APC_524

Generated by Doxygen 1.8.12

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

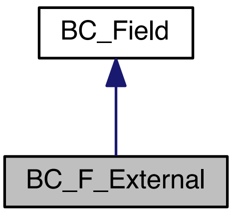
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

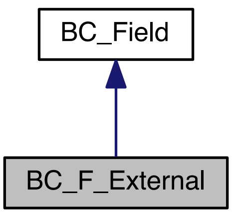
Class Documentation

3.1 BC_F_External Class Reference

Inheritance diagram for BC_F_External:



Collaboration diagram for BC_F_External:



Public Member Functions

- **BC_F_External** (int side, Domain ∗domain, Grid ∗grids, Input_Info_t ∗info)
- int **completeBC** ()

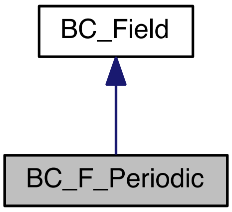
Additional Inherited Members

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

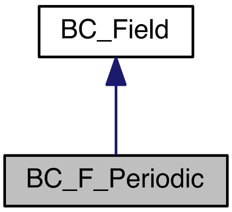
- src/boundaries/b_fields/bc_f_external.cpp

3.2 BC_F_Periodic Class Reference

Inheritance diagram for BC_F_Periodic:



Collaboration diagram for BC_F_Periodic:



Public Member Functions

- **BC_F_Periodic** (int side, Domain *domain, Grid *grids, Input_Info_t *info)
- int **completeBC** ()

Additional Inherited Members

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

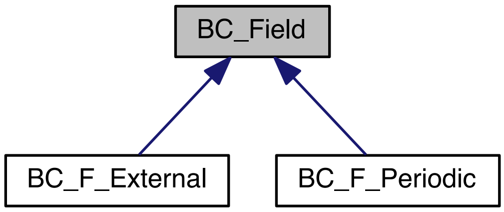
- src/boundaries/b_fields/bc_f_periodic.cpp

3.3 BC_Field Class Reference

Class which defines a field boundary condition.

```
#include <fields_boundary.hpp>
```

Inheritance diagram for BC_Field:



Public Member Functions

- virtual int **completeBC** (void)=0

Protected Attributes

- std::string **type_**
- int **side_**
- double ∗ **ghostTmp_**

3.3.1 Detailed Description

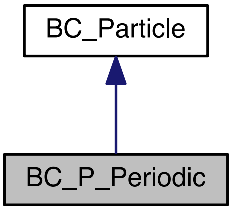
Class which defines a field boundary condition.

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

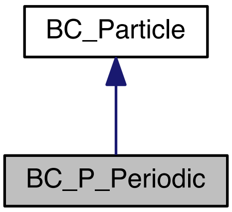
- src/boundaries/fields_boundary.hpp

3.4 BC_P_Periodic Class Reference

Inheritance diagram for BC_P_Periodic:



Collaboration diagram for BC_P_Periodic:



Public Member Functions

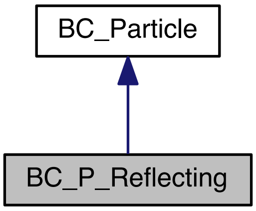
- **BC_P_Periodic** (Domain ∗domain, int dim_Index, short isRight, std::string type)
- void **computeParticleBCs** (std::vector< Particle > ∗pl)
- int **completeBC** (std::vector< Particle > ∗pl)

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

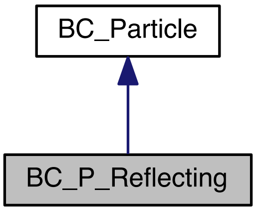
- src/boundaries/b_particles/bc_p_periodic.cpp

3.5 BC_P_Reflecting Class Reference

Inheritance diagram for BC_P_Reflecting:



Collaboration diagram for BC_P_Reflecting:



Public Member Functions

- **BC_P_Reflecting** (Domain ∗domain, int dim_Index, short isRight, std::string type)
- void **computeParticleBCs** (std::vector< Particle > ∗pl)
- int **completeBC** (std::vector< Particle > ∗pl)

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

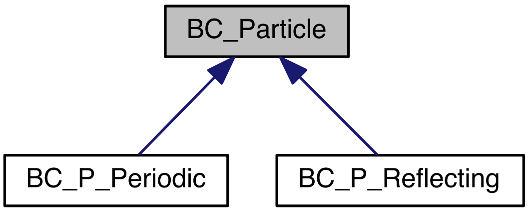
- src/boundaries/b_particles/bc_p_reflecting.cpp

3.6 BC_Particle Class Reference

Class which defines a particle boundary condition.

`#include <particles_boundary.hpp>`

Inheritance diagram for BC_Particle:



Public Member Functions

- int **computeParticleBCs** (std::vector< Particle > ∗pl)

3.6.1 Detailed Description

Class which defines a particle boundary condition.

Boundary conditions have two stages.

1st stage: Cycling through particle list and determining which particles need to have boundary conditions applied, then applies them.

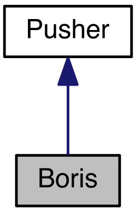
2nd stage: Perform any more auxilliary computations, including MPI calls, creating new ghost particles, shuffling particles ETC...

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

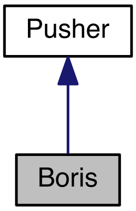
- src/boundaries/particles_boundary.hpp
- src/boundaries/particles_boundary.cpp

3.7 Boris Class Reference

Inheritance diagram for Boris:



Collaboration diagram for Boris:



Public Member Functions

- int **Step** (Particle ∗part, Field_part ∗field, double dt)

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

- src/pusher/boris.hpp
- src/pusher/boris.cpp

3.8 Depositor Class Reference

Public Member Functions

- void **deposit_particle_J** (Particle ∗part, double ∗lcell, double ∗cellverts, double ∗JObj)
- void **deposit_particle_Rho** (Particle ∗part, double ∗lcell, double ∗cellverts, double ∗RhoObj)

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

- src/particles/deposit.hpp
- src/particles/deposit.cpp

3.9 Domain Class Reference

Public Member Functions

- **Domain** (Input_Info_t *input_info)
- int **getnGhosts** (void)
- int * **getnxyz** (void)
- int * **getn2xyz** (void)
- double * **getxyz0** (void)
- double * **getLxyz** (void)
- double getmindx (void)

    Find minimum grid size.
- void mallocGhosts (int xgsize, int ygsize, int zgsize)

    Allocate ghost buffers for MPI.
- void **freeGhosts** (void)
- void PassFields (Grid *grids, Input_Info_t *input_info, int sendID, int op)

    Pass fields across boundaries, or execute physical boundary conditions.
- double GetMaxValueAcrossDomains (double send_val)

    Find maximum of values across MPI domains.
- int * **getnProcxyz** (void)
- int * **getmyijk** (void)
- int * **getNeighbours** ()
- int **getxl** (void)
- int **getyl** (void)
- int **getzl** (void)
- int **getxr** (void)
- int **getyr** (void)
- int **getzr** (void)
- int ijkToRank (int i, int j, int k)

    return rank for assigned i,j,k
- void RankToijk (int rank, int *myijk)

    assign value to allocated myijk[3]

3.9.1 Member Function Documentation

3.9.1.1 mallocGhosts()

```
void Domain::mallocGhosts (
            int xgsize,
            int ygsize,
            int zgsize )
```

Allocate ghost buffers for MPI.

xgsize : size of ghost buffer in x direction ygsize : size of ghost buffer in y direction zgsize : size of ghost buffer in z direction

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

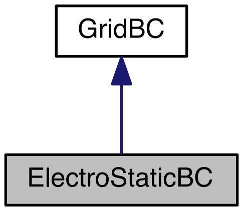
- src/domain/domain.hpp
- src/domain/domain.cpp
- src/domain/ghosts.cpp
- src/domain/pass_fields.cpp

3.10 ElectroStaticBC Class Reference

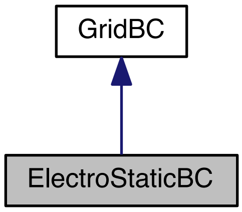
Class for supplying electrostatic boundary conditions in a single field to field grid.

`#include <estaticBC.hpp>`

Inheritance diagram for ElectroStaticBC:



Collaboration diagram for ElectroStaticBC:



Public Member Functions

- **ElectroStaticBC** (int side, Input_Info_t ∗input_info)
- void applyBCs (double t, double dt, Grid ∗grids)

  Inject electrostatic wave boundary condition to grid.

Additional Inherited Members

3.10.1 Detailed Description

Class for supplying electrostatic boundary conditions in a single field to field grid.

Boundary conditions are of linear superpositions of the wave and constant
The wave is of the form: peakamp ∗ cos(omega ∗ t + phase) ∗ exp(-(t-delay)^2∗invWidth^2)
along plane perpendicular to side side = -1: x left, side = +1: x right
side = -2: y left, side = +2: y right
side = -3: z left, side = +3: y right

3.10.2 Member Function Documentation

3.10.2.1 applyBCs()

```
void ElectroStaticBC::applyBCs (
            double t,
            double dt,
            Grid * grids )  [virtual]
```

Inject electrostatic wave boundary condition to grid.

Uses setFieldAlongEdge method in grid to add field to grid.

Implements GridBC.

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

- src/grid/estaticBC.hpp
- src/grid/estaticBC.cpp

3.11 Field_BC_Factory Class Reference

A singleton class to handle registration of field boundaries/.

`#include <field_bc_factory.hpp>`

Public Types

- typedef BC_Field ∗(∗ **Factory**) (int side, Domain ∗domain, Grid ∗grids, Input_Info_t ∗info)

Public Member Functions

- void Construct (Domain ∗domain, Grid ∗grids, Input_Info_t ∗input_info)
- void **declare** (const std::string &type, Factory factory)
- Factory **lookup** (const std::string &type)
- std::vector< const std::string ∗ > **types** () const

Static Public Member Functions

- static Field_BC_Factory & **getInstance** ()

3.11.1 Detailed Description

A singleton class to handle registration of field boundaries/.

3.11.2 Member Function Documentation

3.11.2.1 Construct()

```
void Field_BC_Factory::Construct (
            Domain * domain,
            Grid * grids,
            Input_Info_t * input_info )
```

Construct the boundary condition array (must be freed!) Takes in an array of size 6.

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

- src/boundaries/field_bc_factory.hpp
- src/boundaries/field_bc_factory.cpp

3.12 Field_part Struct Reference

Public Attributes

- double **e1**
- double **e2**
- double **e3**
- double **b1**
- double **b2**
- double **b3**

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

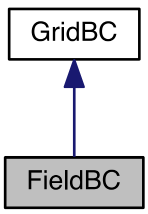
- src/particles/particle.hpp

3.13 FieldBC Class Reference

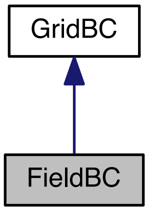
Class for supplying boundary conditions in a single field to field grid.

`#include <fieldBC.hpp>`

Inheritance diagram for FieldBC:



Collaboration diagram for FieldBC:



Public Member Functions

- **FieldBC** (std::string &fieldStr, int dim, bool edge, double amp, double omega, double phase)
- void applyBCs (double t, double dt, Grid &grid)

    Apply boundary condition to grid.

Additional Inherited Members

3.13.1 Detailed Description

Class for supplying boundary conditions in a single field to field grid.

Boundary conditions are of form:
amp $*$ cos(omega $*$ t + phase)
along plane perpendicular to dimension dim (0 = x, 1 = y, 2 = z) on edge (false = left, true = right)
fieldStr one of Ex, Ey, Ez, Bx, By, Bz

3.13.2 Member Function Documentation

3.13.2.1 applyBCs()

```
void FieldBC::applyBCs (
            double t,
            double dt,
            Grid & grid )
```

Apply boundary condition to grid.

Uses setFieldAlongEdge method in grid to add field to grid.

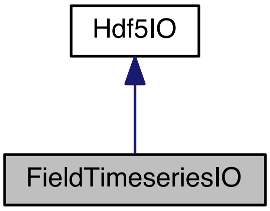
The documentation for this class was generated from the following files:

- src/grid/fieldBC.hpp
- src/grid/fieldBC.cpp

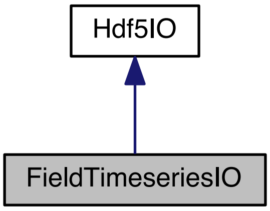
3.14 FieldTimeseriesIO Class Reference

```
#include <hdf5io.hpp>
```

Inheritance diagram for FieldTimeseriesIO:



Collaboration diagram for FieldTimeseriesIO:



Public Member Functions

- **FieldTimeseriesIO** (Hdf5IO ∗io, Grid ∗grid, Domain ∗domain, Input_Info_t ∗input, const int totWrites)
- int **getnFieldDatasets** ()
- int **writeAField** (const int fieldID, double ∗∗∗data, const int iwrite)
- int **writeFields** (Grid ∗grid, Input_Info_t ∗input, const int iwrite)

Additional Inherited Members

3.14.1 Detailed Description

Class for writing fields in time series to hdf5

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

- src/IO/hdf5io.hpp
- src/IO/hdf5io.cpp

3.15 Gaussian_Pulses_t Struct Reference

```
#include <GaussianPulse.hpp>
```

Public Attributes

- int **nwaves**
- double $*$ **peakamps**
- double $*$ **omegas**
- double $*$ **phases**
- double $*$ **delays**
- double $*$ **invWidths**

3.15.1 Detailed Description

This parameter struct generating field boudary values. The values are superpositions of Gaussian pulses of the form: peakamps cos(omegas $*$ t + phases) exp(-(t-delays)^2$*$invWidths^2) The coefficients peakamps, omegas, phases, delays,and invWidths are double arrays of length nwaves.

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

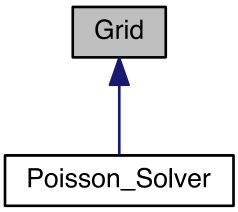
- src/utils/GaussianPulse.hpp

3.16 Grid Class Reference

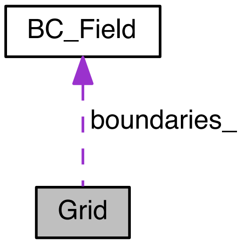
Class representing grid on which E and B fields and currents are defined.

```
#include <grid.hpp>
```

Inheritance diagram for Grid:



Collaboration diagram for Grid:



Public Member Functions

- Grid (int ∗nxyz, int nGhosts, double ∗xyz0, double ∗Lxyz)

  Grid constructor.
- virtual ∼Grid ()

  Grid destructor.
- int evolveFields (double dt)

  Evolve Electric and Magnetic fields in time.
- int evolveFieldsES (double dt)

  Evolve Electric Fields Electrostatically.
- virtual void InitializeFields (Input_Info_t ∗input_info)

  Initialize E and B fields.
- void constJ (double vx, double vy, double vz)

  sets J to a constant value
- void constE (double vx, double vy, double vz)

  sets E to a constant value
- void constB (double vx, double vy, double vz)

  sets B to a constant value
- void constRho (double v)

  sets rho to a constant value
- int addJ (int cellID, double ∗Jvec)

  Add currents from particle to grid.
- int addRho (int cellID, double ∗Rhovec)

  Add charge from particle to grid.
- int getFieldInterpolatorVec (int cellID, double ∗InterpolatorVec)

  Return vector for field interpolation.
- int getCellID (double x, double y, double z)

  Get cell ID based on particle position.
- int getCellVertex (int cellID, double ∗xyz)

  Returns vertex corresponding to cell ID.
- int getNumberOfCells ()

  Get total number of cells in grid.
- int getNumCells3D (double ∗nvec)

  Get # of cells in each dimension of grid.

- double getStepSize (int dimension)

  Get step size along dimension in grid.
- int getnxyzTot (int ∗nxyzTot)

  get total dimensions
- void getDimPhys (const int fieldID, int ∗dim)

  get dimensions of physical region of field
- int setFieldAlongEdge (std::string &fieldStr, int dim, bool edge, double fieldVal)

  Set field along a certain edge.
- virtual int getGhostVecSize (const int sendID)

  returns size of ghost cell data to send
- virtual void getGhostVec (const int side, double ∗ghostVec, int sendID)

  bundles the data in the ghost cells to send
- virtual void setGhostVec (const int side, double ∗ghostVec, int sendID, int op)

  unbundles the data in the ghost cells to send
- void updatePeriodicGhostCells ()

  updates E,B ghost cells in y/z directions with periodic boundary conditions
- void **setBoundaryVec** (const int side, const double ∗ghostVec)
- double ∗∗∗∗ **getFieldPtr** ()
- int **getExID** ()
- int **getEyID** ()
- int **getEzID** ()
- int **getBxID** ()
- int **getByID** ()
- int **getBzID** ()
- int **getBx_tm1ID** ()
- int **getBy_tm1ID** ()
- int **getBz_tm1ID** ()
- int **getJxID** ()
- int **getJyID** ()
- int **getJzID** ()
- int **getrhoID** ()
- void executeBC (void)

  Execute field boundary conditions.
- void **setBoundaries** (BC_Field ∗∗bc)
- void **freeBoundaries** (void)

Public Attributes

- double ∗ **sliceTmp_**
- double ∗ **ghostTmp_**

Protected Member Functions

- double ∗∗∗ newField_ (int ifield)

  allocates memory for a single field
- void deleteField_ (double ∗∗∗fieldPt, int ifield)

  frees memory for a single field
- int ∗∗ setFieldSize_ ()

  constructs and returns fieldSize_ array
- void deleteFieldSize_ ()

deletes fieldSize_ array
- int ∗ setFieldType_ ()

  constructs and returns fieldType_ array
- void deleteFieldType_ ()

  deletes fieldType_ array
- double ∗∗∗∗ setFieldPtr_ ()

  constructs and returns fieldPtr__ array
- void deleteFieldPtr_ ()

  deletes fieldPtr_ array
- void constField_ (const int fieldID, const double val)

  sets field corresponding to fieldID to specified value
- int sideToIndex_ (const int side, const int fieldID)

  function to convert -/+ 1 left/right side indicator to index in x direction (description out of date)
- void checkInput_ ()

  checks validity of input parameters for Grid constructor
- void sliceMatToVec_ (const int fieldID, const int side, const int offset, double ∗vec)

  slices a physical plane in the specified direction (excludes ghosts)
- void unsliceMatToVec_ (const int fieldID, const int side, const int offset, double ∗vec, const int op)

  unslices a physical plane in the specified direction (excludes ghosts)
- int setFieldInPlane_ (int dim, int indx, double ∗∗∗field, double fieldVal)

  Internal method to set field along a plane.
- **FRIEND_TEST** (oGridInternalTest, EMWave)
- **FRIEND_TEST** (oGridInternalTest, EMWaveLong)
- **FRIEND_TEST** (GridPrivateTest, fieldSizeTest)
- **FRIEND_TEST** (GridPrivateTest, fieldPtrTest)
- **FRIEND_TEST** (GridPrivateTest, zeroFields)
- **FRIEND_TEST** (GridPrivateTest, sideToIndexTest)
- **FRIEND_TEST** (GridPrivateTest, periodicUpdateTest)
- **FRIEND_TEST** (GridPrivateTest, ghostVecSizeTest)

Protected Attributes

- BC_Field ∗∗ **boundaries_**
- const int **nx_**
- const int **ny_**
- const int **nz_**
- const int **nGhosts_**
- const int **nxTot_**
- const int **nyTot_**
- const int **nzTot_**
- const double **x0_**
- const double **y0_**
- const double **z0_**
- const double **Lx_**
- const double **Ly_**
- const double **Lz_**
- const int **iBeg_**
- const int **jBeg_**
- const int **kBeg_**
- const double **dx_**
- const double **dy_**
- const double **dz_**

- const double **idx_**
- const double **idy_**
- const double **idz_**
- const int **maxPointsInPlane_**
- const int **nFieldsTotal_**
- const int **ExID_**
- const int **EyID_**
- const int **EzID_**
- const int **BxID_**
- const int **ByID_**
- const int **BzID_**
- const int **JxID_**
- const int **JyID_**
- const int **JzID_**
- const int **Bx_tm1ID_**
- const int **By_tm1ID_**
- const int **Bz_tm1ID_**
- const int **rhoID_**
- const int **nTypes_**
- const int **edgeXID_**
- const int **edgeYID_**
- const int **edgeZID_**
- const int **faceXID_**
- const int **faceYID_**
- const int **faceZID_**
- const int **vertID_**
- double *** **Ex_**
- double *** **Ey_**
- double *** **Ez_**
- double *** **Bx_**
- double *** **By_**
- double *** **Bz_**
- double *** **Bx_tm1_**
- double *** **By_tm1_**
- double *** **Bz_tm1_**
- double *** **Jx_**
- double *** **Jy_**
- double *** **Jz_**
- double *** **rho_**
- int * **fieldType_**
- int ** **fieldSize_**
- double **** **fieldPtr_**
- int * **fieldIsContiguous_**

Friends

- class **oGridInternalTest**
- class **GridPrivateTest**

3.16.1 Detailed Description

Class representing grid on which E and B fields and currents are defined.

Grid has ghost cells on each face. The ghost cell updating in y and z arises from periodic boundary conditions. x-direction ghost cells allow communication between MPI domains.

Following Yee (1966), electric fields and currents reside on edges, and magnetic fields on faces. Fields are updated using a set of finite-difference equations approximating Ampere's and Faraday's Laws.

A set of getters are available to allow particles to interpolate electric fields based on their position.

3.16.2 Constructor & Destructor Documentation

3.16.2.1 Grid()

```
Grid::Grid (
            int * nxyz,
            int nGhosts,
            double * xyz0,
            double * Lxyz )
```

Grid constructor.

Input arguments:
nxyz: integer array [nx,ny,nz] where nx is the total number of cells (physical + ghost) in the x direction in the simulation, and the same for ny,nz.
nGhosts: integer number of ghost cells on each side of the domain. This should always be at least 1. Currently the code does not support nGhosts>1, though it may in the future (to take advantage of higher order finite difference and interpolation methods, for instance).
xyz0: integer array [x0,y0,z0] where x0 is the initial x position, and the same for y0,z0
Lxyz0: double array [Lx,Ly,Lz] where Lx is the physical length of each cell in the x direction, and the same for Ly,Lz

3.16.2.2 ∼Grid()

```
Grid::∼Grid ( ) [virtual]
```

Grid destructor.

calls deleteField_ on each of the double∗∗∗ fields

3.16.3 Member Function Documentation

3.16.3.1 addJ()

```
int Grid::addJ (
            int cellID,
            double * Jvec )
```

Add currents from particle to grid.

Currents added to cell with ID cellID via input vector of form:
[Jx((0,0,0) -> (1,0,0)), Jx((0,1,0) -> (1,1,0)), Jx((0,1,1) -> (1,1,1)), Jx((0,0,1) -> (1,0,1)),...
Jy((0,0,0) -> (0,1,0)), Jy((0,0,1) -> (0,1,1)), Jy((1,0,1) -> (1,1,1)), Jy((1,0,0) -> (1,1,0)),...
Jz((0,0,0) -> (0,0,1)), Jz((1,0,0) -> (1,0,1)), Jz((1,1,0) -> (1,1,1)), Jz((0,1,0) -> (0,1,1))]

3.16.3.2 checkInput_()

```
void Grid::checkInput_ ( )  [protected]
```

checks validity of input parameters for Grid constructor

asserts necessary conditions on each input (mainly positivity of many parameters). Terminates program if inputs are incorrect.

3.16.3.3 deleteField_()

```
void Grid::deleteField_ (
             double *** fieldPt,
             int fieldID )  [protected]
```

frees memory for a single field

Uses fieldIsContiguous_ to determine contiguous or noncontiguous deltion method

3.16.3.4 evolveFields()

```
int Grid::evolveFields (
             double dt )
```

Evolve Electric and Magnetic fields in time.

Uses Yee algorithm to advance E and B fields. Assumes Gaussian-style Maxwell equation, with c = 1.

3.16.3.5 evolveFieldsES()

```
int Grid::evolveFieldsES (
             double dt )
```

Evolve Electric Fields Electrostatically.

Ignores "light wave" contribution (curl terms), effectively only solves poisson equation.

3.16.3.6 getCellID()

```
int Grid::getCellID (
             double x,
             double y,
             double z )
```

Get cell ID based on particle position.

Cell ID is uniquely given by (ny_∗nz_)∗ix + nz_∗iy + iz.
If particle is in a ghost cell or off the grid entirely, returns
-1 if off (-z), -2 if off (+z)
-3 if off (-y), -4 if off (+y)
-5 if off (-x), -6 if off (+x)

3.16.3.7 getFieldInterpolatorVec()

```
int Grid::getFieldInterpolatorVec (
            int cellID,
            double * InterpolatorVec )
```

Return vector for field interpolation.

Based on cellID, return relevant edge E and face B fields and cell origin, in format:
[x, y, z, ...
Ex(ix, iy, iz), Ex(ix, iy+1,iz), Ex(ix, iy+1, iz+1), Ex(ix, iy, iz+1), ...
Ey(ix, iy, iz), Ey(ix, iy, iz+1), Ey(ix+1, iy, iz+1), Ey(ix+1, iy, iz), ...
Ez(ix, iy, iz), Ez(ix+1, iy, iz), Ez(ix+1, iy+1, iz), Ez(ix, iy+1, iz), ...
Bx(ix, iy, iz), Bx(ix+1, iy, iz), ...
By(ix, iy, iz), By(ix, iy+1, iz), ...
Bz(ix, iy, iz), Bz(ix, iy, iz+1), ...]
where ix, iy, and iz are the row indices for each of the three dimensions (calculated from the cellID)

3.16.3.8 getGhostVec()

```
void Grid::getGhostVec (
            const int side,
            double * ghostVec,
            int sendID ) [virtual]
```

bundles the data in the ghost cells to send

side = -/+ 1 for left/right x direction, -/+ 2 for y, -/+ 3 for z
ghostVec is the vector to store the data in, which must be of length ghostVecSize_ (can be determined with get↩GhostVecSize)
sendID = -2 to get Jrho fields, -1 to get EB fields, or sendID = an individual field ID (e.g. ExID_) to get just that field (used for Poisson updating for example)
Gets the data of the E,B,J fields along the specified boundary plane from the 1D array ghostVec to be sent with a single MPI call. If sendID = -1 (as used in each time step update), stores in order: Ex,Ey,Ez,Bx,By,Bz. If sendID = -2, stores in order: Jx,Jy,Jz,rho.
ghostVec can (and should) be unpacked with setGhostVec function

Reimplemented in Poisson_Solver.

3.16.3.9 getGhostVecSize()

```
int Grid::getGhostVecSize (
            const int sendID ) [virtual]
```

returns size of ghost cell data to send

sendID is an integer specifying which fields are intended to be packaged into the ghost vector.
-2: for J/rho package, -1 for E/B package, fieldID for any individual field (e.g. ExID_)
It is of length equal to the number of fields being sent times the maximum number of total points in any plane, so that it will be large enough to send the maximum amount of data in a single plane of any of the fields.

Reimplemented in Poisson_Solver.

3.16.3.10 getNumberOfCells()

```
int Grid::getNumberOfCells ( )
```

Get total number of cells in grid.

Includes ghost cells.

3.16.3.11 getNumCells3D()

```
int Grid::getNumCells3D (
            double * nvec )
```

Get # of cells in each dimension of grid.

Includes ghost cells.

3.16.3.12 getStepSize()

```
double Grid::getStepSize (
            int dimension )
```

Get step size along dimension in grid.

Returns step size along dimension according to; dimension = 0: x dimension = 1: y dimension = 2: z Returns -1 if invalid dimension.

3.16.3.13 InitializeFields()

```
void Grid::InitializeFields (
            Input_Info_t * input_info )  [virtual]
```

Initialize E and B fields.

Use restart file to set values of initial E,B,J fields

3.16.3.14 newField_()

```
double *** Grid::newField_ (
            int fieldID )  [protected]
```

allocates memory for a single field

Returns double*** of size [nx_+1][ny_+1][nz_+1].
First attempts to allocate contiguously. If that fails, issues a warning and attempts to allocate with several calls to new.

3.16.3.15 setFieldAlongEdge()

```
int Grid::setFieldAlongEdge (
            std::string & fieldStr,
            int dim,
            bool edge,
            double fieldVal )
```

Set field along a certain edge.

Inputs:
fieldStr: string of format "Ex", "Bz", etc
dim: dimension along which to apply boundary condition
edge: side along which to apply boundary condition

3.16.3.16 setFieldInPlane_()

```
int Grid::setFieldInPlane_ (
            int dim,
            int indx,
            double *** field,
            double fieldVal )  [protected]
```

Internal method to set field along a plane.

Inputs:
dimension perpendicular to plane.
For example, if dim=0 (x direction), then this program set field in one yz plane.

indx along dimenstion perpendicular to plane.
For example, if dim=0 and indx =14, then set field for the 14th yz plane.

field to set along dimension
value to set field

3.16.3.17 setFieldPtr_()

```
double **** Grid::setFieldPtr_ ( )  [protected]
```

constructs and returns fieldPtr__ array

fieldPtr_ is an nFieldsTotal_ array storing each field, so that they can be accessed via fieldID
e.g. int fieldID = ExID_;
double*** field = fieldPtr_[fieldID];

3.16.3.18 setFieldSize_()

```
int ** Grid::setFieldSize_ ( )  [protected]
```

constructs and returns fieldSize_ array

fieldSize_ is an ntypes by ndim array storing the number of physical + ghost points in each direction. This is necessary because although all field arrays are allocated to be the same size (nx+1,ny+1,nz+1), due to the different locations of each type of field on the grid (3 types of edge locations, 3 types of face locations, vertices) which leads to differences in the number of points needed for nx,ny,nz cells.
rows correspond to fieldType: 0: x edge (Ex/Jx), 1: y edge (Ey/Jy), 2: z edge (Ez/Jz),
3: x face (Bx) , 4: y face (By) , 5: z face (Bz),
6: vertices (rho)
columns correspond to the direction (0,1,2)=(x,y,z)

3.16.3.19 setFieldType_()

```
int * Grid::setFieldType_ ( ) [protected]
```

constructs and returns fieldType_ array

fieldType_ is an nFieldsTotal_ array of ints storing the type of each field (edgeX, faceZ, vertex, etc).
e.g. int typeOfBx = fieldType_[BxID_];

3.16.3.20 setGhostVec()

```
void Grid::setGhostVec (
            const int side,
            double * ghostVec,
            int sendID,
            int op ) [virtual]
```

unbundles the data in the ghost cells to send

side = -/+ 1 for left/right x direction, -/+ 2 for y, -/+ 3 for z
ghostVec is the vector to read the data from, which must be of length ghostVecSize_ (can be determined with get↩ GhostVecSize)
sendID = -2 to set Jrho fields, -1 to set EB fields, or sendID = an individual field ID (e.g. ExID_) to set just that field (used for Poisson updating for example)
Sets the data of the E,B,J fields along the specified boundary plane from the 1D array ghostVec to be received with a single MPI call. If sendID = -1 (as used in each time step update), fields are read and set in order↩ : Ex,Ey,Ez,Bx,By,Bz. If sendID = -2, fields are read and set in order: Jx,Jy,Jz,rho.
op is a flag determining how the field will be set. op = 0 replaces the current current values in the field's ghost points with the values in ghostVec. op = 1 sums the current values and the values in ghostVec.
ghostVec can (and should) be generated with getGhostVec function

Reimplemented in Poisson_Solver.

3.16.3.21 sideToIndex_()

```
int Grid::sideToIndex_ (
            const int side,
            const int fieldID ) [protected]
```

function to convert -/+ 1 left/right side indicator to index in x direction (description out of date)

For use with ghost cell methods. side=-1 indicates operations on the left side of the domain, side=+1 indicates operations on the right side of the domain. This method converts side into the correct index i to reference ghost cells on that side of the domain. For instance, called by getGhostVec and setGhostVec. Generalizes to any number of ghost cells so long as iBeg_ and iEnd_ are initialized correctly. function to convert (-/+)(1,2,3) side indicator into (left/right)(x,y,z) index of boundary physical data point

Helper function for public ghost cell methods which accept side indicator as argument.
Side $<$ 0 will return index of first physical point, side $>$ 0 will return index of last physical point
abs(side) == 1 returns value in x direction, 2 in y, 3 in z
This function is necessary because different field types have a different number of physical grid points in each direction.
fieldID is a private fieldID such as ExID_

3.16.3.22 sliceMatToVec_()

```
void Grid::sliceMatToVec_ (
            const int fieldID,
            const int side,
            const int offset,
            double * vec )  [protected]
```

slices a physical plane in the specified direction (excludes ghosts)

mat is 3D array whose real (non-ghost) data on one side will be stored in vec as a 1D array. vec must be of size maxPointsInPlane_. side is an integer -/+ 1 to indicate the location on the left/right side in the x direction, -/+ 2 in y, -/+ 3 in z. offset is an integer offset from the first/last physical index determined by side (e.g. side=-1 and offset=0 gives the yz plane of the 1st physical grid points in x direction, whereas offset=-1 would have returned the adjacent ghost cells and offset = 3 would have returned the 4th physical yz plane from the left). unsliceMatToVec_ is the inverse function.

3.16.3.23 unsliceMatToVec_()

```
void Grid::unsliceMatToVec_ (
            const int fieldID,
            const int side,
            const int offset,
            double * vec,
            const int op )  [protected]
```

unslices a physical plane in the specified direction (excludes ghosts)

mat is 3D array whose real (non-ghost) data on one side will be replaced by data in the 1D array vec. vec must be of size maxPointsInPlane_. side is an integer -/+ 1 to indicate the location on the left/right side in the x direction, -/+ 2 in y, -/+ 3 in z. offset is an integer offset from the first/last physical index determined by side (e.g. side=-1 and offset=0 gives the yz plane of the 1st physical grid points in x direction, whereas offset=-1 would have returned the adjacent ghost cells and offset = 3 would have returned the 4th physical yz plane from the left). op=0 replaces the values in mat with those in vec, op=1 adds the values in vec to thos in mat. sliceMatToVec_ is the inverse function.

3.16.3.24 updatePeriodicGhostCells()

```
void Grid::updatePeriodicGhostCells ( )
```

updates E,B ghost cells in y/z directions with periodic boundary conditions

Makes 4 calls each to get/setGhostVec for EB fields all at once

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

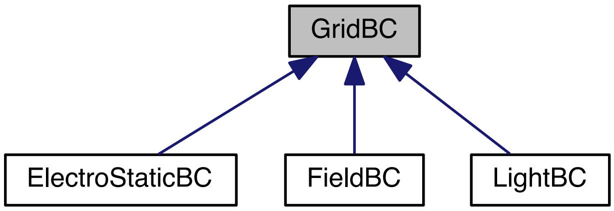
- src/grid/grid.hpp
- src/grid/grid.cpp
- src/grid/gridOutput.cpp
- src/grid/oGrid.cpp
- src/grid/spookyGrid.cpp

3.17 GridBC Class Reference

Abstract class for supplying boundary conditions to field grid.

```
#include <gridBC.hpp>
```

Inheritance diagram for GridBC:



Public Member Functions

- virtual void **applyBCs** (double t, double dt, Grid ∗grids)=0

Protected Attributes

- int **side_**
- int **dim_**

3.17.1 Detailed Description

Abstract class for supplying boundary conditions to field grid.

side = -1: x left, side = +1: x right
side = -2: y left, side = +2: y right
side = -3: z left, side = +3: y right
dim_ = abs(side_)-1 dim_ = 0: x boundaries dim_ = 1: y boundaries dim_ = 2: z boundaries

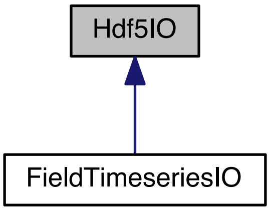
The documentation for this class was generated from the following file:

- src/grid/gridBC.hpp

3.18 Hdf5IO Class Reference

```
#include <hdf5io.hpp>
```

Inheritance diagram for Hdf5IO:



Public Member Functions

- **Hdf5IO** (const char *filename)
- hid_t **getFileID** ()
- hid_t **getFileAccessPlist** ()
- hid_t **getDataXferPlist** ()

Protected Attributes

- hid_t **file_id_**
- hid_t **file_access_plist_**
- hid_t **data_xfer_plist_**

3.18.1 Detailed Description

Class that creates an hdf5 file and sets basic props

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

- src/IO/hdf5io.hpp
- src/IO/hdf5io.cpp

3.19 Input Class Reference

```
#include <input.hpp>
```

Public Member Functions

- int **readinfo** (char ∗inputname)
- int checkinfo (void)

    Check input self-consistency and sufficiency.
- Input_Info_t ∗ **getinfo** (void)

3.19.1 Detailed Description

Class handeling input information

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

- src/IO/input.hpp
- src/IO/check.cpp
- src/IO/input.cpp
- src/IO/readinfo.cpp

3.20 Input_Info_t Struct Reference

Structure storing info in the input file.

```
#include <input.hpp>
```

Public Attributes

- int **nCell** [NDIM]
- int nProc [NDIM]
- int nt
- int restart
- int debug
- int relativity
- int electrostatic
- int output_pCount
- int **nspecies**
- int nwaves
- int inSide [NWAVE]
- int inPolE [NWAVE]
- int nwrite
- int **write_field_timeseries**
- int **write_all_fields**
- int **write_E**
- int **write_B**
- int **write_J**
- int **write_rho**
- long **np**
- long output_dStep

    number of particles in each domain
- double t0

number of steps between outputs
- double output_dT

  start time of simulation
- double mass_ratio [NSPEC]

  physical time between outputs
- double charge_ratio [NSPEC]
- double dens_frac [NSPEC]
- double temp [NSPEC]
- double peakamps [NWAVE]
- double **omegas** [NWAVE]
- double **phases** [NWAVE]
- double **invWidths** [NWAVE]
- double **delays** [NWAVE]
- double **E0** [NDIM]
- double B0 [NDIM]

  background electric field
- double xyz0 [NDIM]

  background magnetic field
- double **Lxyz** [NDIM]
- char **distname** [NCHAR]
- char parts_init [NCHAR]

  name of file containing distribution function
- char fields_init [NCHAR]

  particle initialization method
- char parts_bound [2 ∗NDIM][NCHAR]

  field initialization method
- char fields_bound [2 ∗NDIM][NCHAR]

  particle boundary conditions for 6 sides of box

3.20.1 Detailed Description

Structure storing info in the input file.

3.20.2 Member Data Documentation

3.20.2.1 charge_ratio

```
double Input_Info_t::charge_ratio[NSPEC]
```

mass of each type of particle in unit of electron mass array of length nspecies eg. in electron-proton plasma mass↩
_ratio[0]=1; mass_ratio[1]=1830;

3.20.2.2 debug

```
int Input_Info_t::debug
```

How many previous runs? restart = 0: initial run

3.20.2.3 dens_frac

```
double Input_Info_t::dens_frac[NSPEC]
```

charge of each type of particle in unit of $|e|$ array of length nspecies eq. in electron-proton plasma chargeratio[0]=-1; chargeratio[1]=1

3.20.2.4 electrostatic

```
int Input_Info_t::electrostatic
```

1: use relativistic pusher 0: use nonrelativistic pusher

3.20.2.5 inPolE

```
int Input_Info_t::inPolE[NWAVE]
```

from which sides are waves injected eg. inSide[0]=-1: 1st wave injected in x direction(1) from left(-) inside[1]=+3: 2nd wave injected in z direction(3) from right(+)

3.20.2.6 inSide

```
int Input_Info_t::inSide[NWAVE]
```

How many waves to inject into the system nwave$<$=NWAVE

3.20.2.7 nProc

```
int Input_Info_t::nProc[NDIM]
```

number of cells in each direction

3.20.2.8 nt

```
int Input_Info_t::nt
```

number of processors to use in each direction

3.20.2.9 nwaves

```
int Input_Info_t::nwaves
```

How many species of particles eg. nspecies=2 in electron-proton plasma nspecies $<$=NSPEC

3.20.2.10 nwrite

`int Input_Info_t::nwrite`

polarization of E field of injected waves eg. inPolE[0]=2: 1st wave E field is in y direction(2) inPolE[1]=3: 2nd wave E field is in Z direction(3) inPolE should only take value of 1,2,3

3.20.2.11 output_pCount

`int Input_Info_t::output_pCount`

1: use electrostatic field solve 0: use electromagnetic field solve

3.20.2.12 peakamps

`double Input_Info_t::peakamps[NWAVE]`

Maxwellian temperature in unit of eV if specified array of length nspecies eq. in cold ion and hot electron plasma, possible value temp[0]=100; temp[1]=1.2;

3.20.2.13 relativity

`int Input_Info_t::relativity`

0: do not print debug statements 1: print minimal debug statements 2: print more debug statements 3: write debug files

3.20.2.14 restart

`int Input_Info_t::restart`

number of time steps

3.20.2.15 temp

`double Input_Info_t::temp[NSPEC]`

fractional density, array of length nspecies eg. in quasineutral electron-proton plasma frac_dens[0]=0.5; frac_↩ dens[1]=0.5;

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

- src/IO/input.hpp

3.21 Interpolator Class Reference

Public Member Functions

- void **interpolate_fields** (double ∗pos, double ∗lcell, double ∗cellvars, Field_part ∗field)

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

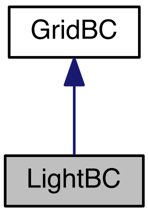
- src/particles/interpolate.hpp
- src/particles/interpolate.cpp

3.22 LightBC Class Reference

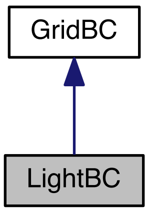
Class for supplying light-wave boundary conditions to field grid.

`#include <lightBC.hpp>`

Inheritance diagram for LightBC:



Collaboration diagram for LightBC:



Public Member Functions

- **LightBC** (int side, Input_Info_t ∗input_info)
- void applyBCs (double t, double dt, Grid ∗grids)

  Apply transverse light wave boundary condition to grid.

Additional Inherited Members

3.22.1 Detailed Description

Class for supplying light-wave boundary conditions to field grid.

Boundary conditions are of linear superpositions of the wave and constant
The wave is of the form: peakamp ∗ cos(omega ∗ t + phase) ∗ exp(-(t-delay)^2∗invWidth^2)
along plane perpendicular to side side = -1: x left, side = +1: x right
side = -2: y left, side = +2: y right
side = -3: z left, side = +3: y right

3.22.2 Member Function Documentation

3.22.2.1 applyBCs()

```
void LightBC::applyBCs (
            double t,
            double dt,
            Grid ∗ grids )  [virtual]
```

Apply transverse light wave boundary condition to grid.

Uses setFieldAlongEdge method in grid to add field to grid.

Implements GridBC.

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

- src/grid/lightBC.hpp
- src/grid/lightBC.cpp

3.23 Part_BC_Factory Class Reference

```
#include <particle_bc_factory.hpp>
```

Public Types

- typedef BC_Particle ∗(∗ **Factory**) (Domain ∗domain, int dim_Index, short isRight, std::string type)

Public Member Functions

- BC_Particle ** **constructConditions** (Domain *domain, const char(*bound)[NCHAR])
- void **declare** (const std::string &type, Factory factory)
- Factory **lookup** (const std::string &type)
- std::vector< const std::string * > **types** () const

Static Public Member Functions

- static Part_BC_Factory & **getInstance** ()

3.23.1 Detailed Description

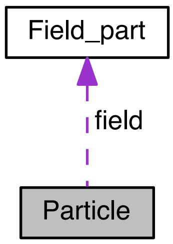
A singleton class to handle registration of particle boundaries

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

- src/boundaries/particle_bc_factory.hpp
- src/boundaries/particle_bc_factory.cpp

3.24 Particle Struct Reference

Collaboration diagram for Particle:



Public Attributes

- double **x** [3]
- double **v** [3]
- double **gamma**
- double **xo** [3]
- double **vo** [3]
- double **dx** [3]
- double **q**
- double **m**
- long **my_id**
- int **initRank**
- short **isGhost**
- Field_part **field**

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

- src/particles/particle.hpp

3.25 Particle_Compare Class Reference

```
#include <particle_utils.hpp>
```

Public Member Functions

- **Particle_Compare** (Grid ∗grid)
- bool **operator()** (Particle const a, Particle const b) const

3.25.1 Detailed Description

Function to use are a comparator in std::sort for std::vec<Particle> Idea: Particle is sorted by outer index first (i.e. particle at [2][12][43] should be closer to beginning of array than particle at [24][12][43])

At the moment, implemented very slowly!!! Should be modified two ways:

```
1) Instead of comparing cell ID, compare i,j,k indice locations individually
   to save time
2) Bring the code to calculating i,j,k into the comparison routine
```

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

- src/particles/particle_utils.hpp

3.26 Particle_Handler Class Reference

Class that handles all particle-relevant operations.

```
#include <particle_handler.hpp>
```

Public Member Functions

- void Load (Input_Info_t ∗input_info, Domain ∗domain)

    Load and initialize the particle handler. Should be called at the beginning of the run.
- void **Push** (double dt)
- long **nParticles** ()
- void **incrementNParticles** (int inc)
- void SortParticles (Particle_Compare comp)

    Sort particles based on grid location.
- void **setPusher** (Pusher ∗pusher)
- void clearGhosts ()

    Clear all ghost particles. Uses a swap-to-back and pop-last-element for speed.
- void **InterpolateEB** (Grid ∗grid)
- void **depositRhoJ** (Grid ∗grid, bool depositRho, Domain ∗domain, Input_Info_t ∗input_info)
- std::vector< Particle > **getParticleVector** ()
- double **computeCFLTimestep** (Domain ∗domain)
- void **setParticleBoundaries** (BC_Particle ∗∗bc)
- void **executeParticleBoundaryConditions** ()
- void outputParticles (long nstep, Input_Info_t ∗input_info)

    Output particles.
- void **outputParticleVel** ()

3.26.1 Detailed Description

Class that handles all particle-relevant operations.

Particle handler handles all the particle operations. This includes deposition, boundary conditions, particle pushing, and communication between MPI nodes if needed

3.26.2 Member Function Documentation

3.26.2.1 outputParticles()

```
void Particle_Handler::outputParticles (
            long step,
            Input_Info_t * input_info )
```

Output particles.

Output particles. Currently outputs time, position and velocity.

Can work with either cadencing on time (output every dT) or cadencing on steps (output ever dsteps), or both. Either are optional parameters in the input file and will default to -1.

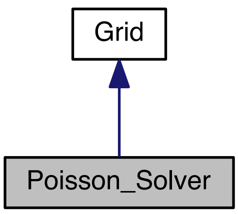
Particles are written to the tracks/ directory and named with initial rank and id.

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

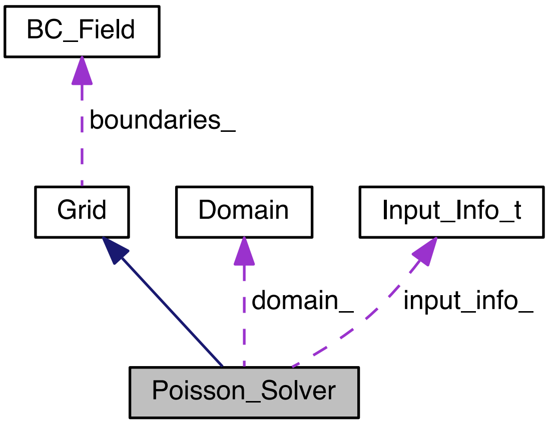
- src/particles/particle_handler.hpp
- src/particles/particle_handler.cpp

3.27 Poisson_Solver Class Reference

Inheritance diagram for Poisson_Solver:



Collaboration diagram for Poisson_Solver:



Public Member Functions

- **Poisson_Solver** (Domain ∗domain, Input_Info_t ∗input_info)
- void **InitializeFields** ()
- int getGhostVecSize (const int sendID)

  returns size of ghost cell data to send
- void getGhostVec (const int side, double ∗ghostVec, int sendID)

  bundles the data in the ghost cells to send
- void setGhostVec (const int side, double ∗ghostVec, int sendID, int op)

  unbundles the data in the ghost cells to send
- void phiToE ()

  derives E from scalar potential phi: E = -grad phi
- void AToB ()

  derives A from vector potential A: B = curl A
- void constA (const double vx, const double vy, const double vz)

  Set vector potential to constant values.
- void constPhi (const double v)

  Set scalar potential to constant value.

Protected Member Functions

- void **run_poisson_solver_** (const int fieldID, double ∗∗∗u0, double ∗∗∗u1, double ∗∗∗R, double convergenceTol, double sourceMult)
- void setPoissonFieldType_ ()

  Same as Grid::setFieldType_ for phi,A arrays unique to Poisson.
- void setPoissonFieldPtr_ ()

  Same as Grid::setFieldPtr_ for phi,A arrays unique to Poisson.
- void phiToESingleComp_ (const int fieldID, const int dir)

  Calculates a single component of E from phi.
- void AToBSingleComp_ (const int fieldID, const int dir)

  calculates a single component of B from A
- **FRIEND_TEST** (ConvertPrivateTest, constantPhiTest)

Protected Attributes

- Domain ∗ **domain_**
- Input_Info_t ∗ **input_info_**
- double ∗∗∗ **phi1_**
- double ∗∗∗ **phi2_**
- double ∗∗∗ **Ax1_**
- double ∗∗∗ **Ay1_**
- double ∗∗∗ **Az1_**
- double ∗∗∗ **Ax2_**
- double ∗∗∗ **Ay2_**
- double ∗∗∗ **Az2_**
- const int **nFieldsPoisson_**
- const int **phi1ID_**
- const int **phi2ID_**
- const int **Ax1ID_**
- const int **Ay1ID_**
- const int **Az1ID_**
- const int **Ax2ID_**
- const int **Ay2ID_**
- const int **Az2ID_**
- const int **xdir_**
- const int **ydir_**
- const int **zdir_**

Friends

- class **ConvertPrivateTest**

Additional Inherited Members

3.27.1 Member Function Documentation

3.27.1.1 AToB()

```
void Poisson_Solver::AToB ( )
```

derives A from vector potential A: B = curl A

Makes three separate calls to AToBSingleComp to perform calculation

3.27.1.2 AToBSingleComp_()

```
void Poisson_Solver::AToBSingleComp_ (
            const int fieldID,
            const int dir )  [protected]
```

calculates a single component of B from A

fieldID is the field ID of the component to be solved for (BxID_, ByID_, or BzID_)
dir is the direction corresonding to the component being solved for

3.27.1.3 getGhostVec()

```
void Poisson_Solver::getGhostVec (
            const int side,
            double * ghostVec,
            int sendID )  [virtual]
```

bundles the data in the ghost cells to send

side = -/+ 1 for left/right x direction, -/+ 2 for y, -/+ 3 for z
ghostVec is the vector to store the data in, which must be of length ghostVecSize_ (can be determined with get↩ GhostVecSize)
sendID = -1 to get EB fields, -2 for rho/J sources, -3 for phi/A potentials, or sendID = an individual field ID (e.g. ExID_) to get just that field (used for Poisson updating for example)
Stores the data of the E,B,J fields along the specified boundary plane into a 1D array to be sent with a single MPI call.
If sendID = -1 (as used in each time step update), stores in order: Ex,Ey,Ez,Bx,By,Bz,Jx,Jy,Jz.
If sendID = -2 (as used in Poisson iteration), stores in order: phi1,phi2,Ax1,Ay1,Az1,Ax2,Ay2,Az2
If sendID = -3 (as used in Poisson initialization), stores in order: Jx,Jy,Jz,rho
ghostVec can (and should) be unpacked with setGhostVec function

Reimplemented from Grid.

3.27.1.4 getGhostVecSize()

```
int Poisson_Solver::getGhostVecSize (
            const int sendID )  [virtual]
```

returns size of ghost cell data to send

sendID is an integer specifying which fields are intended to be packaged into the ghost vector. -3 for potentials (phi,A), -2 for sources (rho,J), -1 for fields (EB), fieldID for any individual field (e.g. ExID_)
It is of length equal to the number of fields being sent times the maximum number of total points in any plane, so that it will be large enough to send the maximum amount of data in a single plane of any of the fields.

Reimplemented from Grid.

3.27.1.5 phiToE()

```
void Poisson_Solver::phiToE ( )
```

derives E from scalar potential phi: E = -grad phi

Makes three calls to phiToESingleComp which performs actual computation

3.27.1.6 phiToESingleComp_()

```
void Poisson_Solver::phiToESingleComp_ (
            const int fieldID,
            const int dir )  [protected]
```

Calculates a single component of E from phi.

fieldID is a field's fieldID (ExID_, EyID_, or EzID_)
dir is (0,1,2) for (x,y,z) which must match the component of the fieldID being solved for

3.27.1.7 setGhostVec()

```
void Poisson_Solver::setGhostVec (
            const int side,
            double * ghostVec,
            int sendID,
            int op )  [virtual]
```

unbundles the data in the ghost cells to send

side = -/+ 1 for left/right x direction, -/+ 2 for y, -/+ 3 for z
ghostVec is the vector to read the data from, which must be of length ghostVecSize_ (can be determined with get↩
GhostVecSize)
sendID = -1 to set JEB fields, or sendID = an individual field ID (e.g. ExID_) to set just that field (used for Poisson updating for example)
Sets the data of the E,B,J fields along the specified boundary plane from the 1D array ghostVec to be received with a single MPI call.
If sendID = -1 (as used in each time step update), fields are read and set in order: Ex,Ey,Ez,Bx,By,Bz,Jx,Jy,Jz.
If sendID = -2 (as used in Poisson iteration), fields are read and set in order: phi1,phi2,Ax1,Ay1,Az1,Ax2,Ay2,Az2
If sendID = -3 (as used in Poisson initialization), stores in order: Jx,Jy,Jz,rho
ghostVec can (and should) be generated with getGhostVec function

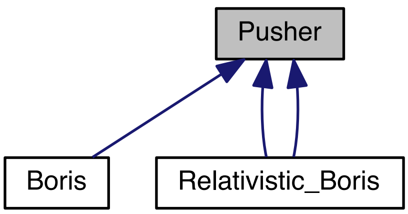
Reimplemented from Grid.

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

- src/poisson/poisson.hpp
- src/poisson/convertFields.cpp
- src/poisson/poisson.cpp

3.28 Pusher Class Reference

Inheritance diagram for Pusher:



Public Member Functions

- virtual int **Step** (Particle ∗part, Field_part ∗field, double dt)=0

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

- src/pusher/pusher.hpp

3.29 Random_Number_Generator Class Reference

Class that provides methods to generate random numbers.

```
#include <RNG.hpp>
```

Public Member Functions

- **Random_Number_Generator** (long seed)
- double getUniform ()

    Get a random number in the range (0,1), exclusive.
- long getInteger (long min, long max)

    Draw a random number, inclusive of both min and max.
- double getStandardNormal ()

    Draw a number from a standard normal distribution.
- double getGaussian (double mu, double sigma)

    Draw a number from a normal distribution.
- RNG_State ∗ **getRNGState** ()
- void **setRNGState** (RNG_State ∗state)
- void **setUserPDF** (bool isDiscrete, long size, double ∗userVal, double ∗userProb)
- void loadUserPDFfromFile (const bool isDiscrete, const char ∗fname)

    Load a user distribution from file.
- double getUserNumber ()

    Get a random number from the user distribution.

3.29.1 Detailed Description

Class that provides methods to generate random numbers.

The Random Number generator class uses the ran2 algorithm from Numerical recipes. The algorithm provides fast random numbers over (0,1) exclusive with a period of over 10^15.

This is then used in the implementation for numerous other distrituions (standard normal, integer...)

This class can also be loaded with a user defined PDF, either discrete or continuous. User provided PDF does not need to be normalized, but must be positive everywhere.

Discrete PDF is treat as a histogram, while the continuous PDF is treated as piecewise linear and continous. The CDF is calculated (simple partial sum for discrete, triangle rule for continuous) and then used for value sampling. This is done using a binary search.

3.29.2 Member Function Documentation

3.29.2.1 getGaussian()

```
double Random_Number_Generator::getGaussian (
            double mu,
            double sigma )
```

Draw a number from a normal distribution.

Adapted from Wikipedia, annotated by Denis St-Onge Box Mueller generates numbers in pairs, so store both, return one at a time.

3.29.2.2 getUniform()

```
double Random_Number_Generator::getUniform ( )
```

Get a random number in the range (0,1), exclusive.

Uses Numerical Recipes ran2 algorithm.

3.29.2.3 getUserNumber()

```
double Random_Number_Generator::getUserNumber ( )
```

Get a random number from the user distribution.

Uses binary search (O(log n)) and quadratic interpolation for continuous distributions.

3.29.2.4 loadUserPDFfromFile()

```
void Random_Number_Generator::loadUserPDFfromFile (
            const bool isDiscrete,
            const char * fname )
```

Load a user distribution from file.

File is in ascii format with two columns. First column representes the value while the second column represents the probability associated with that value.

Values do not need to be equally spaced.

Continous PDFs are treated as piecewise linear between points. Therefore the resulting CDFs are continuous in both the zeroth and first derivatives.

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

- src/utils/RNG.hpp
- src/utils/RNG.cpp

3.30 RegisterFieldBoundary Struct Reference

An object which, when instantiated, registers a field boundary condition.

```
#include <field_bc_factory.hpp>
```

Public Member Functions

- **RegisterFieldBoundary** (const std::string &type, Field_BC_Factory::Factory factory)

3.30.1 Detailed Description

An object which, when instantiated, registers a field boundary condition.

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

- src/boundaries/field_bc_factory.hpp

3.31 RegisterParticleBoundary Struct Reference

Public Member Functions

- **RegisterParticleBoundary** (const std::string &type, Part_BC_Factory::Factory factory)

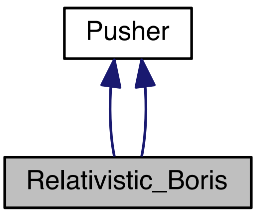
The documentation for this struct was generated from the following file:

- src/boundaries/particle_bc_factory.hpp

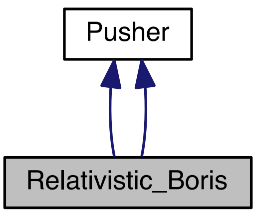
3.32 Relativistic_Boris Class Reference

Relativistic Boris pusher.

Inheritance diagram for Relativistic_Boris:



Collaboration diagram for Relativistic_Boris:



Public Member Functions

- int **Step** (Particle *part, Field_part *field, double dt)
- int **Step** (Particle *part, Field_part *field, double dt)

3.32.1 Detailed Description

Relativistic Boris pusher.

Uses the pusher described in "Simulation of beams or plasmas crossing at relativistic velocity"

J.-L. Vay, Phys. Plasmas 15 (5) 2007

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

- src/pusher/relativisticBoris.cpp
- src/pusher/relativisticBoris.hpp

3.33 RNG_State Struct Reference

```
#include <RNG.hpp>
```

Public Attributes

- long int **initialSeed**
- long int **idum**
- long int **idum2**
- long int **iy**
- long int **iv** [RNG_NTAB]
- double **z0**
- double **z1**
- bool **generate**

3.33.1 Detailed Description

Structure that contains all the infomration for a random number generator. Can be written/read using fwrite/fread.

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

- src/utils/RNG.hpp