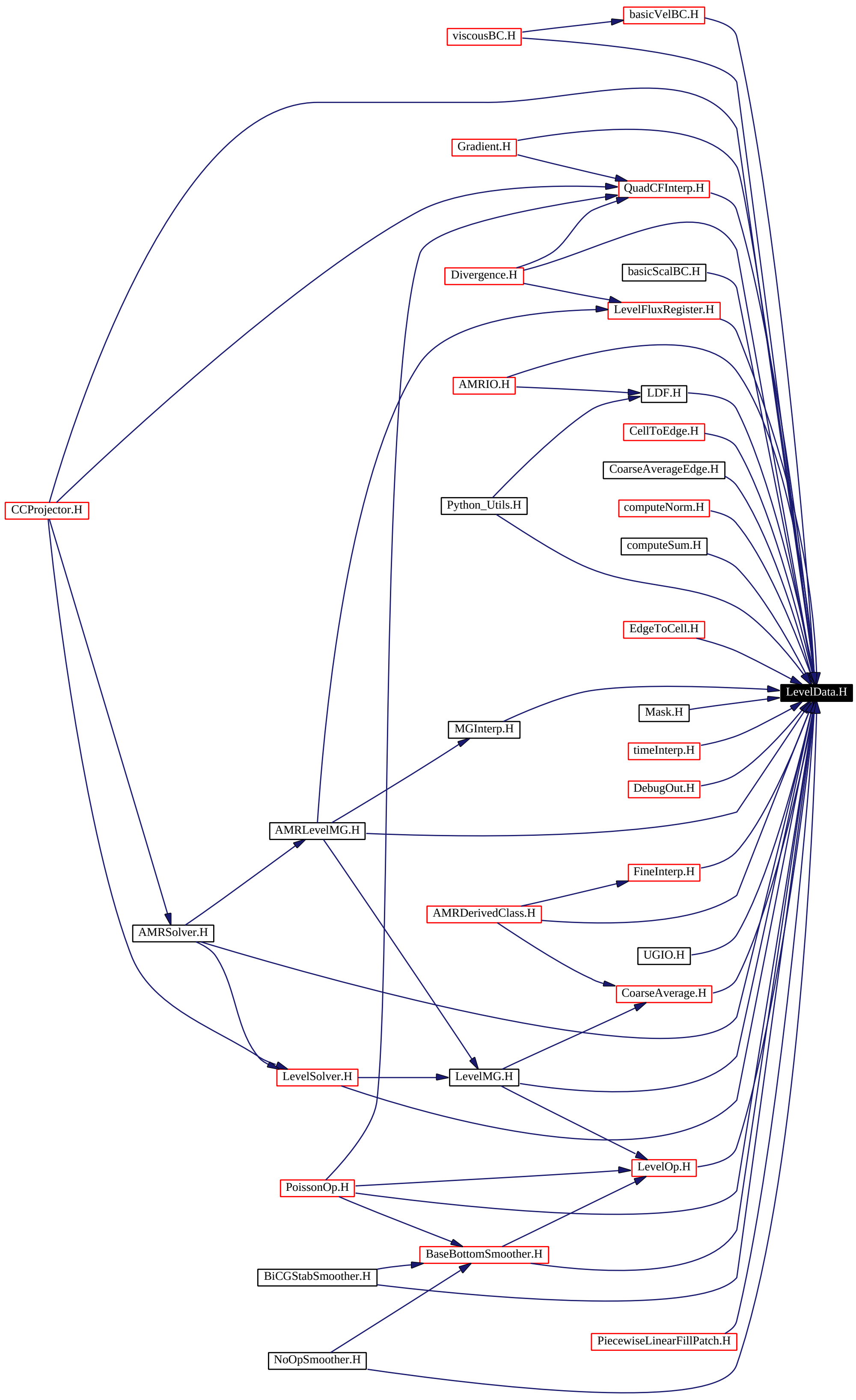
7.88 LevelData.H File Reference

```
#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:


viscousBC.H
basicVelBC.H
Gradient.H
QuadCFInterp.H
Divergence.H
basicScalBC.H
LevelFluxRegister.H
AMRIO.H
LDF.H
CellToEdge.H
CoarseAverageEdge.H
CCProjector.H
Python_Utils.H
computeNorm.H
computeSum.H
EdgeToCell.H
LevelData.H
Mask.H
MGInterp.H
timeInterp.H
DebugOut.H
AMRLevelMG.H
FineInterp.H
AMRDerivedClass.H
AMRSolver.H
UGIO.H
CoarseAverage.H
LevelSolver.H
LevelMG.H
LevelOp.H
PoissonOp.H
BaseBottomSmoother.H
BiCGStabSmoother.H
PiecewiseLinearFillPatch.H
NoOpSmoother.H

Compounds

- class **LevelData**

  Data over a disjoint union of rectangles.

7.89 LevelDataI.H File Reference

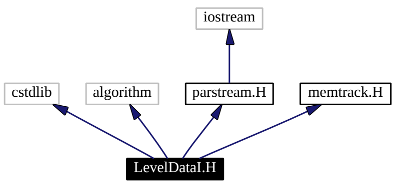
`#include <cstdlib>`

`#include <algorithm>`

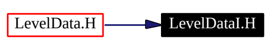
`#include "parstream.H"`

`#include "memtrack.H"`

Include dependency graph for LevelDataI.H:



This graph shows which files directly or indirectly include this file:



7.90 LevelFluxRegister.H File Reference

`#include "REAL.H"`

`#include "Vector.H"`

`#include "FArrayBox.H"`

`#include "IntVectSet.H"`

`#include "CFStencil.H"`

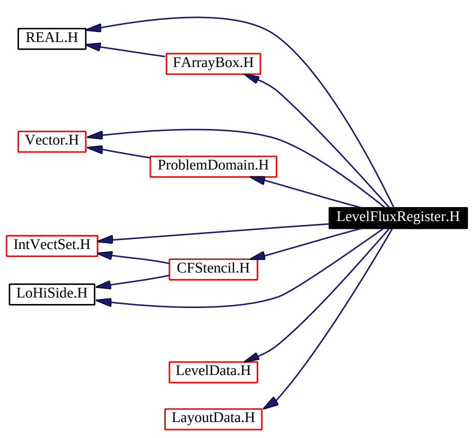
`#include "LoHiSide.H"`

`#include "LevelData.H"`

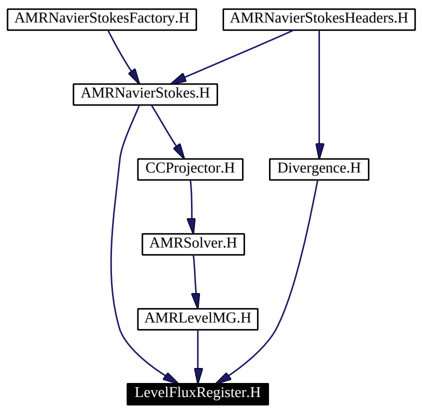
`#include "LayoutData.H"`

`#include "ProblemDomain.H"`

Include dependency graph for LevelFluxRegister.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **LevelFluxRegister**

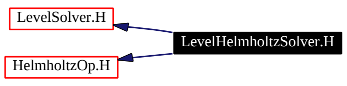
  LevelFluxRegister-A class to encapsulate a levels worth of flux registers.

7.91 LevelHelmholtzSolver.H File Reference

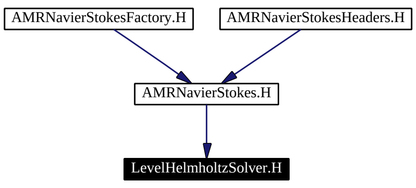
`#include <LevelSolver.H>`

`#include <HelmholtzOp.H>`

Include dependency graph for LevelHelmholtzSolver.H:



This graph shows which files directly or indirectly include this file:



Compounds

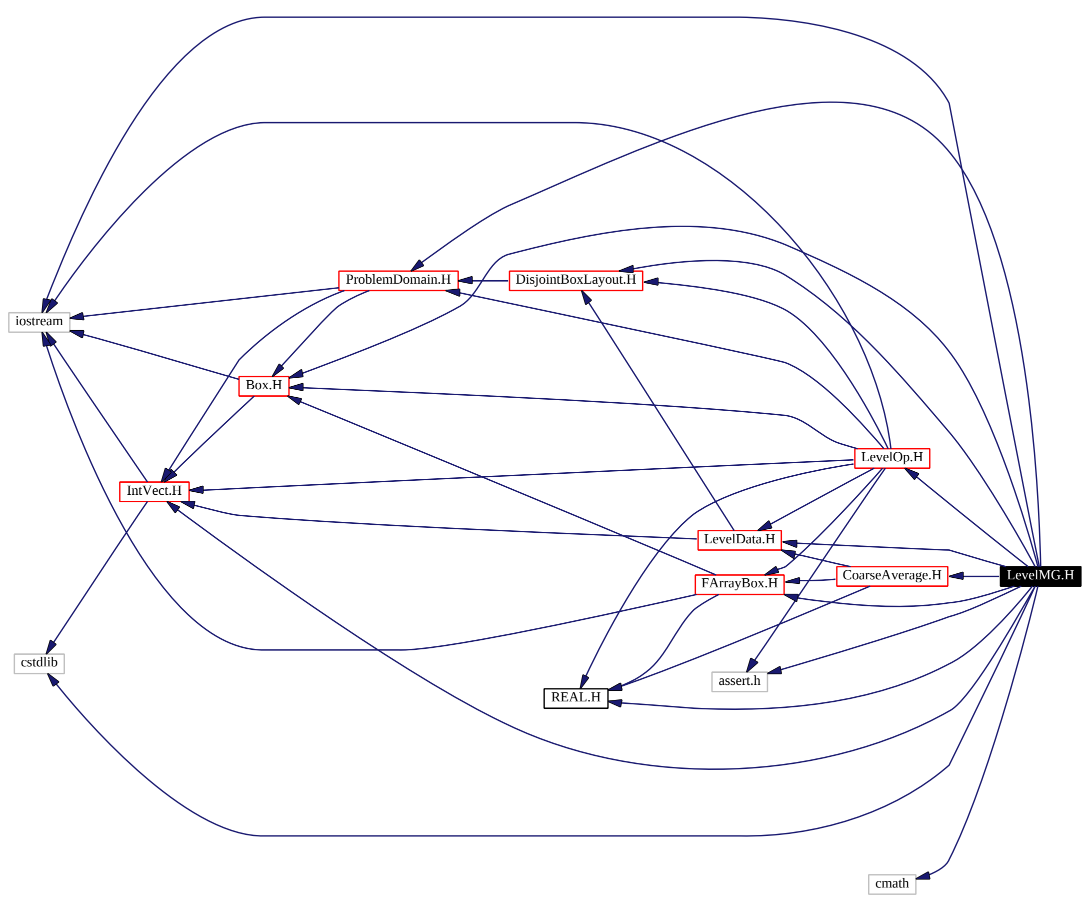
- class **LevelHelmholtzSolver**

  special case **LevelSolver** (p. 319) *to solve Helmholtz eqn*

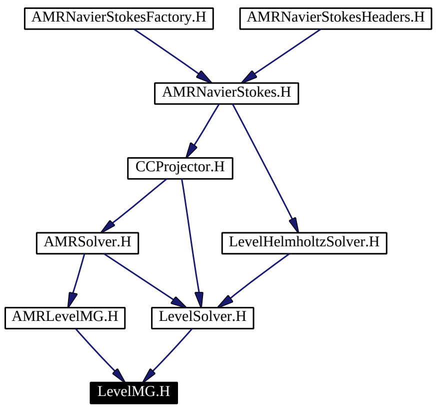
7.92 LevelMG.H File Reference

```
#include <iostream>
#include <cmath>
#include <assert.h>
#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**

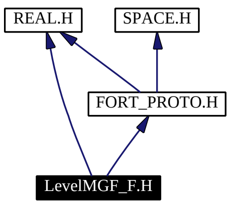
  Multigrid solver on a level.

7.93 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))

7.93.1 Define Documentation

7.93.1.1 #define FORT_INTERPMG FORTRAN_NAME(INTERPMG ,interpmg)

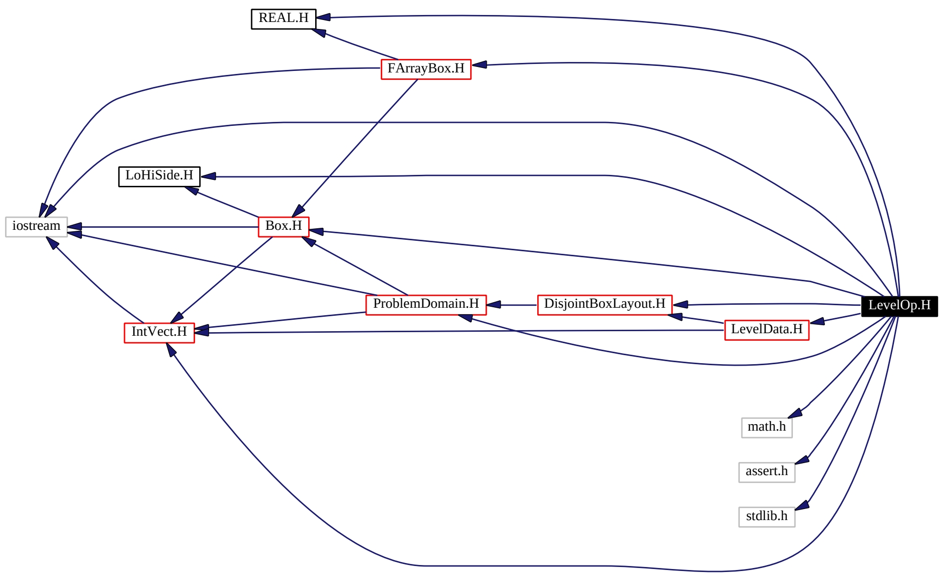
7.93.2 Function Documentation

7.93.2.1 void FORT_INTERPMG (CHFp_FRA(fine), CHFp_CONST_FRA(crse), CHFp_BOX(region), CHFp_CONST_INT(nref), CHFp_BOX(bref))

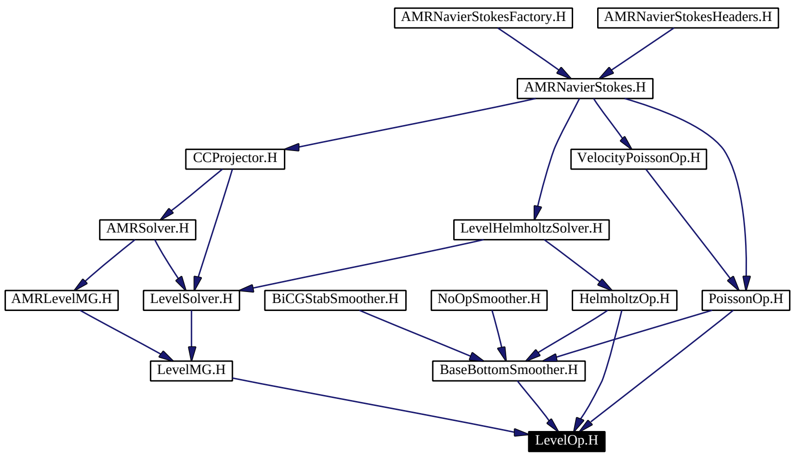
7.94 LevelOp.H File Reference

```
#include <iostream>
#include <math.h>
#include <assert.h>
#include <stdlib.h>
#include <REAL.H>
#include <IntVect.H>
#include <Box.H>
#include <DisjointBoxLayout.H>
#include <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

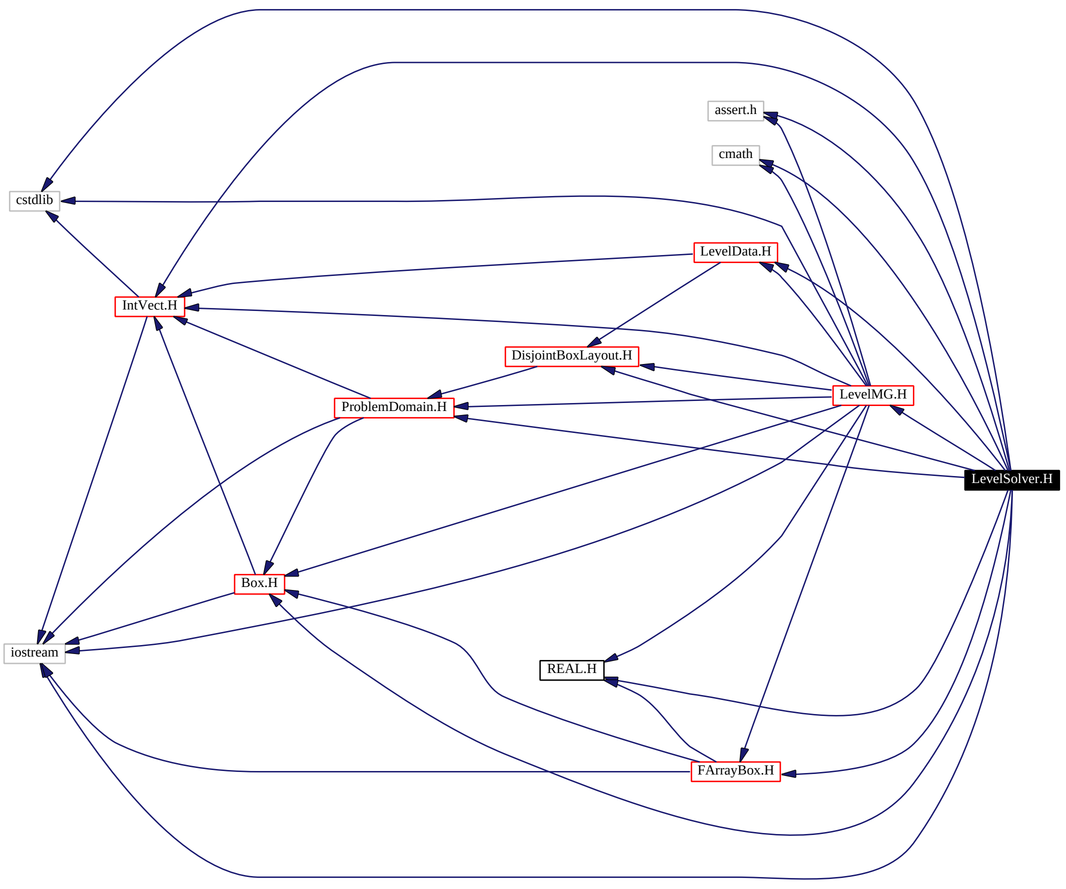
- class **LevelOp**

  Pure base class to encapsulate level operations API for elliptic solvers.

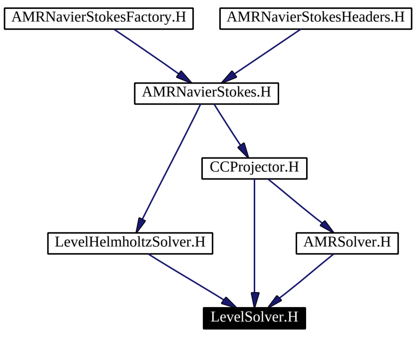
7.95 LevelSolver.H File Reference

```
#include <cstdlib>
#include <iostream>
#include <assert.h>
#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**

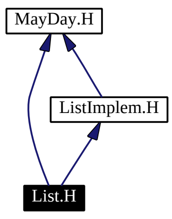
  Solves elliptic equations on a level.

7.96 List.H File Reference

`#include "MayDay.H"`

`#include "ListImplem.H"`

Include dependency graph for List.H:



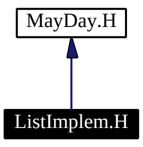
Compounds

- class **List**
- class **ListIterator**
- class **ListLink**

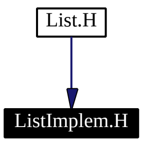
7.97 ListImplem.H File Reference

`#include "MayDay.H"`

Include dependency graph for ListImplem.H:



This graph shows which files directly or indirectly include this file:



7.98 LoadBalance.H File Reference

`#include "SPACE.H"`

`#include "REAL.H"`

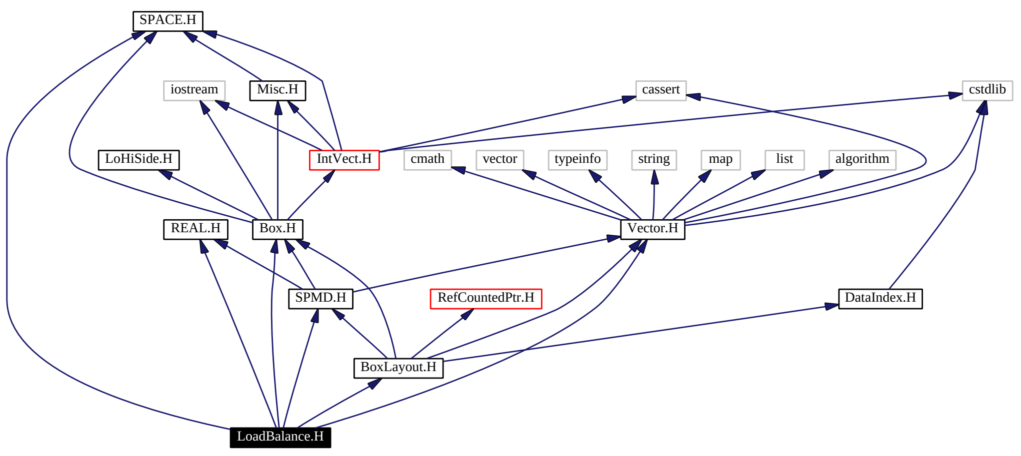
`#include "Box.H"`

`#include "Vector.H"`

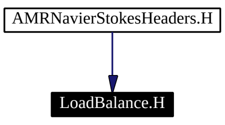
`#include "BoxLayout.H"`

`#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 > > &procAssignments,**Real** &effRatio,const **Vector**< **Vector**< **Box** > > &Grids,const **Vector**< **Vector**< long > > &ComputeLoads,const **Vector**< int > &RefRatios,int 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)

7.98.1 Function Documentation

7.98.1.1 int LoadBalance (Vector< int > & *procAssignments*, const Vector< Box > & *boxes*)

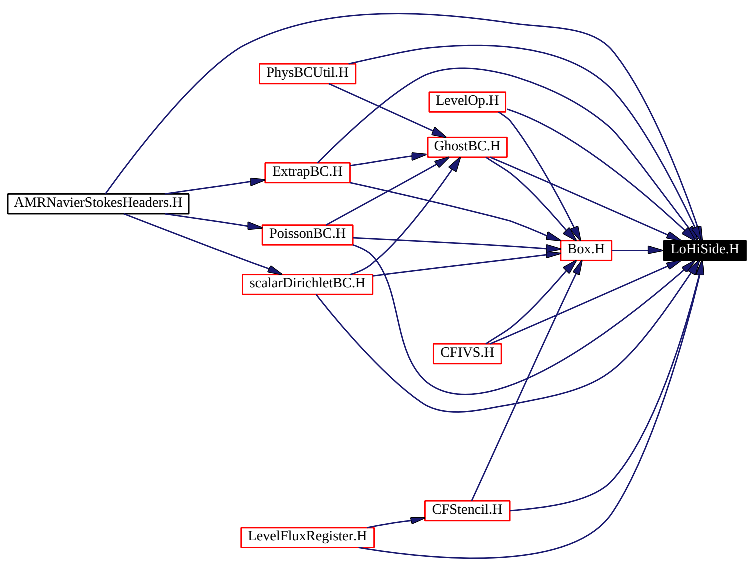
convenience function to load balance a **Vector** (p. 471) of Boxes based on load=box.numPts()

7.98.1.2 int LoadBalance (Vector< BoxLayout > & *Grids*, Real & *effRatio*, const Vector< Vector< long > > & *ComputeLoads*, const Vector< int > & *RefRatios*, int *nProc* = numProc())

7.98.1.3 int LoadBalance (Vector< Vector< int > > & *procAssignments*, Real & *effRatio*, const Vector< Vector< Box > > & *Grids*, const Vector< Vector< long > > & *ComputeLoads*, const Vector< int > & *RefRatios*, int *nProc* = numProc())

7.99 LoHiSide.H File Reference

This graph shows which files directly or indirectly include this file:



Compounds

- class **Side**

  encapsulation of high or low side

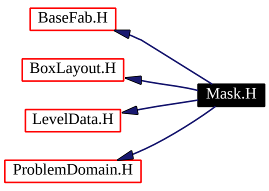
- class **SideIterator**

  Iterator for low and high side.

7.100 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**

7.101 MASKVAL.H File Reference

Defines

- #define **MASKCOVERED** -2
- #define **MASKPHYS** -1
- #define **MASKCOPY** 0
- #define **MASKCOARSE** 1

7.101.1 Define Documentation

7.101.1.1 #define MASKCOARSE 1

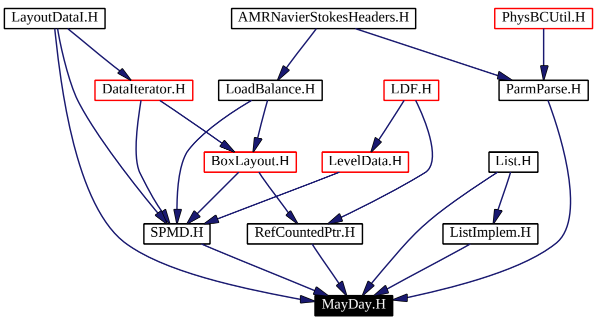
7.101.1.2 #define MASKCOPY 0

7.101.1.3 #define MASKCOVERED -2

7.101.1.4 #define MASKPHYS -1

7.102 MayDay.H File Reference

This graph shows which files directly or indirectly include this file:

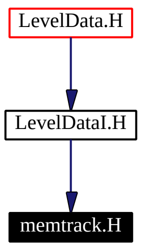


Compounds

- class **MayDay**

7.103 memtrack.H File Reference

This graph shows which files directly or indirectly include this file:



7.104 MeshRefine.H File Reference

`#include <climits>`

`#include "Vector.H"`

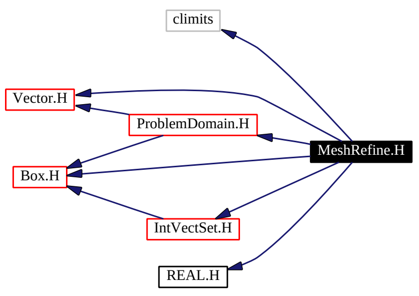
`#include "Box.H"`

`#include "IntVectSet.H"`

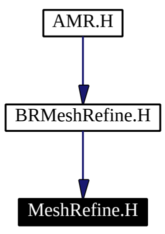
`#include "REAL.H"`

`#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

7.105 MGInterp.H File Reference

`#include "REAL.H"`

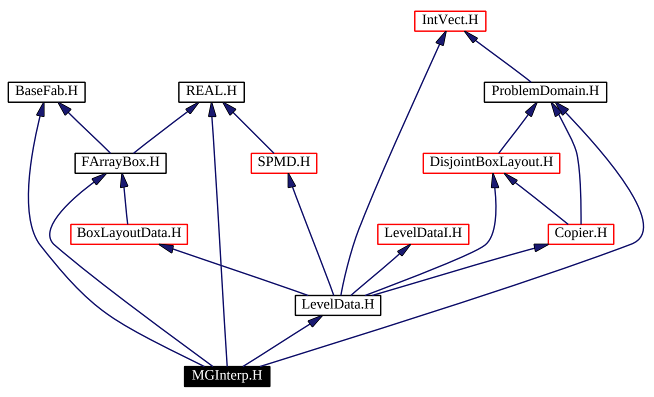
`#include "LevelData.H"`

`#include "BaseFab.H"`

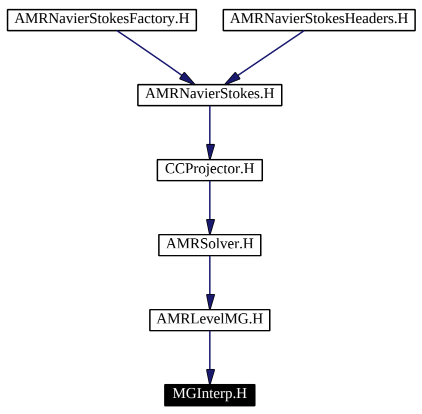
`#include "FArrayBox.H"`

`#include "ProblemDomain.H"`

Include dependency graph for MGInterp.H:



This graph shows which files directly or indirectly include this file:



Compounds

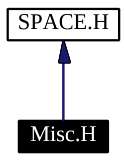
- class **MGInterp**

  class to interpolate coarse level data to a fine level

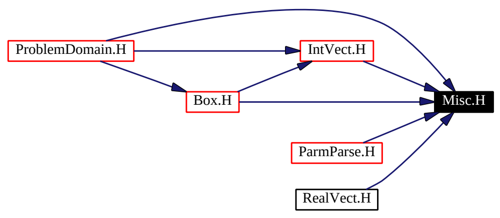
7.106 Misc.H File Reference

`#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)

7.106.1 Function Documentation

7.106.1.1 template<class T> T Abs (const T & *a*) `[inline]`

7.106.1.2 template<class T> T Max (const T & *a*, const T & *b*) `[inline]`

7.106.1.3 template<class T> T Min (const T & *a*, const T & *b*) `[inline]`

7.106.1.4 template<class T> void Swap (T & *a*, T & *b*) `[inline]`

7.107 NoOpSmoother.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>`

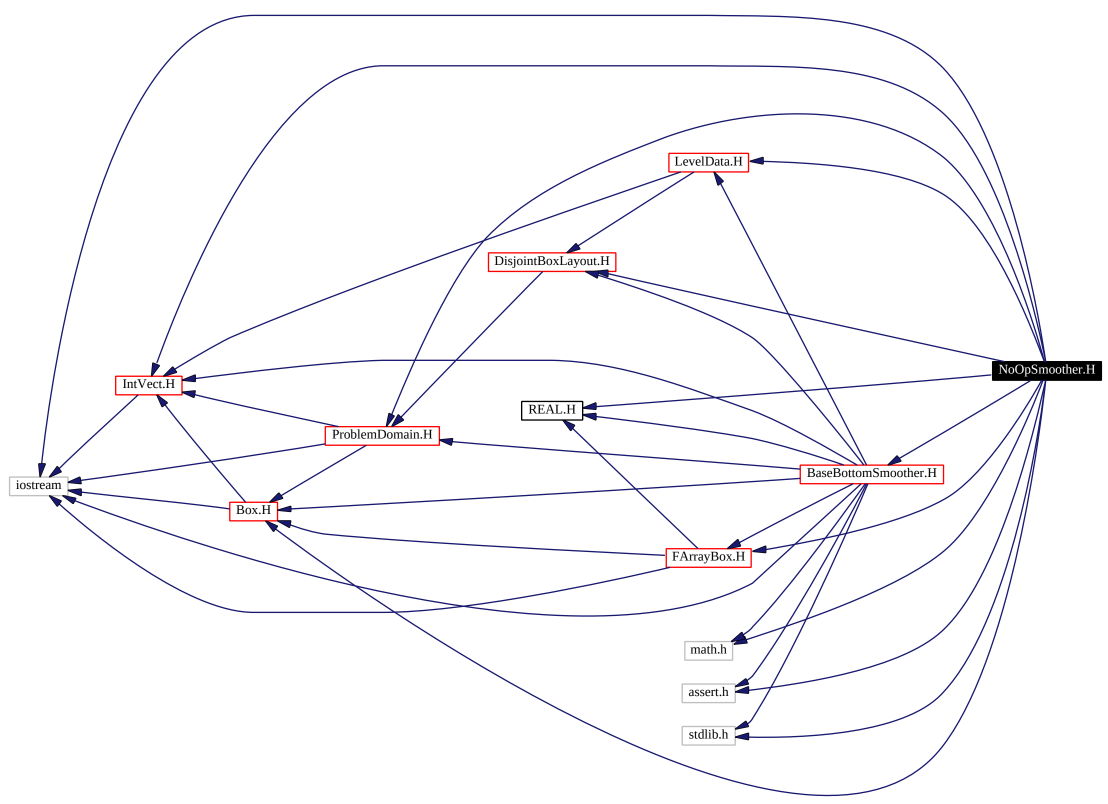
`#include <LevelData.H>`

`#include <FArrayBox.H>`

`#include <ProblemDomain.H>`

`#include "BaseBottomSmoother.H"`

Include dependency graph for NoOpSmoother.H:



Compounds

- class **NoOpSmoother**

  does nothing as a bottom smoother.

7.108 ParmParse.H File Reference

`#include <iostream>`

`#include <iomanip>`

`#include <cstdlib>`

`#include <string>`

`#include <vector>`

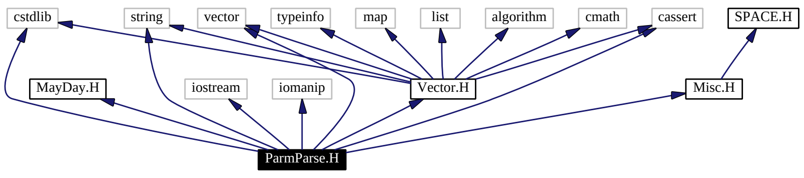
`#include <cassert>`

`#include "MayDay.H"`

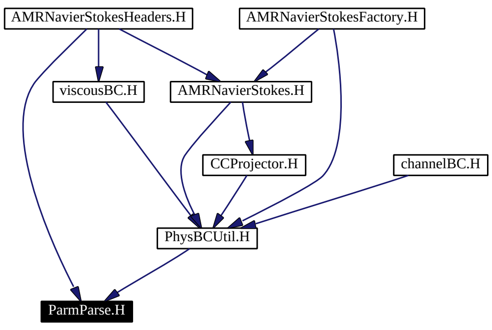
`#include "Misc.H"`

`#include "Vector.H"`

Include dependency graph for ParmParse.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **ParmParse**
- class **PP_Array**
- class **PP_CpClassPtr**
- class **PP_CpPtr**
- class **PP_entry**
- class **PP_List**
- class **PP_ListIterator**
- class **PP_ListLink**
- class **PP_LnClassPtr**
- class **PP_LnPtr**

- class **PP_String**
- 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)

7.108.1 Function Documentation

7.108.1.1 bool operator!= (const char * *left*, const PP_String & *right*) `[inline]`

7.108.1.2 bool operator!= (const PP_String & *left*, const char * *right*) `[inline]`

7.108.1.3 bool operator!= (const PP_String & *left*, const PP_String & *right*) `[inline]`

7.108.1.4 PP_String operator+ (const PP_String & *left*, const PP_String & *right*) `[inline]`

7.108.1.5 bool operator< (const char * *left*, const PP_String & *right*) `[inline]`

7.108.1.6 bool operator< (const PP_String & *left*, const char * *right*) `[inline]`

7.108.1.7 bool operator< (const PP_String & *left*, const PP_String & *right*) `[inline]`

7.108.1.8 bool operator<= (const char * *left*, const PP_String & *right*) `[inline]`

7.108.1.9 bool operator<= (const PP_String & *left*, const char * *right*) `[inline]`

7.108.1.10 bool operator<= (const PP_String & *left*, const PP_String & *right*) `[inline]`

7.108.1.11 bool operator== (const char * *left*, const PP_String & *right*) `[inline]`

7.108.1.12 bool operator== (const PP_String & *left*, const char * *right*) `[inline]`

7.108.1.13 bool operator== (const PP_String & *left*, const PP_String & *right*) `[inline]`

7.108.1.14 bool operator> (const char * *left*, const PP_String & *right*) `[inline]`

7.108.1.15 bool operator> (const PP_String & *left*, const char * *right*) `[inline]`

7.108.1.16 bool operator> (const PP_String & *left*, const PP_String & *right*) `[inline]`

7.108.1.17 bool operator>= (const char * *left*, const PP_String & *right*) `[inline]`

7.108.1.18 bool operator>= (const PP_String & *left*, const char * *right*) `[inline]`

7.108.1.19 bool operator>= (const PP_String & *left*, const PP_String & *right*) `[inline]`

7.109 ParmParseIO.H File Reference

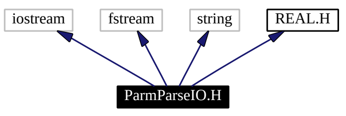
`#include <iostream>`

`#include <fstream>`

`#include <string>`

`#include "REAL.H"`

Include dependency graph for ParmParseIO.H:



Compounds

- class **ParmParseIO**

Functions

- void **write** (**ParmParseIO** &a_ppio, const **LevelData**< **FArrayBox** > &a_data, const std::string &a_name)
- int **read** (**ParmParseIO** &a_ppio, **LevelData**< **FArrayBox** > &a_data, const std::string &a_name, const **DisjointBoxLayout** &a_layout)

7.109.1 Function Documentation

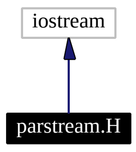
7.109.1.1 int read (ParmParseIO & *a_ppio*, LevelData< FArrayBox > & *a_data*, const std::string & *a_name*, const DisjointBoxLayout & *a_layout*)

7.109.1.2 void write (ParmParseIO & *a_ppio*, const LevelData< FArrayBox > & *a_data*, const std::string & *a_name*)

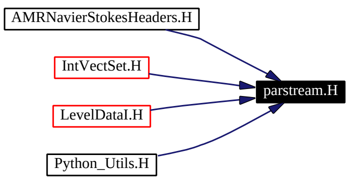
7.110 parstream.H File Reference

`#include <iostream>`

Include dependency graph for parstream.H:



This graph shows which files directly or indirectly include this file:



Functions

- std::ostream & **pout** ()

7.110.1 Function Documentation

7.110.1.1 std::ostream& pout ()

7.111 PhysBCUtil.H File Reference

`#include "GhostBC.H"`

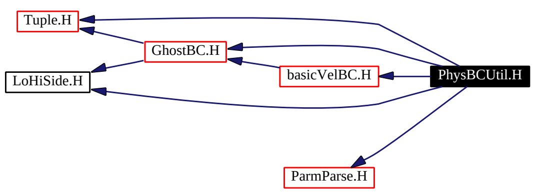
`#include "Tuple.H"`

`#include "ParmParse.H"`

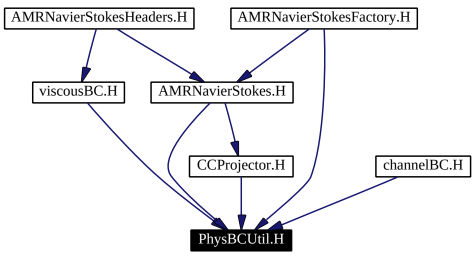
`#include "LoHiSide.H"`

`#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.

7.112 PiecewiseLinearFillPatch.H File Reference

`#include <iostream>`

`#include <fstream>`

`#include "REAL.H"`

`#include "Box.H"`

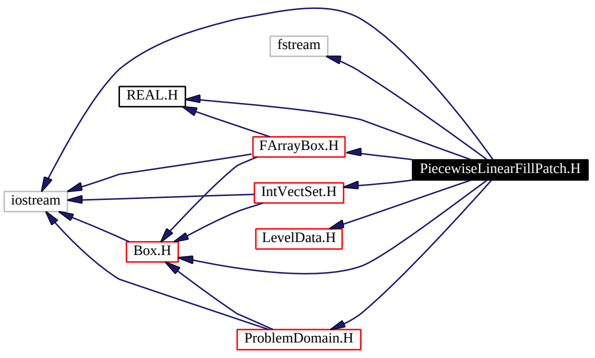
`#include "FArrayBox.H"`

`#include "LevelData.H"`

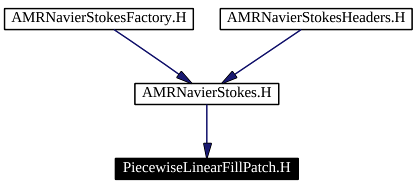
`#include "IntVectSet.H"`

`#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.

7.113 PoissonBC.H File Reference

`#include "Box.H"`

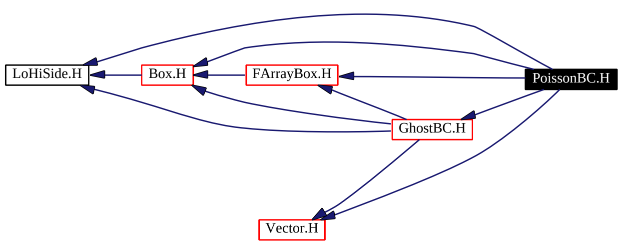
`#include "FArrayBox.H"`

`#include "Vector.H"`

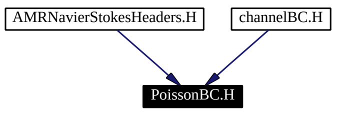
`#include "LoHiSide.H"`

`#include "GhostBC.H"`

Include dependency graph for PoissonBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DircheletBC**

  Class to enforce homogeneous dirchelet boundary conditions.

- class **NeumannBC**

  Class to enforce homogeneous neumann boundary conditions.

7.114 PoissonOp.H File Reference

#include <iostream>

#include <cmath>

#include <assert.h>

#include <cstdlib>

#include <REAL.H>

#include <IntVect.H>

#include <Box.H>

#include <DisjointBoxLayout.H>

#include <LevelData.H>

#include <BaseFab.H>

#include <FArrayBox.H>

#include "GhostBC.H"

#include "LevelOp.H"

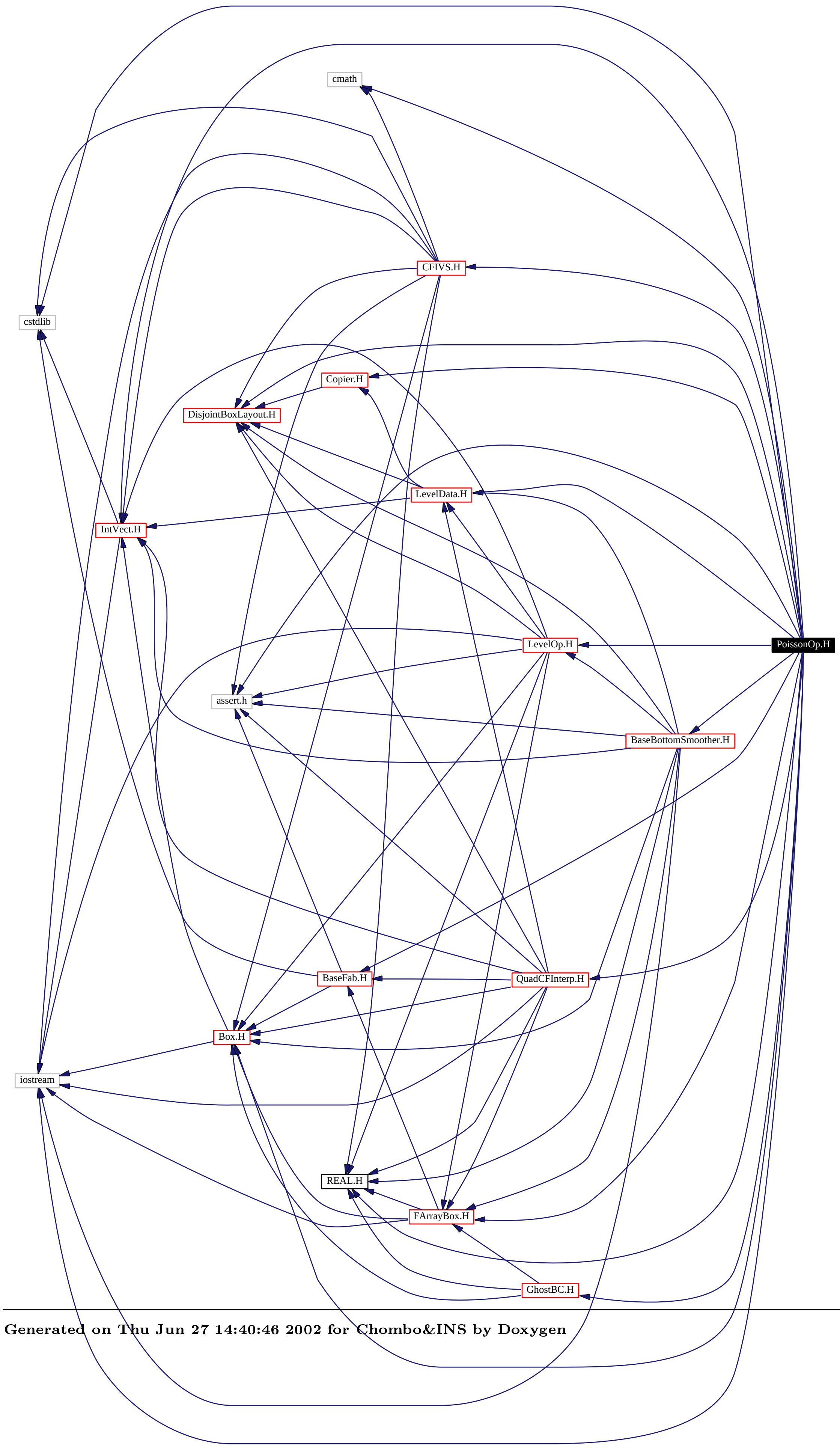
#include "QuadCFInterp.H"

#include "CFIVS.H"

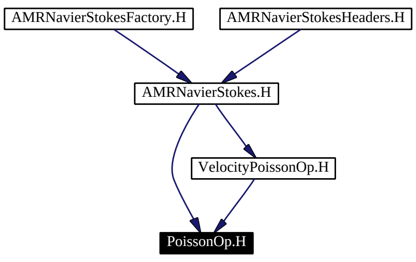
#include "Copier.H"

#include "BaseBottomSmoother.H"

Include dependency graph for PoissonOp.H:


cmath
CFIVS.H
cstdlib
Copier.H
DisjointBoxLayout.H
LevelData.H
IntVect.H
LevelOp.H
PoissonOp.H
assert.h
BaseBottomSmoother.H
BaseFab.H
QuadCFInterp.H
Box.H
iostream
REAL.H
FArrayBox.H
GhostBC.H

This graph shows which files directly or indirectly include this file:



Compounds

- class **PoissonOp**

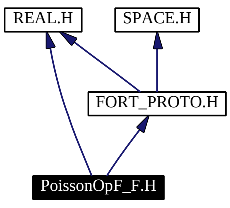
  one-sided differencing version of poisson operator

7.115 PoissonOpF_F.H File Reference

`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for PoissonOpF_F.H:



Defines

- #define **FORT_GSRBLEVELLAP** FORTRAN_NAME(GSRBLEVELLAP ,gsrblevellap)
- #define **FORT_OPERATORLAP** FORTRAN_NAME(OPERATORLAP ,operatorlap)

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))

7.115.1 Define Documentation

7.115.1.1 #define FORT_GSRBLEVELLAP FORTRAN_NAME(GSRBLEVELLAP ,gsrblevellap)

7.115.1.2 #define FORT_OPERATORLAP FORTRAN_NAME(OPERATORLAP ,operatorlap)

7.115.2 Function Documentation

7.115.2.1 void FORT_GSRBLEVELLAP (CHFp_FRA(phi), CHFp_CONST_-FRA(rhs), CHFp_BOX(region), CHFp_CONST_REAL(dx), CHFp_CONST_INT(redBlack))

7.115.2.2 void FORT_OPERATORLAP (CHFp_FRA(lofphi), CHFp_CONST_-FRA(phi), CHFp_BOX(region), CHFp_CONST_REAL(dx))

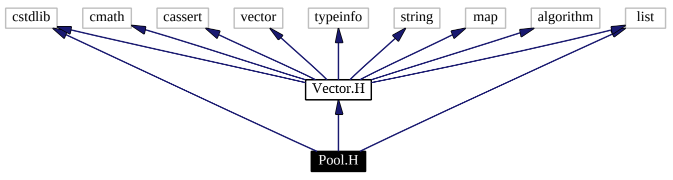
7.116 Pool.H File Reference

`#include <cstdlib>`

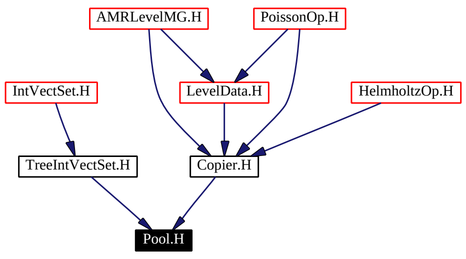
`#include "Vector.H"`

`#include <list>`

Include dependency graph for Pool.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Pool**

Typedefs

- typedef std::list< **Pool** * > **PoolList**

7.116.1 Typedef Documentation

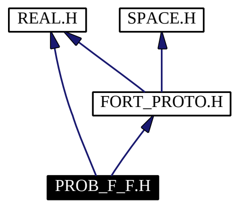
7.116.1.1 typedef std::list<Pool*> PoolList

7.117 PROB_F_F.H File Reference

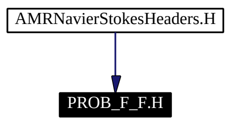
`#include "FORT_PROTO.H"`

`#include "REAL.H"`

Include dependency graph for PROB_F_F.H:



This graph shows which files directly or indirectly include this file:



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)

Functions

- void **FORT_INITVEL** (CHFp_FRA(vel),CHFp_REAL(dx))
- void **FORT_SETTAGGINGVAR** (CHFp_FRA1(tagvar),CHFp_CONST_REAL(dx))
- void **FORT_INITSCAL** (CHFp_FRA1(scal),CHFp_REAL(dx),CHFp_INT(comp),CHFp_BOX(domain),CHFp_INT(isEC))
- void **FORT_INITVORT** (CHFp_FRA1(vort),CHFp_INT(comp),CHFp_REAL(dx))

7.117.1 Define Documentation

7.117.1.1 #define FORT_INITSCAL FORTRAN_NAME(INITSCAL ,initscal)

7.117.1.2 #define FORT_INITVEL FORTRAN_NAME(INITVEL ,initvel)

7.117.1.3 #define FORT_INITVORT FORTRAN_NAME(INITVORT ,initvort)

7.117.1.4 #define FORT_SETTAGGINGVAR FORTRAN_NAME(SETTAGGINGVAR ,settaggingvar)

7.117.2 Function Documentation

7.117.2.1 void FORT_INITSCAL (CHFp_FRA1(scal), CHFp_REAL(dx), CHFp_INT(comp), CHFp_BOX(domain), CHFp_INT(isEC))

7.117.2.2 void FORT_INITVEL (CHFp_FRA(vel), CHFp_REAL(dx))

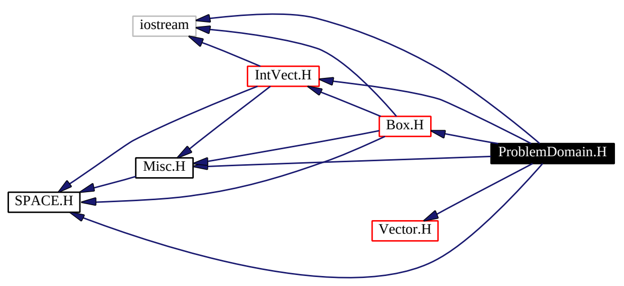
7.117.2.3 void FORT_INITVORT (CHFp_FRA1(vort), CHFp_INT(comp), CHFp_REAL(dx))

7.117.2.4 void FORT_SETTAGGINGVAR (CHFp_FRA1(tagvar), CHFp_CONST_REAL(dx))

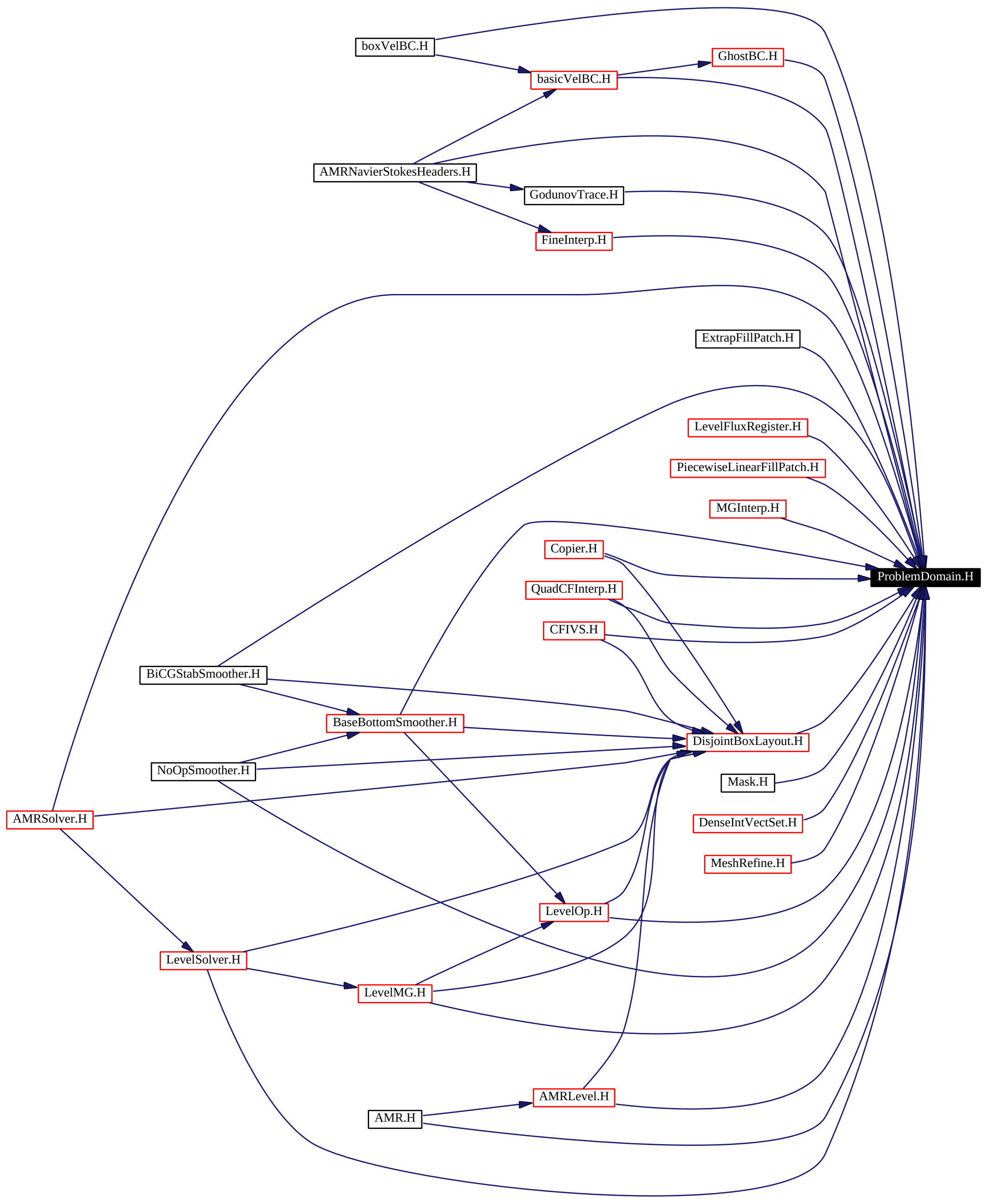
7.118 ProblemDomain.H File Reference

```
#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**
- class **ShiftIterator**

  clase 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)

7.118.1 Function Documentation

7.118.1.1 ProblemDomain grow (const ProblemDomain & *pd*, const IntVect & *v*) [inline]

Returns a **ProblemDomain** (p. 418) that is the argument **ProblemDomain** (p. 418) with a DomainBox grown by the given amount.

7.118.1.2 ProblemDomain grow (const ProblemDomain & *pd*, int *i*) [inline]

returns a **ProblemDomain** (p. 418) with a domainBox grown by the given amount

7.119 Python_Utils.H File Reference

`#include "Vector.H"`

`#include "Box.H"`

`#include <string>`

`#include "DisjointBoxLayout.H"`

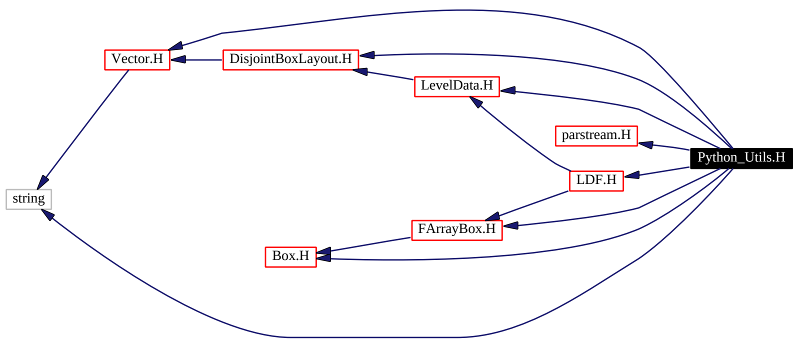
`#include "LevelData.H"`

`#include "FArrayBox.H"`

`#include "LDF.H"`

`#include "parstream.H"`

Include dependency graph for Python_Utils.H:



Defines

- #define **VECTOR_MACRO**(class)

Functions

- **VECTOR_MACRO** (**Box**)
- **VECTOR_MACRO** (int)
- **VECTOR_MACRO** (string)
- **VECTOR_MACRO** (**DisjointBoxLayout**)
- **VECTOR_MACRO** (**LDF**)

7.119.1 Define Documentation

7.119.1.1 #define VECTOR_MACRO(class)

Value:

```
class& vget_##class(Vector<class>& vector, int index) { return vector[index];} \
;\
void vset_##class(Vector<class>& vector, int i, class val) { vector[i]=val;} \
```

```
;\
class val_##class(const Vector<class>& vector, int index) { return vector[index];} \
;\
int vsize_##class(const Vector<class>& vector) { return vector.size();} \
;\
Vector<class> vcopy_##class(const Vector<class>& vector) {return vector;} \
;\
void vpush_back_##class(Vector<class>& vec, const class& in){vec.push_back(in);}\
;\
void vappend_##class(Vector<class>& mod, const Vector<class>& in){mod.append(in);}\
;\
void vresize_##class(Vector<class>& mod, int s) {mod.resize(s);}\
;\
Vector<class> vnew_##class(int s) {return Vector<class>(s);}\
```

7.119.2 Function Documentation

7.119.2.1 VECTOR_MACRO (LDF)

7.119.2.2 VECTOR_MACRO (DisjointBoxLayout)

7.119.2.3 VECTOR_MACRO (string)

7.119.2.4 VECTOR_MACRO (int)

7.119.2.5 VECTOR_MACRO (Box)

7.120 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"`

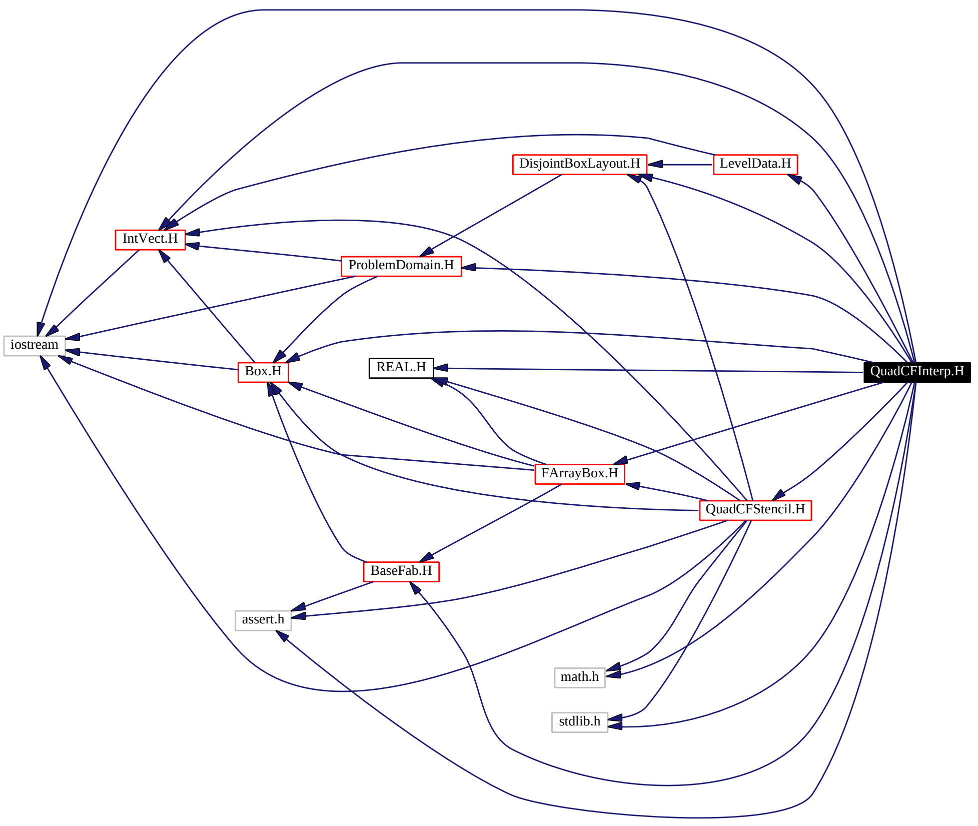
`#include "LevelData.H"`

`#include "FArrayBox.H"`

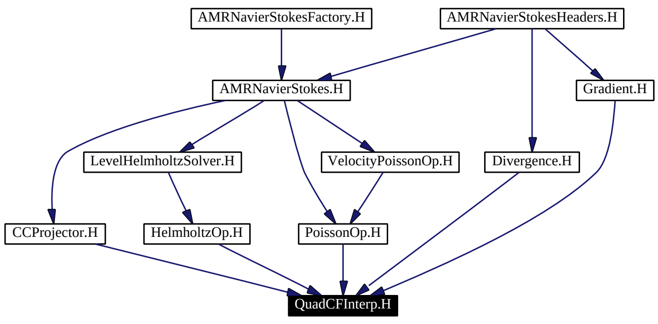
`#include "QuadCFStencil.H"`

`#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.

7.121 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"`

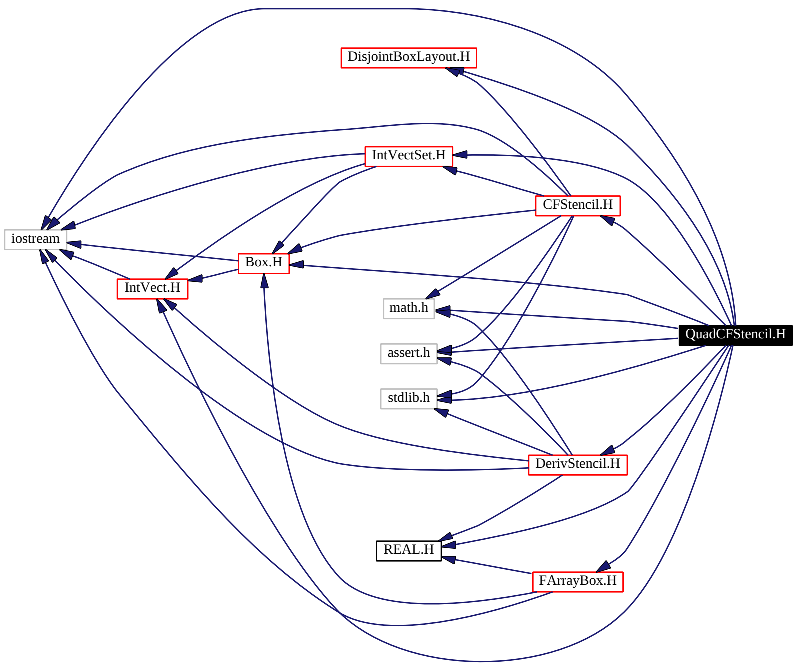
`#include "FArrayBox.H"`

`#include "IntVectSet.H"`

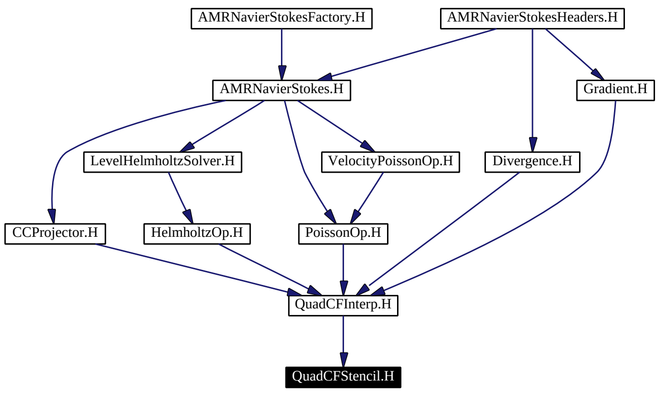
`#include "CFStencil.H"`

`#include "DerivStencil.H"`

Include dependency graph for QuadCFStencil.H:



This graph shows which files directly or indirectly include this file:



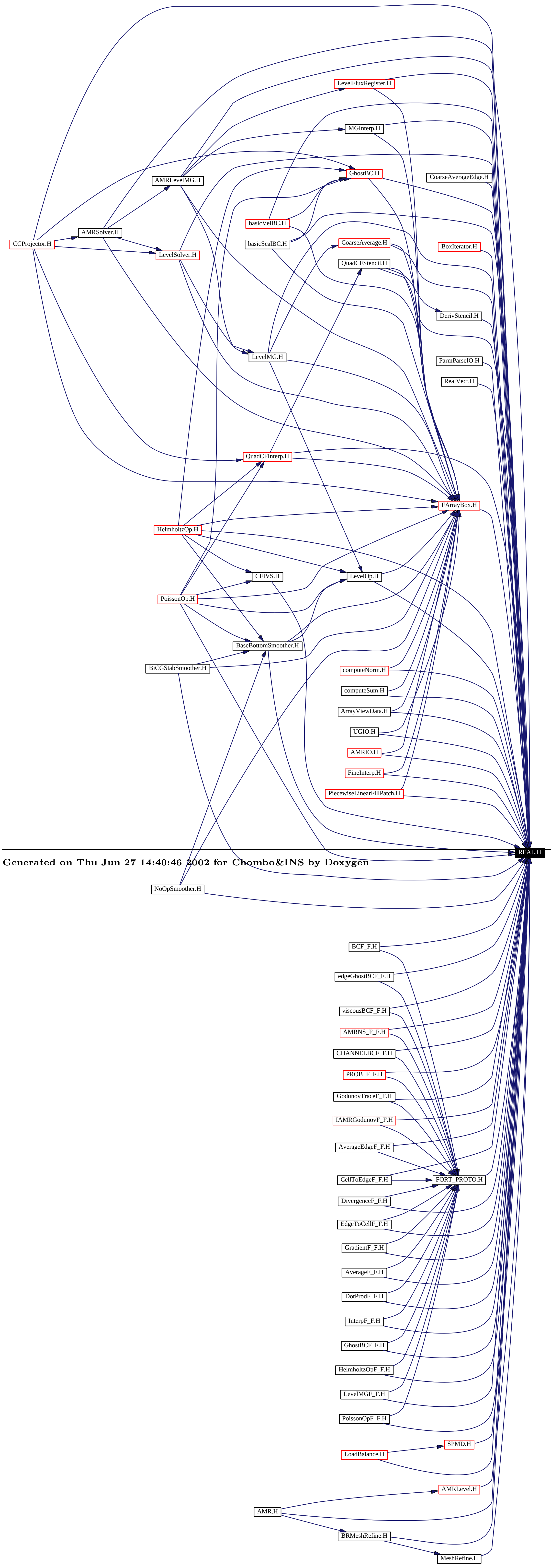
Compounds

- class **QuadCFStencil**

    class to encapsulate CF info for quadratic interpolation

7.122 REAL.H File Reference

This graph shows which files directly or indirectly include this file:


LevelFluxRegister.H
MGInterp.H
GhostBC.H
CoarseAverageEdge.H
AMRLevelMG.H
basicVelBC.H
AMRSolver.H
CCProjector.H
basicScalBC.H
CoarseAverage.H
BoxIterator.H
LevelSolver.H
QuadCFStencil.H
DerivStencil.H
LevelMG.H
ParmParseIO.H
RealVect.H
QuadCFInterp.H
FArrayBox.H
HelmholtzOp.H
CFIVS.H
LevelOp.H
PoissonOp.H
BaseBottomSmoother.H
BiCGStabSmoother.H
computeNorm.H
computeSum.H
ArrayViewData.H
UGIO.H
AMRIO.H
FineInterp.H
PiecewiseLinearFillPatch.H
REAL.H
NoOpSmoother.H
BCF_F.H
edgeGhostBCF_F.H
viscousBCF_F.H
AMRNS_F_F.H
CHANNELBCF_F.H
PROB_F_F.H
GodunovTraceF_F.H
IAMRGodunovF_F.H
AverageEdgeF_F.H
CellToEdgeF_F.H
FORT_PROTO.H
DivergenceF_F.H
EdgeToCellF_F.H
GradientF_F.H
AverageF_F.H
DotProdF_F.H
InterpF_F.H
GhostBCF_F.H
HelmholtzOpF_F.H
LevelMGF_F.H
PoissonOpF_F.H
LoadBalance.H
SPMD.H
AMRLevel.H
AMR.H
BRMeshRefine.H
MeshRefine.H

Defines

- #define **CH_USE_DOUBLE** 1
- #define **MPI_CH_REAL** MPI_DOUBLE
- #define **H5T_NATIVE_REAL** H5T_NATIVE_DOUBLE

Typedefs

- typedef double **Real**

7.122.1 Define Documentation

7.122.1.1 #define CH_USE_DOUBLE 1

7.122.1.2 #define H5T_NATIVE_REAL H5T_NATIVE_DOUBLE

7.122.1.3 #define MPI_CH_REAL MPI_DOUBLE

7.122.2 Typedef Documentation

7.122.2.1 typedef double Real

7.123 RealVect.H File Reference

#include <cstddef>

#include <cstdlib>

#include <cstring>

#include <iostream>

#include "SPACE.H"

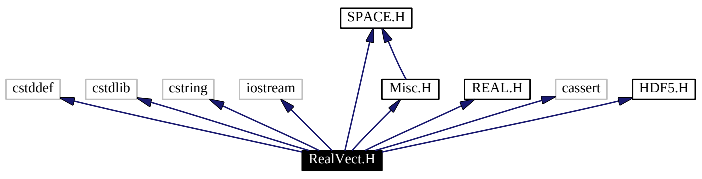
#include "Misc.H"

#include "REAL.H"

#include <cassert>

#include "HDF5.H"

Include dependency graph for RealVect.H:



Compounds

- class **RealVect**

Functions

- 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)

7.123.1 Function Documentation

7.123.1.1 template<> void linearIn (RealVect & *a_outputT*, const void *const *inBuf*)

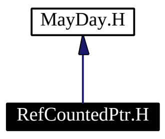
7.123.1.2 template<> void linearOut (void *const *a_outBuf*, const RealVect & *a_inputT*)

7.123.1.3 template<> int linearSize (const RealVect & *vindex*)

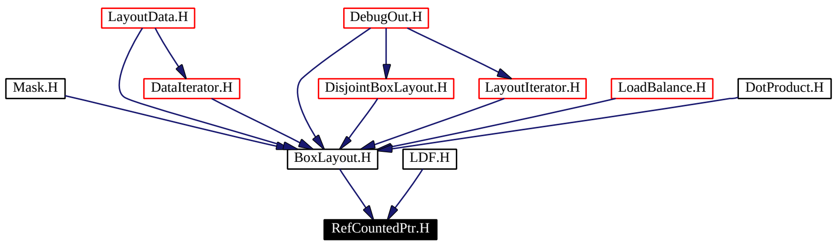
7.124 RefCountedPtr.H File Reference

`#include "MayDay.H"`

Include dependency graph for RefCountedPtr.H:



This graph shows which files directly or indirectly include this file:



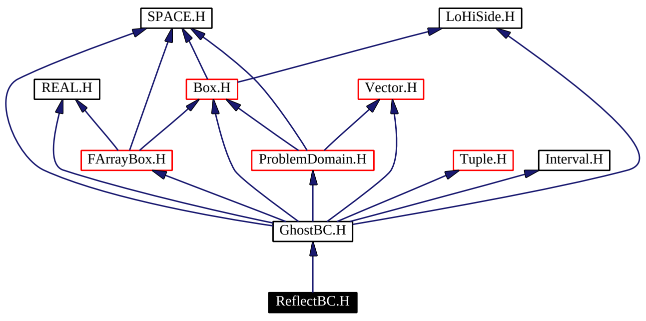
Compounds

- class **RefCountedPtr**

7.125 ReflectBC.H File Reference

`#include "GhostBC.H"`

Include dependency graph for ReflectBC.H:



Compounds

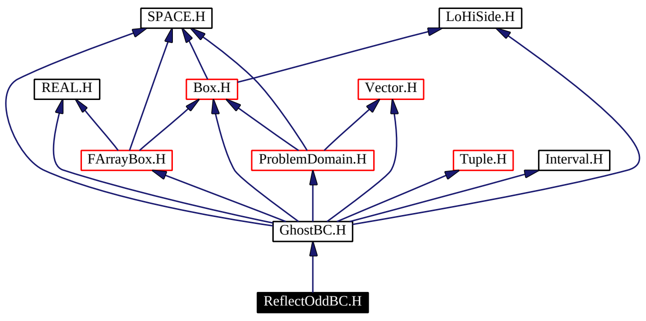
- class **ReflectBC**

  class to do extrapolation BC's

7.126 ReflectOddBC.H File Reference

`#include "GhostBC.H"`

Include dependency graph for ReflectOddBC.H:



Compounds

- class **ReflectOddBC**

  class to do extrapolation BC's

7.127 scalarDirichletBC.H File Reference

`#include "Box.H"`

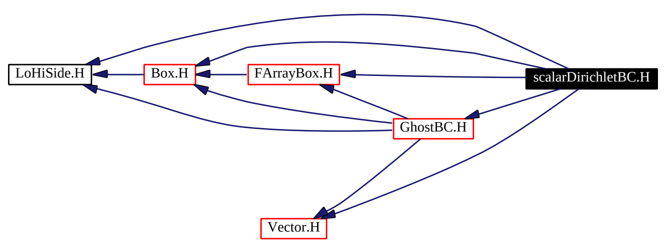
`#include "FArrayBox.H"`

`#include "Vector.H"`

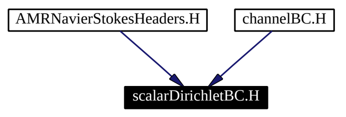
`#include "LoHiSide.H"`

`#include "GhostBC.H"`

Include dependency graph for scalarDirichletBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

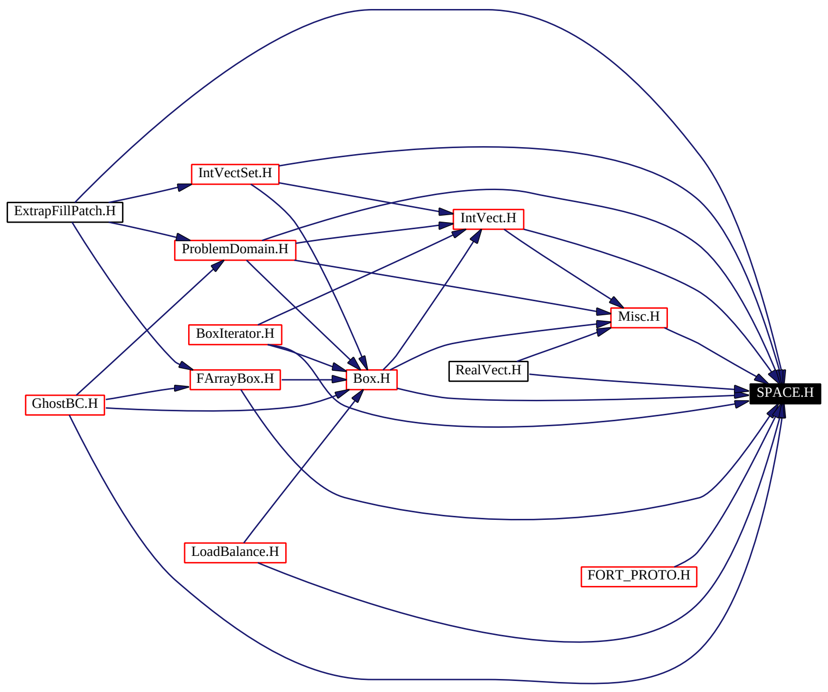
- class **scalarDirichletBC**

    implements dirichlet BCs for scalars

- class **scalarInflowBC**

7.128 SPACE.H File Reference

This graph shows which files directly or indirectly include this file:



Variables

- const int **SpaceDim** = CH_SPACEDIM

7.128.1 Variable Documentation

7.128.1.1 const int SpaceDim = CH_SPACEDIM

7.129 SPMD.H File Reference

`#include "REAL.H"`

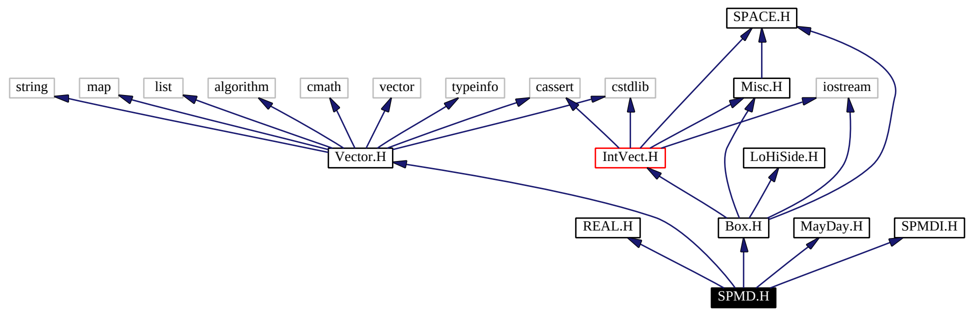
`#include "Vector.H"`

`#include "Box.H"`

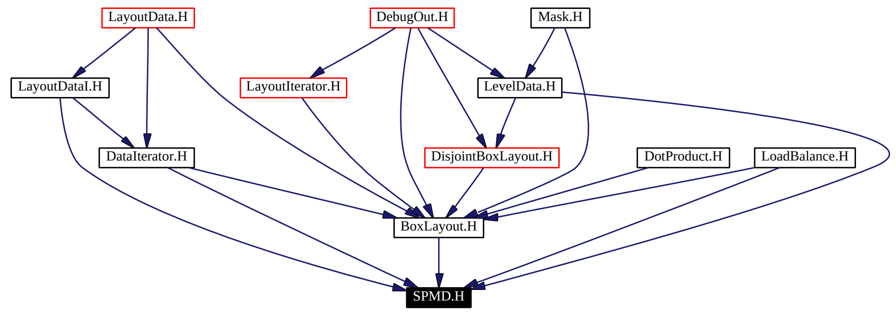
`#include "MayDay.H"`

`#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<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**

7.129.1 Function Documentation

7.129.1.1 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**() (p. 695). \

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); \

7.129.1.2 template<> void linearIn (Vector< Vector< Box > > & *a_outputT*, const void *const *inBuf*)

7.129.1.3 template<> void linearIn (Vector< Box > & *a_outputT*, const void *const *inBuf*)

7.129.1.4 template<> void linearIn (Vector< Real > & *a_outputT*, const void *const *inBuf*)

7.129.1.5 template<> void linearIn (Vector< int > & *a_outputT*, const void *const *inBuf*)

7.129.1.6 template<> void linearIn (Box & *a_outputT*, const void *const *a_inBuf*)

7.129.1.7 template<> void linearIn (Real & *a_outputT*, const void *const *a_inBuf*)

7.129.1.8 template<> void linearIn (int & *a_outputT*, const void *const *inBuf*)

7.129.1.9 template<class T> void linearIn (T & *a_outputT*, const void *const *inBuf*)

7.129.1.10 template<class T> void linearListIn (Vector< T > & *a_outputT*, const void *const *a_inBuf*)

7.129.1.11 template<class T> void linearListOut (void *const *a_outBuf*, const Vector< T > & *a_inputT*)

7.129.1.12 template<class T> int linearListSize (const Vector< T > & *a_input*)

7.129.1.13 template<> void linearOut (void *const *a_outBuf*, const Vector< Vector< Box > > & *a_inputT*)

7.129.1.14 template<> void linearOut (void *const *a_outBuf*, const Vector< Box > & *a_inputT*)

7.129.1.15 template<> void linearOut (void *const *a_outBuf*, const Vector< Real > & *a_inputT*)

7.129.1.16 template<> void linearOut (void *const *a_outBuf*, const Vector< int > & *a_inputT*)

7.129.1.17 template<> void linearOut (void *const *a_outBuf*, const Box & *a_inputT*)

7.129.1.18 template<> void linearOut (void *const *a_outBuf*, const Real & *a_inputT*)

7.129.1.19 template<> void linearOut (void *const *a_outBuf*, const int & *a_inputT*)

7.129.1.20 template<class T> void linearOut (void *const *a_outBuf*, const T & *inputT*)

7.129.1.21 template<> int linearSize (const Vector< Vector< Box > > & *a_input*)

7.129.1.22 template<> int linearSize (const Vector< Box > & *a_input*)

7.129.1.23 template<> int linearSize (const Vector< Real > & *a_input*)

7.129.1.24 template<> int linearSize (const Vector< int > & *a_input*)

7.129.1.25 template<> int linearSize (const Box & *a_input*)

7.129.1.26 template<> int linearSize (const Real & *a_input*)

7.129.1.27 template<> int linearSize (const int & *a_input*)

7.129.1.28 template<class T> int linearSize (const T & *inputT*)

7.129.1.29 unsigned int numProc ()

number of parallel processes

Returns the number of parallel processes running. Always returns at least 1.

7.129.1.30 int procID ()

local process ID

Returns the ID of the locally running process in the range 0 <= **procID**() (p. 696) < **numProc**() (p. 695). This has no relation to the operating system pid. There is always a **procID**() (p. 696) == 0.

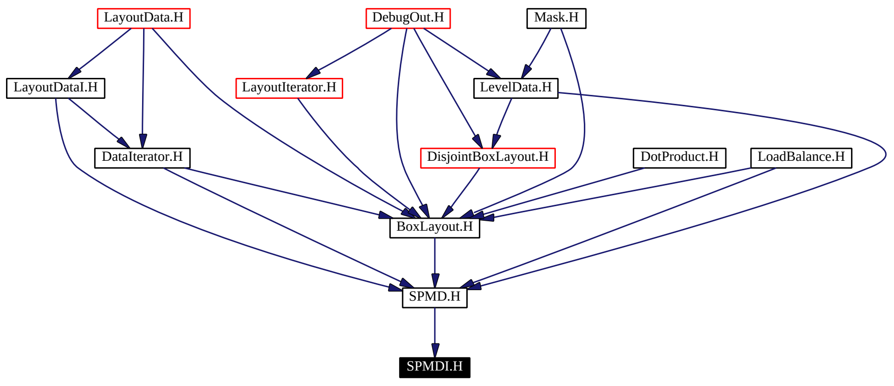
7.129.1.31 int uniqueProc (const SerialTask::task & *a_task*)

7.129.2 Variable Documentation

7.129.2.1 int num_procs

7.130 SPMDI.H File Reference

This graph shows which files directly or indirectly include this file:



Functions

- 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)

7.130.1 Function Documentation

7.130.1.1 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**() (p. 695). \

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); \

7.130.1.2 template<class T> void gather (Vector< T > & *a_outVec*, const T & *a_input*, int *a_dest*) [inline]

7.130.1.3 template<class T> void linearListIn (Vector< T > & *a_outputT*, const void ∗const *a_inBuf*)

7.130.1.4 template<class T> void linearListOut (void ∗const *a_outBuf*, const Vector< T > & *a_input*)

7.130.1.5 template<class T> int linearListSize (const Vector< T > & *a_input*)

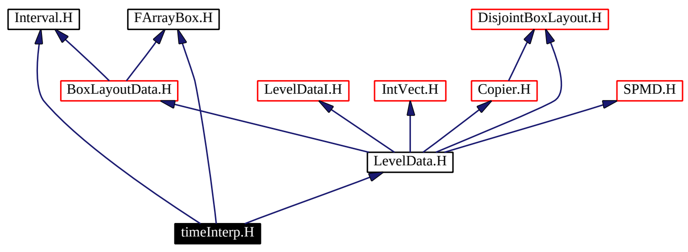
7.131 timeInterp.H File Reference

`#include "LevelData.H"`

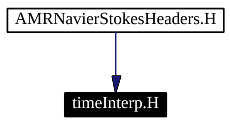
`#include "FArrayBox.H"`

`#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

7.131.1 Function Documentation

7.131.1.1 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

7.131.1.2 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*)

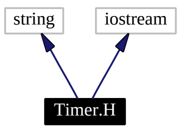
interpolate data in time – assumes one-to-one correspondence in comps

7.132 Timer.H File Reference

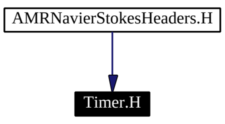
`#include <string>`

`#include <iostream>`

Include dependency graph for Timer.H:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **std**

Compounds

- class **Timer**

Defines

- #define **DIAG_TABLE1** 1
- #define **DIAG_TABLE2** 2
- #define **DIAG_TABLE3** 3

7.132.1 Define Documentation

7.132.1.1 #define DIAG_TABLE1 1

7.132.1.2 #define DIAG_TABLE2 2

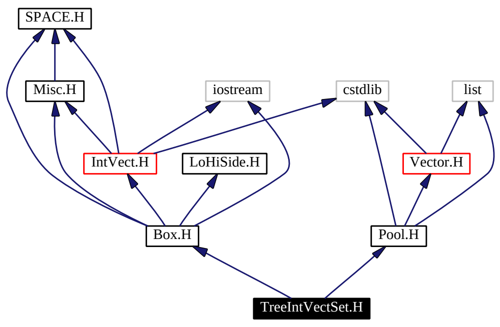
7.132.1.3 #define DIAG_TABLE3 3

7.133 TreeIntVectSet.H File Reference

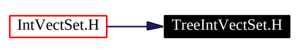
`#include "Box.H"`

`#include "Pool.H"`

Include dependency graph for TreeIntVectSet.H:



This graph shows which files directly or indirectly include this file:



Compounds

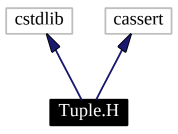
- class **TreeIntVectSet**
- struct **TreeNode**
- class **TreeIntVectSetIterator**

7.134 Tuple.H File Reference

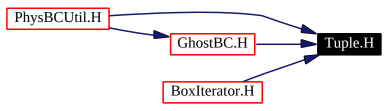
`#include <cstdlib>`

`#include <cassert>`

Include dependency graph for Tuple.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Tuple**

7.135 UGIO.H File Reference

`#include "LevelData.H"`

`#include "REAL.H"`

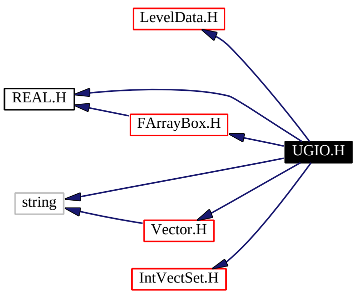
`#include "Vector.H"`

`#include "FArrayBox.H"`

`#include "IntVectSet.H"`

`#include <string>`

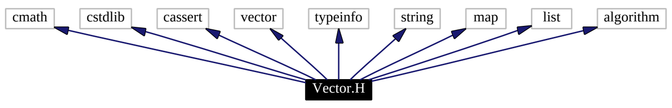
Include dependency graph for UGIO.H:



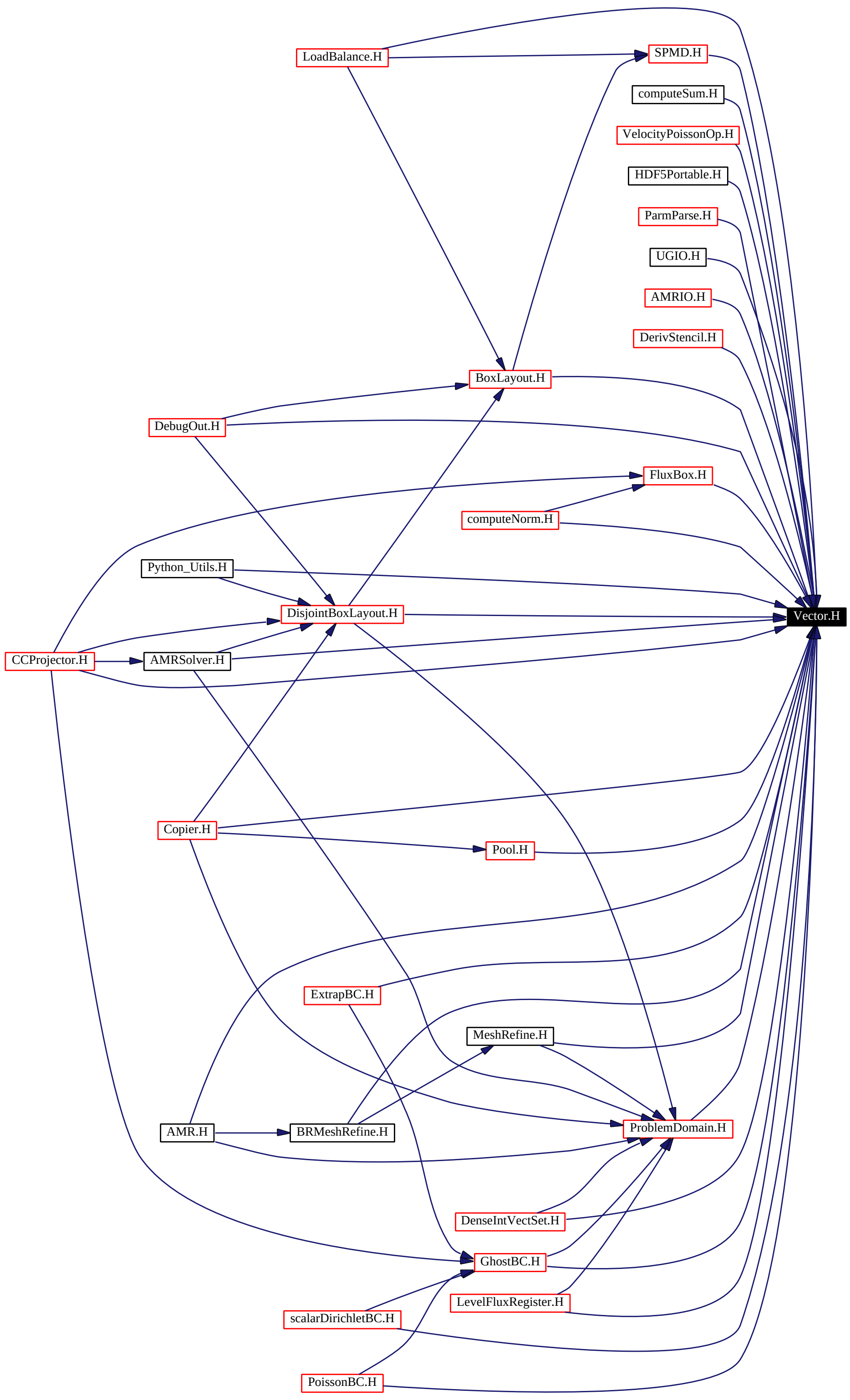
7.136 Vector.H File Reference

```
#include <cmath>
#include <cstdlib>
#include <cassert>
#include <vector>
#include <typeinfo>
#include <string>
#include <map>
#include <list>
#include <algorithm>
```

Include dependency graph for Vector.H:



This graph shows which files directly or indirectly include this file:


LoadBalance.H
SPMD.H
computeSum.H
VelocityPoissonOp.H
HDF5Portable.H
ParmParse.H
UGIO.H
AMRIO.H
DerivStencil.H
BoxLayout.H
DebugOut.H
FluxBox.H
computeNorm.H
Python_Utils.H
DisjointBoxLayout.H
Vector.H
CCProjector.H
AMRSolver.H
Copier.H
Pool.H
ExtrapBC.H
MeshRefine.H
AMR.H
BRMeshRefine.H
ProblemDomain.H
DenseIntVectSet.H
GhostBC.H
LevelFluxRegister.H
scalarDirichletBC.H
PoissonBC.H

Compounds

- class **Vector**

  one dimensional dynamic array

Typedefs

- typedef std::pair< long int *, long int * > **ppair**
- typedef std::map< std::string, **ppair** > **VectorList**
- typedef std::list< int > **lst**
- typedef std::map< std::string, **lst** * > **incr**

Functions

- template<class T> ostream & **operator**<< (ostream &os, const **Vector**< T > &vec)

Variables

- **VectorList * vectorList_**
- **incr * vectorIncr_**

7.136.1 Typedef Documentation

7.136.1.1 typedef std::map<std::string, lst*> incr

7.136.1.2 typedef std::list<int> lst

7.136.1.3 typedef std::pair<long int*, long int*> ppair

7.136.1.4 typedef std::map<std::string, ppair > VectorList

7.136.2 Function Documentation

7.136.2.1 template<class T> ostream& operator<< (ostream & *os*, const Vector< T > & *vec*)

7.136.3 Variable Documentation

7.136.3.1 incr* vectorIncr_

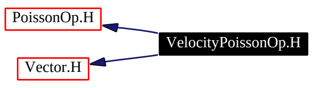
7.136.3.2 VectorList* vectorList_

7.137 VelocityPoissonOp.H File Reference

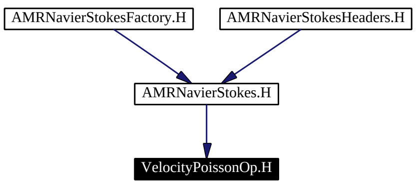
`#include "PoissonOp.H"`

`#include "Vector.H"`

Include dependency graph for VelocityPoissonOp.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **VelocityPoissonOp**

7.138 viscousBC.H File Reference

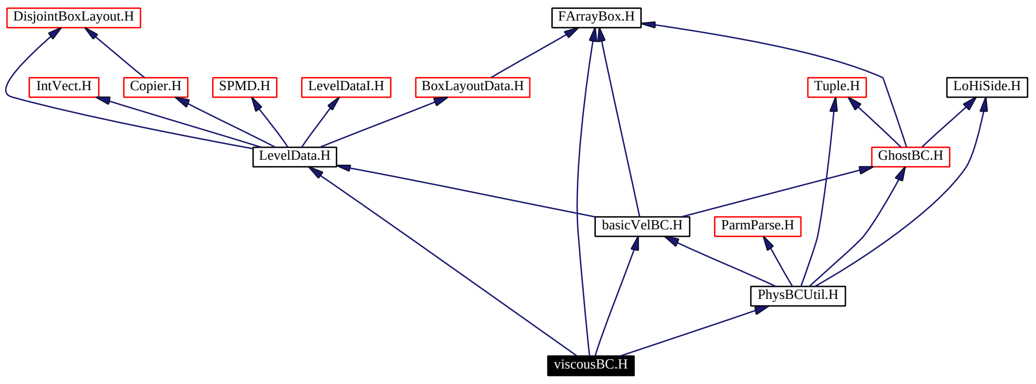
`#include <LevelData.H>`

`#include <FArrayBox.H>`

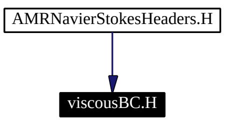
`#include "PhysBCUtil.H"`

`#include "basicVelBC.H"`

Include dependency graph for viscousBC.H:



This graph shows which files directly or indirectly include this file:



Compounds

- class **viscousBC**

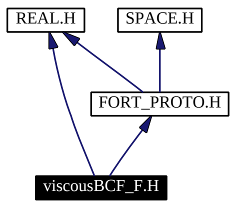
  implements boundary conditions on viscous terms themselves

7.139 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))

7.139.1 Define Documentation

7.139.1.1 #define FORT_VISCOUSBC FORTRAN_NAME(VISCOUSBC ,viscousbc)

7.139.2 Function Documentation

7.139.2.1 void FORT_VISCOUSBC (CHFp_FRA(viscous), CHFp_BOX(gridBox), CHFp_BOX(loBox), CHFp_BOX(hiBox), CHFp_INT(dir), CHFp_INT(nComp))

Index