Manual Entries

1

enum **maskValues**

Members

COVERED
PHYS
COPY
COARSE

2

enum **probType**

Members

LAPLACIAN
VAR_COEFF
HELMHOLTZ

3

typedef BaseFab<int> **MaskFab**

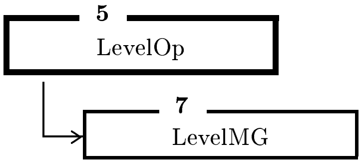
4

typedef FabArray<int,MaskFab> **MaskMultiFab**

5

class **LevelOp**

Inheritance



Public Members

	LevelOp ()
	LevelOp (BoxArray& Bx, BOX domain, const Geometry& _geom, PhysBCFcn F, int n_comp = 1) *Constructor for Laplacian.*
void	**define** (BoxArray& Bx, BOX domain, const Geometry& _geom, PhysBCFcn F, int n_comp = 1) *Corresponding define function.*
	LevelOp (BoxArray& Bx, BOX domain, const Geometry& _geom, LevelFlux& Beta, PhysBCFcn F, int n_comp = 1) *Constructor for variable coefficient case.*
void	**define** (BoxArray& Bx, BOX domain, const Geometry& _geom, LevelFlux& Beta, PhysBCFcn F, int n_comp = 1) *Corresponding define function.*
	LevelOp (BoxArray& Bx, BOX domain, const Geometry& _geom, MultiFab& Alpha, LevelFlux& Beta, PhysBCFcn F, int n_comp = 1) *Constructor for variable Helmholtz case.*
void	**define** (BoxArray& Bx, BOX domain, const Geometry& _geom, MultiFab& Alpha, LevelFlux& Beta, PhysBCFcn F, int n_comp = 1) *Corresponding define function.*
	LevelOp (LevelOp& L, int refRatio) *Constructor for coarsening from finer LevelOp.*
void	**resetAlpha** (MultiFab& Alpha)

		Reset coefficients. Can only be done if the original constructor had these coefficients defined.
void	**resetBeta** (LevelFlux& Beta)	
	~LevelOp ()	*Destructor .*
void	**smooth** (MultiFab& phi, MultiFab& rhs, const REAL* dxCrse)	*Smoother. Assumes that problem has already been put in residual correction form, so that C/F boundary conditions are homogeneous.*
void	**ApplyOp** (MultiFab& phi, MultiFab& phiCoarse, MultiFab& LOfPhi, int refRatio, bool isHomogeneous = false)	*Evaluate Operator, inhomogeneous C/F boundary conditions. – isHomogeneous refers to form of the physical boundary conditions to be applied*
void	**ApplyOp** (MultiFab& phi, MultiFab& LOfPhi, const REAL* dxCrse, bool isHomogeneous = true)	*Evaluate Operator, homogeneous C/F boundary conditions. – isHomogeneous refers to the form of the physical boundary condition to be applied*
void	**CFInterp** (MultiFab& phi, MultiFab& phiCoarse, int refRatio)	*Coarse / Fine interpolation operator.*
void	**homogeneousCFInterp** (MultiFab& phi, const REAL* dxCrse)	*homogeneous coarse/fine interpolation operator*
void	**getFluxes** (FluxRegister& F, MultiFab& finePhi, MultiFab& crsePhi, int refRatio)	*Get fluxes at outer boundaries.*
void	**addFineFluxes** (FluxRegister& flux, MultiFab& finePhi, MultiFab& crsePhi, int refRatio, REAL mult)	*get fine-level fluxes – assumption that grids for mask and finePhi are correlated*
void	**addCrseFluxes** (FluxRegister& flux, MultiFab& Phi, int refRatio, REAL mult)	*add crse-level fluxes – again, assumption that mask and finePhi are correlated*
void	**BottomSmoother** (MultiFab& phi, MultiFab& rhs, const REAL* dxCrse)	*smoother at bottom level – for now, just a lotta gsrb*
		Access Functions
BOX	**Domain** () const	
const BoxArray&	**grids** () const	
const Geometry&	**Geom** () const	
const int&	**numComp** () const	
MaskMultiFab&	**getMasks** ()	*this one is probably temporary*

Protected Members

BOX	**DProblem**	
MultiFab	**alpha**	
LevelFlux	**beta**	
BoxArray	**bx**	
Geometry	**geom**	
int	**nComp**	
PhysBCFcn	**f_bc**	
MaskMultiFab	**masks**	
int	**prob_type**	
BoxAssoc	**ba**	
		worker functions:
virtual void	**buildMasks** ()	*function to build mask arrays*
void	**levelGSRB** (MultiFab& phi, MultiFab& rhs, const REAL* dxCrse)	*perform a GSRB iteration on this level*
bool	**doPhysBC** (FARRAYBOX& phi, BaseFab<int>& mask, bool& isHomogeneous)	*function to do physical BC's; returns 1 if physical bc is applied*

this class contains stuff to solve (alpha + div(beta*grad))phi = rhs

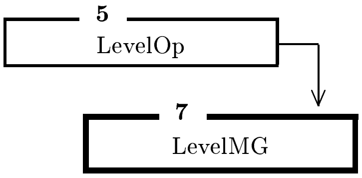
6

void **AvgDown** (MultiFab& fine, MultiFab& crse, int nref)

7

class **LevelMG** : public LevelOp

Inheritance



Public Members

	LevelMG ()	*Default constructor.*
	LevelMG (BoxArray ba, BOX domain, const Geometry& _geom, PhysBCFcn F, int n_coarser_levels=0, int N_comp = 1, int n_ref = 2)	*Constructor with BoxArray, number of coarser levels.*
void	**define** (BoxArray ba, BOX domain, const Geometry& _geom, PhysBCFcn F, int n_coarser_levels=0, int N_comp = 1, int n_ref = 2)	*Corresponding define function.*
	LevelMG (LevelMG& L, int refCoarse = 2)	

Constructor for coarsened version of object.

	~LevelMG ()	*Destructor.*
		Access functions. These are used to pass arguments to the multigrid relaxation step, since the multigrid solver itself does not own the data.
MultiFab&	**getSoln** ()	*Solution.*
MultiFab&	**getRhs** ()	*Right-hand side.*
void	**assignRhs** (MultiFab* rhs)	
void	**mgRelax** (int NumIter, const REAL* dxCrse, bool bottomSolveFlag = false)	*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.*
void	**CrseCorrect** (MultiFab& soln, MultiFab& crseCorr, int nref)	*apply correction from crse level to soln on this level*

Protected Members

MultiFab*	**Soln**	
MultiFab*	**Rhs**	
int	**nref**	*refinement ratio to coarser level*
int	**nCoarserLevels**	
LevelMG*	**lCoarse**	*coarser level*

8

class **LevelPoisson**

Public Members

	LevelPoisson ()	*constructor*
	~LevelPoisson ()	*destructor*
	LevelPoisson (AMRPoisson* papa, const BoxArray& _grids, BOX _domain, const Geometry& _geom, int level, int nRef=2)	*complete constructor*
void	**define** (AMRPoisson* papa, const BoxArray& _grids, BOX _domain, const Geometry& _geom, int level, int nRef=2)	*define level*
void	**LNf** ()	*computes LNf(corr) at the present level, using coarser grid values for coarse/fine boundary conditions – places result in LofPhi*
void	**L** (MultiFab& L_of_phi)	*complete operator*

void	**LevelRelax** (int numMGiter)	*relax on this level*
void	**initData** ()	*initialize data on new (not preexisting) level*
void	**initData** (LevelPoisson& oldLevel)	*re-initialize data on pre-existing level after regridding*
REAL	**residual** ()	*compute residual on this level, returns maximum on this level*
void	**errorEst** (TagBoxArray& tb, int clearval, int tagval)	*tag cells for refinement*
void	**richardsonExtrap** (TagBoxArray& tb, int clearval, int tagval)	*tag cells for refinement based on Richardson extrapolation*
REAL	**maxRHS** ()	*returns maximum of RHS on this level*
void	**writePlotFile** (ostream& os)	*write plotfile for this level*
		access functions
BOX	**Domain** () const	
const BoxArray&	**getGrids** () const	
const BoxArray&	**getCrseGrids** () const	*grids, coarsened down to next lower level*
MultiFab&	**get_phi** ()	
MultiFab&	**get_rhs** ()	
const Geometry&	**Geom** () const	*returns solver.Geom()*

Protected Members

LevelPoisson *	**lower**	*pointers to level above, below*
LevelPoisson *	**higher**	
AMRPoisson *	**parent**	*pointer to parent AMR*
BoxArray	**grids**	*grids at this level*
BoxArray	**crseGrids**	*grids coarsened by nRefine (projection onto coarser level)*
BOX	**domain**	*problem domain at this level*
MultiFab	**phi**	
MultiFab	**res**	*residual*
MultiFab	**corr**	*correction*
MultiFab	**phiSave**	*extra copy of solution*
MultiFab	**dCorr**	*correction to the correction*
MultiFab	**LofPhi**	*place to store result of operator acting on solution*
MultiFab	**corrC**	*correction copied from coarser grids – defined on coarser domain*
MultiFab	**resC**	*residual averaged onto coarsened grid; defined on coarsened domain*
MultiFab	**error**	*estimate of truncation error computed using Richardson Extrapolation*
FluxRegister	**Freg**	*Flux Registers*
BoxAssoc	**baiota**	*Intersection Lists:*

int	**numGrids**	
LevelMG	**solver**	*LevelMG level solver object*
MaskMultiFab& **buildMasks** ()		*builds masks for Richardson Extrapolation*

9 class **AMRPoisson**

Public Members

	AMRPoisson ()	*constructor*
	~AMRPoisson ()	*destructor*
void	**init** ()	*initialize AMRPoisson object from inputs file*
void	**bldBaseLevel** (BoxArray & _ba)	*build and initialize coarsest level*
int	**bldFineLevel** (int level)	*build and initialize finer levels – returns 1 if new level built, 0 else*
int	**buildNewLevel** ()	*build a new finest level?*
void	**resize** (int _numLevels, int nref)	*resize levelData when grids will be determined adaptively (don't know them a priori)*
void	**resize** (Array<BoxArray>& DI, BOX DProblem, int nref)	*resize levelData when entire grid hierarchy is provided*
void	**AMRRelax** ()	*starts multigrid relaxation at finest existing level*
REAL	**residual** ()	*computes residual & returns infinity norm (maximum) of residual on all levels*
REAL	**maxRHS** ()	*returns max RHS over the grid hierarchy*
int	**totalNumCells** ()	*returns total number of cells in AMR hierarchy*
void	**writePlotFile** (char* fname)	*writes plot file (suitable for AMRVis) access functions*
LevelPoisson &	**get_LevelPoisson** (int level)	
int &	**num_levels** ()	
int &	**finest_level** ()	
int &	**refineRatio** ()	

Protected Members

int	**numLevels**	*Maximum number of levels allowed*
int	**nRefine**	*refinement ratio.*
int	**finestLevel**	*finest existing level. (may not have created levels all the way up to numLevels)*
int	**do_richardson_extrap**	*flag – whether or not to do R.E.*
REAL	**ErrorTolerance**	*Error Tolerance for R.E.*
REAL	**max_rhs**	*max RHS (norm for R.E.)*
int	**numMGiter**	*number of relaxation iterations per multigrid*

int **debug** *turns on debugging options*
int **do_plot_file**
ArrayLevelPoisson **levelData** *array of LevelPoisson's containing data at each refinement level*

10

typedef void (* **PhysBCFcn)** (FARRAYBOX& BCInputs, const int StartComp, const int NumComp, const INTVECT Idir, const int IsHomogeneous, const REAL Time, FARRAYBOX& GridPoints, FARRAYBOX& BCData, BaseFab<int>& BCFlags, BOX domain)

Arguments

FARRAYBOX&	**BCInputs**	*contains REAL inputs for BC computation*
const int	**StartComp**	*subrange for which BC's are being computed*
const int	**NumComp**	*number of components for which BC's are being computed*
const INTVECT	**Idir**	*signed unit vector for which BC's are being computed (i.e. an orientation)*
const int	**IsHomogeneous**	*TRUE if homogeneous BC's, FALSE otherwise*
const REAL	**Time**	*time at which BC is centerd*
FARRAYBOX&	**GridPoints**	*Location of the grid*
FARRAYBOX&	**BCData**	*Contains REAL data required to compute BC (returned from function)*
BaseFab<int>&	**BCFlags**	*contains int/flag data for BC*
BOX	**domain**	*physical domain*

11

class **GridGenerator**

Public Members

GridGenerator () *default constructor*

GridGenerator (int _nref) *constructor*

˜GridGenerator () *destructor*

int **TagArrayGrow** (const BoxArray& fineGrids, const BoxArray& crseGrids)
determine number of cells need to grow TagBoxArray by in order to ensure proper nesting is satisfied.

int **grid_places** (PArray<TagBoxArray> & Tags, BoxArray &Domains, PArray<BoxArray> &levelGrids, int lbase, int finest_level, Array<BoxArray> &new_grids)
construct grids from base level lbase – returns finest level after regridding

12 class **CLUSTER**

Public Members

		administrative
	CLUSTER ()	*construct invalid Cluster*
	CLUSTER (Array<INTVECT> *a)	*construct a cluster from an array of INTVECTS NOTE: Cluster object takes over memory management of array*
	CLUSTER (CLUSTER &c, const BOX& b)	*construct new cluster by removing all points from c that lie in box b. Cluster c is modified and may become invalid.*
	~CLUSTER ()	*destructor*
const BOX&	**box** () const	*return minimal box containing all tagged points*
INTVECT	**operator**[] (int i)	*return ith tagged point in cluster*
int	**ok** () const	*does cluster contain any points?*
int	**numTag** () const	*returns number of tagged points in cluster*
int	**numTag** (const BOX& b)	*return number of tagged points in intersection of cluster and BOX b.*
CLUSTER*	**chop** ()	*This operation splits a cluster into two pieces by selecting a cutting plane with certain optimal characteristics then dividing the tagged points into clusters on either side of the plane. One cluster is returned as a new object the other is the modified calling object.*
void	**distribute** (ClusterList &clst, const BoxList &bl)	*constructs a list of cluster objects obtained by intersecting this cluster with each box in bl. The list is retruned in the argument clst. For each intersection that includes tagged points, construct a new cluster by removing those points from this object. Empty intersections or those that contain no tagged points will not generate a new cluster. NOTE that this cluster will be modified and possibly become invalid in the process.*
REAL	**eff** () const	*compute ratio of tagged to total number og points in cluster*

Protected Members

void	**minBox** ()	*compute and store minimal box containing tagged points*

this class is basically a fortran array of INTEGERSs the array is defined to exist for a set of points inside a box at each point an integer value is defined.

13 class **ClusterList**

Public Members

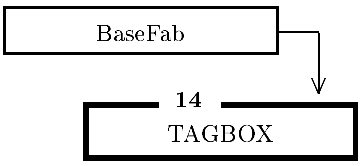
	ClusterList ()	
	ClusterList (Array<INTVECT> *pts)	
	~ClusterList ()	
int	**length** () const	*return number of clusters in list*
void	**append** (CLUSTER* c)	*add cluster to end of list*
BoxArray	**boxArray** ()	*return array of boxes corresponding to clusters*
void	**boxArray** (BoxArray &ba)	*same but modifies argument*
BoxList	**boxList** ()	*return list of boxes corresponding to clusters*
void	**boxList** (BoxList &blst)	*same but modifies argument*
void	**chop** (REAL eff)	*chop all clusters in list that have poor efficiency*
void	**intersect** (const BoxDomain& dom)	*intersect clusters with BoxDomain to insure cluster boxes are interior to domain*

Protected Members

List<CLUSTER*>	**lst**

14 class **TAGBOX** : public BaseFab<int>

Inheritance



Public Members

enum	**TagVal**	*administrative*
	TAGBOX ()	*construct an invalid FAB with no memory*
	TAGBOX (const BOX &bx, int n = 1)	
virtual	**~TAGBOX** ()	*destructor*
void	**resize** (const BOX& b)	
int&	**operator()** (const INTVECT& p)	
const int&	**operator()** (const INTVECT& p) const	

TAGBOX*	**coarsen** (int ratio)	*construct and return a new tagbox in which the coarsened cell is tagged of any of the corresponding fine cells are tagged*
void	**buffer** (int nbuff, int nwid)	*mark neighbors of every tagged cell a distance nbuff away only search interior for initial tagged points where nwid is given as the width of the bndry region.*
void	**merge** (const TAGBOX& src)	*tag cells on intersect with src if corresponding src cell is tagged.*
int	**colate** (Array<INTVECT> &ar, int start) const	*add location of every tagged cell to INTVECT array, starting at given location*
int	**numTags** (const BOX& bx) const	
int	**numTags** () const	

15 class **TagBoxArray**

Public Members

	TagBoxArray (PArrayPolicy _managed=PArrayManage)
	TagBoxArray (int len, PArrayPolicy _managed=PArrayManage)
	TagBoxArray (const BoxArray& bs, int _ngrow=0, PArrayPolicy _managed = PArrayManage)
virtual	**~TagBoxArray** ()
int	**borderSize** ()
void	**buffer** (int nbuf)
void	**clear** ()
void	**mergeUnique** ()
void	**setVal** (const BoxDomain& bd, TAGBOX::TagVal val)
void	**setVal** (const BoxArray& ba, TAGBOX::TagVal val)
void	**coarsen** (int ratio)
TAGBOX&	**operator[]** (int i)
const TAGBOX&	**operator[]** (int i) const
int	**numTags** () const
Array<INTVECT>*	**colate** () const

Protected Members

PArray<TAGBOX>	**tb**
int	**ngrow**

16 class **LevelFlux**

Public Members

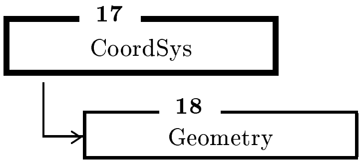
	LevelFlux ()	
	LevelFlux (BoxArray& boxes, int num_var, int growfactor)	*constructor – growfactor is amount by which each set of edges is to be grown or shrunk in direction parallel to edges*
	~LevelFlux ()	
MultiFab&	**operator**[] (int dir)	
void	**define** (BoxArray& boxes, int num_var, int growfactor)	
BoxArray	**ba**	*BoxArray around which fluxes are defined*
int	**ncomp**	
MultiFab*	**Flux**	

LevelFlux is basically a fancy MultiFab to handle edge-centered data

17

class **CoordSys**

Inheritance



Public Members

		——— I/O friend functions ———-
friend ostream&	**operator <<** (ostream&, const CoordSys&)	
friend istream&	**operator >>** (istream&, CoordSys&)	
	CoordSys ()	*Default constructor, CoordSys is undefined*
	CoordSys (const REAL* cell_dx)	*specify a cell size, and low end of index range*
virtual	**~CoordSys** ()	
void	**define** (const REAL* cell_dx)	*initialize after construction*
static void	**SetCoord** (CoordType coord)	*functions to set and access static members*
static CoordType	**Coord** ()	
static int	**IsRZ** ()	
static int	**IsCartesian** ()	
static void	**SetOffset** (const REAL* x_lo)	
static const REAL*	**Offset** ()	
static REAL	**Offset** (int dir)	
const REAL*	**CellSize** () const	*functions to access non-static members*
REAL	**CellSize** (int dir) const	

Index space -> Physical space mappings

void **CellCenter** (const INTVECT& point, Array<REAL>& loc) const

void **CellCenter** (const INTVECT& point, REAL *loc) const
REAL **CellCenter** (int point, int dir) const
REAL **LoEdge** (int point, int dir) const
REAL **LoEdge** (const INTVECT& point, int dir) const
REAL **HiEdge** (int point, int dir) const
REAL **HiEdge** (const INTVECT& point, int dir) const

location of lo and hi cell faces

void **LoFace** (const INTVECT& point, int dir, Array<REAL>& loc) const

void **LoFace** (const INTVECT& point, int dir, REAL *loc) const
void **HiFace** (const INTVECT& point, int dir, Array<REAL>& loc) const

void **HiFace** (const INTVECT& point, int dir, REAL *loc) const

return location of lower left hand corner

void **LoNode** (const INTVECT& point, Array<REAL>& loc) const
void **LoNode** (const INTVECT& point, REAL *loc) const

return location of upper right hand corner

void **HiNode** (const INTVECT& point, Array<REAL>& loc) const
void **HiNode** (const INTVECT& point, REAL *loc) const

INTVECT **CellIndex** (const REAL* point) const
Physical Space -> index space mappings WARNING:: May return undesired results if point is on a cell boundary. returns cell centered index of cell containing point

INTVECT **LowerIndex** (const REAL* point) const
returns node centered index of lower left hand corner of cell containing this point

INTVECT **UpperIndex** (const REAL* point) const
returns node centered index of upper right hand corner of cell containing this point

void **GetVolume** (FARRAYBOX& vol, const BOX& region) const
compute cell volumes in given region – this version resizes input FAB

FARRAYBOX* **GetVolume** (const BOX& region) const
this version returns volume in new FAB

REAL **Volume** (const INTVECT& point) const
these versions return volume of single cell

REAL **Volume** (const REAL xlo[BL_SPACEDIM], const REAL xhi[BL_SPACEDIM]) const
likewise

void **GetFaceArea** (FARRAYBOX& area, const BOX& region, int dir) const
compute area of cell faces in given region and given index direction

FARRAYBOX*

GetFaceArea (const BOX& region, int dir) const
likewise

REAL **AreaLo** (const INTVECT& point, int dir) const
these versions return face areas of given cell in direction dir

REAL **AreaHi** (const INTVECT& point, int dir) const

void **GetEdgeLoc** (Array<REAL>& loc, const BOX& region, int dir) const
return array of physical locations of cell edges – resizes input array

void **GetCellLoc** (Array<REAL>& loc, const BOX& region, int dir) const
return array of physical locations of cell centers – resizes input array

void **getGridPlaces** (FARRAYBOX& gridPlaces) const
return FAB of physical locations of cell centers – assumes extents of FAB have already been defined...

void **GetEdgeVolCoord** (Array<REAL>& loc, const BOX& region, int dir) const
return array of volume coordinates at cell edges for region in given direction

void **GetCellVolCoord** (Array<REAL>& loc, const BOX& region, int dir) const
return array of volume coordinates at cell centers for region in given direction

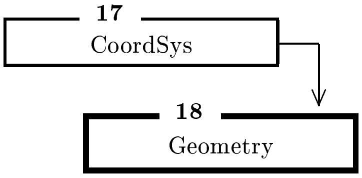
Protected Members

static CoordType **c_sys**
static REAL **offset[BL_SPACEDIM]**
int **ok**
REAL **dx[BL_SPACEDIM]**

18

class **Geometry** : public CoordSys

Inheritance



Public Members

——— I/O friend functions ———-

friend ostream& **operator <<** (ostream&, const Geometry&)
friend istream& **operator >>** (istream&, Geometry&)

	Geometry ()	
	Geometry (const BOX& dom)	
	Geometry (const Geometry& g)	
	~Geometry ()	
static void	**Setup** ()	*read static values from ParmParse database.*
void	**define** (const BOX& dom)	
static const REALBOX&	**ProbDomain** ()	
static const REAL*	**ProbLo** ()	
static const REAL*	**ProbHi** ()	
static REAL	**ProbLo** (int dir)	
static REAL	**ProbHi** (int dir)	
static const REAL*	**ProbLength** ()	
static REAL	**ProbLength** (int dir)	
const BOX&	**Domain** () const	
void	**GetVolume** (MultiFab& vol, const BoxArray& grds, int grow) const	*define a multifab of areas and volumes with given grow factor*
void	**GetFaceArea** (MultiFab& area, const BoxArray& grds, int dir, int grow) const	
void	**refine** (int nref)	
void	**coarsen** (int nref)	

Protected Members

static REALBOX	**prob_domain**
BOX	**domain**
void	**read_params** ()

19

class **FluxRegister**

Public Members

	FluxRegister ()	*default constructor*
	FluxRegister (const BoxArray& fine_boxes, int ref_ratio, int fine_lev, int nvar)	*constructor given boxArray of fine grids*
	~FluxRegister ()	*destructor*
void	**define** (const BoxArray& fine_boxes, int ref_ratio, int fine_lev, int nvar)	
		access functions
int	**refRatio** () const	
int	**fineLevel** () const	
int	**crseLevel** () const	
int	**nComp** () const	

const BoxArray&
coarsenedBoxes () const
const PArray<FARRAYBOX>&
loReg (int dir) const
const PArray<FARRAYBOX>&
hiReg (int dir) const
const FARRAYBOX&
loReg (int boxno, int dir) const
const FARRAYBOX&
hiReg (int boxno, int dir) const
void **initReg** (REAL val) *intialize registers to val*
REAL **SumReg** (int comp) const
returns sum over all flux registers of component comp
void **copyTo** (FARRAYBOX& flx, int dir, int src_comp, int dest_comp, int num_comp)
copy from flux register to fab
void **copyTo** (MultiFab& mf, int dir, int src_comp, int dest_comp, int num_comp)
copy from flux register to multifab
void **addTo** (FARRAYBOX& flx, int dir, REAL mult, int src_comp, int dest_comp, int num_comp)
*flx = flx + mult*register*
void **addTo** (MultiFab& mf, int dir, REAL mult, int src_comp, int dest_comp, int num_comp)
*mf = fm + mult*register*
void **scaleAddTo** (FARRAYBOX& flx, const FARRAYBOX& area, int dir, REAL mult, int src_comp, int dest_comp, int num_comp)
*flx = flx + mult*register/area*
void **scaleAddTo** (MultiFab& mf, const MultiFab& area, int dir, REAL mult, int src_comp, int dest_comp, int num_comp)
*mf = fm + mult*register/area*
void **CrseAdd** (const MultiFab& mflx, int dir, int srccomp, int destcomp, int numcomp, REAL mult = -1.0)
*register(destcomp) = register(destcomp) + mult*mflx(srccomp)*
void **CrseAdd** (const MultiFab& mflx, const MultiFab& area, int dir, int srccomp, int destcomp, int numcomp, REAL mult = -1.0)
*register(destcomp) = register(destcomp) + mult*area*mflx(srccomp)*
void **CrseAdd** (const FARRAYBOX& flux, const BOX& subbox, int dir, int srccomp, int destcomp, int numcomp, REAL mult = -1.0)
CrseAdd with FAB's and subboxes
void **CrseAdd** (const FARRAYBOX& flux, const FARRAYBOX& area, const BOX& subbox, int dir, int srccomp, int destcomp, int numcomp, REAL mult = -1.0)

void **FineAdd** (const MultiFab& mflx, int dir, int srccomp, int destcomp, int numcomp, REAL mult) *register(destcomp) = register(destcomp) + mult*(sum of appropriate mflx (which is defined on fine grid)*

void **FineAdd** (const MultiFab& mflx, const MultiFab& area, int dir, int srccomp, int destcomp, int numcomp, REAL mult)
*register(destcomp) = register(destcomp) + mult*area*mflx(srccomp)*

void **FineAdd** (const FARRAYBOX& flux, int dir, int boxno, int srccomp, int destcomp, int numcomp, REAL mult)

void **FineAdd** (const FARRAYBOX& flux, const FARRAYBOX& area, int dir, int boxno, int srccomp, int destcomp, int numcomp, REAL mult)

void **Reflux** (MultiFab& mf, const MultiFab& volume, REAL scale, int srccomp, int destcomp, int numcomp)
*do refluxing along boundary of fine grid (changes values on crse grid) – mf(destcomp) = mf(destcomp) + mult*reg(srccomp)/vol*

void **Reflux** (MultiFab& mf, const REAL* dx, REAL scale, int srccomp, int destcomp, int numcomp)
constant volume version

void **print** (ostream& os)

Protected Members

BoxArray **crse_boxes** *fine boxes projected to crse level*
int **ratio**
int **fine_level**
int **ncomp**
PArray<FARRAYBOX> **lo[BL_SPACEDIM]**
PArray<FARRAYBOX> **hi[BL_SPACEDIM]**
void **increment** (const FARRAYBOX& fab, int dir)
helper member functions

The FluxRegister class is designed to keep track of the differences between coarse and fine solutions at the coarse-fine interfaces by storing the differences in the fluxes of the coarse-level and fine-level solutions across the cell edges making up the coarse-fine interface. This is used to enforce conservation where appropriate, although they can also be used more generally to keep track of any differences between edge-centered quantities. They are designed to be "owned" by the fine level, although they actually contain coarsened data.

20

class **REALBOX**

Public Members

——— I/O friend functions ———-

friend ostream&

	operator << (ostream&, const REALBOX&)	
friend istream&	**operator >>** (istream&, REALBOX&)	
		———— constructors —————
	REALBOX ()	*construct invalid region*
	REALBOX (const REAL *lo, const REAL* hi)	*construct region given diagonal points*
	REALBOX (const BOX& bx, const REAL* dx, const REAL* base)	*construct region given index box, cell spacing and physical location of index (0, 0, 0)*
	REALBOX (istream &is)	*read region from input file*
	REALBOX (D_DECL(REAL x0, REAL y0, REAL z0), D_DECL(REAL x1, REAL y1, REAL z1))	*explicit dimension specific constructors*
		———- static functions ————
static void	**setEpsilon** (REAL epsilon)	*set epsilon for determining intersections*
static REAL	**epsilon** ()	
		———— access functions ———-
const REAL*	**lo** () const	
const REAL*	**hi** () const	
const REAL*	**length** () const	
REAL	**lo** (int dir) const	
REAL	**hi** (int dir) const	
REAL	**length** (int dir) const	
		———- Modify functions ————-
void	**setLo** (const REAL* lo)	
void	**setLo** (const Array<REAL> &lo)	
void	**setHi** (const REAL* hi)	
void	**setHi** (const Array<REAL> &hi)	
void	**setLo** (int indx, REAL lo)	
void	**setHi** (int indx, REAL hi)	
		———- Boolean functions ————
int	**ok** () const	
int	**contains** (const REAL* point)	
int	**contains** (const REALBOX &bx)	
int	**intersects** (const REALBOX &bx)	
		——- intersection functions ———-
REALBOX&	**operator &=** (const REALBOX& bx)	
REALBOX	**operator &** (const REALBOX& bx)	
		——— conversion functions ———-
BOX	**CCBox** (const REAL *dx, const REAL *base)	
BOX	**NodeBox** (const REAL *dx, const REAL* base)	

Protected Members

REAL	**xlo[BL_SPACEDIM]**
REAL	**len[BL_SPACEDIM]**
static REAL	**eps**
void	**computeBoxLen** ()

Class Graph