Bemsolver

2.0

Generated by Doxygen 1.7.3

Wed Feb 9 2011 15:37:33

Contents

Chapter 1

Class 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

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 calcD3triangle Class Reference

`#include <bem.h>`

Public Member Functions

- **calcD3triangle** ()
- ∼**calcD3triangle** ()
- double **GetIntL1** (double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3)
- double **GetIntL1OutOfPlane** (double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3, double rp)
- double **GetIntL1divR3** (double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3, double rp)
- double **triangleint** (**trianglefn** ∗fn, double x0, double y0, double z0, double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3)
- double **G3Danalytic** (double xp, double yp, double zp, double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3, bool inversenorm)
- double **G3DdnAnalytic** (double xp, double yp, double zp, double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3, bool inversenorm)
- void **triangleint_pot_exyz** (double x0, double y0, double z0, double x1, double y1, double z1, double x2, double y2, double z2, double x3, double y3, double z3, bool inversenorm, double &pot, double &ex, double &ey, double &ez)

Static Public Member Functions

- static double **G3D** (double x0, double y0, double z0, double x1, double y1, double z1)

- static double **dG3Ddx** (double x0, double y0, double z0, double x1, double y1, double z1)

- static double **dG3Ddy** (double x0, double y0, double z0, double x1, double y1, double z1)

- static double **dG3Ddz** (double x0, double y0, double z0, double x1, double y1, double z1)

Protected Member Functions

- void **gauleg** (double x1, double x2, double x[ ], double **w**[ ], int **n**)

- void **gauslegsimplextriangle** (double **xi**[ ], double **eta**[ ], double c[ ], int **n**)

Protected Attributes

- double **EPS**

- double ∗ **xi**

- double ∗ **eta**

- double ∗ **w**

- int **numquad**

- int **n**

4.1.1 Constructor & Destructor Documentation

4.1.1.1 calcD3triangle::calcD3triangle ( )

4.1.1.2 calcD3triangle::~calcD3triangle ( )

4.1.2 Member Function Documentation

4.1.2.1 static double calcD3triangle::dG3Ddx (double *x0,* double *y0,* double *z0,* double *x1,* double *y1,* double *z1*) [static]

4.1.2.2 static double calcD3triangle::dG3Ddy (double *x0,* double *y0,* double *z0,* double *x1,* double *y1,* double *z1*) [static]

4.1.2.3 static double calcD3triangle::dG3Ddz (double *x0,* double *y0,* double *z0,* double *x1,* double *y1,* double *z1*) [static]

4.1.2.4 static double calcD3triangle::G3D (double *x0,* double *y0,* double *z0,* double *x1,* double *y1,* double *z1*) [static]

4.1.2.5 double calcD3triangle::G3Danalytic (double *xp,* double *yp,* double *zp,* double *x1,* double *y1,* double *z1,* double *x2,* double *y2,* double *z2,* double *x3,* double *y3,* double *z3,* bool *inversenorm*)

4.1.2.6 double calcD3triangle::G3DdnAnalytic (double *xp,* double *yp,* double *zp,* double *x1,* double *y1,* double *z1,* double *x2,* double *y2,* double *z2,* double *x3,* double *y3,* double *z3,* bool *inversenorm*)

4.1.2.7 void calcD3triangle::gauleg (double *x1,* double *x2,* double *x[],* double *w[],* int *n*) [protected]

4.1.2.8 void calcD3triangle::gauslegsimplextriangle (double *xi[],* double *eta[],* double *c[],* int *n*) [protected]

4.1.2.9 double calcD3triangle::GetIntL1 (double *x1,* double *y1,* double *z1,* double *x2,* double *y2,* double *z2,* double *x3,* double *y3,* double *z3*)

4.1.2.10 double calcD3triangle::GetIntL1divR3 (double *x1,* double *y1,* double *z1,* double *x2,* double *y2,* double *z2,* double *x3,* double *y3,* double *z3,* double *rp*)

4.1.2.11 double calcD3triangle::GetIntL1OutOfPlane (double *x1,* double *y1,* double *z1,* double *x2,* double *y2,* double *z2,* double *x3,* double *y3,* double *z3,* double *rp*)

4.1.2.12 double calcD3triangle::triangleint (trianglefn * *fn,* double *x0,* double *y0,* double *z0,* double *x1,* double *y1,* double *z1,* double *x2,* double *y2,* double *z2,* double *x3,* double *y3,* double *z3*)

4.1.2.13 void calcD3triangle::triangleint_pot_exyz (double *x0,* double *y0,* double *z0,* double *x1,* double *y1,* double *z1,* double *x2,* double *y2,* double *z2,* double *x3,* double *y3,* double *z3,* bool *inversenorm,* double & *pot,* double & *ex,* double & *ey,* double & *ez*)

4.1.3 Member Data Documentation

4.1.3.1 double calcD3triangle::EPS [protected]

4.1.3.2 double* calcD3triangle::eta [protected]

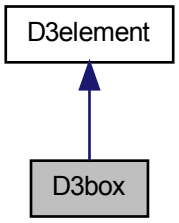
4.1.3.3 int calcD3triangle::n [protected]

- **bem.h**

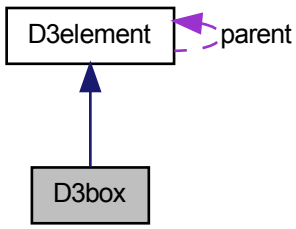
4.2 D3box Class Reference

`#include <bem.h>`

Inheritance diagram for D3box:



Collaboration diagram for D3box:



Public Member Functions

- **D3box** (double xl_, double yl_, double zl_, double **x**, double **y**, double **z**, double phi_, double theta_, double psi_, bool **inversenorm**=false, **PD3element parent**=NULL)

  Creates a 3-dimensional symmetric box.

- ~**D3box** ()

Protected Attributes

- double **xl**
- double **yl**
- double **zl**

4.2.1 Constructor & Destructor Documentation

4.2.1.1 D3box::D3box (double *xl_*, double *yl_*, double *zl_*, double *x*, double *y*, double *z*, double *phi_*, double *theta_*, double *psi_*, bool *inversenorm* = `false`, **PD3element** *parent* = `NULL`) `[inline]`

Creates a 3-dimensional symmetric box.

Initial orientation of the Box and coordinate system:

x-axis: towards the viewer

y-axis: upwards

z-axis: to the right

x,y,z: Coordinates of the boxes origin

x1_: defines length of box in x-direction

y1_: defines length of box in y-direction

z1_: defines length of box in z-direction

negative lengths are also possible

x + x1_, y + y1_, z + z1_ define the edges of the box

The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

4.2.1.2 D3box::~D3box ( ) [inline]

4.2.2 Member Data Documentation

4.2.2.1 double D3box::xl [protected]

4.2.2.2 double D3box::yl [protected]

4.2.2.3 double D3box::zl [protected]

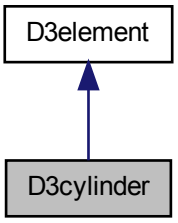
The documentation for this class was generated from the following file:

- **bem.h**

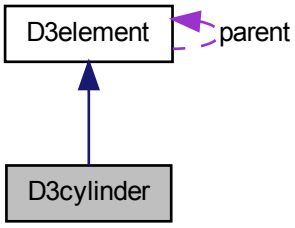
4.3 D3cylinder Class Reference

`#include <bem.h>`

Inheritance diagram for D3cylinder:



Collaboration diagram for D3cylinder:



Public Member Functions

- **D3cylinder** (double r_, double l_, double **x**, double **y**, double **z**, double phi_-, double theta_, double psi_, int sectors_=12, bool subdivide=true, bool **inversenorm**=false, **PD3element parent**=NULL)

  Creates a cylinder with radius r_ and length l_.

- ~**D3cylinder** ()

Protected Attributes

- double **r**
- double **l**
- double **sectors**

4.3.1 Constructor & Destructor Documentation

4.3.1.1 D3cylinder::D3cylinder (double *r_*, double *l_*, double *x*, double *y*, double *z*, double *phi_*, double *theta_*, double *psi_*, int *sectors_* = 12, bool *subdivide* = true, bool *inversenorm* = false, **PD3element** *parent* = NULL) [inline]

Creates a cylinder with radius r_ and length l_.

The bottom of the cylinder is centered at (x,y,z).

The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

For phi_ = theta_ = psi_ = 0, the bottom of the cylinder lies in the x,y-plane and the z-extension correspond to the length l_.

The parameter sectors_ provides an option for refinement. It is set to 12 by initialization but can also be set manually. A value of sectors_ = 8 creates a cylinder with 8 edges on the bottom for example. sectors_ is not limited to powers of two or even numbers!! 5 or 11 works as well.

4.3.1.2 D3cylinder::~D3cylinder ( ) [inline]

4.3.2 Member Data Documentation

4.3.2.1 double D3cylinder::l [protected]

4.3.2.2 double D3cylinder::r [protected]

4.3.2.3 double D3cylinder::sectors [protected]

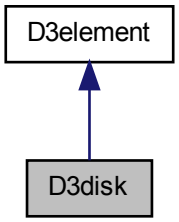
The documentation for this class was generated from the following file:

- **bem.h**

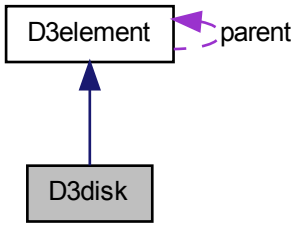
4.4 D3disk Class Reference

`#include <bem.h>`

Inheritance diagram for D3disk:



Collaboration diagram for D3disk:



Public Member Functions

- **D3disk** (double r_, double **x**, double **y**, double **z**, double phi_, double theta_-, double psi_, int sectors_=32, bool subdivide=true, bool **inversenorm**=false, **PD3element parent**=NULL)

  Creates a disk with radius r_.

- ~**D3disk** ()

Protected Attributes

- double **r**
- double **sectors**

4.4.1 Constructor & Destructor Documentation

4.4.1.1 D3disk::D3disk (double *r_*, double *x*, double *y*, double *z*, double *phi_*, double *theta_*, double *psi_*, int *sectors_* = `32`, bool *subdivide* = `true`, bool *inversenorm* = `false`, **PD3element** *parent* = `NULL`) `[inline]`

Creates a disk with radius r_.

The disk is centered at (x,y,z).

The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

For phi_ = theta_ = psi_ = 0, the disk lies in the x,y-plane and there is no extension in z-direction

The parameter sectors_ provides an option for refinement. It is set to 32 by initialization but can also be set manually.

A value of sectors_ = 8 creates a disk with 8 edges. sectors_ is not limited to powers of two or even numbers!! 5 or 11 for example work as well.

4.4.1.2 D3disk::~D3disk ( ) [inline]

4.4.2 Member Data Documentation

4.4.2.1 double D3disk::r [protected]

4.4.2.2 double D3disk::sectors [protected]

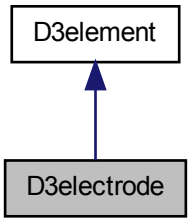
The documentation for this class was generated from the following file:

- **bem.h**

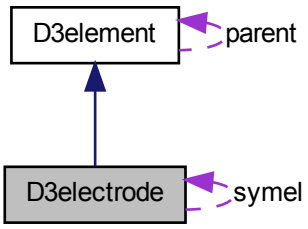
4.5 D3electrode Class Reference

`#include <bem.h>`

Inheritance diagram for D3electrode:



Collaboration diagram for D3electrode:



Public Member Functions

- **D3electrode** ()

  Constructor for an electrode.

- ∼**D3electrode** ()
- void **insert** (**PD3element** el)

  Allocates an D3element (p. ***20) to an electrode, multiple elements can be allocated to one electrode.***

- void **SetVoltage** (double voltage_)

  Sets the voltage for the electrode. Unit: Volt.

- double **GetVoltage** ()

  Gives the voltage of the electrode. Unit: Volt.

- void **SetSymmetryWith** (bool xsym, bool ysym, bool zsym, bool diagmirror, **D3electrode** ∗symel2, int rotsym=0, int **totalrot**=0)

  Eliminates small numerical deviations from perfect symmetry.

- void **SetSymmetry** (bool xsym, bool ysym, bool zsym, bool diagmirror, int rotsym=0, int totalrot2=0)

  Eliminates small numerical deviations from perfect symmetry.

- **D3electrode** ∗ **GetSymmetryWith** (int num)
- int **GetTotalrot** ()
- void **SetCardinalNumber** (int num)
- int **GetCardinalNumber** ()

Protected Attributes

- int **cardinalnum**
- double **voltage**
- **D3electrode** ∗∗ **symel**
- int **totalrot**

4.5.1 Constructor & Destructor Documentation

4.5.1.1 D3electrode::D3electrode ( )

Constructor for an electrode.

The father class is **D3element** (p. 20), which means an electrode consists of at least one 3-dimensional element and additional attributes such as voltage etc.

4.5.1.2 D3electrode::∼D3electrode ( )

4.5.2 Member Function Documentation

4.5.2.1 int D3electrode::GetCardinalNumber ( )

4.5.2.2 D3electrode∗ D3electrode::GetSymmetryWith (int *num*)

4.5.2.3 int D3electrode::GetTotalrot ( )

4.5.2.4 double D3electrode::GetVoltage ( )

Gives the voltage of the electrode. Unit: Volt.

4.5.2.5 void D3electrode::insert (PD3element *el*)

Allocates an D3element (p. **20) to an electrode, multiple elements can be allocated to one electrode.**

Parameters

el	**D3element** (p. 20) that is being allocated to the electrode

4.5.2.6 void D3electrode::SetCardinalNumber (int *num*)

4.5.2.7 void D3electrode::SetSymmetry (bool *xsym,* bool *ysym,* bool *zsym,* bool *diagmirror,* int *rotsym* = 0, int *totalrot2* = 0)

Eliminates small numerical deviations from perfect symmetry.

If you need an electrode that is symmetric to a certain plane or direction, you have to

place it properly at first.

The method will now only correct small numerical deviations from perfect symmetry! It cannot provide large-scale translations or rotations!

xsym=true means symmetric with respect to x=0 plane

ysym=true means symmetric with respect to y=0 plane

zsym=true means symmetric with respect to z=0 plane

diagmirror=true means symmetric with respect of exchange of x and y (if z is set as axis in SymmetrizeCharges())

diagmirror=true means symmetric with respect of exchange of y and z (if x is set as axis in SymmetrizeCharges())

diagmirror=true means symmetric with respect of exchange of x and z (if y is set as axis in SymmetrizeCharges())

rotsym means that symmetric with respect to rotsym/totalsym∗360 degree rotation (set in a loop if you have rotational symmetry)

Parameters

xsym	true, if symmetric with respect to x=0 plane
ysym	true, if symmetric with respect to y=0 plane
zsym	true, if symmetric with respect to z=0 plane
diagmirror	true, if see above
rotsym	?? int ??
totalrot2	??

4.5.2.8 void D3electrode::SetSymmetryWith (bool *xsym,* bool *ysym,* bool *zsym,* bool *diagmirror,* D3electrode ∗ *symel2,* int *rotsym* = 0, int *totalrot* = 0)

Eliminates small numerical deviations from perfect symmetry.

If you need an electrode that is symmetric to a certain plane or direction, you have to place it properly at first.

The method will now only correct small numerical deviations from perfect symmetry! It cannot provide large-scale translations or rotations!

Additionally to the methode SetSymmetry , this method also provides to set the current electrode in perfect symmetry with the one specified by ∗syme12

xsym=true means symmetric with respect to x=0 plane

ysym=true means symmetric with respect to y=0 plane

zsym=true means symmetric with respect to z=0 plane

diagmirror=true means symmetric with respect of exchange of x and y (if z is set as axis in SymmetrizeCharges())

diagmirror=true means symmetric with respect of exchange of y and z (if x is set as axis in SymmetrizeCharges())

diagmirror=true means symmetric with respect of exchange of x and z (if y is set as axis in SymmetrizeCharges())

symel2 is the other electrode unequal to ∗this that the symmetry refers to

rotsym means that symmetric with respect to rotsym/totalsym∗360 degree rotation (set in a loop if you have rotational symmetry)

Parameters

xsym	true, if symmetric with respect to x=0 plane
ysym	true, if symmetric with respect to y=0 plane
zsym	true, if symmetric with respect to z=0 plane
diagmirror	true, if see above
symel2	other electrode unequal to ∗this that the symmetry refers to
rotsym	?? int ??
totalrot	??

4.5.2.9 void D3electrode::SetVoltage (double *voltage_*)

Sets the voltage for the electrode. Unit: Volt.

Parameters

*voltage_*	voltage for the electrode, unit: Volt.

4.5.3 Member Data Documentation

4.5.3.1 int D3electrode::cardinalnum [protected]

4.5.3.2 D3electrode∗∗ D3electrode::symel [protected]

4.5.3.3 int D3electrode::totalrot [protected]

4.5.3.4 double D3electrode::voltage [protected]

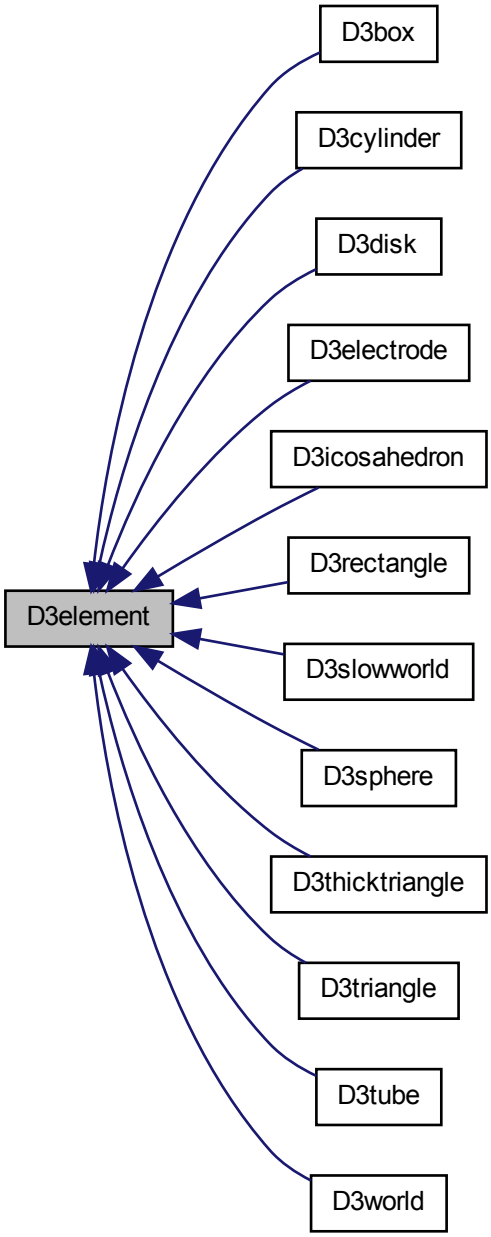
The documentation for this class was generated from the following file:

- **bem.h**

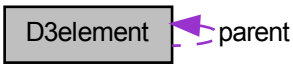
4.6 D3element Class Reference

```
#include <bem.h>
```

Inheritance diagram for D3element:



Collaboration diagram for D3element:



Public Member Functions

- virtual void **correctNorm** (double x0, double y0, double z0)

  Uses ray-shooting to determine the norm and should only be used for externally imported geometrical figures.

- virtual bool **IntersectWithRay** (double x0, double y0, double z0, double xdir, double ydir, double zdir, double &mul)
- virtual void **InsertTGeoVolume** (TGeoVolume ∗top, TGeoMedium ∗mat, TGeoMedium ∗matVak, TGeoManager ∗geom)
- virtual double **GetSelfPotential** ()
- virtual double **GetPotentialAt** (double **x**, double **y**, double **z**)
- virtual double **GetSelfDoubleLayerPotential** ()
- virtual double **GetDoubleLayerPotentialAt** (double **x**, double **y**, double **z**)
- virtual void **GetPotentialAndFieldAt** (double **x**, double **y**, double **z**, double &pot, double &ex, double &ey, double &ez)
- virtual double **GetArea** ()
- virtual void **GetCenter** (double &**x**, double &**y**, double &**z**)
- **D3element** (double x_, double y_, double z_, double phi_, double theta_, double psi_, bool inversenorm_=false, bool refineable_=true, **D3element** ∗parent_=NULL, double **epsilon**=0, string **name**="")
- ∼**D3element** ()
- virtual void **GetReferencePoint** (double &**x**, double &**y**, double &**z**)
- virtual void **init** (bool ignorefirst=false)
- virtual void **refine** (double length)

  Refines a geometrical figure.

- virtual void **refine** (double length, int num)
- int **GetAmountOfSubelements** ()
- int **PrintAmountOfSubelements** ()
- void **GetListOfBaseElements** (**PD3element** ∗el, int &cnt)
- virtual void **rotate** (double phi_, double theta_, double psi_)
- virtual void **shift** (double xs, double ys, double zs)
- virtual void **rotate2** (double phi_, double theta_, double psi_, double **x**, double **y**, double **z**, double &x2, double &y2, double &z2)

- virtual void **rotateinv** (double phi_, double theta_, double psi_, double **x**, double **y**, double **z**, double &x2, double &y2, double &z2)
- void **Add** (**PD3element** el)
- virtual void **GetTriangle** (double ∗A, double ∗B, double ∗C, double ∗COL)
- virtual void **GetRectangle** (double ∗A, double ∗B, double ∗C, double ∗D, double ∗COL)
- virtual void **SetNormTowards** (double **x**, double **y**, double **z**, bool towards)
- virtual void **createNewSubelements** (double length)
- virtual string **getName** ()
- virtual void **setName** (string name2)
- virtual bool **GetInversenorm** ()

Public Attributes

- **D3element** ∗ **parent**
- int **Color**
- string **name**
- list< **D3element** ∗ > **subelement**

Protected Member Functions

- virtual void **deleteSubelements** ()

Protected Attributes

- double **epsilon**
- TGeoHMatrix **h**
- double **x**
- double **y**
- double **z**
- double **phi**
- double **theta**
- double **psi**
- bool **isBaseElement**
- bool **refineable**
- bool **inversenorm**

4.6.1 Constructor & Destructor Documentation

4.6.1.1 D3element::D3element (double *x_*, double *y_*, double *z_*, double *phi_*, double *theta_*, double *psi_*, bool *inversenorm_* = false, bool *refineable_* = true, D3element * *parent_* = NULL, double *epsilon* = 0, string *name* = " ")

4.6.1.2 D3element::~D3element ( )

4.6.2 Member Function Documentation

4.6.2.1 void D3element::Add (PD3element *el*)

4.6.2.2 virtual void D3element::correctNorm (double *x0*, double *y0*, double *z0*) [virtual]

Uses ray-shooting to determine the norm and should only be used for externally imported geometrical figures.

From Point (x0, y0, z0) Rays are being shot towards the object in order to determine in which direction the normal vector of each surface is pointing. Generally, all normal vectors can point either to the inside of the object or to the outside of the object. We should make sure that all normal vectors are pointing in the same direction!

A blue color on each outside-surface indicates a correct norm, which means that all normal Vectors are pointing in the same direction. The ions that fly by the electrodes see only the outside-surfaces of the electrodes. If an outside-surface has an incorrect norm, the ion will not be influenced correctly by the surface. **This means that all outside-surfaces should be blue!**

A red color on at least one outside-surface indicates that at least one normal vector is pointing in a different direction than the others. In this case we need to modify the parameters x0/y0/z0.

Important for the "shooting point" (x0,y0,z0): It must not be in the same plane as one of the surfaces!! Furthermore it must not be inside a geometrical figure!! Otherwise the Ray-Shooting method will not work properly.

Parameters

x0	x-coordinate of the shooting-point
y0	y-coordinate of the shooting-point
z0	z-coordinate of the shooting-point

4.6.2.3 virtual void D3element::createNewSubelements (double *length*) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 37).

4.6.2.4 virtual void D3element::deleteSubelements ( ) [protected, virtual]

4.6.2.5 int D3element::GetAmountOfSubelements ( )

4.6.2.6 virtual double D3element::GetArea ( ) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 37).

4.6.2.7 virtual void D3element::GetCenter (double & *x*, double & *y*, double & *z*) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 37).

4.6.2.8 virtual double D3element::GetDoubleLayerPotentialAt (double *x*, double *y*, double *z*) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 37).

4.6.2.9 virtual bool D3element::GetInversenorm ( ) [virtual]

4.6.2.10 void D3element::GetListOfBaseElements (**PD3element** * *el*, int & *cnt*)

4.6.2.11 virtual string D3element::getName ( ) [virtual]

4.6.2.12 virtual void D3element::GetPotentialAndFieldAt (double *x*, double *y*, double *z*, double & *pot*, double & *ex*, double & *ey*, double & *ez*) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 38).

4.6.2.13 virtual double D3element::GetPotentialAt (double *x*, double *y*, double *z*) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 38).

4.6.2.14 virtual void D3element::GetRectangle (double * *A*, double * *B*, double * *C*, double * *D*, double * *COL*) [virtual]

Reimplemented in **D3rectangle** (p. 38).

4.6.2.15 virtual void D3element::GetReferencePoint (double & *x*, double & *y*, double & *z*) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 38).

4.6.2.16 virtual double D3element::GetSelfDoubleLayerPotential ( ) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 38).

4.6.2.17 virtual double D3element::GetSelfPotential ( ) [virtual]

Reimplemented in **D3triangle** (p. 54), and **D3rectangle** (p. 38).

4.6.2.18 virtual void D3element::GetTriangle (double * *A*, double * *B*, double * *C*, double * *COL*) [virtual]

Reimplemented in **D3triangle** (p. 55).

4.6.2.19 virtual void D3element::init (bool *ignorefirst* = false) [virtual]

4.6.2.20 virtual void D3element::InsertTGeoVolume (TGeoVolume * *top*, TGeoMedium * *mat*, TGeoMedium * *matVak*, TGeoManager * *geom*) [virtual]

Reimplemented in **D3triangle** (p. 55), and **D3rectangle** (p. 38).

4.6.2.21 virtual bool D3element::IntersectWithRay (double *x0*, double *y0*, double *z0*, double *xdir*, double *ydir*, double *zdir*, double & *mul*) [virtual]

Reimplemented in **D3triangle** (p. 55), and **D3rectangle** (p. 38).

4.6.2.22 int D3element::PrintAmountOfSubelements ( )

4.6.2.23 virtual void D3element::refine (double *length*, int *num*) [virtual]

4.6.2.24 virtual void D3element::refine (double *length*) [virtual]

Refines a geometrical figure.

This means that the original geometrical object will be filled with smaller versions of itself. Works good with cubic objects.

Does not work good with other objects like cylinders etc..?? //x

The parameter length determines the size of the smaller objects.

Example: Cube with sides (1x1x1) and parameter length 0.1 => 10 small cubes will be fit along each side, so the cube will then contain 10∗10∗10=1000 small cubes.

Press ctrl+w to see the lattice model and get an impression of the refinement. Press ctrl+r to see the original model again

Parameters

length	size of the smaller objects that fill up the incident object

4.6.2.25 virtual void D3element::rotate (double *phi_*, double *theta_*, double *psi_*) [virtual]

Reimplemented in **D3triangle** (p. 55), and **D3rectangle** (p. 38).

4.6.2.26 virtual void D3element::rotate2 (double *phi_*, double *theta_*, double *psi_*, double *x*, double *y*, double *z*, double & *x2*, double & *y2*, double & *z2*) [virtual]

4.6.2.27 virtual void D3element::rotateinv (double *phi_*, double *theta_*, double *psi_*, double *x*, double *y*, double *z*, double & *x2*, double & *y2*, double & *z2*) [virtual]

4.6.2.28 virtual void D3element::setName (string *name2*) [virtual]

4.6.2.29 virtual void D3element::SetNormTowards (double *x*, double *y*, double *z*, bool *towards*) [virtual]

Reimplemented in **D3triangle** (p. 55), and **D3rectangle** (p. 39).

4.6.2.30 virtual void D3element::shift (double *xs*, double *ys*, double *zs*) [virtual]

Reimplemented in **D3triangle** (p. 55), and **D3rectangle** (p. 39).

4.6.3 Member Data Documentation

4.6.3.1 int D3element::Color

4.6.3.2 double D3element::epsilon [protected]

4.6.3.3 TGeoHMatrix D3element::h [protected]

4.6.3.4 bool D3element::inversenorm [protected]

4.6.3.5 bool D3element::isBaseElement [protected]

4.6.3.6 string D3element::name

4.6.3.7 D3element* D3element::parent

4.6.3.8 double D3element::phi [protected]

4.6.3.9 double D3element::psi [protected]

4.6.3.10 bool D3element::refineable [protected]

4.6.3.11 list<D3element*> D3element::subelement

4.6.3.12 double D3element::theta [protected]

4.6.3.13 double D3element::x [protected]

Reimplemented in **D3world** (p. 66).

4.6.3.14 double D3element::y [protected]

4.6.3.15 double D3element::z [protected]

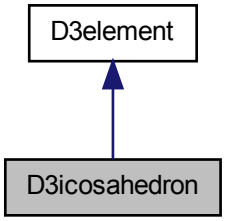
The documentation for this class was generated from the following file:

- **bem.h**

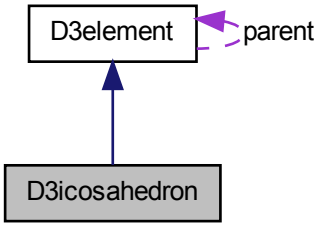
4.7 D3icosahedron Class Reference

```
#include <bem.h>
```

Inheritance diagram for D3icosahedron:



Collaboration diagram for D3icosahedron:



Public Member Functions

- **D3icosahedron** (double edgelength, double **refinement**, double **x**, double **y**, double **z**, double phi_, double theta_, double psi_, bool **inversenorm**=false, **PD3element parent**=NULL)
- ~**D3icosahedron** ()
- void **triangle** (int a, int b, int c, bool **inversenorm**=false)

Protected Attributes

- double **r**
- double **refinement**
- double(* **V**)[12][3]

4.7.1 Constructor & Destructor Documentation

4.7.1.1 D3icosahedron::D3icosahedron (double *edgelength,* double *refinement,* double *x,* double *y,* double *z,* double *phi_,* double *theta_,* double *psi_,* bool *inversenorm =* `false`, **PD3element** *parent =* `NULL`) `[inline]`

4.7.1.2 D3icosahedron::∼D3icosahedron ( ) `[inline]`

4.7.2 Member Function Documentation

4.7.2.1 void D3icosahedron::triangle (int *a,* int *b,* int *c,* bool *inversenorm =* `false`) `[inline]`

4.7.3 Member Data Documentation

4.7.3.1 double D3icosahedron::r `[protected]`

4.7.3.2 double D3icosahedron::refinement `[protected]`

4.7.3.3 double(∗ D3icosahedron::V)[12][3] `[protected]`

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

- **bem.h**

4.8 D3ImportedElectrodes Class Reference

`#include <bem.h>`

Public Member Functions

- **D3ImportedElectrodes** ()

  Can be used to import an Electrode from an external .dxf file, that was created by AutoCAD.

- ∗ ∼**D3ImportedElectrodes** ()
- bool **Import** (const char ∗file, bool ignore3DFace_=false, bool ignorePolyline_-=false)

  Loads the .dxf file into memory.

- virtual void **addLayer** (const DL_LayerData &data)
- virtual void **addPoint** (const DL_PointData &data)
- virtual void **addLine** (const DL_LineData &data)
- virtual void **addArc** (const DL_ArcData &data)
- virtual void **addCircle** (const DL_CircleData &data)
- virtual void **addPolyline** (const DL_PolylineData &data)

- virtual void **addVertex** (const DL_VertexData &data)
- virtual void **add3DFace** (const DL_3DFaceData &data)
- virtual void **addBlockRecord** (const DL_BlockRecordData &data)
- void **endSequence** ()
- void **ListElectrodes** ()

  Iterates over all electrodes and displays their names in a command line output.

- **D3electrode** & **FindElectrode** (const char ∗name)
- void **printAttributes** ()

Public Attributes

- map< string, **D3electrode** > **electrode**

Protected Attributes

- int **Model_Space_Handle**
- bool **ignore_Model_Space_Handle**
- bool **ignore3DFace**
- bool **ignorePolyline**
- bool **importingPolyline**
- int **vertexcnt**
- double **x** [4]
- double **y** [4]
- double **z** [4]

4.8.1 Constructor & Destructor Documentation

4.8.1.1 D3ImportedElectrodes::D3ImportedElectrodes ( )

Can be used to import an Electrode from an external .dxf file, that was created by AutoCAD.

The lines below show an example for proper use of the method

D3ImportedElectrodes (p. 30) ∗impel=new **D3ImportedElectrodes()** (p. 31);

TString importfilename;

logfile.AbsoluteFileName("./Ansaug_endcap.dxf",importfilename);

if(!impel->Import(importfilename)) return 0;

endcapl=& (impel->FindElectrode("endcap"));

endcapl->correctNorm(0,0,-10);

endcapl->refine(0.02);

so genug??

4.8.1.2 * D3ImportedElectrodes::~D3ImportedElectrodes ( )

4.8.2 Member Function Documentation

4.8.2.1 virtual void D3ImportedElectrodes::add3DFace (const DL_3DFaceData & *data*) [virtual]

4.8.2.2 virtual void D3ImportedElectrodes::addArc (const DL_ArcData & *data*) [virtual]

4.8.2.3 virtual void D3ImportedElectrodes::addBlockRecord (const DL_BlockRecordData & *data*) [virtual]

4.8.2.4 virtual void D3ImportedElectrodes::addCircle (const DL_CircleData & *data*) [virtual]

4.8.2.5 virtual void D3ImportedElectrodes::addLayer (const DL_LayerData & *data*) [virtual]

4.8.2.6 virtual void D3ImportedElectrodes::addLine (const DL_LineData & *data*) [virtual]

4.8.2.7 virtual void D3ImportedElectrodes::addPoint (const DL_PointData & *data*) [virtual]

4.8.2.8 virtual void D3ImportedElectrodes::addPolyline (const DL_PolylineData & *data*) [virtual]

4.8.2.9 virtual void D3ImportedElectrodes::addVertex (const DL_VertexData & *data*) [virtual]

4.8.2.10 void D3ImportedElectrodes::endSequence ( )

4.8.2.11 D3electrode& D3ImportedElectrodes::FindElectrode (const char * *name*)

4.8.2.12 bool D3ImportedElectrodes::Import (const char * *file*, bool *ignore3DFace_* = false, bool *ignorePolyline_* = false)

Loads the .dxf file into memory.

Displays the filename given by file , in a command line output.

Returns false and gives an error message if the file cannot be opened.

so genug??

Parameters

file	name of the .dxf file

4.8.2.13 void D3ImportedElectrodes::ListElectrodes ( )

Iterates over all electrodes and displays their names in a command line output.

4.8.2.14 void D3ImportedElectrodes::printAttributes ( )

4.8.3 Member Data Documentation

4.8.3.1 map<string,D3electrode> D3ImportedElectrodes::electrode

4.8.3.2 bool D3ImportedElectrodes::ignore3DFace [protected]

4.8.3.3 bool D3ImportedElectrodes::ignore_Model_Space_Handle [protected]

4.8.3.4 bool D3ImportedElectrodes::ignorePolyline [protected]

4.8.3.5 bool D3ImportedElectrodes::importingPolyline [protected]

4.8.3.6 int D3ImportedElectrodes::Model_Space_Handle [protected]

4.8.3.7 int D3ImportedElectrodes::vertexcnt [protected]

4.8.3.8 double D3ImportedElectrodes::x[4] [protected]

4.8.3.9 double D3ImportedElectrodes::y[4] [protected]

4.8.3.10 double D3ImportedElectrodes::z[4] [protected]

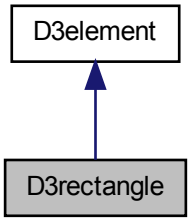
The documentation for this class was generated from the following file:

- **bem.h**

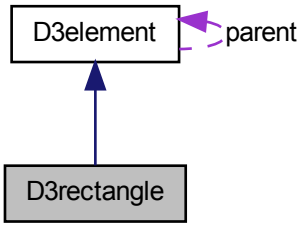
4.9 D3rectangle Class Reference

```
#include <bem.h>
```

Inheritance diagram for D3rectangle:



Collaboration diagram for D3rectangle:



Public Member Functions

- virtual void **InsertTGeoVolume** (TGeoVolume ∗top, TGeoMedium ∗matVak, TGeoMedium ∗mat, TGeoManager ∗geom)
- virtual bool **IntersectWithRay** (double x0, double y0, double z0, double xdir, double ydir, double zdir, double &mul)
- virtual void **SetNormTowards** (double **x**, double **y**, double **z**, bool towards)
- virtual void **GetReferencePoint** (double &**x**, double &**y**, double &**z**)
- virtual double **GetArea** ()
- virtual void **GetCenter** (double &**x**, double &**y**, double &**z**)
- virtual double **GetSelfPotential** ()
- virtual double **GetPotentialAt** (double **x**, double **y**, double **z**)
- virtual double **GetSelfDoubleLayerPotential** ()

- virtual double **GetDoubleLayerPotentialAt** (double **x**, double **y**, double **z**)
- virtual void **GetPotentialAndFieldAt** (double **x**, double **y**, double **z**, double &pot, double &ex, double &ey, double &ez)
- virtual void **GetRectangle** (double ∗A, double ∗B, double ∗C, double ∗D, double ∗COL)
- virtual void **rotate** (double phi_, double theta_, double psi_)
- virtual void **shift** (double xs, double ys, double zs)
- **D3rectangle** (double xa_, double ya_, double xc_, double yc_, double **x**, double **y**, double **z**, double phi_, double theta_, double psi_, bool **inversenorm**=false, **PD3element parent**=NULL, bool subdivideInSqares=false)

  Creates a rectangle from 2 2-dimensional vectors, 1 3-dimensional center vector and 3 Euler angles.

- **D3rectangle** (double xa_, double ya_, double xb_, double yb_, double xc_, double yc_, double **x**, double **y**, double **z**, double phi_, double theta_, double psi_, bool **inversenorm**=false, **PD3element parent**=NULL)

  Creates a rectangle from 3 2-dimensional vectors, 1 center vector and 3 Euler angles.

- **D3rectangle** (char ∗str, double x1_, double y1_, double z1_, double x2_, double y2_, double z2_, double x3_, double y3_, double z3_, double x4_, double y4_, double z4_, bool **inversenorm**=false, **PD3element parent**=NULL)

  Creates a rectangle from coordinates for each corner.

- ∼**D3rectangle** ()
- virtual void **createNewSubelements** (double length)

Public Attributes

- double **x1**
- double **y1**
- double **z1**
- double **x2**
- double **y2**
- double **z2**
- double **x3**
- double **y3**
- double **z3**
- double **x4**
- double **y4**
- double **z4**

Protected Attributes

- double **xa**
- double **ya**
- double **xb**

- double **yb**
- double **xc**
- double **yc**

4.9.1 Constructor & Destructor Documentation

4.9.1.1 D3rectangle::D3rectangle (double *xa_*, double *ya_*, double *xc_*, double *yc_*, double *x*, double *y*, double *z*, double *phi_*, double *theta_*, double *psi_*, bool *inversenorm* = `false`, PD3element *parent* = `NULL`, bool *subdivideInSqares* = `false`)

Creates a rectangle from 2 2-dimensional vectors, 1 3-dimensional center vector and 3 Euler angles.

The parameters x,y,z determine the 1. corner of the rectangle. The Euler-rotations are also performed with respect to this corner.

(x,y,z) can also be interpreted as a center of a coordinate system.

The 2. corner is determined by (xa_ + x, ya_ + y).

The 3. corner is determined by (xb_ + x, yb_ + y).

The 4. corner is determined by (xa_ + xb_ + x, ya_ + yb_ + y). This represents a vector addition of the vectors for the first 2 corners.

The ordering of the corners does not matter!!

The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

They can be used to generate z-coordinates for the rectangle

The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

4.9.1.2 D3rectangle::D3rectangle (double *xa_*, double *ya_*, double *xb_*, double *yb_*, double *xc_*, double *yc_*, double *x*, double *y*, double *z*, double *phi_*, double *theta_*, double *psi_*, bool *inversenorm* = `false`, PD3element *parent* = `NULL`)

Creates a rectangle from 3 2-dimensional vectors, 1 center vector and 3 Euler angles.

The parameters x,y,z determine the 1. corner of the rectangle. The Euler-rotations are also performed with respect to this corner.

(x,y,z) can also be interpreted as a center of a coordinate system.

The 2. corner is determined by (xa_ + x, ya_ + y).

The 3. corner is determined by (xb_ + x, yb_ + y).

The 4. corner is not being calculated automatically, it is specified via (xc_,yc_)!

The ordering of the corners does not matter!!

If the 3 corner-vectors would result in a non-convex object only a triangle will be drawn!

The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

They can be used to generate z-coordinates for the rectangle

The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

4.9.1.3 D3rectangle::D3rectangle (char * *str*, double *x1_*, double *y1_*, double *z1_*, double *x2_*, double *y2_*, double *z2_*, double *x3_*, double *y3_*, double *z3_*, double *x4_*, double *y4_*, double *z4_*, bool *inversenorm* = `false`, **PD3element** *parent* = `NULL`)

Creates a rectangle from coordinates for each corner.

Parameters x1_, y1_, z1_ define the coordinates of the first corner

The remaining paramters define the other corners

wofür der string???--> nur wegen überladung?

4.9.1.4 D3rectangle::~D3rectangle ( ) `[inline]`

4.9.2 Member Function Documentation

4.9.2.1 virtual void D3rectangle::createNewSubelements (double *length*) `[virtual]`

Reimplemented from **D3element** (p. 24).

4.9.2.2 virtual double D3rectangle::GetArea ( ) `[virtual]`

Reimplemented from **D3element** (p. 25).

4.9.2.3 virtual void D3rectangle::GetCenter (double & *x*, double & *y*, double & *z*) `[virtual]`

Reimplemented from **D3element** (p. 25).

4.9.2.4 virtual double D3rectangle::GetDoubleLayerPotentialAt (double *x*, double *y*, double *z*) `[inline, virtual]`

Reimplemented from **D3element** (p. 25).

4.9.2.5 virtual void D3rectangle::GetPotentialAndFieldAt (double *x*, double *y*, double *z*, double & *pot*, double & *ex*, double & *ey*, double & *ez*) [virtual]

Reimplemented from **D3element** (p. 25).

4.9.2.6 virtual double D3rectangle::GetPotentialAt (double *x*, double *y*, double *z*) [inline, virtual]

Reimplemented from **D3element** (p. 25).

4.9.2.7 virtual void D3rectangle::GetRectangle (double * *A*, double * *B*, double * *C*, double * *D*, double * *COL*) [virtual]

Reimplemented from **D3element** (p. 25).

4.9.2.8 virtual void D3rectangle::GetReferencePoint (double & *x*, double & *y*, double & *z*) [virtual]

Reimplemented from **D3element** (p. 25).

4.9.2.9 virtual double D3rectangle::GetSelfDoubleLayerPotential ( ) [virtual]

Reimplemented from **D3element** (p. 26).

4.9.2.10 virtual double D3rectangle::GetSelfPotential ( ) [virtual]

Reimplemented from **D3element** (p. 26).

4.9.2.11 virtual void D3rectangle::InsertTGeoVolume (TGeoVolume * *top*, TGeoMedium * *matVak*, TGeoMedium * *mat*, TGeoManager * *geom*) [virtual]

Reimplemented from **D3element** (p. 26).

4.9.2.12 virtual bool D3rectangle::IntersectWithRay (double *x0*, double *y0*, double *z0*, double *xdir*, double *ydir*, double *zdir*, double & *mul*) [virtual]

Reimplemented from **D3element** (p. 26).

4.9.2.13 virtual void D3rectangle::rotate (double *phi_*, double *theta_*, double *psi_*) [virtual]

Reimplemented from **D3element** (p. 27).

4.9.2.14 virtual void D3rectangle::SetNormTowards (double *x*, double *y*, double *z*, bool *towards*) [virtual]

Reimplemented from **D3element** (p. 27).

4.9.2.15 virtual void D3rectangle::shift (double *xs*, double *ys*, double *zs*) [virtual]

Reimplemented from **D3element** (p. 27).

4.9.3 Member Data Documentation

4.9.3.1 double D3rectangle::x1

4.9.3.2 double D3rectangle::x2

4.9.3.3 double D3rectangle::x3

4.9.3.4 double D3rectangle::x4

4.9.3.5 double D3rectangle::xa [protected]

4.9.3.6 double D3rectangle::xb [protected]

4.9.3.7 double D3rectangle::xc [protected]

4.9.3.8 double D3rectangle::y1

4.9.3.9 double D3rectangle::y2

4.9.3.10 double D3rectangle::y3

4.9.3.11 double D3rectangle::y4

4.9.3.12 double D3rectangle::ya [protected]

4.9.3.13 double D3rectangle::yb [protected]

4.9.3.14 double D3rectangle::yc [protected]

4.9.3.15 double D3rectangle::z1

4.9.3.16 double D3rectangle::z2

4.9.3.17 double D3rectangle::z3

4.9.3.18 double D3rectangle::z4

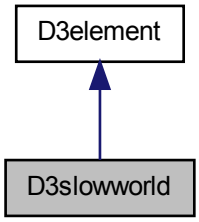
The documentation for this class was generated from the following file:

- **bem.h**

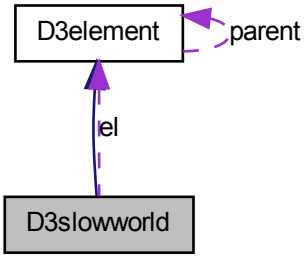
4.10 D3slowworld Class Reference

```
#include <bem.h>
```

Inheritance diagram for D3slowworld:



Collaboration diagram for D3slowworld:



Public Member Functions

- **D3slowworld** ()
- ~**D3slowworld** ()
- void **insert** (**D3electrode** ∗**el**)
- void **GetListOfBaseElements** (**PD3element** ∗**el**, int ∗**electrodeIndexLimit**, int &cnt)
- void **solve** ()
- double **calc** (double **x**, double **y**, double **z**)
- void **draw** ()

Protected Attributes

- double * **f**
- double * **dfdnAll**
- int * **electrodeIndexLimit**
- **PD3element** * **el**
- int **n**
- int **amountOfElectrodes**

4.10.1 Constructor & Destructor Documentation

4.10.1.1 D3slowworld::D3slowworld ( )

4.10.1.2 D3slowworld::~D3slowworld ( )

4.10.2 Member Function Documentation

4.10.2.1 double D3slowworld::calc (double *x*, double *y*, double *z*)

4.10.2.2 void D3slowworld::draw ( )

4.10.2.3 void D3slowworld::GetListOfBaseElements (**PD3element** * *el*, int * *electrodeIndexLimit*, int & *cnt*)

4.10.2.4 void D3slowworld::insert (**D3electrode** * *el*)

4.10.2.5 void D3slowworld::solve ( )

4.10.3 Member Data Documentation

4.10.3.1 int D3slowworld::amountOfElectrodes [protected]

4.10.3.2 double* D3slowworld::dfdnAll [protected]

4.10.3.3 PD3element* D3slowworld::el [protected]

4.10.3.4 int* D3slowworld::electrodeIndexLimit [protected]

4.10.3.5 double* D3slowworld::f [protected]

4.10.3.6 int D3slowworld::n [protected]

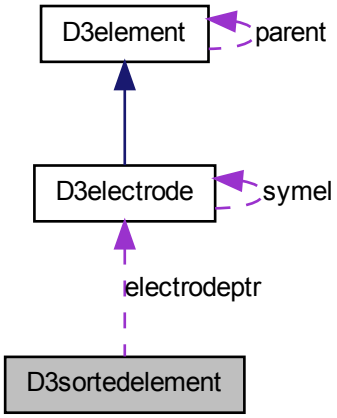
The documentation for this class was generated from the following file:

- **bem.h**

4.11 D3sortedelement Class Reference

`#include <bem.h>`

Collaboration diagram for D3sortedelement:



Public Member Functions

- **D3sortedelement** ()
- ~**D3sortedelement** ()

Public Attributes

- bool **done**
- double **axis**
- double **radius**
- int **index**
- **D3electrode** * **electrodeptr**

Friends

- bool **SmallerThan** (**D3sortedelement** &a, **D3sortedelement** &b, double eps)

4.11.1 Constructor & Destructor Documentation

4.11.1.1 D3sortedelement::D3sortedelement ( ) [inline]

4.11.1.2 D3sortedelement::~D3sortedelement ( ) [inline]

4.11.2 Friends And Related Function Documentation

4.11.2.1 bool SmallerThan (D3sortedelement & *a*, D3sortedelement & *b*, double *eps*) [friend]

4.11.3 Member Data Documentation

4.11.3.1 double D3sortedelement::axis

4.11.3.2 bool D3sortedelement::done

4.11.3.3 D3electrode* D3sortedelement::electrodeptr

4.11.3.4 int D3sortedelement::index

4.11.3.5 double D3sortedelement::radius

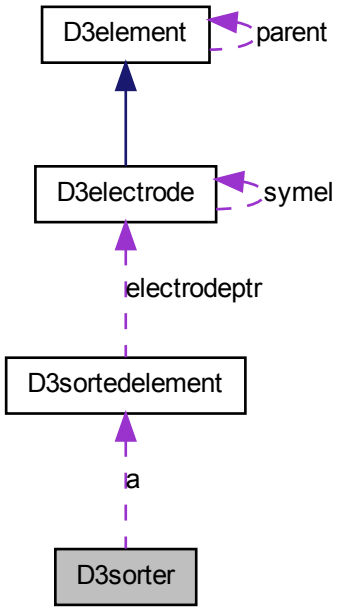
The documentation for this class was generated from the following file:

- **bem.h**

4.12 D3sorter Class Reference

```
#include <bem.h>
```

Collaboration diagram for D3sorter:



Public Member Functions

- void **sort** (**D3sortedelement** ∗arr, int size, double epsi)
- void **quicksort** (int l, int r)
- void **exchange** (int i, int j)

Protected Attributes

- double **eps**

Private Attributes

- **D3sortedelement** ∗ **a**
- int **n**

4.12.1 Member Function Documentation

4.12.1.1 void D3sorter::exchange (int *i*, int *j*) [inline]

4.12.1.2 void D3sorter::quicksort (int *l*, int *r*) [inline]

4.12.1.3 void D3sorter::sort (D3sortedelement * *arr*, int *size*, double *epsi*) [inline]

4.12.2 Member Data Documentation

4.12.2.1 D3sortedelement* D3sorter::a [private]

4.12.2.2 double D3sorter::eps [protected]

4.12.2.3 int D3sorter::n [private]

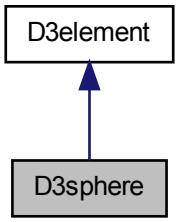
The documentation for this class was generated from the following file:

- **bem.h**

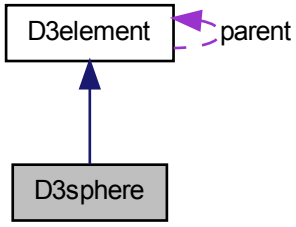
4.13 D3sphere Class Reference

`#include <bem.h>`

Inheritance diagram for D3sphere:



Collaboration diagram for D3sphere:



Public Member Functions

- **D3sphere** (double **r**, double **refinement**, double **x**, double **y**, double **z**, double phi_, double theta_, double psi_, bool **inversenorm**=false, **PD3element parent**=NULL)

  Creates a sphere with radius r.

- ~**D3sphere** ()

Protected Attributes

- double **r**
- double **refinement**

4.13.1 Constructor & Destructor Documentation

4.13.1.1 D3sphere::D3sphere (double *r*, double *refinement*, double *x*, double *y*, double *z*, double *phi_*, double *theta_*, double *psi_*, bool *inversenorm* = `false`, PD3element *parent* = `NULL`) `[inline]`

Creates a sphere with radius r.

The sphere is centered at (x,y,z).

The parameter refinement gives the order of magnitude for the small triangles which assemble the sphere.

Press ctrl+w to see the lattice model and get an impression of the refinement.

Press ctrl+r to see the original model again.

The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

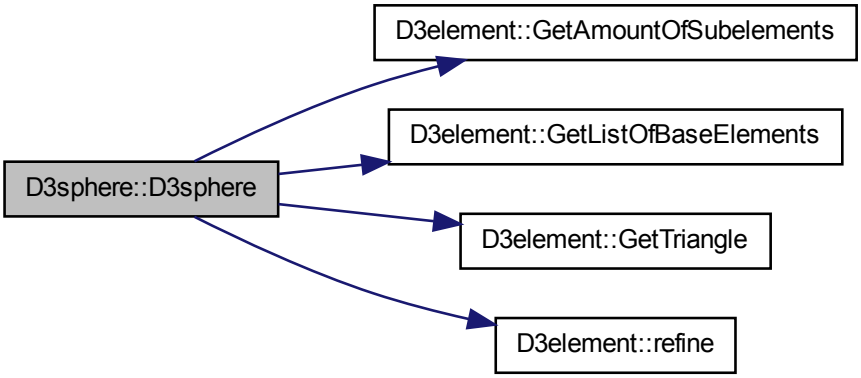
The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

Here is the call graph for this function:



4.13.1.2 D3sphere::~D3sphere ( ) [inline]

4.13.2 Member Data Documentation

4.13.2.1 double D3sphere::r [protected]

4.13.2.2 double D3sphere::refinement [protected]

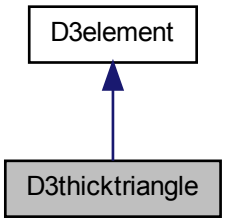
The documentation for this class was generated from the following file:

- **bem.h**

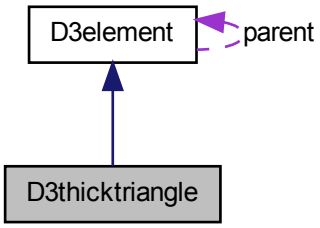
4.14 D3thicktriangle Class Reference

`#include <bem.h>`

Inheritance diagram for D3thicktriangle:



Collaboration diagram for D3thicktriangle:



Public Member Functions

- **D3thicktriangle** (double **xa**, double **ya**, double **xb**, double **yb**, double **htriangle**, double **x**, double **y**, double **z**, double **phi**, double **theta**, double **psi**, bool **inversenorm**=false, **PD3element parent**=NULL)

  Creates a thick triangle from 2 2-dimensional vectors, 1 3-dimensional center vector and 3 Euler-angles.

- ~**D3thicktriangle** ()

Protected Attributes

- double **xa**

- double **ya**
- double **xb**
- double **yb**
- double **htriangle**

4.14.1 Constructor & Destructor Documentation

4.14.1.1 D3thicktriangle::D3thicktriangle (double *xa,* double *ya,* double *xb,* double *yb,* double *htriangle,* double *x,* double *y,* double *z,* double *phi,* double *theta,* double *psi,* bool *inversenorm* = `false`, **PD3element** *parent* = `NULL`) `[inline]`

Creates a thick triangle from 2 2-dimensional vectors, 1 3-dimensional center vector and 3 Euler-angles.

The parameter htriangle determines the thickness of the triangle. For phi_ = theta_= psi_ = 0 the thickness corresponds to the extension in z-direction.

The parameters x,y,z determine the 1. corner of the triangle. The Euler-rotations are also performed with respect to this corner.

(x,y,z) can also be interpreted as a center of a coordinate system.

The 2. corner is determined by (xa_ + x, ya_ + y). The 3. corner is determined by (xb_ + x, yb_ + y).

The ordering does not matter!! The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

4.14.1.2 D3thicktriangle::~D3thicktriangle ( ) `[inline]`

4.14.2 Member Data Documentation

4.14.2.1 double D3thicktriangle::htriangle `[protected]`

4.14.2.2 double D3thicktriangle::xa `[protected]`

4.14.2.3 double D3thicktriangle::xb `[protected]`

4.14.2.4 double D3thicktriangle::ya `[protected]`

4.14.2.5 double D3thicktriangle::yb `[protected]`

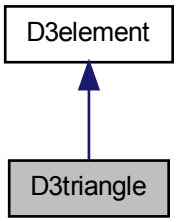
The documentation for this class was generated from the following file:

- **bem.h**

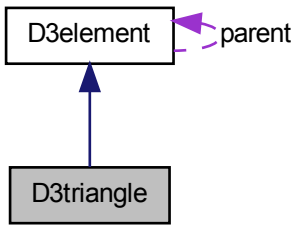
4.15 D3triangle Class Reference

`#include <bem.h>`

Inheritance diagram for D3triangle:



Collaboration diagram for D3triangle:



Public Member Functions

- virtual void **InsertTGeoVolume** (TGeoVolume ∗top, TGeoMedium ∗matVak, TGeoMedium ∗mat, TGeoManager ∗geom)
- virtual bool **IntersectWithRay** (double x0, double y0, double z0, double xdir, double ydir, double zdir, double &mul)

- virtual void **SetNormTowards** (double **x**, double **y**, double **z**, bool towards)
- virtual void **GetReferencePoint** (double &**x**, double &**y**, double &**z**)
- virtual double **GetArea** ()
- virtual void **GetCenter** (double &**x**, double &**y**, double &**z**)
- virtual double **GetSelfPotential** ()
- virtual double **GetPotentialAt** (double **x**, double **y**, double **z**)
- virtual void **GetPotentialAndFieldAt** (double **x**, double **y**, double **z**, double &pot, double &ex, double &ey, double &ez)
- virtual double **GetSelfDoubleLayerPotential** ()
- virtual double **GetDoubleLayerPotentialAt** (double **x**, double **y**, double **z**)
- **D3triangle** (double xa_, double ya_, double xb_, double yb_, double **x**, double **y**, double **z**, double phi_, double theta_, double psi_, bool **inversenorm**=false, **PD3element parent**=NULL, bool **stripeit**=true)

  Creates a triangle from 2 2-dimensional vectors, 1 3-dimensional center vector and 3 Euler-angles.

- **D3triangle** (double x1_, double y1_, double z1_, double x2_, double y2_, double z2_, double x3_, double y3_, double z3_, bool **inversenorm**=false, **PD3element parent**=NULL, bool **stripeit**=true)

  Creates a triangle from 3 3-dimensional vectors.

- ∼**D3triangle** ()
- virtual void **rotate** (double phi_, double theta_, double psi_)
- virtual void **shift** (double xs, double ys, double zs)
- virtual void **GetTriangle** (double ∗A, double ∗B, double ∗C, double ∗COL)
- virtual void **createNewSubelements** (double length)

Static Public Member Functions

- static bool **IntersectWithRay** (double ax, double ay, double az, double bx, double by, double bz, double cx, double cy, double cz, double xdir, double ydir, double zdir, double &mul)

Public Attributes

- double **x1**
- double **y1**
- double **z1**
- double **x2**
- double **y2**
- double **z2**
- double **x3**
- double **y3**
- double **z3**

Protected Member Functions

- void **Stripeit** (double length)
- void **Newtriangles** (double length)

Protected Attributes

- bool **stripeit**
- double **xa**
- double **ya**
- double **xb**
- double **yb**

4.15.1 Constructor & Destructor Documentation

4.15.1.1 D3triangle::D3triangle (double *xa_*, double *ya_*, double *xb_*, double *yb_*, double *x*, double *y*, double *z*, double *phi_*, double *theta_*, double *psi_*, bool *inversenorm* = `false`, **PD3element** *parent* = `NULL`, bool *stripeit* = `true`)

Creates a triangle from 2 2-dimensional vectors, 1 3-dimensional center vector and 3 Euler-angles.

The parameters x,y,z determine the 1. corner of the triangle. The Euler-rotations are also performed with respect to this corner.

(x,y,z) can also be interpreted as a center of a coordinate system.

The 2. corner is determined by (xa_ + x, ya_ + y). The 3. corner is determined by (xb_ + x, yb_ + y). The ordering does not matter!!

The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

They can be used to generate z-coordinates for the triangle

The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

4.15.1.2 D3triangle::D3triangle (double *x1_*, double *y1_*, double *z1_*, double *x2_*, double *y2_*, double *z2_*, double *x3_*, double *y3_*, double *z3_*, bool *inversenorm* = `false`, **PD3element** *parent* = `NULL`, bool *stripeit* = `true`)

Creates a triangle from 3 3-dimensional vectors.

(x1_,y1_z1_), (x2_,y2_,z2_), (x3_,y3_,z3_) define the corners of the triangle. The ordering does not matter!!

4.15.1.3 D3triangle::~D3triangle ( )

4.15.2 Member Function Documentation

4.15.2.1 virtual void D3triangle::createNewSubelements (double *length*) [virtual]

Reimplemented from **D3element** (p. 24).

4.15.2.2 virtual double D3triangle::GetArea ( ) [virtual]

Reimplemented from **D3element** (p. 25).

4.15.2.3 virtual void D3triangle::GetCenter (double & *x*, double & *y*, double & *z*) [virtual]

Reimplemented from **D3element** (p. 25).

4.15.2.4 virtual double D3triangle::GetDoubleLayerPotentialAt (double *x*, double *y*, double *z*) [inline, virtual]

Reimplemented from **D3element** (p. 25).

4.15.2.5 virtual void D3triangle::GetPotentialAndFieldAt (double *x*, double *y*, double *z*, double & *pot*, double & *ex*, double & *ey*, double & *ez*) [virtual]

Reimplemented from **D3element** (p. 25).

4.15.2.6 virtual double D3triangle::GetPotentialAt (double *x*, double *y*, double *z*) [inline, virtual]

Reimplemented from **D3element** (p. 25).

4.15.2.7 virtual void D3triangle::GetReferencePoint (double & *x*, double & *y*, double & *z*) [virtual]

Reimplemented from **D3element** (p. 25).

4.15.2.8 virtual double D3triangle::GetSelfDoubleLayerPotential ( ) [virtual]

Reimplemented from **D3element** (p. 26).

4.15.2.9 virtual double D3triangle::GetSelfPotential ( ) [virtual]

Reimplemented from **D3element** (p. 26).

4.15.2.10 virtual void D3triangle::GetTriangle (double * *A*, double * *B*, double * *C*, double * *COL*) [virtual]

Reimplemented from **D3element** (p. 26).

4.15.2.11 virtual void D3triangle::InsertTGeoVolume (TGeoVolume * *top*, TGeoMedium * *matVak*, TGeoMedium * *mat*, TGeoManager * *geom*) [virtual]

Reimplemented from **D3element** (p. 26).

4.15.2.12 virtual bool D3triangle::IntersectWithRay (double *x0*, double *y0*, double *z0*, double *xdir*, double *ydir*, double *zdir*, double & *mul*) [virtual]

Reimplemented from **D3element** (p. 26).

4.15.2.13 static bool D3triangle::IntersectWithRay (double *ax*, double *ay*, double *az*, double *bx*, double *by*, double *bz*, double *cx*, double *cy*, double *cz*, double *xdir*, double *ydir*, double *zdir*, double & *mul*) [static]

4.15.2.14 void D3triangle::Newtriangles (double *length*) [protected]

4.15.2.15 virtual void D3triangle::rotate (double *phi_*, double *theta_*, double *psi_*) [virtual]

Reimplemented from **D3element** (p. 27).

4.15.2.16 virtual void D3triangle::SetNormTowards (double *x*, double *y*, double *z*, bool *towards*) [virtual]

Reimplemented from **D3element** (p. 27).

4.15.2.17 virtual void D3triangle::shift (double *xs*, double *ys*, double *zs*) [virtual]

Reimplemented from **D3element** (p. 27).

4.15.2.18 void D3triangle::Stripeit (double *length*) [protected]

4.15.3 Member Data Documentation

4.15.3.1 bool D3triangle::stripeit [protected]

4.15.3.2 double D3triangle::x1

4.15.3.3 double D3triangle::x2

4.15.3.4 double D3triangle::x3

4.15.3.5 double D3triangle::xa [protected]

4.15.3.6 double D3triangle::xb [protected]

4.15.3.7 double D3triangle::y1

4.15.3.8 double D3triangle::y2

4.15.3.9 double D3triangle::y3

4.15.3.10 double D3triangle::ya [protected]

4.15.3.11 double D3triangle::yb [protected]

4.15.3.12 double D3triangle::z1

4.15.3.13 double D3triangle::z2

4.15.3.14 double D3triangle::z3

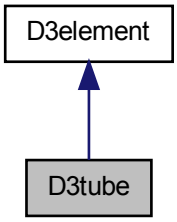
The documentation for this class was generated from the following file:

- **bem.h**

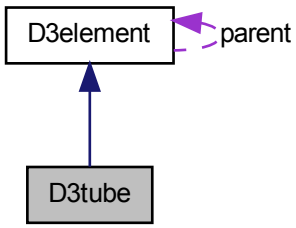
4.16 D3tube Class Reference

`#include <bem.h>`

Inheritance diagram for D3tube:



Collaboration diagram for D3tube:



Public Member Functions

- **D3tube** (double r_, double r2_, double l_, double **x**, double **y**, double **z**, double phi_, double theta_, double psi_, int sectors_=12, bool subdivide=true, bool **inversenorm**=false, **PD3element parent**=NULL)

  Creates a tube with outer radius r_, inner radius r2_ and length l_.

- ~**D3tube** ()

Protected Attributes

- double **r**
- double **r2**

- double **l**
- double **sectors**

4.16.1 Constructor & Destructor Documentation

4.16.1.1 D3tube::D3tube (double *r_*, double *r2_*, double *l_*, double *x*, double *y*, double *z*, double *phi_*, double *theta_*, double *psi_*, int *sectors_* = 12, bool *subdivide* = true, bool *inversenorm* = false, **PD3element** *parent* = NULL) [inline]

Creates a tube with outer radius r_, inner radius r2_ and length l_.

r2_ should be smaller than r_ for the correct norm.

The bottom of the tube is centered at (x,y,z).

The angles phi_, theta_ and psi_ can be declared in degrees and represent Euler-angles.

The Center (x,y,z) specifies the center of the Euler-rotations

The Euler-rotations occur in the following order:

phi_:.....Rotation around z-axis.....x -> x'.....y -> y'.....z -> z

theta_:.....Rotation around the rotated x-axis (x'):.....x'-> x'.....y' -> y".....z -> z'

psi_:.....Rotation around z'-axis:.....x'-> x".....y"-> y"'.....z'-> z'

For phi_ = theta_ = psi_ = 0, the bottom of the tube lies in the x,y-plane and the z-extension correspond to the length l_.

The parameter sectors_ provides an option for refinement. It is set to 12 by initialization but can also be set manually.

A value of sectors_ = 8 creates a cylinder with 8 edges on the bottom for example. sectors_ is not limited to powers of two or even numbers!! 5 or 11 works as well.

4.16.1.2 D3tube::~D3tube ( ) [inline]

4.16.2 Member Data Documentation

4.16.2.1 double D3tube::l [protected]

4.16.2.2 double D3tube::r [protected]

4.16.2.3 double D3tube::r2 [protected]

4.16.2.4 double D3tube::sectors [protected]

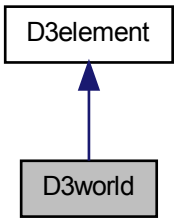
The documentation for this class was generated from the following file:

- **bem.h**

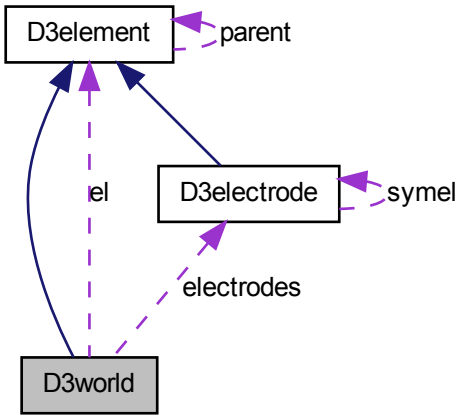
4.17 D3world Class Reference

`#include <bem.h>`

Inheritance diagram for D3world:



Collaboration diagram for D3world:



Public Member Functions

- **D3world** (unsigned long num1, unsigned long num2, unsigned long num3, unsigned long num4, unsigned long d, unsigned long m, unsigned long **y**)

- **D3world** (const char ∗_cachefilename, double **tol**=0.001, int **maxit**=32, int **numMom**=2, int **numLev**=4, double **spaceunit**=0.001, int **segmentation**=1000)
- ∼**D3world** ()
- void **insert** (**D3electrode** ∗**el**)
- void **GetListOfBaseElements** (**PD3element** ∗**el**, int ∗**electrodeIndexLimit**, int &cnt)
- void **SymmetrizeCharges** (int axis=1, double **epsilon**=0.00001, bool ignoremirror=false)
- void **solve** ()
- double **calc** (double **x**, double **y**, double **z**)

  Computes matrices that are required for the calculations of the potentials.

- void **calc** (double xxx, double **y**, double **z**, double &pot, double &feldx, double &feldy, double &feldz)
- void **calc** (double **xmin**, double **xmax**, int **nx**, double **ymin**, double **ymax**, int **ny**, double **zmin**, double **zmax**, int **nz**)

  Caching data for access with calc(x,y,z) or calc(x,y,z,&pot,&feldx,&feldy,&feldz). //x.

- void **calc** (int xxxnum, double ∗xxx, double ∗Potential)

  Calculating only p otentials. //x.

- void **calc** (int xxxnum, double ∗xxx, double ∗Potential, double ∗FieldX, double ∗FieldY, double ∗FieldZ)

  Calculating potentials and fields. //x.

- double **calc_slow** (double **x**, double **y**, double **z**)
- void **calc_slow** (double **xmin**, double **xmax**, int **nx**, double **ymin**, double **ymax**, int **ny**, double **zmin**, double **zmax**, int **nz**)
- void **calc_slow** (int xxxnum, double ∗xxx, double ∗Potential, double ∗FieldX, double ∗FieldY, double ∗FieldZ)

  Calculating potentials and fields. //x.

- void **SetScalePostCalc** (double xscale2, double yscale2, double zscale2)
- bool **IsEqualSurfaceElement** (int amountOfVertices, double eps, double ∗**x**, double ∗X)
- void **draw** ()
- void **Dcentroid** (int **shape**, double ∗pc, double ∗xcout)
- void **save** (char ∗fname)
- bool **load** (char ∗fname)
- int **cut** (int **n**, int max)
- void **savecalc** (char ∗fname)
- bool **loadcalc** (char ∗fname)
- void **AssignColors** ()
- unsigned long **update_adler32double** (unsigned long old, double ∗buf, unsigned long len, int ignorebytes=3)
- void **RefreshChecksum** ()
- void **exportGeometry** (const char ∗fname)

- void **propagateForwardVerlet** (double **x**[3], double v[3], double **h**, double qDivM=eee/mmm, bool onedim=false)
- void **propagateForwardEuler** (double **x**[3], double v[3], double **h**, double qDivM=eee/mmm)
- void **propagateForwardVerletRotSymX** (double **x**[3], double v[3], double **h**, double qDivM=eee/mmm)
- void **propagateForwardVerletRotSymY** (double **x**[3], double v[3], double **h**, double qDivM=eee/mmm)
- void **propagateForwardVerletRotSymZ** (double **x**[3], double v[3], double **h**, double qDivM=eee/mmm)
- void **calc2** (int xxxnum, double ∗xxx, double ∗Potential, double ∗FieldX, double ∗FieldY, double ∗FieldZ)

  Calculating potentials and fields. //x.

- void **calc_slow2** (int xxxnum, double ∗xxx, double ∗Potential, double ∗FieldX, double ∗FieldY, double ∗FieldZ)

  Calculating potentials and fields. //x.

Public Attributes

- bool **rangeerror**

Protected Member Functions

- void **exch** (double &a, double &b)
- void **RotMirrorSurfaceElement** (int axis, int sym, double ∗xx)
- unsigned long **update_adler32** (unsigned long old, unsigned char ∗buf, unsigned long len)

Protected Attributes

- double **xscale**
- double **yscale**
- double **zscale**
- int **totalrot**
- bool **refreshchecksum**
- int **currentel**
- int **segmentation**
- double **spaceunit**
- char **cachefilename** [2000]
- bool **docache**
- double **xmin**
- double **xmax**
- int **nx**
- double **ymin**

- double **ymax**
- int **ny**
- double **zmin**
- double **zmax**
- int **nz**
- double * **potcache**
- double * **feldxcache**
- double * **feldycache**
- double * **feldzcache**
- int **size**
- int **nlhs**
- int **nrhs**
- int **numMom**
- int **numLev**
- int **i**
- int **j**
- char * **shapechar**
- double * **x**
- double * **poten**
- double * **dbydnpotenAll**
- double * **dbydbpoten**
- double * **xcoll**
- double * **xnrm**
- double * **lhsvect**
- double * **rhsvect**
- int * **shape**
- int * **type**
- int * **dtype**
- int * **rhstype**
- int * **lhstype**
- int * **rhsindex**
- int * **lhsindex**
- int **job**
- int **fljob**
- double **error**
- double **max_diri**
- double **ave_diri**
- double **max_neum**
- double **ave_neum**
- double **cnt_diri**
- double **cnt_neum**
- double **tol**
- int **maxit**
- int **numit**
- unsigned long **checksum**
- unsigned long **checksumcalc**

- double * **f**
- double * **dfdnAll**
- int * **electrodeIndexLimit**
- **PD3element** * **el**
- **D3electrode** ** **electrodes**
- int **n**
- int **amountOfElectrodes**

4.17.1 Constructor & Destructor Documentation

4.17.1.1 D3world::D3world (unsigned long *num1,* unsigned long *num2,* unsigned long *num3,* unsigned long *num4,* unsigned long *d,* unsigned long *m,* unsigned long *y*)

4.17.1.2 D3world::D3world (const char * *_cachefilename,* double *tol* = 0.001, int *maxit* = 32, int *numMom* = 2, int *numLev* = 4, double *spaceunit* = 0.001, int *segmentation* = 1000)

aus der cpp: num is registration key, d is day, m is month, y is year

???

4.17.1.3 D3world::~D3world ( )

4.17.2 Member Function Documentation

4.17.2.1 void D3world::AssignColors ( )

4.17.2.2 double D3world::calc (double *x,* double *y,* double *z*)

Computes matrices that are required for the calculations of the potentials.

4.17.2.3 void D3world::calc (int *xxxnum,* double * *xxx,* double * *Potential,* double * *FieldX,* double * *FieldY,* double * *FieldZ*)

Calculating potentials and fields. //x.

4.17.2.4 void D3world::calc (int *xxxnum,* double * *xxx,* double * *Potential*)

Calculating only p otentials. //x.

4.17.2.5 void D3world::calc (double *xxx,* double *y,* double *z,* double & *pot,* double & *feldx,* double & *feldy,* double & *feldz*)

4.17.2.6 void D3world::calc (double *xmin,* double *xmax,* int *nx,* double *ymin,* double *ymax,* int *ny,* double *zmin,* double *zmax,* int *nz*)

Caching data for access with calc(x,y,z) or calc(x,y,z,&pot,&feldx,&feldy,&feldz). //x.

4.17.2.7 void D3world::calc2 (int *xxxnum,* double * *xxx,* double * *Potential,* double * *FieldX,* double * *FieldY,* double * *FieldZ*)

Calculating potentials and fields. //x.

4.17.2.8 double D3world::calc_slow (double *x,* double *y,* double *z*)

4.17.2.9 void D3world::calc_slow (double *xmin,* double *xmax,* int *nx,* double *ymin,* double *ymax,* int *ny,* double *zmin,* double *zmax,* int *nz*)

4.17.2.10 void D3world::calc_slow (int *xxxnum,* double * *xxx,* double * *Potential,* double * *FieldX,* double * *FieldY,* double * *FieldZ*)

Calculating potentials and fields. //x.

4.17.2.11 void D3world::calc_slow2 (int *xxxnum,* double * *xxx,* double * *Potential,* double * *FieldX,* double * *FieldY,* double * *FieldZ*)

Calculating potentials and fields. //x.

4.17.2.12 int D3world::cut (int *n*, int *max*)

4.17.2.13 void D3world::Dcentroid (int *shape*, double * *pc*, double * *xcout*)

4.17.2.14 void D3world::draw ( )

4.17.2.15 void D3world::exch (double & *a*, double & *b*) [protected]

4.17.2.16 void D3world::exportGeometry (const char * *fname*)

4.17.2.17 void D3world::GetListOfBaseElements (PD3element * *el*, int * *electrodeIndexLimit*, int & *cnt*)

4.17.2.18 void D3world::insert (D3electrode * *el*)

4.17.2.19 bool D3world::IsEqualSurfaceElement (int *amountOfVertices*, double *eps*, double * *x*, double * *X*)

4.17.2.20 bool D3world::load (char * *fname*)

4.17.2.21 bool D3world::loadcalc (char * *fname*)

4.17.2.22 void D3world::propagateForwardEuler (double *x[3]*, double *v[3]*, double *h*, double *qDivM* = eee/mmm)

4.17.2.23 void D3world::propagateForwardVerlet (double *x[3]*, double *v[3]*, double *h*, double *qDivM* = eee/mmm, bool *onedim* = false)

Propagates particle at position x[3] one timestep h forward.

Parameters

x[3]	contains x,y,z position at current timestep. After call contains the position at next timestep.
v[3]	contains x,y,z velocity at current timestep.
h	is timestep in seconds.
qDivM	is charge of particle divided by mass.

4.17.2.24 void D3world::propagateForwardVerletRotSymX (double *x[3],* double *v[3],* double *h,* double *qDivM =* eee/mmm)

4.17.2.25 void D3world::propagateForwardVerletRotSymY (double *x[3],* double *v[3],* double *h,* double *qDivM =* eee/mmm)

4.17.2.26 void D3world::propagateForwardVerletRotSymZ (double *x[3],* double *v[3],* double *h,* double *qDivM =* eee/mmm)

4.17.2.27 void D3world::RefreshChecksum ( )

4.17.2.28 void D3world::RotMirrorSurfaceElement (int *axis,* int *sym,* double * *xx*) [protected]

4.17.2.29 void D3world::save (char * *fname*)

4.17.2.30 void D3world::savecalc (char * *fname*)

4.17.2.31 void D3world::SetScalePostCalc (double *xscale2,* double *yscale2,* double *zscale2*)

4.17.2.32 void D3world::solve ( )

4.17.2.33 void D3world::SymmetrizeCharges (int *axis =* 1, double *epsilon =* 0.00001, bool *ignoremirror =* false)

4.17.2.34 unsigned long D3world::update_adler32 (unsigned long *old,* unsigned char * *buf,* unsigned long *len*) [protected]

4.17.2.35 unsigned long D3world::update_adler32double (unsigned long *old,* double * *buf,* unsigned long *len,* int *ignorebytes =* 3)

4.17.3 Member Data Documentation

4.17.3.1 int **D3world::amountOfElectrodes** [protected]

4.17.3.2 double **D3world::ave_diri** [protected]

4.17.3.3 double **D3world::ave_neum** [protected]

4.17.3.4 char **D3world::cachefilename[2000]** [protected]

4.17.3.5 unsigned long **D3world::checksum** [protected]

4.17.3.6 unsigned long **D3world::checksumcalc** [protected]

4.17.3.7 double **D3world::cnt_diri** [protected]

4.17.3.8 double **D3world::cnt_neum** [protected]

4.17.3.9 int **D3world::currentel** [protected]

4.17.3.10 double* **D3world::dbydbpoten** [protected]

4.17.3.11 double* **D3world::dbydnpotenAll** [protected]

4.17.3.12 double* **D3world::dfdnAll** [protected]

4.17.3.13 bool **D3world::docache** [protected]

4.17.3.58 double* D3world::xcoll [protected]

4.17.3.59 double D3world::xmax [protected]

4.17.3.60 double D3world::xmin [protected]

4.17.3.61 double* D3world::xnrm [protected]

4.17.3.62 double D3world::xscale [protected]

4.17.3.63 double D3world::ymax [protected]

4.17.3.64 double D3world::ymin [protected]

4.17.3.65 double D3world::yscale [protected]

4.17.3.66 double D3world::zmax [protected]

4.17.3.67 double D3world::zmin [protected]

4.17.3.68 double D3world::zscale [protected]

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

- **bem.h**

4.18 xv Struct Reference

Public Attributes

- double **x**
- double **y**
- double **z**
- double **vx**
- double **vy**
- double **vz**

4.18.1 Member Data Documentation

4.18.1.1 double xv::vx

4.18.1.2 double xv::vy

4.18.1.3 double xv::vz

4.18.1.4 double xv::x

4.18.1.5 double xv::y

4.18.1.6 double xv::z

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

- **linearPaulTrap_edit.cxx**
- **ringTrap_edit.cxx**

Chapter 5

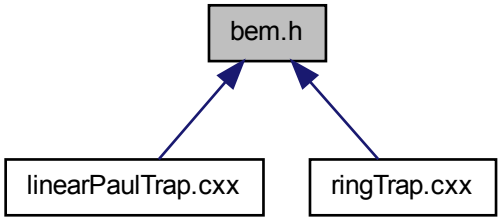
File Documentation

5.1 bem.h File Reference

```
#include <complex>
#include <TGeoMatrix.h>
#include <TGeoManager.h>
#include <TGeoArb8.h>
#include "dl_creationadapter.h"
#include "lapack.h"
#include <float.h>
#include <fftw3.h>
#include <gsl/gsl_sf_bessel.h>
#include <gsl/gsl_sf_ellint.h>
#include <math.h>
#include <gsl/gsl_integration.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_linalg.h>
#include "stdafx.h"
#include <vector>
#include <list>
```

Include dependency graph for bem.h:

This graph shows which files directly or indirectly include this file:



Classes

- class **calcD3triangle**
- class **D3element**
- class **D3electrode**
- class **D3sortedelement**
- class **D3sorter**
- class **D3ImportedElectrodes**
- class **D3triangle**
- class **D3rectangle**
- class **D3box**
- class **D3icosahedron**
- class **D3sphere**
- class **D3disk**
- class **D3cylinder**
- class **D3tube**
- class **D3world**
- class **D3slowworld**
- class **D3thicktriangle**

Defines

- #define **BEMREVISION** 2
- #define **sqr**(x) ((x)∗(x))
- #define **size_** 3
- #define **eee** 1.602e-19
- #define **mmm** (40.078∗1.66e-27)

Typedefs

- typedef double **trianglefn** (double, double, double, double, double, double)
- typedef **D3element** ∗ **PD3element**
- typedef list< **PD3element** > **PD3elementList**

Functions

- double **testfn** (double x, double y, double z)
- void **q** ()

Variables

- const double **pi** = 3.14159265358979323846264338327950288419716939937510582097494459230781640628620
- **calcD3triangle calc**

5.1.1 Define Documentation

5.1.1.1 #define BEMREVISION 2

5.1.1.2 #define eee 1.602e-19

5.1.1.3 #define mmm (40.078∗1.66e-27)

5.1.1.4 #define size_ 3

5.1.1.5 #define sqr(*x*) ((x)∗(x))

5.1.2 Typedef Documentation

5.1.2.1 typedef D3element∗ PD3element

5.1.2.2 typedef list<PD3element> PD3elementList

5.1.2.3 typedef double trianglefn(double, double, double, double, double, double)

5.1.3 Function Documentation

5.1.3.1 void q ( )

5.1.3.2 double testfn (double *x,* double *y,* double *z*)

5.1.4 Variable Documentation

5.1.4.1 calcD3triangle calc

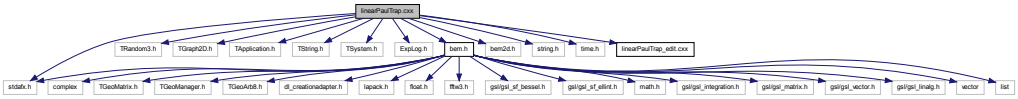
5.1.4.2 const double pi = 3.14159265358979323846264338327950288419716939937510582097494459230781640628620 90

5.2 linearPaulTrap.cxx File Reference

```
#include "stdafx.h"
#include "TRandom3.h"
#include "TGraph2D.h"
#include <TApplication.h>
#include <TString.h>
#include <TSystem.h>
#include "ExpLog.h"
#include "bem.h"
#include "bem2d.h"
```

```
#include <string.h>
#include <time.h>
#include "linearPaulTrap_edit.cxx"
```

Include dependency graph for linearPaulTrap.cxx:



Defines

- #define **ELCNT** 5

Functions

- int **main** (int argc, char ∗argv[ ])

Variables

- **D3world** ∗ **wr**
- **D3electrode** ∗ **eldca** [ELCNT]
- **D3electrode** ∗ **eldcb** [ELCNT]
- **D3electrode** ∗ **elrfa**
- **D3electrode** ∗ **elrfb**

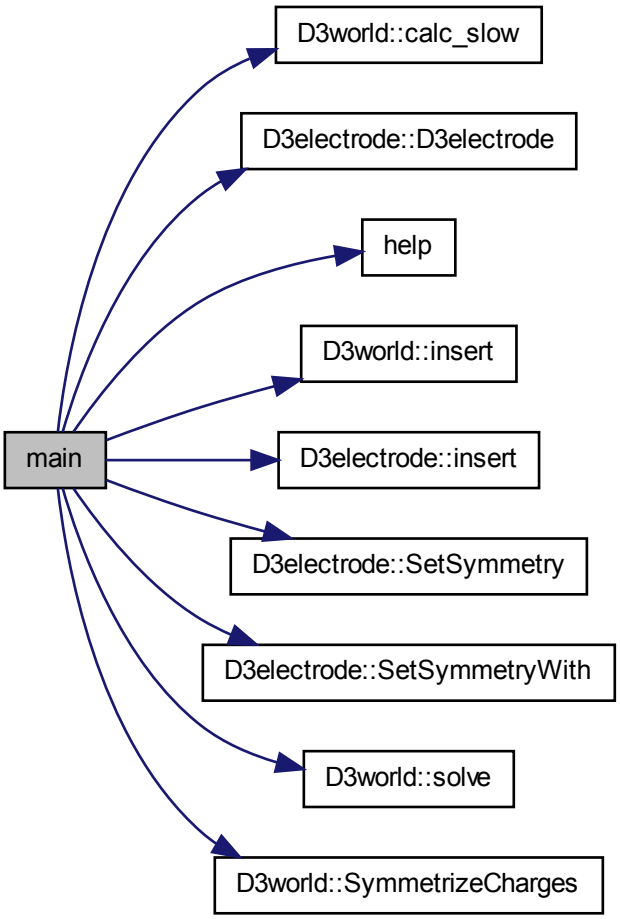
5.2.1 Define Documentation

5.2.1.1 #define ELCNT 5

5.2.2 Function Documentation

5.2.2.1 int main (int *argc*, char * *argv[]*)

Here is the call graph for this function:



5.2.3 Variable Documentation

5.2.3.1 D3electrode* eldca[ELCNT]

5.2.3.2 D3electrode * eldcb[ELCNT]

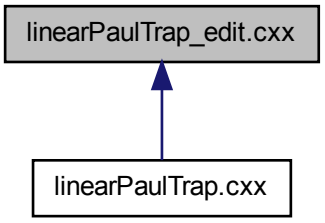
5.2.3.3 D3electrode * elrfa

5.2.3.4 D3electrode * elrfb

5.2.3.5 D3world* wr

5.3 linearPaulTrap_edit.cxx File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct **xv**

Defines

- #define **PI** 3.141592653589793

Functions

- void **help** ()
- void **free** ()
- void **setTrappingVoltage** (double dcac=0, double dcbc=0, double rfac=0, double rfbc=0, double urf=0)

- void **setZero** (double dcac=0, double dcbc=0, double rfac=0, double rfbc=0, double urf=0)
- double **calcit** (double x, double y, double z)
- void **ExportAxialPot** (double zstart=-15, double zstop=15, double zcnt=3000)
- void **PlotPot** (double x1=0, double y1=0, double z1=-5, double x2=0, double y2=0, double z2=5, int num=200, bool plotfield=true)
- void **PlotPseudo** ()
- void **Trajectory** (double rfac=0., double rfbc=0., double dcac=0., double dcbc=0., double x=0.00002, double y=0.00002, double z=0.0, double vx=0, double vy=0, double vz=0, double stoptime=1000./12e6, int steps=500000, double RFVoltage=400./2., double RFfreq=12155000, bool verlet=false)

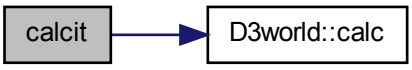
5.3.1 Define Documentation

5.3.1.1 #define PI 3.141592653589793

5.3.2 Function Documentation

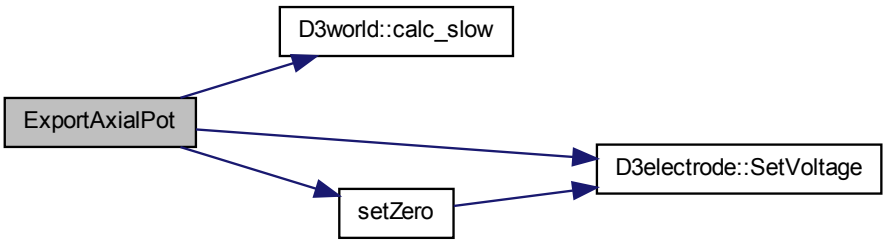
5.3.2.1 double calcit (double *x*, double *y*, double *z*)

Here is the call graph for this function:



5.3.2.2 void ExportAxialPot (double *zstart* = −15, double *zstop* = 15, double *zcnt* = 3000)

Here is the call graph for this function:

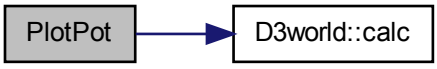


5.3.2.3 void free ( )

5.3.2.4 void help ( )

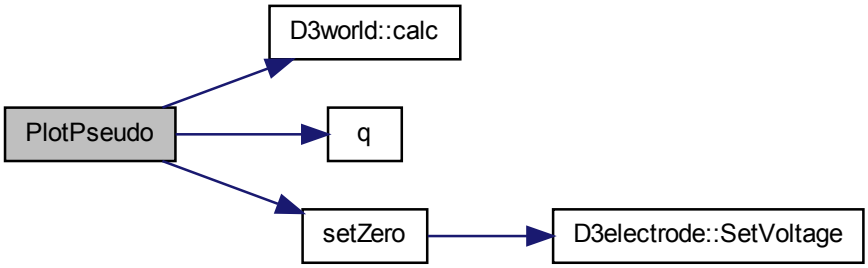
5.3.2.5 void PlotPot (double *x1* = 0, double *y1* = 0, double *z1* = −5, double *x2* = 0, double *y2* = 0, double *z2* = 5, int *num* = 200, bool *plotfield* = true)

Here is the call graph for this function:



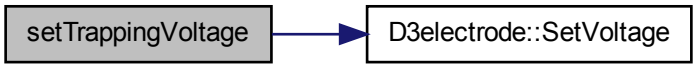
5.3.2.6 void PlotPseudo ( )

Here is the call graph for this function:



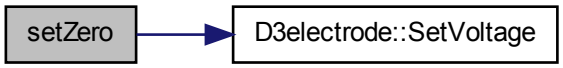
5.3.2.7 void setTrappingVoltage (double *dcac* = 0, double *dcbc* = 0, double *rfac* = 0, double *rfbc* = 0, double *urf* = 0)

Here is the call graph for this function:



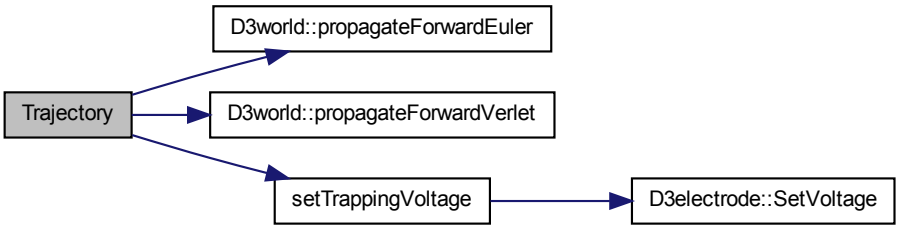
5.3.2.8 void setZero (double *dcac* = 0, double *dcbc* = 0, double *rfac* = 0, double *rfbc* = 0, double *urf* = 0)

Here is the call graph for this function:



5.3.2.9 void Trajectory (double *rfac* = 0., double *rfbc* = 0., double *dcac* = 0., double *dcbc* = 0., double *x* = 0.00002, double *y* = 0.00002, double *z* = 0.0, double *vx* = 0, double *vy* = 0, double *vz* = 0, double *stoptime* = 1000./12e6, int *steps* = 500000, double *RFVoltage* = 400./2., double *RFfreq* = 12155000, bool *verlet* = false)

Here is the call graph for this function:

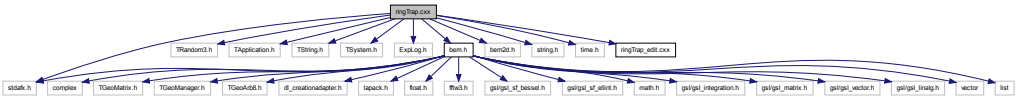


5.4 ringTrap.cxx File Reference

```
#include "stdafx.h"
#include "TRandom3.h"
#include <TApplication.h>
#include <TString.h>
#include <TSystem.h>
```

```
#include "ExpLog.h"
#include "bem.h"
#include "bem2d.h"
#include <string.h>
#include <time.h>
#include "ringTrap_edit.cxx"
```

Include dependency graph for ringTrap.cxx:



Defines

- #define **ELCNT** 5

Functions

- int **main** (int argc, char ∗argv[ ])

Variables

- **D3world** ∗ **wr**
- **D3electrode** ∗ **eldrf**
- **D3electrode** ∗ **eldc1**
- **D3electrode** ∗ **eldc2**

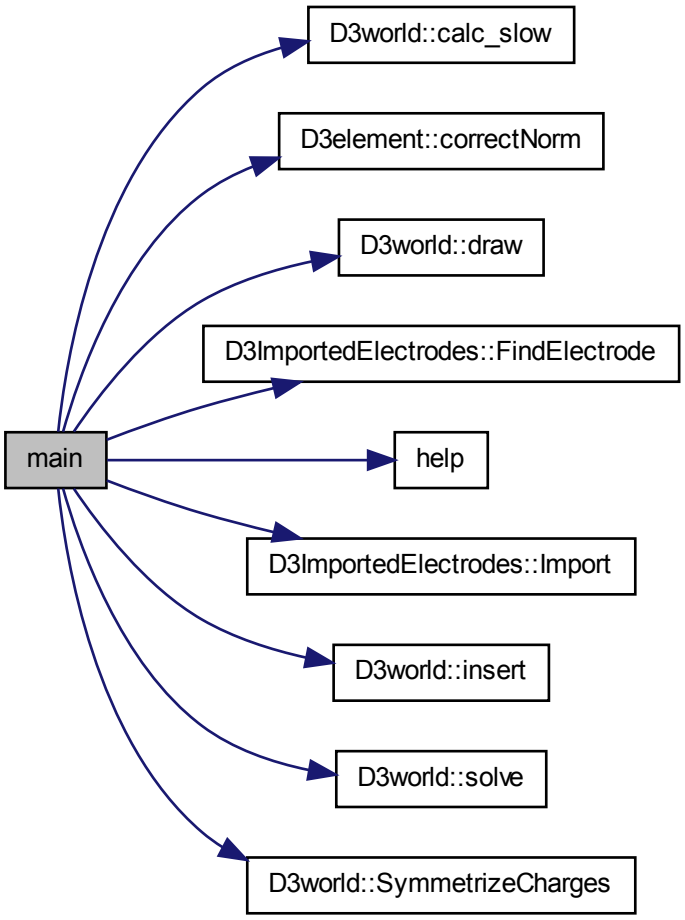
5.4.1 Define Documentation

5.4.1.1 #define ELCNT 5

5.4.2 Function Documentation

5.4.2.1 int main (int *argc*, char * *argv[]*)

Here is the call graph for this function:



5.4.3 Variable Documentation

5.4.3.1 D3electrode * eldc1

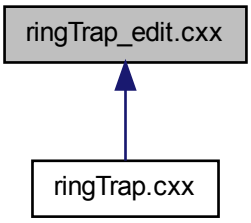
5.4.3.2 D3electrode * eldc2

5.4.3.3 D3electrode* eldrf

5.4.3.4 D3world* wr

5.5 ringTrap_edit.cxx File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct **xv**

Defines

- #define **PI** 3.141592653589793

Functions

- void **help** ()
- void **free** ()
- void **setTrappingVoltage** (double dcac=0, double dcbc=0, double rfac=0, double rfbc=0, double urf=0)
- void **setZero** (double dcac=0, double dcbc=0, double rfac=0, double rfbc=0, double urf=0)
- double **calcit** (double x, double y, double z)

- void **ExportAxialPot** (double zstart=-15, double zstop=15, double zcnt=3000)
- void **PlotPot** (double x1=0, double y1=0, double z1=-5, double x2=0, double y2=0, double z2=5, int num=200, bool plotfield=true)
- void **Trajectory** (double rfac=0., double rfbc=0., double dcac=0., double dcbc=0., double x=0, double y=0, double z=0.0, double vx=0, double vy=0, double vz=0, double stoptime=1000./12e6, int steps=500000, double RFVoltage=400./2., double RFfreq=12155000)

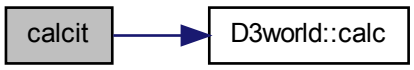
5.5.1 Define Documentation

5.5.1.1 #define PI 3.141592653589793

5.5.2 Function Documentation

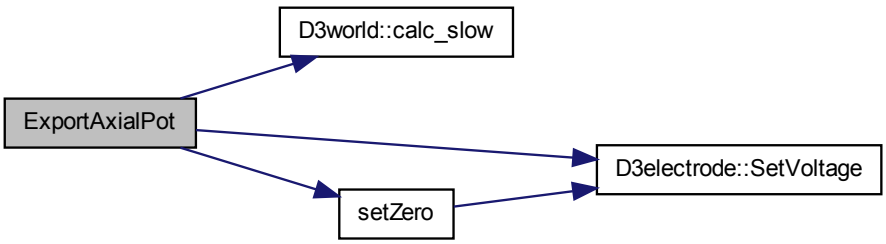
5.5.2.1 double calcit (double *x*, double *y*, double *z*)

Here is the call graph for this function:



5.5.2.2 void ExportAxialPot (double *zstart* = -15, double *zstop* = 15, double *zcnt* = 3000)

Here is the call graph for this function:

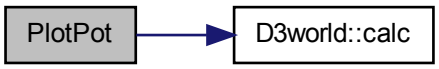


5.5.2.3 void free ( )

5.5.2.4 void help ( )

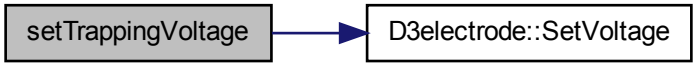
5.5.2.5 void PlotPot (double *x1* = 0, double *y1* = 0, double *z1* = -5, double *x2* = 0, double *y2* = 0, double *z2* = 5, int *num* = 200, bool *plotfield* = true)

Here is the call graph for this function:



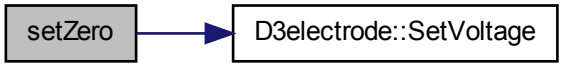
5.5.2.6 void setTrappingVoltage (double *dcac* = 0, double *dcbc* = 0, double *rfac* = 0, double *rfbc* = 0, double *urf* = 0)

Here is the call graph for this function:



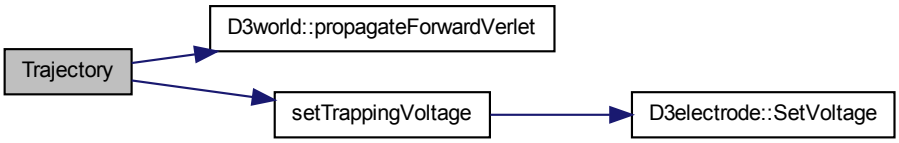
5.5.2.7 void setZero (double *dcac* = 0, double *dcbc* = 0, double *rfac* = 0, double *rfbc* = 0, double *urf* = 0)

Here is the call graph for this function:



5.5.2.8 void Trajectory (double *rfac* = 0., double *rfbc* = 0., double *dcac* = 0., double *dcbc* = 0., double *x* = 0, double *y* = 0, double *z* = 0.0, double *vx* = 0, double *vy* = 0, double *vz* = 0, double *stoptime* = 1000./12e6, int *steps* = 500000, double *RFVoltage* = 400./2., double *RFfreq* = 12155000)

Here is the call graph for this function:



Index