ReactorFsim Reference Manual
0_0_5

Generated by Doxygen 1.2.14

Mon Feb 21 16:11:18 2005

Contents

1 ReactorFsim Compound Index **1**

1.1 ReactorFsim Compound List 1

2 ReactorFsim File Index **3**

2.1 ReactorFsim File List 3

3 ReactorFsim Class Documentation **5**

3.1 MuonPropagator Class Reference 5

3.2 MyBiSource Class Reference 17

3.3 MyCfSource Class Reference 22

3.4 MyHeSource Class Reference 30

3.5 MyLiSource Class Reference 33

3.6 MyTlSource Class Reference 36

3.7 ReactorDetector Class Reference 41

3.8 ReactorEvent Class Reference 45

3.9 ReactorNtuple Class Reference 69

3.10 ReactorTree Class Reference 75

3.11 ReactorTrigger Class Reference 76

4 ReactorFsim File Documentation **77**

4.1 cf252_readme.txt File Reference 77

4.2 DetectorDefaults.hh File Reference 78

4.3 event_readme.txt File Reference 79

4.4 GammaComptonRange.cpp File Reference 80

4.5 MuonPropagator.cpp File Reference 81

4.6 MyBiSource.cpp File Reference 82

4.7 MyBiSource.hh File Reference 83

4.8 MyCfSource.cpp File Reference 84

4.9 MyCfSource.hh File Reference 85

4.10 MyHeSource.cpp File Reference . . . 86
4.11 MyHeSource.hh File Reference . . . 87
4.12 MyLiSource.cpp File Reference . . . 88
4.13 MyLiSource.hh File Reference . . . 89
4.14 MyTlSource.cpp File Reference . . . 90
4.15 MyTlSource.hh File Reference . . . 91
4.16 ReactorConstants.hh File Reference . . . 92
4.17 ReactorDetector.cpp File Reference . . . 93
4.18 ReactorDetector.hh File Reference . . . 94
4.19 ReactorEvent.cpp File Reference . . . 95
4.20 ReactorEvent.hh File Reference . . . 96
4.21 ReactorFortran.hh File Reference . . . 97
4.22 ReactorFsim.cpp File Reference . . . 98
4.23 ReactorNtuple.cpp File Reference . . . 104
4.24 ReactorNtuple.hh File Reference . . . 105
4.25 ReactorTree.cpp File Reference . . . 106
4.26 ReactorTree.hh File Reference . . . 107
4.27 ReactorTrigger.cpp File Reference . . . 108
4.28 ReactorTrigger.hh File Reference . . . 109
4.29 ReactorXC.hh File Reference . . . 110
4.30 RMCEvent.hh File Reference . . . 112
4.31 RMCPMT.hh File Reference . . . 114
4.32 RMCSecondary.hh File Reference . . . 115

Chapter 1

ReactorFsim Compound Index

1.1 ReactorFsim Compound List

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

MuonPropagator (MuonPropogator handles muon transit for **ReactorEvent** (p. 45)) 5
MyBiSource (MyBiSource generates photons from radioactive Thallium decays) . . . 17
MyCfSource (MyCfSource generates prompt photons and delayed neutrons from radioactive Californium decays) 22
MyHeSource (MyHeSource generates delayed neutrons from radioactive He decays) . 30
MyLiSource (MyLiSource generates delayed neutrons from radioactive Li decays) . . . 33
MyTlSource (MyTlSource generates photons from radioactive Thallium decays) . . . 36
ReactorDetector (ReactorDetector handles the detector implementation in the ReactorFsim simulation) 41
ReactorEvent (ReactorEvent handles all of the event generation in the ReactorFsim simulation) 45
ReactorNtuple (ReactorNtuple creates and fills the ntuple for the ReactorFsim simulation) 69
ReactorTree (ReactorTreee creates and fills the root tree for the ReactorFsim simulation) 75
ReactorTrigger (ReactorTrigger performs a simple trigger simulation for the ReactorFsim simulation) 76

Chapter 2

ReactorFsim File Index

2.1 ReactorFsim File List

Here is a list of all documented files with brief descriptions:

DetectorDefaults.hh (DetectorDefaults header file) 78
GammaCompton.cpp . . . . ??
GammaComptonRange.cpp (GammaComptonRange C++ file) 80
InverseBeta.cpp . . . . ??
MuonPropagator.cpp (**MuonPropagator** (p. 5) C++ file) 81
MuonPropagator.hh . . . . ??
MyBiSource.cpp (**MyBiSource** (p. 17) C++ file) 82
MyBiSource.hh (**MyBiSource** (p. 17) header file) 83
MyCfSource.cpp (**MyCfSource** (p. 22) C++ file) 84
MyCfSource.hh (**MyCfSource** (p. 22) header file) 85
MyGdDecayModel.cpp . . . . ??
MyHeSource.cpp (**MyHeSource** (p. 30) C++ file) 86
MyHeSource.hh (**MyHeSource** (p. 30) header file) 87
MyLiSource.cpp (**MyLiSource** (p. 33) C++ file) 88
MyLiSource.hh (**MyLiSource** (p. 33) header file) 89
MyTlSource.cpp (**MyTlSource** (p. 36) C++ file) 90
MyTlSource.hh (**MyTlSource** (p. 36) header file) 91
NeutronPropagator.cpp . . . . ??
PositronRange.cpp . . . . ??
ReactorConstants.hh (ReactorConstants header file) 92
ReactorDetector.cpp (**ReactorDetector** (p. 41) C++ file) 93
ReactorDetector.hh (**ReactorDetector** (p. 41) header file) 94
ReactorEvent.cpp (**ReactorEvent** (p. 45) C++ file) 95
ReactorEvent.hh (**ReactorEvent** (p. 45) header file) 96
ReactorFortran.hh (Defines the Fortran prototypes for calling from C++) 97
ReactorFsim.cpp (ReactorFsim simulation for MWtheta13) 98
ReactorNeutrons.cpp . . . . ??
ReactorNeutrons.hh . . . . ??
ReactorNtuple.cpp (**ReactorNtuple** (p. 69) C++ file) 104
ReactorNtuple.hh (**ReactorNtuple** (p. 69) header file) 105
ReactorScint.cpp . . . . ??
ReactorScint.hh . . . . ??
ReactorTree.cpp (**ReactorTree** (p. 75) C++ file) 106

ReactorTree.hh (**ReactorTree** (p. 75) header file) 107
ReactorTrigger.cpp (**ReactorTrigger** (p. 76) C++ file) 108
ReactorTrigger.hh (**ReactorTrigger** (p. 76) header file) 109
ReactorXC.hh (ReactorXC defines several useful functions for ReactorFsim) 110
RMCEvent.cpp . . . . **??**
RMCEvent.hh (RMCEvent header file) 112
RMCMacro.cpp . . . . **??**
RMCPMT.cpp . . . . **??**
RMCPMT.hh (RMCPMT header file) 114
RMCSecondary.cpp . . . . **??**
RMCSecondary.hh (RMCSecondary header file) 115
RMPflux.cpp . . . . **??**
test.cpp . . . . **??**
U235flux.cpp . . . . **??**

Chapter 3

ReactorFsim Class Documentation

3.1 MuonPropagator Class Reference

MuonPropogator handles muon transit for **ReactorEvent** (p. 45).

`#include <MuonPropagator.hh>`

Public Methods

- **MuonPropagator** (double newRmaxgen=RMAXGEN, double newDepth=450., double newEnergy=0., double newCosTheta=1.)

  MuonPropagator constructor.

- **MuonPropagator** (int test=0, double newRmaxgen=RMAXGEN, double newEnergy=0., double newX=0., double newY=0., double newZ=0., double newVx=0., double newVy=0., double newVz=0.)
- **~MuonPropagator** ()

  MuonPropagator destructor.

- **MuonPropagator** (const MuonPropagator &Muon)

  MuonPropagator copy constructor.

- MuonPropagator & **operator=** (const MuonPropagator &rhs)

  MuonPropagator overloaded = operator.

- double * **Intersection** (double, double, double, double, double, double, double, double, double, double)
- double * **Rotate** (char, double, double, double, double)
- double **RadCorr** (double, double)
- double **GetMuonE** () const
- double **GetMuonCosTheta** () const
- double **GetMuonPhi** () const
- int **GetNumNeutron** () const

  Returns the neutron multiplicity.

- double **GetNeutronKE** (int &n) const

- double **GetNeutronTime** (int &n) const
- double *const **GetNeutronVertexXYZ** ()
- int **GetNumElectron** () const

  Returns the electron multiplicity.

- double **GetElectronKE** (int &n) const
- double **GetElectronTime** (int &n) const
- double *const **GetElectronVertexXYZ** ()
- int **GetNumGamma** () const

  Returns the number of photons.

- double **GetGammaKE** (int &n) const
- double **GetGammaTime** (int &n) const
- double *const **GetGammaVertexXYZ** ()
- double **GetWeight** () const

Static Public Methods

- float **IsotopeDistance** (const float &r)

  The lateral activation profile of muons passing through the detector to produce radioactive isotopes, which describes the probability that an isotope is produced at a given distance r (in cm) perpendicular to the muon track.

3.1.1 Detailed Description

MuonPropogator handles muon transit for **ReactorEvent** (p. 45).

It returns particle information from the muon itself, as well as any byproducts of the muon shower, including neutrons and electrons from 9Li/8He beta decay.

Definition at line 17 of file MuonPropagator.hh.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 MuonPropagator::MuonPropagator (double *newRmaxgen* = RMAXGEN, double *newDepth* = 450., double *newEnergy* = 0., double *newCosTheta* = 1.)

MuonPropagator constructor.

Generates a muon from a given energy and zenith angle. The muon starts at a random point above the detector and at a random azimuthal angle to decide the path of the muon track. Energy from the muon inside the detector is created incrementally along the muon tragectory as optical photons. One heavy isotope 9Li or 8He is produced per muon from the muon spallation and forced to beta decay to create and electron and neutron. The events are weighted by the 9Li/8He production cross sections and the decay branching ratio of the 9Li/8He beta decay. The neutron, electron, and photon energies, positions and times are saved.

Definition at line 402 of file MuonPropagator.cpp.

References MyHeSource::GetHeBetaEnergy, MyHeSource::GetHeDecayTime, MyHeSource::GetHeNeutronEnergy, MyLiSource::GetLiBetaEnergy, MyLiSource::GetLiDecayTime, MyLiSource::GetLiNeutronEnergy, and IsotopeDistance.

```
                                                {

Rmaxgen = newRmaxgen;
Depth = newDepth;
energy = newEnergy;
cosg = newCosTheta;

//cout << "MuonPropagator::MuonPropagator(" << Rmaxgen << ","
//     << Depth << "," << energy << "," << cosg << ")" << endl;

static double re  = RELECTRON;
static double me  = MELECTRON;
static double mmu = MMUON;
static double Na  = NAVOGADRO;
//
// For electron density
//
static double nel[2];
static double A[2] = {A0,A1};
static double Z[2] = {Z0,Z1};
static double f[2] = {f0,f1};
static double I[2] = {I0,I1};
//
static double ncar;
//
// calculate the fluxes and densities
//
bool static first = true;
double flux;
int static muonstop;
float static fspace[200];
//
if(first){

  // flux information
  if(Depth == 300.)
    flux = 0.374;  // per m**2 per sec
  else if(Depth == 450.)
    flux = 0.160;  // per m**2 per sec
  else if(Depth == 500.)
    flux = 0.145;  // per m**2 per sec
  else{
    cout << "  ERROR: no muon flux for depth of " << Depth << " mwe" << endl;
    return;
  }

  // electron density
  for(int i=0;i<2;i++) nel[i] = Na*density*f[i]*Z[i]/A[i];
  //cout << " nel " << nel[0]+nel[1] << endl;

  // 12C atom density
  ncar = Na*density*f[1]/A[1];
  //cout << " ncar " << ncar << endl;

  // prepare the lateral activation profile
  float xlo = 0.;
  float xhi = 50.;
  funlxp_(IsotopeDistance,fspace,xlo,xhi);

  Weight = 0.;
  muonstop = 0;

  first = false;
}

```

```
/* pick a starting point: define a square above the detector
   through which the muons must pass.  The square is centered
   on the z-axis at Rmaxgen and the area is A = (Rmaxgen+delta)**2.
   Randomly pick a point on the square to start the muon and use
   cos(theta) and a randomly selected azimuthal angle to decide
   how the muon hits the detector.  Only keep events where the
   muon hits the scint.
*/
int inter = 0;
double r[6],t[3];
double x[2],y[2],z[2];
double delta = Rmaxgen*20.;
bool passed = false;
//
while(!passed){

  const int len=3;
  float rv[len];
  ranlux_(rv,len);
  r[0] = (2*rv[0]-1)*delta;
  r[1] = (2*rv[1]-1)*delta;
  r[2] = Rmaxgen;
  phig = 2*PI*rv[2];

  // pick the direction
  t[0] = cos(phig)*sqrt(1-cosg*cosg);
  t[1] = sin(phig)*sqrt(1-cosg*cosg);
  t[2] = cosg;
  //for(int i=0;i<3;i++) cout << "t["<<i<<"] " << t[i] << " ";
  //cout << endl;

  // find the far end of the line a unit distance
  // from (x,y,z) along the t direction
  for(int i=0;i<3;i++) r[i+3] = r[i]-t[i];
  //
  //for(int i=0;i<6;i++) cout << "r["<<i<<"] " << r[i] << " ";
  //cout << endl;

  // solve for intersections with the scint region
  double p[7];
  double* ptr = p;
  int num;

  ptr = Intersection(r[0], r[1], r[2],  // first end of line
                     r[3], r[4], r[5],  // second end of line
                     0, 0, 0, Rmaxgen); // detector centered at 0

  num = (int)*ptr++;
  if(num>1){

    inter = 2;
    for(int i=0; i<inter; i++){
      x[i] = *ptr++;
      y[i] = *ptr++;
      z[i] = *ptr++;
    }
    passed = true;
  }
}
/*
for(int i=0; i<inter; i++){
  cout << " X-coordinate of point " << i << " : " << x[i] << endl;
  cout << " Y-coordinate of point " << i << " : " << y[i] << endl;
  cout << " Z-coordinate of point " << i << " : " << z[i] << endl;
}
```

```
*/
// path length
double L = sqrt((x[1]-x[0])*(x[1]-x[0])+
                (y[1]-y[0])*(y[1]-y[0])+
                (z[1]-z[0])*(z[1]-z[0]));
//cout << " path length " << L << endl;

/* iterate through the muon energy loss dE/dx in dx cm steps,
   making a gamma at the center of the dx cm segment of the
   line along the muon path equal to the energy lost in that
   segment; always check that the muon is still in the
   detector and has not stopped
*/
double step=5.0;
double Emin=0.01;
//
double range = 0.;
double time = 0.;
//
// define energy and position arrays
//
int maxgammafromisotope = 5;
int maxgamma = int(Rmaxgen*2/step)+maxgammafromisotope;
double* egamma = new double[maxgamma];
double* rgamma = new double[maxgamma*3];
double* tgamma = new double[maxgamma];
//
double* egptr=egamma;
double* rgptr=rgamma;
double* tgptr=tgamma;
//
// Begin energy loss loop
//
double dE=0;
double eloss;
int ngamma = 0;
//
bool done = false;
while(!done){

  // zero the energy loss for this segment
  eloss = 0;
  //
  // check that the muon hasn't stopped
  //
  if(energy-dE > Emin){
    //
    // check that we aren't leaving the detector
    //
    if(range+step < L){
      //
      // iterate range and time
      //
      double gamma = 1+(energy-dE)/mmu;
      double beta = sqrt(1-1/gamma/gamma);
      range+=step;
      time+=step/beta/c;
      //
      // increment energy loss
      //
      double te = gamma-1;
      double tc = Emin/mmu;
      double tmax = te;
      double tup = tc;
      if(tc>tmax)tup=tmax;
```

```
      //
      for (int k=0;k<2;k++){
        eloss+=2*PI*re*re*me*nel[k]*step/beta/beta*
          (log(2*(gamma+1)/(I[k]*I[k]/me/me))
           +RadCorr(te,tup));
      }
      //
      // add the energy loss to dE and set the "photon" energy
      //
      if(dE+eloss > energy){
        *egptr++=energy-dE; // the muon stops, limit dE and egamma
        dE=energy;
      }
      else{
        *egptr++=eloss; // the muon is still travelling
        dE+=eloss;
      }
      //
      // put the photon at the center of the segment
      //
      *rgptr++ = x[1] - (range-(step/2))*t[0];
      *rgptr++ = y[1] - (range-(step/2))*t[1];
      *rgptr++ = z[1] - (range-(step/2))*t[2];
      //
      *tgptr++ = time;
      //
      // count the photon
      ngamma++;
    }
    else{
      /* we have reached the end of the detector; put one final
         gamma to cover the last remaining segment of energy loss
      */
      double laststep = L-range;
      //
      double gamma = 1+(energy-dE)/mmu;
      double beta = sqrt(1-1/gamma/gamma);
      range+=laststep;
      time+=laststep/beta/c;
      //
      // increment energy loss
      //
      double te = gamma-1;
      double tc = Emin/mmu;
      double tmax = te;
      double tup = tc;
      if(tc>tmax)tup=tmax;
      //
      for (int k=0;k<2;k++){
        eloss+=2*PI*re*re*me*nel[k]*laststep/beta/beta*
          (log(2*(gamma+1)/(I[k]*I[k]/me/me))
           +RadCorr(te,tup));
      }
      //
      // add the energy loss to dE and set the "photon" energy
      //
      if(dE+eloss > energy){
        *egptr++=energy-dE; // the muon stops, limit dE and egamma
        dE=energy;
      }
      else{
        *egptr++=eloss; // the muon is still travelling
        dE+=eloss;
      }
      //
```

```
      // put the photon at the center of the segment
      //
      *rgptr++ = x[1] - (range-(laststep/2))*t[0];
      *rgptr++ = y[1] - (range-(laststep/2))*t[1];
      *rgptr++ = z[1] - (range-(laststep/2))*t[2];
      //
      *tgptr++ = time;
      //
      // count the last photon
      ngamma++;
      //
      // finish the loop
      done = true;
    }
  }
  else{
    /* the muon has stopped in the detector, make one last photon
       with energy equal the mass of the muon
    */
    muonstop++;

    // set the "photon" energy
    //
    *egptr++=mmu;
    //
    // put the photon at point the muon stopped
    //
    *rgptr++ = x[1] - range*t[0];
    *rgptr++ = y[1] - range*t[1];
    *rgptr++ = z[1] - range*t[2];
    //
    *tgptr++ = time;
    //
    // count the last photon
    ngamma++;
    //
    // finish the loop
    done = true;

    /*
    cout << "Muon " << muonstop << " has stopped at "
         << x[1] - range*t[0] << ", "
         << y[1] - range*t[1] << ", "
         << z[1] - range*t[2] << " in "
         << time << " ns" << endl;
    */
  }
}
//cout << " muon range " << range << " cm, time " << time
//     << " ns and energy loss " << dE << " MeV" << endl;

/* now deal with isotope production; start with the cross-section
   for 9Li+8He prodution (as a function of incident muon energy
   in MeV) and generate an isotope; this isn't perfect because the
   muon changes energy throughout its path, altering the XC, but
   it's okay for now
 */
double XC;
double mfp;
double alpha = 0.73; // from the Hagner paper
double XCmeas = 2.12;
double Emeas = 190000.; // 190 GeV
double ubarntocmsq = 1e-30;
//cout << " sf " << (XCmeas/pow(Emeas,alpha))*ubarntocmsq << endl;
//
```

```
XC = (XCmeas/pow(Emeas,alpha))*pow(energy,alpha)*ubarntocmsq;
//cout << " XC " << XC << " cm^2" << endl;
//
// compute mean free path to produce 9Li+8He
//
int ntrial = 0;
passed = false;
while(!passed){

  mfp = 1/ncar/XC;
  //
  // skip if ranlux doesn't have the necessary precision
  //
  double test = 1/pow(10.0,(range/mfp));
  if(test > 0.9999998){
    mfp = range/2.;
    ntrial = 0;
  }
  else{
    const int len=1;
    float rv[len];
    ranlux_(rv,len);
    mfp*=log(1/rv[0]);
  }

  // check that the isotope is inside the detector
  if(mfp <= range) passed = true;

  // count the attempt
  ntrial++;
}
//cout << " mfp " << mfp << " cm in " << ntrial << " trials" << endl;

/* start at the mean free path distance from where the muon
   entered the detector along the muon track in the direction
   of the muon flight to produce the isotope; define a new
   coordinate system with the muon track being the +z'-axis,
   find the perpendicular distance from the muon track via
   Hagner, and pick a random azimuthal angle; thus the
   perpendicular distance and azimuthal angle give the new
   x' and y' values for the isotope and the point of isotope
   production along the muon track is z' = 0; then rotate the
   new coordinate system into the detector x, y, and z
*/
double ri[3];
ri[0] = x[1] - mfp*t[0];
ri[1] = y[1] - mfp*t[1];
ri[2] = z[1] - mfp*t[2];
//
// pick the direction, relative to point ri
//
const int len=2;
float rw[len];
ranlux_(rw,len);

double newphig = 2*PI*rw[0];
double newcosg = 0.;
double newt[3];
newt[0] = cos(newphig)*sqrt(1-newcosg*newcosg);
newt[1] = sin(newphig)*sqrt(1-newcosg*newcosg);
newt[2] = newcosg;
//
// select the lateral distance from the muon track, relative to ri
//
float lateraldistance;
```

```
int len2 = 1;
funlux_(fspace,lateraldistance,len2);
//
// define point rt by x',y', and z' in the new coordinate system
//
double rt[3];
rt[0] = lateraldistance*newt[0];
rt[1] = lateraldistance*newt[1];
rt[2] = lateraldistance*newt[2];
//for(int i=0; i<3; i++)cout << " rt[" << i << "] " << rt[i];
//cout << endl;

/* always assume that the x'-axis of the new coordinate system is
   in the direction chosen by angle phig of the muon track above;
   thus we rotate point rt around the y'-axis (which is now chosen
   always perpendicular to the detector coordinate system's x-y
   plane) by the angle given by cosg above subtracted from pi; we
   then rotate the new point qt around the z'-axis by angle pi
   minus phig to get the new point pt, which is rt in the
   detector coordinate system
*/
double angle = PI - acos(cosg);
double qt[3];
double* qtptr = qt;
qtptr = Rotate('Y',angle,rt[0],rt[1],rt[2]);
for(int i=0; i<3; i++) qt[i] = *qtptr++;
//
// second rotation
//
angle = PI - phig;
double pt[3];
double* ptptr = pt;
ptptr = Rotate('Z',angle,qt[0],qt[1],qt[2]);
for(int i=0; i<3; i++) pt[i] = *ptptr++;
//
// add the new lateral displacement to the point on the muon
// track at which the isotope is produced
//
for(int i=0; i<3; i++)ri[i] += pt[i];
//for(int i=0; i<3; i++)cout << " ri[" << i << "] " << ri[i];
//cout << endl;
//
// decide if we have 9Li or 8He
//
char* isotope;
if(rw[1] <= 0.5)
  isotope = "9Li";
else
  isotope = "8He";
//
// make energy and position arrays
//
int maxneutron = 1;
double* eneutron = new double[maxneutron];
double* rneutron = new double[maxneutron*3];
double* tneutron = new double[maxneutron];
//
double* enptr = eneutron;
double* rnptr = rneutron;
double* tnptr = tneutron;
//
int maxelectron = 1;
double* eelectron = new double[maxelectron];
double* relectron = new double[maxelectron*3];
double* telectron = new double[maxelectron];
```

```
//
double* eeptr = eelectron;
double* reptr = relectron;
double* teptr = telectron;
//
// let the isotope decay
//
int nneutron = 0;
int nelectron = 0;
//
if(isotope == "9Li"){
  MyLiSource LiSource(9,Rmaxgen);

  nneutron = 1;
  *enptr++ = LiSource.GetLiNeutronEnergy();
  for(int i=0; i<3; i++) *rnptr++ = ri[i];
  *tnptr++ = LiSource.GetLiDecayTime();

  nelectron = 1;
  *eeptr++ = LiSource.GetLiBetaEnergy();
  for(int i=0; i<3; i++) *reptr++ = ri[i];
  *teptr++ = LiSource.GetLiDecayTime();

  //    Weight = 1./float(ntrial)*LiSource.GetLiWeight();
  Weight = 1./float(ntrial);
}
else if(isotope == "8He"){
  MyHeSource HeSource(8,Rmaxgen);

  nneutron = 1;
  *enptr++ = HeSource.GetHeNeutronEnergy();
  for(int i=0; i<3; i++) *rnptr++ = ri[i];
  *tnptr++ = HeSource.GetHeDecayTime();

  nelectron = 1;
  *eeptr++ = HeSource.GetHeBetaEnergy();
  for(int i=0; i<3; i++) *reptr++ = ri[i];
  *teptr++ = HeSource.GetHeDecayTime();

  //    Weight = 1./float(ntrial)*HeSource.GetHeWeight();
  Weight = 1./float(ntrial);
}
else{
  cout << "  ERROR: unknown isotope " << isotope
       << ".  Options are '9Li' and '8He'." << endl;
}
//
// transfer temp data to storage
//
Nneutron = nneutron;
Nelectron = nelectron;
Ngamma   = ngamma;

Eneutron = new double[nneutron];
Rneutron = new double[nneutron*3];
Tneutron = new double[nneutron];

Eelectron = new double[nelectron];
Relectron = new double[nelectron*3];
Telectron = new double[nelectron];

Egamma = new double[ngamma];
Rgamma = new double[ngamma*3];
Tgamma = new double[ngamma];

```

```
  double* ptr;

  ptr = Eneutron+nneutron;
  while(ptr>Eneutron)*--ptr=*--enptr;

  ptr = Rneutron+3*nneutron;
  while(ptr>Rneutron)*--ptr=*--rnptr;

  ptr = Tneutron+nneutron;
  while(ptr>Tneutron)*--ptr=*--tnptr;

  ptr = Eelectron+nelectron;
  while(ptr>Eelectron)*--ptr=*--eeptr;

  ptr = Relectron+3*nelectron;
  while(ptr>Relectron)*--ptr=*--reptr;

  ptr = Telectron+nelectron;
  while(ptr>Telectron)*--ptr=*--teptr;

  ptr = Egamma+ngamma;
  while(ptr>Egamma)*--ptr=*--egptr;

  ptr = Rgamma+3*ngamma;
  while(ptr>Rgamma)*--ptr=*--rgptr;

  ptr = Tgamma+ngamma;
  while(ptr>Tgamma)*--ptr=*--tgptr;

  /*
  cout << Nneutron << " neutrons" << endl;
  for(int n=0; n<Nneutron; n++){
    cout << " neutron " << n << " energy " << Eneutron[n] << " MeV, and x, y, z:";
    for(int i=0; i<3; i++) cout << "  " << Rneutron[3*n+i];
    cout << endl << " and time " << Tneutron[n] << " nsec" << endl;
  }

  cout << Nelectron << " electrons" << endl;
  for(int n=0; n<Nelectron; n++){
    cout << " electron " << n << " energy " << Eelectron[n] << " MeV, and x, y, z:";
    for(int i=0; i<3; i++) cout << "  " << Relectron[3*n+i];
    cout << endl << " and time " << Telectron[n] << " nsec" << endl;
  }

  cout << Ngamma << " photons" << endl;
  for(int n=0; n<Ngamma; n++){
    cout << " photon " << n << " energy " << Egamma[n] << " MeV, and x, y, z:";
    for(int i=0; i<3; i++) cout << "  " << Rgamma[3*n+i];
    cout << endl << " and time " << Tgamma[n] << " nsec" << endl;
  }
  */

  // clean up
  delete[] eneutron;
  delete[] rneutron;
  delete[] tneutron;
  delete[] eelectron;
  delete[] relectron;
  delete[] telectron;
  delete[] egamma;
  delete[] rgamma;
  delete[] tgamma;
}
```

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

- **MuonPropagator.hh**
- **MuonPropagator.cpp**

3.2 MyBiSource Class Reference

MyBiSource generates photons from radioactive Thallium decays.

#include <MyBiSource.hh>

Public Methods

- **MyBiSource** (int newIsotope=ISOTOPE)
  MyBiSource constructor.
- ~**MyBiSource** ()
  MyBiSource destructor.
- **MyBiSource** (const MyBiSource &BiSource)
  MyBiSource copy constructor.
- MyBiSource & **operator=** (const MyBiSource &rhs)
  MyBiSource overloaded = operator.
- int **GetNumBiGammas** () const
  Returns the number of photons produced by the Bi decay.
- float **GetBiGammaEnergy** (int &n) const
  Returns the energy of each photon produced by the Bi decay.

3.2.1 Detailed Description

MyBiSource generates photons from radioactive Thallium decays.

The supported isotopes are Bi212 and Bi214. The decay table is read in from tl_bi_decays.txt and parsed by MyBiSource to create an array of the decay branch probabilites and the number and energies of the photons generated by each branch.

Definition at line 18 of file MyBiSource.hh.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 MyBiSource::MyBiSource (int *newIsotope* = ISOTOPE)

MyBiSource constructor.

Called with the desired Thallium isotope: 212 or 214. Default is the isotope given in Reactor-Constants, 252, which does nothing.

Definition at line 17 of file MyBiSource.cpp.

```
17                                        {
18 
19   Isotope = newIsotope;
20 
21   //cout << "MyBiSource::MyBiSource(" << Isotope << ")" << endl;
```

```

// input the source number and position
fstream fin("data/tl_bi_decays.txt", ios::in);
//cout << "Bi input file: data/tl_bi_decays.txt" << endl;

char paramLine[256];
string dummyString;
int dummyInt;
float dummyFloat;
bool done = false;
bool Set = false;

int Z, A;
int Nbranch = 0;
float branchProb[130];
int Ngamma[130];
float gammaE[130][6];

// zero the permanent information
numGamma = 0;
for(int i=0;i<6; i++) gammaEnergy[i] = 0.;

// Bi212
if(Isotope == 212){

  while(!done){

    // '#' specifies a comment line -- ignore it
    if(fin.peek() == '#'){

      fin.getline(paramLine, 256);
      continue;
    }

    // 'E'ND or 'e'nd specifies end of file -- set done flag to true
    if(fin.peek() == 'e' || fin.peek() == 'E'){

      done = true;
      continue;
    }

    // 'N'UCLEUS = or 'n'ucleus = specifies Z and A of daughter nucleus
    if(fin.peek() == 'n' || fin.peek() == 'N'){

      if(Set){

        fin.getline( paramLine, 256 );
        continue;
      }
      fin >> dummyString;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> Z >> A;
      //cout << Z << " " << A << endl;

      if(A == 212){

        // read in Bi212 decay table, 6 branches
        Nbranch = 6;
        for(int n=0; n<Nbranch; n++){

          fin >> branchProb[n] >> dummyInt >> dummyFloat >> Ngamma[n];
          for(int i=0; i<6; i++) fin >> gammaE[n][i];
          fin >> dummyFloat >> dummyFloat >> dummyFloat;
```

```

           /*
           cout << "  branchProb[" << n << "] " << branchProb[n]
                << " and Ngamma[" << n << "] " << Ngamma[n] << endl;
           for(int i=0; i<Ngamma[n]; i++){
             cout << "  gammaE[" << n << "][" << i << "] " << gammaE[n][i];
           }
           cout << endl;
           */
         }

         Set = true;
       }

        continue;
      }

      // ignore lines starting with numbers if they are not in our decay table
      if(fin.peek() == '0' || fin.peek() == '1'){

        fin.getline(paramLine, 256);
        continue;
      }

      // ignore extra whitespace
      if(fin.peek() == ' ' || fin.peek() == '\n'){

        fin.ignore(1);
        continue;
      }
    }

  } // done with Bi212

  // Bi214
  if(Isotope == 214){

    while(!done){

      // '#' specifies a comment line -- ignore it
      if(fin.peek() == '#'){

        fin.getline(paramLine, 256);
        continue;
      }

      // 'E'ND or 'e'nd specifies end of file -- set done flag to true
      if(fin.peek() == 'e' || fin.peek() == 'E'){

        done = true;
        continue;
      }

      // 'N'UCLEUS = or 'n'ucleus = specifies Z and A of daughter nucleus
      if(fin.peek() == 'n' || fin.peek() == 'N'){

        if(Set){

          fin.getline( paramLine, 256 );
          continue;
        }
        fin >> dummyString;
        while(fin.peek() == ' ' || fin.peek() == '=')
          fin.ignore(1);

```

```
        fin >> Z >> A;
        //cout << Z << " " << A << endl;

        if(A == 214){

          // read in Bi214 decay table, 127 branches
          Nbranch = 127;
          for(int n=0; n<Nbranch; n++){

            fin >> branchProb[n] >> dummyInt >> dummyFloat >> Ngamma[n];
            for(int i=0; i<6; i++) fin >> gammaE[n][i];
            fin >> dummyFloat >> dummyFloat >> dummyFloat;

            /*
            cout << "  branchProb[" << n << "] " << branchProb[n]
                 << " and Ngamma[" << n << "] " << Ngamma[n] << endl;
            for(int i=0; i<Ngamma[n]; i++){
              cout << "  gammaE[" << n << "][" << i << "] " << gammaE[n][i];
            }
            cout << endl;
            */
          }

          Set = true;
        }

        continue;
      }

      // ignore lines starting with numbers if they are not in our decay table
      if(fin.peek() == '0' || fin.peek() == '1'){

        fin.getline(paramLine, 256);
        continue;
      }

      // ignore extra whitespace
      if(fin.peek() == ' ' || fin.peek() == '\n'){

        fin.ignore(1);
        continue;
      }
    }

  } // done with Bi214

  // pick a decay branch and get the gamma energies
  const int len = 1;
  float r[len];
  ranlux_(r,len);

  bool passed = false;
  for(int n=0;n<Nbranch; n++){

    if(branchProb[n] > r[0] && !passed){

      // save the number and energies of the gammas
      numGamma = Ngamma[n];
      for(int i=0;i<numGamma; i++) gammaEnergy[i] = gammaE[n][i];

       if(numGamma)passed = true;
    }
  }


```

```
217 }
```

3.2.3 Member Function Documentation

3.2.3.1 float MyBiSource::GetBiGammaEnergy (int & *n*) const `[inline]`

Returns the energy of each photon produced by the Bi decay.

Called with the integer index for each photon in the gammaEnergy array, from 0 to the total number of photons.

Definition at line 38 of file MyBiSource.hh.

```
38 {return gammaEnergy[n];}
```

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

- **MyBiSource.hh**
- **MyBiSource.cpp**

3.3 MyCfSource Class Reference

MyCfSource generates prompt photons and delayed neutrons from radioactive Californium decays.

`#include <MyCfSource.hh>`

Public Methods

- **MyCfSource** (int newIsotope=ISOTOPE, double newRmaxgen=RMAXGEN)

  MyCfSource constructor.

- ~**MyCfSource** ()

  MyCfSource destructor.

- **MyCfSource** (const MyCfSource &CfSource)

  MyCfSource copy constructor.

- MyCfSource & **operator=** (const MyCfSource &rhs)

  MyCfSource overloaded = operator.

- int **GetNumCfSources** () const

  Returns the number of Cf sources.

- int **GetNumNeutron** (int &n) const

  Returns the neutron multiplicity of each source.

- int **GetNumGamma** (int &n) const

  Returns the prompt photon multiplicity of each source.

- double **GetCfNeutronEnergy** (int &n, int &nn) const

  Returns the energy of the neutrons produced by each Cf decay.

- double **GetCfNeutronTime** (int &n, int &nn) const
- double **GetCfGammaEnergy** (int &n, int &nn) const

  Returns the energy of the gammas produced by each Cf decay.

- double **GetCfGammaTime** (int &n, int &nn) const
- double **GetCfVertexR** (int &n) const

  Returns the radius of the neutron produced by each Cf decay.

- double **GetCfVertexCosTheta** (int &n) const

  Returns cos(theta) of the neutron produced by each Cf decay.

- double **GetCfVertexPhi** (int &n) const

  Returns the phi of the neutron produced by each Cf decay.

Static Public Methods

- float **Cf252NeutronSpectrum** (const float &x)

  The probability density of the prompt neutrons from the Cf decay as a function of neutron energy.

- float **Cf252GammaMultiplicity** (const int &x)

  The probability density of the prompt gammas from the Cf decay as a function of integer gamma multiplicity (exact).

- float **Cf252GammaMultiplicityFit** (const float &x)

  The probability density of the prompt gammas from the Cf decay as a function of float gamma multiplicity (fit).

- float **Cf252GammaSpectrum** (const float &x)

  The probability density of the prompt gammas from the Cf decay as a function of gamma energy.

3.3.1 Detailed Description

MyCfSource generates prompt photons and delayed neutrons from radioactive Californium decays.

The supported isotope is 252, which is also the default isotope given ReactorConstants. The neutrons are generated with the correct multiplicity and each neutron is given an energy by the Cf252NeutronSpectrum function, which the probability density of the produced neutrons as a function of neutron energy. The prompt photons are generated by the Cf252GammaMultiplicity-Fit and Cf252GammaSpectrum functions.

The number and locations of the Cf sources are read in from cf252_position.txt and parsed by MyCfSource to yield the source vertex R, cos(theta), and phi.

Definition at line 26 of file MyCfSource.hh.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 MyCfSource::MyCfSource (int *newIsotope* = ISOTOPE, double *newRmaxgen* = RMAXGEN)

MyCfSource constructor.

Called with the desired Cf isotope: 252, which is the default isotope given in ReactorConstants and the maximum allowed radius for generating particles in the simulation, Rmaxgen (default = 200 cm). If the source in cf252_position.txt is located with R > Rmaxgen the source is ignored and an error message is printed.

Definition at line 19 of file MyCfSource.cpp.

References Cf252GammaMultiplicityFit, Cf252GammaSpectrum, and Cf252NeutronSpectrum.

```
                                      {

  Isotope = newIsotope;
  Rmaxgen = newRmaxgen;

  //  cout << "MyCfSource::MyCfSource(" << Isotope << "," << Rmaxgen
  //       << ")" << endl;

```

```
// Cf252
if(Isotope == 252){

  // setup the probability density as a function of energy
  bool static first=1;
  float static fspace[200];
  float static mspace[200];
  float static gspace[200];

  if(first){
    float xlo = 0.;
    float xhi = 50.;
    funlxp_(Cf252NeutronSpectrum,fspace,xlo,xhi);
    xhi = 25.;
    funlxp_(Cf252GammaMultiplicityFit,mspace,xlo,xhi);
    funlxp_(Cf252GammaSpectrum,gspace,xlo,xhi);
    first = 0;
  }

  // input the source number and position
  fstream cf252_position("data/cf252_position.txt", ios::in);
  //cout << "Cf input file: data/cf252_position.txt" << endl;

  Nsources = 0;
  char paramLine[256];
  string dummy;
  bool done = false;
  bool numSet = false;
  int positionNum = 0;

  while(!done){

    // '#' specifies a comment line -- ignore it
    if(cf252_position.peek() == '#'){

      cf252_position.getline(paramLine, 256);
      continue;
    }

    // 'E'ND or 'e'nd specifies end of file -- set done flag to true
    if(cf252_position.peek() == 'e' || cf252_position.peek() == 'E'){

      if(Nsources == 0) cout << "   Cf252 ERROR: "
                             << "number of cf252 sources = 0" << endl;
      done = true;
      continue;
    }

    // 'N'SOURCES = or 'n'sources = specifies number of sources
    if(cf252_position.peek() == 'n' || cf252_position.peek() == 'N'){

      if(numSet){

        cf252_position.getline( paramLine, 256 );
        continue;
      }
      cf252_position >> dummy;
      while(cf252_position.peek() == ' ' || cf252_position.peek() == '=')
        cf252_position.ignore(1);

      cf252_position >> Nsources;
      if(Nsources == 0){
        cout << "   Cf252 ERROR: number of cf252 sources = 0" << endl;
        done = true;
        continue;
```

```
   }

   numSet = true;
   continue;
 }

 // 'P'OSITION = or 'p'osition = specifies position of the sourse(s)
  if(cf252_position.peek() == 'p' || cf252_position.peek() == 'P'){

    if(Nsources == 0 || positionNum >= Nsources){

      cout << "   Cf252 ERROR: too many positions " << positionNum+1
           << " given for the number of sources " << Nsources << endl;
      cf252_position.getline( paramLine, 256 );
      positionNum++;
      continue;
    }
    cf252_position >> dummy;
    while(cf252_position.peek() == ' ' || cf252_position.peek() == '=')
      cf252_position.ignore(1);

    cf252_position >> R[positionNum] >> zR[positionNum]
                   >> phiR[positionNum];

    // check for unphysical source positions
    if(R[positionNum] > Rmaxgen){
      cout << "   Cf252 ERROR: R[" << positionNum << "] of "
           << R[positionNum] << " > Rmaxgen" << endl;
      done = true;
      continue;
    }
    if(zR[positionNum] < -1. || zR[positionNum] > 1.){
      cout << "   Cf252 ERROR: CosTheta["
           << positionNum << "] of " << zR[positionNum] << endl;
      done = true;
      continue;
    }
    if(phiR[positionNum] > 2*PI){
      cout << "   Cf252 ERROR: Phi[" << positionNum << "] of "
           << phiR[positionNum] << " > 2*pi" << endl;
      done = true;
      continue;
    }

    positionNum++;
    continue;
  }

  // ignore extra whitespace
  if(cf252_position.peek() == ' ' || cf252_position.peek() == '\n'){

    cf252_position.ignore(1);
    continue;
  }
}

if(positionNum < Nsources){
  cout << "   Cf252 ERROR: not enough positions " << positionNum
       << " given for the number of sources " << Nsources << endl;
}
//
// neutron multiplicity distribution (via Los Alamo manual)
//
double p[maxNeutron+1] = {0.002,0.028,0.155,0.428,
                          0.732,0.917,0.983,0.998,1.000};
```

```
    //
    for(int n=0;n<Nsources;n++){
      //cout << "Source " << n+1 << endl << "   Cf Source Position = "
      //           << R[n] << ", " << zR[n] << ", " << phiR[n] << endl;

      // pick a neutron multiplicity
      bool passed = false;
      int len;
      while(!passed){
        const int len = 1;
        float r[len];
        ranlux_(r,len);

        for(int i=0;i<maxNeutron+1; i++){

          if(p[i] > r[0] && !passed){
            Nneutron[n] = i;
            if(Nneutron[n])passed = true;
          }
        }
      }
      //cout << "   " << Nneutron[n] << " neutrons" << endl;
      //
      // pick an energy for each neutron
      //
      for(int nn=0;nn<Nneutron[n]; nn++){
        len = 1;
        funlux_(fspace,neutronE[n][nn],len);
        //cout << "   Cf neutron energy = " << neutronE[n][nn] << endl;
        Tneutron[n][nn] = 0.;
      }
      /*
        The total energy in the prompt gammas is a function of the
        material and the number of prompt neutrons produced; these
        formulae come from T. Valentine's summary
      */
      float Z = 98.;
      float A = 252.;
      //
      double avge;
      double phi;
      double etot;
      //
      avge = -1.33 + 119.6*pow(Z,0.3333)/A;
      phi = 2.51 - 1.13e-5*Z*Z*sqrt(A);
      etot = phi*Nneutron[n]+4.0;
      //
      // use the Brunson multiplicity (also via Valentine)
      //
      len = 1;
      float ng = 0.;
      funlux_(mspace,ng,len);
      Ngamma[n] = (int)ng;
      //cout << "   " << Ngamma[n] << " photons" << endl;
      //
      // pick an energy for each gamma
      //
      double tote = 0.;
      for(int nn=0;nn<Ngamma[n]; nn++){
        int len = 1;
        funlux_(gspace,gammaE[n][nn],len);
        //cout << "   252Cf photon energy = " << gammaE[n][nn] << endl;
        tote += gammaE[n][nn];

        const int len2 = 2;
```

```
        float rv[len2];
        ranlux_(rv,len2);
        //
        // 80% of gammas have T_1/2 = 0.01 ns and 20% have T_1/2 = 1 ns
        //
        double halflife;
        if(rv[0] < 0.8)
          halflife = 0.01;
        else
          halflife = 1.0;
        Tgamma[n][nn] = halflife*log(1/rv[1]);
      }
      //cout << "          total energy = " << tote << endl;
    } // done looping over sources

    // zero out remaining energy and position arrays
    for(int n=0;n<Nsources;n++){
      for(int nn=Nneutron[n];nn<maxNeutron;nn++){
        neutronE[n][nn] = 0.;
        Tneutron[n][nn] = 0.;
      }
      for(int nn=Ngamma[n];nn<maxGamma;nn++){
        gammaE[n][nn] = 0.;
        Tgamma[n][nn] = 0.;
      }
    }
    for(int n=Nsources;n<Ndim;n++){
      Nneutron[n] = 0;
      Ngamma[n] = 0;
      R[n] = 0.;
      zR[n] = 0.;
      phiR[n] = 0.;
      for(int nn=0;nn<maxNeutron;nn++){
        neutronE[n][nn] = 0.;
        Tneutron[n][nn] = 0.;
      }
      for(int nn=0;nn<maxGamma;nn++){
        gammaE[n][nn] = 0.;
        Tgamma[n][nn] = 0.;
      }
    }

  } // done with Cf 252

  if(Isotope == 255){

    Nsources = 0;
    for(int n=Nsources;n<Ndim;n++){
      Nneutron[n] = 0;
      Ngamma[n] = 0;
      R[n] = 0.;
      zR[n] = 0.;
      phiR[n] = 0.;
      for(int nn=0;nn<maxNeutron;nn++){
        neutronE[n][nn] = 0.;
        Tneutron[n][nn] = 0.;
      }
      for(int nn=0;nn<maxGamma;nn++){
        gammaE[n][nn] = 0.;
        Tgamma[n][nn] = 0.;
      }
    }
  } // done with Cf 255
}
```

3.3.3 Member Function Documentation

3.3.3.1 double MyCfSource::GetCfGammaEnergy (int & *n*, int & *nn*) const [inline]

Returns the energy of the gammas produced by each Cf decay.

Called with the integer index for each gamma in the gammaE array, from 0 to the total number of sources and 0 to the total number of gammas/source.

Definition at line 85 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

```
85 {return gammaE[n][nn];}
```

3.3.3.2 double MyCfSource::GetCfNeutronEnergy (int & *n*, int & *nn*) const [inline]

Returns the energy of the neutrons produced by each Cf decay.

Called with the integer index for each neutron in the neutronE array, from 0 to the total number of sources and 0 to the total number of neutrons/source.

Definition at line 78 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

```
78 {return neutronE[n][nn];}
```

3.3.3.3 double MyCfSource::GetCfVertexCosTheta (int & *n*) const [inline]

Returns cos(theta) of the neutron produced by each Cf decay.

Called with the integer index for each neutron in the zR array, from 0 to the total number of sources.

Definition at line 96 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

```
96 {return zR[n];}
```

3.3.3.4 double MyCfSource::GetCfVertexPhi (int & *n*) const [inline]

Returns the phi of the neutron produced by each Cf decay.

Called with the integer index for each neutron in the phiR array, from 0 to the total number of sources.

Definition at line 101 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

```
101 {return phiR[n];}
```

3.3.3.5 double MyCfSource::GetCfVertexR (int & *n*) const `[inline]`

Returns the radius of the neutron produced by each Cf decay.

Called with the integer index for each neutron in the R array, from 0 to the total number of sources.

Definition at line 91 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

```
91 {return R[n];}
```

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

- **MyCfSource.hh**
- **MyCfSource.cpp**

3.4 MyHeSource Class Reference

MyHeSource generates delayed neutrons from radioactive He decays.

`#include <MyHeSource.hh>`

Public Methods

- **MyHeSource** (int newIsotope=ISOTOPE, double newRmaxgen=RMAXGEN)

  MyHeSource constructor.

- **~MyHeSource** ()

  MyHeSource destructor.

- **MyHeSource** (const MyHeSource &HeSource)

  MyHeSource copy constructor.

- MyHeSource & **operator=** (const MyHeSource &rhs)

  MyHeSource overloaded = operator.

- double **GetHeNeutronEnergy** () const

  Returns the energy of the neutrons produced by each He decay.

- double **GetHeBetaEnergy** () const

  Returns the energy of the betas produced by each He decay.

- double **GetHeGammaEnergy** () const

  Returns the energy of the photons produced by each He decay.

- double **GetHeWeight** () const

  Returns the branching ratio weight for each He decay.

- double **GetHeDecayTime** () const

  Returns the decay time for each He decay.

Static Public Methods

- float **BetaHighQ** (const float &x)
- float **BetaMiddleQ** (const float &x)
- float **BetaLowQ** (const float &x)
- double **Fermi** (const int &Z, const int &A, const float &x)

3.4.1 Detailed Description

MyHeSource generates delayed neutrons from radioactive He decays.

The supported isotope is 8. The neutrons are generated by the He8NeutronSpectrum function, which the probability density of the produced neutrons as a function of neutron energy.

Definition at line 18 of file MyHeSource.hh.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 MyHeSource::MyHeSource (int *newIsotope* = ISOTOPE, double *newRmaxgen* = RMAXGEN)

MyHeSource constructor.

Called with the desired He isotope: 8 and the maximum allowed radius for generating particles in the simulation, Rmaxgen (default = 200 cm).

Definition at line 17 of file MyHeSource.cpp.

```
18                                         {
19
20   Isotope = newIsotope;
21   Rmaxgen = newRmaxgen;
22
23   //cout << "MyHeSource::MyHeSource(" << Isotope << "," << Rmaxgen
24   //      << ")" << endl;
25
26   // 8He
27   if(Isotope == 8){
28
29     // setup the decay parameters (from Jon Link)
30     // Nneutron is always 1 and Ngamma always 0
31     Nneutron = 1;
32     Ngamma = 0;
33
34     // branching fraction (the 1.0000 bin contains all decays
35     // which do nothing: neutron E = 0, beta Q = 0)
36     float static br[4] = {0.080,0.160,0.169,1.0000};
37
38     // Q-value for beta decay
39     float static betaQ[4] = {7.442,5.252,1.852,0.000};
40
41     // Q-value for the total decay chain
42     float static totQ;
43     totQ = 8.619;
44
45     // neutron energy
46     float static neuE[4] = {0.697,2.887,6.727,0.000};
47
48     // prepare the beta decay functions
49     bool static first=1;
50     float static fspace[3][200];
51
52     if(first){
53       float xlo = 0.;
54       funlxp_(BetaHighQ,fspace[0],xlo,betaQ[0]);
55       funlxp_(BetaMiddleQ,fspace[1],xlo,betaQ[1]);
56       funlxp_(BetaLowQ,fspace[2],xlo,betaQ[2]);
57
58       first = 0;
59     }
60     //
61     // let the 8He decay
62     //
63     int ntrial = 0;
64     bool passed = false;
65     while(!passed){
66       const int len=1;
67       float rv[len];
68       ranlux_(rv,len);
69       //
```

```
      for(int i=0;i<4; i++){
        if(rv[0] < br[i] && !passed){

          // pick an energy for each neutron
          neutronE = neuE[i];

          if(neutronE){

            // get the energy of the beta
            int len2 = 1;
            funlux_(fspace[i],betaE,len2);
            passed = true;
          }
        }
      }
      ntrial++;
    }
    Weight = 1./float(ntrial);

    //cout << "   8He neutron energy = " << neutronE << " MeV" << endl;
    //cout << "   8He beta energy = " << betaE << " MeV" << endl;
    //cout << "   8He weight = " << Weight << endl;

    // pick a decay time in ns
    double HalfLife = 1.2e8;

    const int len=1;
    float rv[len];
    ranlux_(rv,len);
    DecayTime = HalfLife*log(1/rv[0]);

  } // done with 8He
}
```

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

- **MyHeSource.hh**
- **MyHeSource.cpp**

3.5 MyLiSource Class Reference

MyLiSource generates delayed neutrons from radioactive Li decays.

`#include <MyLiSource.hh>`

Public Methods

- **MyLiSource** (int newIsotope=ISOTOPE, double newRmaxgen=RMAXGEN)
  MyLiSource constructor.
- ~**MyLiSource** ()
  MyLiSource destructor.
- **MyLiSource** (const MyLiSource &LiSource)
  MyLiSource copy constructor.
- MyLiSource & **operator=** (const MyLiSource &rhs)
  MyLiSource overloaded = operator.
- double **GetLiNeutronEnergy** () const
  Returns the energy of the neutrons produced by each Li decay.
- double **GetLiBetaEnergy** () const
  Returns the energy of the betas produced by each Li decay.
- double **GetLiGammaEnergy** () const
  Returns the energy of the photons produced by each Li decay.
- double **GetLiWeight** () const
  Returns the branching ratio weight for each Li decay.
- double **GetLiDecayTime** () const
  Returns the decay time for each Li decay.

Static Public Methods

- float **BetaHighestQ** (const float &x)
- float **BetaHighQ** (const float &x)
- float **BetaMiddleQ** (const float &x)
- float **BetaLowQ** (const float &x)
- double **Fermi** (const int &Z, const int &A, const float &x)

3.5.1 Detailed Description

MyLiSource generates delayed neutrons from radioactive Li decays.

The supported isotope is 9. The neutrons are generated by the Li9NeutronSpectrum function, which the probability density of the produced neutrons as a function of neutron energy.

Definition at line 18 of file MyLiSource.hh.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 MyLiSource::MyLiSource (int *newIsotope* = ISOTOPE, double *newRmaxgen* = RMAXGEN)

MyLiSource constructor.

Called with the desired Li isotope: 9 and the maximum allowed radius for generating particles in the simulation, Rmaxgen (default = 200 cm).

Definition at line 18 of file MyLiSource.cpp.

```
19                                          {
20
21   Isotope = newIsotope;
22   Rmaxgen = newRmaxgen;
23
24   //cout << "MyLiSource::MyLiSource(" << Isotope << "," << Rmaxgen
25   //     << ")" << endl;
26
27   // 9Li
28   if(Isotope == 9){
29
30     // setup the decay parameters (from Jon Link)
31     // Nneutron is always 1 and Ngamma always 0
32     Nneutron = 1;
33     Ngamma = 0;
34
35     // branching fraction (the 1.0000 bin contains all decays
36     // which do nothing: neutron E = 0, beta Q = 0)
37     float static br[8] = {0.0238,0.3400,0.4400,0.4475,
38                           0.4550,0.4894,0.4950,1.0000};
39
40     // Q-value for beta decay
41     float static betaQ[8] = {11.176,11.176,10.826,5.666,
42                              5.666,2.323,2.323,0.000};
43
44     // Q-value for the total decay chain
45     float static totQ = 12.033;
46
47     // neutron energy
48     float static neuE[8] = {0.764,0.856,1.115,3.235,
49                             5.473,8.816,6.578,0.000};
50
51     // prepare the beta decay functions
52     bool static first=1;
53     float static fspace[7][200];
54
55     if(first){
56       float xlo = 0.;
57       funlxp_(BetaHighestQ,fspace[0],xlo,betaQ[0]);
58       funlxp_(BetaHighestQ,fspace[1],xlo,betaQ[1]);
59       funlxp_(BetaHighQ,fspace[2],xlo,betaQ[2]);
60       funlxp_(BetaMiddleQ,fspace[3],xlo,betaQ[3]);
61       funlxp_(BetaMiddleQ,fspace[4],xlo,betaQ[4]);
62       funlxp_(BetaLowQ,fspace[5],xlo,betaQ[5]);
63       funlxp_(BetaLowQ,fspace[6],xlo,betaQ[6]);
64
65       first = 0;
66     }
67     //
68     // let the 9Li decay
69     //
70     int ntrial = 0;
```

```
    bool passed = false;
    while(!passed){
      const int len=1;
      float rv[len];
      ranlux_(rv,len);
      //
      for(int i=0;i<8; i++){
        if(rv[0] < br[i] && !passed){

          // pick an energy for each neutron
          neutronE = neuE[i];

          if(neutronE){

            // get the energy of the beta
            int len2 = 1;
            funlux_(fspace[i],betaE,len2);
            passed = true;
          }
        }
      }
      ntrial++;
    }
    Weight = 1./float(ntrial);

    //cout << "   9Li neutron energy = " << neutronE << " MeV" << endl;
    //cout << "   9Li beta energy = " << betaE << " MeV" << endl;
    //cout << "   9Li weight = " << Weight << endl;

    // pick a decay time in ns
    double HalfLife = 1.8e8;

    const int len=1;
    float rv[len];
    ranlux_(rv,len);
    DecayTime = HalfLife*log(1/rv[0]);

  } // done with 9Li
}
```

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

- **MyLiSource.hh**
- **MyLiSource.cpp**

3.6 MyTlSource Class Reference

MyTlSource generates photons from radioactive Thallium decays.

#include <MyTlSource.hh>

Public Methods

- **MyTlSource** (int newIsotope=ISOTOPE)
  MyTlSource constructor.
- ~**MyTlSource** ()
  MyTlSource destructor.
- **MyTlSource** (const MyTlSource &TlSource)
  MyTlSource copy constructor.
- MyTlSource & **operator=** (const MyTlSource &rhs)
  MyTlSource overloaded = operator.
- int **GetNumTlGammas** () const
  Returns the number of photons produced by the Tl decay.
- float **GetTlGammaEnergy** (int &n) const
  Returns the energy of each photon produced by the Tl decay.

3.6.1 Detailed Description

MyTlSource generates photons from radioactive Thallium decays.

The supported isotopes are Tl208 and Tl210. The decay table is read in from tl_bi_decays.txt and parsed by MyTlSource to create an array of the decay branch probabilites and the number and energies of the photons generated by each branch.

Definition at line 18 of file MyTlSource.hh.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 MyTlSource::MyTlSource (int *newIsotope* = ISOTOPE)

MyTlSource constructor.

Called with the desired Thallium isotope: 208 or 210. Default is the isotope given in Reactor-Constants, 252, which does nothing.

Definition at line 17 of file MyTlSource.cpp.

```
17                                   {
18
19   Isotope = newIsotope;
20
21   //cout << "MyTlSource::MyTlSource(" << Isotope << ")" << endl;
```

```
// input the source number and position
fstream fin("data/tl_bi_decays.txt", ios::in);
//cout << "Tl input file: data/tl_bi_decays.txt" << endl;

char paramLine[256];
string dummyString;
int dummyInt;
float dummyFloat;
bool done = false;
bool Set = false;

int Z, A;
int Nbranch = 0;
float branchProb[40];
int Ngamma[40];
float gammaE[40][6];

// zero the permanent information
numGamma = 0;
for(int i=0;i<6; i++) gammaEnergy[i] = 0.;

// Tl208
if(Isotope == 208){

  while(!done){

    // '#' specifies a comment line -- ignore it
    if(fin.peek() == '#'){

      fin.getline(paramLine, 256);
      continue;
    }

    // 'E'ND or 'e'nd specifies end of file -- set done flag to true
    if(fin.peek() == 'e' || fin.peek() == 'E'){

      done = true;
      continue;
    }

    // 'N'UCLEUS = or 'n'ucleus = specifies Z and A of daughter nucleus
    if(fin.peek() == 'n' || fin.peek() == 'N'){

      if(Set){

        fin.getline( paramLine, 256 );
        continue;
      }
      fin >> dummyString;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> Z >> A;
      //cout << Z << " " << A << endl;

      if(A == 208){

        // read in Tl208 decay table, 40 branches
        Nbranch = 40;
        for(int n=0; n<Nbranch; n++){

          fin >> branchProb[n] >> dummyInt >> dummyFloat >> Ngamma[n];
          for(int i=0; i<6; i++) fin >> gammaE[n][i];
          fin >> dummyFloat >> dummyFloat >> dummyFloat;
```

```

          /*
          cout << "  branchProb[" << n << "] " << branchProb[n]
               << " and Ngamma[" << n << "] " << Ngamma[n] << endl;
          for(int i=0; i<Ngamma[n]; i++){
            cout << "  gammaE[" << n << "][" << i << "] " << gammaE[n][i];
          }
          cout << endl;
          */
        }

        Set = true;
      }

      continue;
    }

    // ignore lines starting with numbers if they are not in our decay table
    if(fin.peek() == '0' || fin.peek() == '1'){

      fin.getline(paramLine, 256);
      continue;
    }

    // ignore extra whitespace
    if(fin.peek() == ' ' || fin.peek() == '\n'){

      fin.ignore(1);
      continue;
    }
  }

} // done with Tl208

// Tl210
if(Isotope == 210){

  while(!done){

    // '#' specifies a comment line -- ignore it
    if(fin.peek() == '#'){

      fin.getline(paramLine, 256);
      continue;
    }

    // 'E'ND or 'e'nd specifies end of file -- set done flag to true
    if(fin.peek() == 'e' || fin.peek() == 'E'){

      done = true;
      continue;
    }

    // 'N'UCLEUS = or 'n'ucleus = specifies Z and A of daughter nucleus
    if(fin.peek() == 'n' || fin.peek() == 'N'){

      if(Set){

        fin.getline( paramLine, 256 );
        continue;
      }
      fin >> dummyString;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

```

```
        fin >> Z >> A;
        //cout << Z << " " << A << endl;

        if(A == 210){

          // read in Tl210 decay table, 12 branches
          Nbranch = 12;
          for(int n=0; n<Nbranch; n++){

            fin >> branchProb[n] >> dummyInt >> dummyFloat >> Ngamma[n];
            for(int i=0; i<6; i++) fin >> gammaE[n][i];
            fin >> dummyFloat >> dummyFloat >> dummyFloat;

            /*
            cout << "  branchProb[" << n << "] " << branchProb[n]
                 << " and Ngamma[" << n << "] " << Ngamma[n] << endl;
            for(int i=0; i<Ngamma[n]; i++){
              cout << "  gammaE[" << n << "][" << i << "] " << gammaE[n][i];
            }
            cout << endl;
            */
          }

          Set = true;
        }

        continue;
      }

      // ignore lines starting with numbers if they are not in our decay table
      if(fin.peek() == '0' || fin.peek() == '1'){

        fin.getline(paramLine, 256);
        continue;
      }

      // ignore extra whitespace
      if(fin.peek() == ' ' || fin.peek() == '\n'){

        fin.ignore(1);
        continue;
      }
    }

  } // done with Tl210

  // pick a decay branch and get the gamma energies
  const int len = 1;
  float r[len];
  ranlux_(r,len);

  bool passed = false;
  for(int n=0;n<Nbranch; n++){

    if(branchProb[n] > r[0] && !passed){

      // save the number and energies of the gammas
      numGamma = Ngamma[n];
      for(int i=0;i<numGamma; i++) gammaEnergy[i] = gammaE[n][i];

      if(numGamma)passed = true;
    }
  }


```

```
217 }
```

3.6.3 Member Function Documentation

3.6.3.1 float MyTlSource::GetTlGammaEnergy (int & *n*) const [inline]

Returns the energy of each photon produced by the Tl decay.

Called with the integer index for each photon in the gammaEnergy array, from 0 to the total number of photons.

Definition at line 38 of file MyTlSource.hh.

```
38 {return gammaEnergy[n];}
```

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

- **MyTlSource.hh**
- **MyTlSource.cpp**

3.7 ReactorDetector Class Reference

ReactorDetector handles the detector implementation in the ReactorFsim simulation.

`#include <ReactorDetector.hh>`

Public Methods

- **ReactorDetector** ()

  ReactorDetector constructor.

- ~**ReactorDetector** ()

  ReactorDetector destructor.

- **ReactorDetector** (const ReactorDetector &Detector)

  ReactorDetector copy constructor.

- ReactorDetector & **operator=** (const ReactorDetector &rhs)

  ReactorDetector overloaded = operator.

- void **LightsOut** (**ReactorEvent** &Event, int eventnum)
- void **CleanPMTOutputArray** ()
- double **GetPHelectron** () const
- double **GetPHpositron** () const
- double **GetPHneutron** () const
- double **GetPHthallium** () const
- double **GetPHbismuth** () const
- double **GetPHcalifornium** () const
- double **GetPHmuon** () const
- double **GetPHeleReco** () const
- double **GetPHposReco** () const
- double **GetPHneuReco** () const
- double **GetPHthaReco** () const
- double **GetPHbisReco** () const
- double **GetPHcalReco** () const
- double **GetPHmuoReco** () const
- double **GetMeanNpePos** () const
- double **GetMeanQPos** () const
- double **GetMeanAttenPos** () const
- double **GetMeanSfactorPos** () const
- double **GetMeanNpeNeu** () const
- double **GetMeanQNeu** () const
- double **GetMeanAttenNeu** () const
- double **GetMeanSfactorNeu** () const
- double **GetMeanNpeTl** () const
- double **GetMeanQTl** () const
- double **GetMeanAttenTl** () const
- double **GetMeanSfactorTl** () const
- double **GetMeanNpeBi** () const
- double **GetMeanQBi** () const

- double **GetMeanAttenBi** () const
- double **GetMeanSfactorBi** () const
- double **GetRefracSc** () const
- double **GetMeanNpeCf** () const
- double **GetMeanQCf** () const
- double **GetMeanAttenCf** () const
- double **GetMeanSfactorCf** () const
- double **GetMeanNpeMuon** () const
- double **GetMeanQMuon** () const
- double **GetMeanAttenMuon** () const
- double **GetMeanSfactorMuon** () const
- double **GetMeanNpeEle** () const
- double **GetMeanQEle** () const
- double **GetMeanAttenEle** () const
- double **GetMeanSfactorEle** () const
- double **GetHitFracEle** ()
- double **GetHitFracPos** ()
- double **GetHitFracNeu** ()
- double **GetHitFracTl** ()
- double **GetHitFracBi** ()
- double **GetHitFracCf** ()
- double **GetHitFracMuon** ()
- double **GetAbsorber** (int &n) const
- double **GetNeutronSteps** (int &n) const
- int **GetNcosbins** () const
- int **GetNphibins** () const
- int **GetNgamma** () const
- double *const **GetEgamma** ()
- double *const **GetTgamma** ()
- int **GetNgamma** (std::string parent)
- void **GetGamma** (std::string parent, double *xyz)
- void **GetGamma** (std::string parent, double *e, double *t, double *r, double *c, double *f)
- double *const **GetPositronDipole** ()
- double *const **GetNeutronDipole** ()
- int **GetNpmt** () const
- void **ReactorDetector::GetPMTdata** (int npmt, double *XYZpmt, int &nele, double *PHele, double *Tele, int &npos, double *PHpos, double *Tpos, int &nneu, double *PHneu, double *Tneu, int &ntha, double *PHtha, double *Ttha, int &nbis, double *PHbis, double *Tbis, int &ncal, double *PHcal, double *Tcal, int &nmuo, double *PHmuo, double *Tmuo, int *npepos, int *npeneu, int *npetha, int *npebis, int *npecal, int *npemuo, int *npeele)
- void **ReactorDetector::GetPMTdata** (std::string parent, int pmtindex, double *XYZpmt, int *hits, double *PH, double *Q, double *Time, int *npe)
- int **ReactorDetector::GetHitPMTsPos** ()
- int **ReactorDetector::GetHitPMTsNeu** ()
- int **ReactorDetector::GetHitPMTsTha** ()
- int **ReactorDetector::GetHitPMTsBis** ()
- int **ReactorDetector::GetHitPMTsCal** ()
- int **ReactorDetector::GetHitPMTsMuo** ()
- int **ReactorDetector::GetHitPMTsEle** ()
- void **SetParam_R0** (double x)

- void **SetParam_R1** (double x)
- void **SetParam_R2** (double x)
- void **SetParam_GdConcentration** (double x)
- void **SetParam_refracGd** (double x)
- void **SetParam_refracSc** (double x)
- void **SetParam_mfpGd** (double x)
- void **SetParam_mfpSc** (double x)
- void **SetParam_MeanNeuDispl** (double x)
- void **SetParam_FastNeutronOption** (bool x)
- void **SetParam_PMTDebugOption** (bool x)
- void **SetParam_PMTMergeTime** (double x)
- void **SetParam_Temperature** (double x)
- void **SetParam_tGd** (double x)
- void **SetParam_tSc** (double x)
- void **SetParam_GdCaptureFraction** (double x)
- void **SetParam_GammasPerGd** (double x)
- void **SetParam_attenlGd** (double x)
- void **SetParam_attenlSc** (double x)
- void **SetParam_PhotonsPerMeV** (double x)
- void **SetParam_PMTcoverage** (double x)
- void **SetParam_PMTdiameter** (double x)
- void **SetParam_PMTqe** (double x)
- void **SetParam_ScintDecayTime** (double x)
- double * **TranslateRadialDistance** (double, double, double, double)

Friends

- class **PMToutput**

3.7.1 Detailed Description

ReactorDetector handles the detector implementation in the ReactorFsim simulation.

The constructor creates a spherical three-zone detector. The member function LightsOut simulates the events in the detector.

Definition at line 23 of file ReactorDetector.hh.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 ReactorDetector::ReactorDetector ()

ReactorDetector constructor.

Called by ReactorFsim to simulate a spherical, three-zone detector. All of the PMT and gamma arrays are initialized and the nominal detector parameters are set (by SetNominalParameters) and the PMTs are arrayed (by SetPMT).

Definition at line 26 of file ReactorDetector.cpp.

```
                              {
  //
  // initialize PMT arrays
  //
  Xpmt = new double[0];
  Ypmt = new double[0];
  Zpmt = new double[0];
  PMTOutputArray = new PMToutput*[0];
  PMTdata = new PMT[0];
  PMTelectron = new PMT[0];
  PMTpositron = new PMT[0];
  PMTneutron = new PMT[0];
  PMTthallium = new PMT[0];
  PMTbismuth = new PMT[0];
  PMTcalifornium = new PMT[0];
  PMTmuon = new PMT[0];
  //
  // Set parameters
  //
  SetNominalParameters();
  SetPMT();
  SetNeutronPropStuff();
  //
  // initialize gamma arrays
  //
  Ngamma = 0;
  Rgamma = new double[Ngamma];
  Egamma = new double[Ngamma];
  Tgamma = new double[Ngamma];
  Ogamma = new std::string[Ngamma];
  //
  // initialize PMT charge spectrum
  //
  float Qlo = 0.001;
  float Qhi = 10.0;
  funlxp_(QSpectrum,Qspace,Qlo,Qhi);

  Nneutron = 0;
  Absorber = new double[Nneutron];
  NeutronSteps = new double[Nneutron];

  pmtRadCount = 0;
  MergeTime = 0;
}
```

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

- **ReactorDetector.hh**
- **ReactorDetector.cpp**

3.8 ReactorEvent Class Reference

ReactorEvent handles all of the event generation in the ReactorFsim simulation.

`#include <ReactorEvent.hh>`

Public Methods

- **ReactorEvent** (int newNevents, double newRmaxgen=RMAXGEN, double newEmin=Emin, double newEmax=Emax, int newXCorder=XCorder)

  ReactorEvent constructor.

- ~**ReactorEvent** ()

  ReactorEvent destructor.

- **ReactorEvent** (const ReactorEvent &Event)

  ReactorEvent copy constructor.

- ReactorEvent & **operator=** (const ReactorEvent &rhs)

  ReactorEvent overloaded = operator.

- double **XCReweight** ()
- double **XCReweight** (double Enu)
- double **XCReweight** (ReactorEvent &Event)
- std::string **GetPmtRad** () const

  Returns the PMT radiation state: ON/OFF.

- int **GetNevent** () const

  Returns the number of events.

- int **GetNumNubar** () const

  Returns the number of anti-neutrinos.

- int **GetNumPositron** () const

  Returns the number of positrons.

- int **GetNumElectron** () const

  Returns the number of electrons.

- int **GetNumNeutron** () const

  Returns the number of neutrons.

- int **GetNumGamma** () const

  Returns the number of gammas.

- double **GetXCWeight** () const
- double **GetFluxWeight** () const
- double **GetEnu** (int &n) const

  Returns the energy of each anti-neutrino produced in the event.

- double **GetNubarParentProcess** (int &n) const

  Returns the parent process used to generate each anti-neutrino in the event.

- double **GetElectronKE** (int &n) const

  Returns the energy of each electron produced in the event.

- double **GetElectronCosTheta** (int &n) const
- double **GetElectronPhi** (int &n) const
- double **GetElectronParentProcess** (int &n) const

  Returns the parent process used to generate each electron in the event.

- double *const **GetElectronPxyz** ()
- double **GetPositronKE** (int &n) const

  Returns the energy of each positron produced in the event.

- double **GetPositronCosTheta** (int &n) const
- double **GetPositronPhi** (int &n) const
- double **GetPositronParentProcess** (int &n) const

  Returns the parent process used to generate each positron in the event.

- double *const **GetPositronPxyz** ()
- double **GetNeutronKE** (int &n) const

  Returns the energy of each neutron produced in the event.

- double **GetNeutronCosTheta** (int &n) const
- double **GetNeutronPhi** (int &n) const
- double **GetNeutronParentProcess** (int &n) const

  Returns the parent process used to generate each neutron in the event.

- double *const **GetNeutronPxyz** ()
- double **GetGammaKE** (int &n) const

  Returns the energy of each photon produced in the event.

- double **GetGammaParentProcess** (int &n) const

  Returns the parent process used to generate each photon in the event.

- double **GetNubarTime** (int &n) const
- double **GetElectronTime** (int &n) const
- double **GetPositronTime** (int &n) const
- double **GetNeutronTime** (int &n) const
- double **GetGammaTime** (int &n) const
- double **GetNubarVertexR** (int &n) const
- double **GetNubarVertexCosTheta** (int &n) const
- double **GetNubarVertexPhi** (int &n) const
- double *const **GetNubarVertexXYZ** ()
- double **GetElectronVertexR** (int &n) const
- double **GetElectronVertexCosTheta** (int &n) const
- double **GetElectronVertexPhi** (int &n) const
- double *const **GetElectronVertexXYZ** ()
- double **GetPositronVertexR** (int &n) const

- double **GetPositronVertexCosTheta** (int &n) const
- double **GetPositronVertexPhi** (int &n) const
- double *const **GetPositronVertexXYZ** ()
- double **GetNeutronVertexR** (int &n) const
- double **GetNeutronVertexCosTheta** (int &n) const
- double **GetNeutronVertexPhi** (int &n) const
- double *const **GetNeutronVertexXYZ** ()
- double **GetGammaVertexR** (int &n) const
- double **GetGammaVertexCosTheta** (int &n) const
- double **GetGammaVertexPhi** (int &n) const
- double *const **GetGammaVertexXYZ** ()
- double **GetGeneratedElectronTime** (int &n) const
- double **GetGeneratedElectronVertexR** (int &n) const
- double **GetGeneratedElectronVertexCosTheta** (int &n) const
- double **GetGeneratedElectronVertexPhi** (int &n) const
- double *const **GetGeneratedElectronVertexXYZ** ()
- double **GetGeneratedPositronTime** (int &n) const
- double **GetGeneratedPositronVertexR** (int &n) const
- double **GetGeneratedPositronVertexCosTheta** (int &n) const
- double **GetGeneratedPositronVertexPhi** (int &n) const
- double *const **GetGeneratedPositronVertexXYZ** ()
- double **GetGeneratedNeutronTime** (int &n) const
- double **GetGeneratedNeutronVertexR** (int &n) const
- double **GetGeneratedNeutronVertexCosTheta** (int &n) const
- double **GetGeneratedNeutronVertexPhi** (int &n) const
- double *const **GetGeneratedNeutronVertexXYZ** ()
- void **SetPositronVertex** (int &n, double *rpos)
- void **SetNeutronVertex** (int &n, double *rneu)

Public Attributes

- std::string **eventType**
- std::string **pmtRad**
- int **Pdim**
- int **Rdim**

3.8.1 Detailed Description

ReactorEvent handles all of the event generation in the ReactorFsim simulation.

It first sets up the event by reading in the type of event from event_setup.txt, the event setup file. The three types of event currently supported are "reactor," "cf252," and "all." For more on event_setup.txt, see **event_readme.txt**.

The "reactor" setting generates an anti-neutrino, positron, neutron from the inverse-beta decay reaction: nubar + proton -> positron + neutron. The "cf252" setting generates a neutron from a Cf252 decay. The "all" setting mixes both types of event.

Definition at line 26 of file ReactorEvent.hh.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 ReactorEvent::ReactorEvent (int *newNevents*, double *newRmaxgen* = RMAXGEN, double *newEmin* = Emin, double *newEmax* = Emax, int *newXCorder* = XCorder)

ReactorEvent constructor.

Called with the number of generated events set in **ReactorFsim.cpp**, the maximum allowed radius for generating particles in the simulation, Rmaxgen (default = 200 cm), and the minimum and maximum allowed energies, respectively, for the generated nubar in the inverse-beta decay reaction. The default minimum energy is 1.85; the maximum is 8.15, set in **ReactorConstants.hh**.

Definition at line 21 of file ReactorEvent.cpp.

References MyCfSource::GetCfGammaEnergy, MyCfSource::GetCfGammaTime, MyCfSource::GetCfNeutronEnergy, MyCfSource::GetCfNeutronTime, MyCfSource::GetCfVertexCosTheta, MyCfSource::GetCfVertexPhi, MyCfSource::GetCfVertexR, MuonPropagator::GetElectronKE, MuonPropagator::GetElectronTime, MuonPropagator::GetElectronVertexXYZ, MuonPropagator::GetGammaKE, MuonPropagator::GetGammaTime, MuonPropagator::GetGammaVertexXYZ, MuonPropagator::GetNeutronKE, MuonPropagator::GetNeutronTime, MuonPropagator::GetNeutronVertexXYZ, MyCfSource::GetNumCfSources, MuonPropagator::GetNumElectron, MuonPropagator::GetNumGamma, MyCfSource::GetNumGamma, MuonPropagator::GetNumNeutron, MyCfSource::GetNumNeutron, and MuonPropagator::GetWeight.

```
                                      {

  Pdim = 7;
  Rdim = 7;

  Nevents = newNevents;
  Rmaxgen_RE = newRmaxgen;
  Emin_RE = newEmin;
  Emax_RE = newEmax;
  XCorder_RE = newXCorder;

  /*
  cout << "ReactorEvent::ReactorEvent(" << Nevents << ","
       << Rmaxgen_RE << "," << Emin << "," << Emax << ","
       << XCorder_RE << ")" << endl;
  */

  int nmax = Ndim;
  int nnu  = 0;
  int npos = 0;
  int nele = 0;
  int nneu = 0;
  int ngam = 0;

  static double FracDone;
  static double FracDisplay;
  static double FracLastDisplay;
  static double NeventLastDisplay;
  static bool display;

  double* pnubar = new double[nmax*Pdim];
  double* pelectron = new double[nmax*Pdim];
  double* ppositron = new double[nmax*Pdim];
  double* pneutron = new double[nmax*Pdim];
  double* pgamma = new double[nmax*Pdim];
```

```

double* pnu  = pnubar;
double* pele = pelectron;
double* ppos = ppositron;
double* pneu = pneutron;
double* pgam = pgamma;

double* rnubar = new double[nmax*Rdim];
double* relectron = new double[nmax*Rdim];
double* rgenele = new double[nmax*Rdim];
double* rpositron = new double[nmax*Rdim];
double* rgenpos = new double[nmax*Rdim];
double* rneutron = new double[nmax*Rdim];
double* rgenneu = new double[nmax*Rdim];
double* rgamma = new double[nmax*Rdim];

double* rnu  = rnubar;
double* rele = relectron;
double* rge  = rgenele;
double* rpos = rpositron;
double* rgp  = rgenpos;
double* rneu = rneutron;
double* rgn  = rgenneu;
double* rgam = rgamma;

double Enu,z,phi;
double R,zR,phiR;

// setup the event
fstream fin("src/event_setup.txt", ios::in);
//cout << "setup file: src/event_setup.txt" << endl;

char paramLine[256];
std::string dummy;
bool done = false;
bool set = false;
std::string static eventtype;
std::string static pmtrad;
double static Depth;
bool static first = true;

 if(first){
   while(!done){

     // '#' specifies a comment line -- ignore it
     if(fin.peek() == '#'){

       fin.getline(paramLine, 256);
       continue;
     }

     // 'E'ND or 'e'nd specifies end of file -- set done flag to true
     if(fin.peek() == 'e' || fin.peek() == 'E'){

       done = true;
       continue;
     }

     // 'K'IND or 'k'ind specifies the event type
     if(fin.peek() == 'k' || fin.peek() == 'K'){

       if(set){

         fin.getline( paramLine, 256 );
         continue;
```

```
      }
      fin >> dummy;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> eventtype;

      if(eventtype == "reactor" || eventtype == "cf252" ||
         eventtype == "pmtrad"  || eventtype == "muon" ||
         eventtype == "all"     || eventtype == "veto"){
        set = true;
      }
      else{
        cout << "SETUP ERROR: unrecognized event type" << eventtype << endl;
      }

      continue;
    }

    // 'P'MTRAD or 'p'mtrad turns on/off radioactive decays in the PMTs
    if(fin.peek() == 'p' || fin.peek() == 'P'){

      fin >> dummy;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> pmtrad;
      continue;
    }

    // 'D'EPTH or 'd'epth sets the detector depth in mwe
    if(fin.peek() == 'd' || fin.peek() == 'D'){

      fin >> dummy;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> Depth;
      continue;
    }

    // skip n, o, r, t, a, b, m, and f
    if(fin.peek() == 'N' || fin.peek() == 'n' ||
       fin.peek() == 'O' || fin.peek() == 'o' ||
       fin.peek() == 'R' || fin.peek() == 'r' ||
       fin.peek() == 'T' || fin.peek() == 't' ||
       fin.peek() == 'A' || fin.peek() == 'a' ||
       fin.peek() == 'B' || fin.peek() == 'b' ||
       fin.peek() == 'M' || fin.peek() == 'm' ||
       fin.peek() == 'F' || fin.peek() == 'f'){

      fin.getline(paramLine, 256);
      continue;
    }

    // ignore extra whitespace
    if(fin.peek() == ' ' || fin.peek() == '\n'){

      fin.ignore(1);
      continue;
    }
  }

  // log file printout
  cout << "=============================" << endl;
```

```
    cout << "  " << eventtype << " type of events" << endl;
    cout << "  PMT radioactivity is " << pmtrad << endl;
    cout << "  Detector depth is " << Depth << " mwe" << endl;
    cout << "=============================" << endl;

    // zero the event counter
    Nevent = 0;

    first = false;
  }
  eventType = eventtype;
  pmtRad = pmtrad;

  // report fraction of processed events
  display = 0;
  if (Nevent==0) {
    // Set up fractions for progress checking
    FracDisplay = 0.01;
    NeventLastDisplay = 0.;
    FracLastDisplay = 0.;
    FracDone = 0.;
    display = 1;
  }
  //
  // report every 1% of processed events
  //
  FracDone = (double) Nevent / (double) Nevents;
  if (FracDone >= 0.){
    if (display || (FracDone-FracLastDisplay) >= FracDisplay){
      cout << Nevent << " events (" << 100.*FracDone
           << "%) processed" << endl;
      FracLastDisplay += FracDisplay;
    }
  }
  else {
    if (display || Nevent-NeventLastDisplay >= FracDisplay*NeventLastDisplay){
      cout << Nevent << " events processed" << endl;
    }
  }
  //
  // count the event
  //
  Nevent++;

  //  if(Nevent%1000 == 0) cout << Nevent << " events processed" << endl;
  //
  // if all events are requested, mix the 'reactor' and 'cf252'
  // events together with 90/10% Cf252/reactor events
  //
  if(eventType == "all"){
    const int len = 1;
    float r[len];
    ranlux_(r,len);

    if(r[0] < 0.90){
      eventType = "cf252";
    }
    else{
      eventType = "reactor";
    }
  }
  //cout << eventType << " type of events" << endl;
  //cout << "PMT radiation " << pmtRad << endl;

  if(eventType == "reactor"){
```

```

XCWeight = 1;

// inverse beta decay reaction
double XCmax = InverseBeta(Emax_RE,-1);
double FluxMax = RMPflux(Emin_RE);

bool passed=0;

while(!passed){
  const int len = 7;
  float r[len];
  ranlux_(r,len);
  Enu = Emin_RE+(Emax_RE-Emin_RE)*r[0];
  z = -1+2*r[1];
  phi = 2*PI*r[2];

  R = Rmaxgen_RE*exp(log(r[3])/3);
  zR = -1+2*r[4];
  phiR = 2*PI*r[5];
  //      cout << R << " " << Rmaxgen_RE << " " << r[3] << endl;

  float XCtest = XCmax*FluxMax*r[6];
  XCWeight = InverseBeta(Enu,z);
  FluxWeight=RMPflux(Enu);
  passed= XCWeight*FluxWeight>XCtest;
}
//      cout << "passed " << R << " " << Rmaxgen_RE << " "<< endl;


double E0 = Enu - DELTA;
double p0 = sqrt(E0*E0-MELECTRON*MELECTRON);
double v0 = p0/E0;
double Ysquared = (DELTA*DELTA-MELECTRON*MELECTRON)/2;
double E1 = E0*(1-Enu/MPROTON*(1-v0*z))-Ysquared/MPROTON;
double p1 = sqrt(E1*E1-MELECTRON*MELECTRON);

*pnu = Enu; pnu++;
*pnu = 0; pnu++;
*pnu = 0; pnu++;
*pnu = Enu; pnu++;
*pnu = Enu; pnu++;
*pnu = Enu; pnu++;
*pnu = 1.; pnu++; // inverse beta decay process Id = 1

*ppos = E1; ppos++;
*ppos = p1*sqrt(1-z*z)*cos(phi); ppos++;
*ppos = p1*sqrt(1-z*z)*sin(phi); ppos++;
*ppos = p1*z; ppos++;
*ppos = E1-MELECTRON; ppos++;
*ppos = p1; ppos++;
*ppos = 1.; ppos++;

*pneu = Enu+MPROTON-E1; pneu++;
*pneu = -p1*sqrt(1-z*z)*cos(phi); pneu++;
*pneu = -p1*sqrt(1-z*z)*sin(phi); pneu++;
*pneu = Enu-p1*z; pneu++;
*pneu = Enu-E1-DELTA; pneu++;
*pneu = sqrt((Enu+MPROTON-E1)*(Enu+MPROTON-E1)-
                    (MPROTON+DELTA)*(MPROTON+DELTA)); pneu++;
*pneu = 1.; pneu++;

*rnu = 0; rnu++;
*rnu = R*sqrt(1-zR*zR)*cos(phiR); rnu++;
*rnu = R*sqrt(1-zR*zR)*sin(phiR); rnu++;
```

```
    *rnu = R*zR; rnu++;
    *rnu = R; rnu++;
    *rnu = zR; rnu++;
    *rnu = phiR; rnu++;

    *rpos = 0; rpos++;
    *rpos = R*sqrt(1-zR*zR)*cos(phiR); rpos++;
    *rpos = R*sqrt(1-zR*zR)*sin(phiR); rpos++;
    *rpos = R*zR; rpos++;
    *rpos = R; rpos++;
    *rpos = zR; rpos++;
    *rpos = phiR; rpos++;

    *rgp = 0; rgp++;
    *rgp = R*sqrt(1-zR*zR)*cos(phiR); rgp++;
    *rgp = R*sqrt(1-zR*zR)*sin(phiR); rgp++;
    *rgp = R*zR; rgp++;
    *rgp = R; rgp++;
    *rgp = zR; rgp++;
    *rgp = phiR; rgp++;

    *rneu = 0; rneu++;
    *rneu = R*sqrt(1-zR*zR)*cos(phiR); rneu++;
    *rneu = R*sqrt(1-zR*zR)*sin(phiR); rneu++;
    *rneu = R*zR; rneu++;
    *rneu = R; rneu++;
    *rneu = zR; rneu++;
    *rneu = phiR; rneu++;

    *rgn = 0; rgn++;
    *rgn = R*sqrt(1-zR*zR)*cos(phiR); rgn++;
    *rgn = R*sqrt(1-zR*zR)*sin(phiR); rgn++;
    *rgn = R*zR; rgn++;
    *rgn = R; rgn++;
    *rgn = zR; rgn++;
    *rgn = phiR; rgn++;

    // count the nubar, positron, and neutron
    nnu++;
    npos++;
    nneu++;

  } // done with reactor neutrino inverse-beta reaction

  // Cf source reaction
  else if(eventType == "cf252"){
    XCWeight = 1;
    FluxWeight = 1;

    int Isotope = 252;
    MyCfSource CfSource(Isotope,RMAXGEN);

    int Nsources = CfSource.GetNumCfSources();
    //cout << "Number of Cf252 sources = " << Nsources << endl;

    for(int n=0;n<Nsources;n++){

      R = CfSource.GetCfVertexR(n);
      zR = CfSource.GetCfVertexCosTheta(n);
      phiR = CfSource.GetCfVertexPhi(n);

      // loop over the neutrons
      for(int nn=0;nn<CfSource.GetNumNeutron(n);nn++){

        double neutronKE = CfSource.GetCfNeutronEnergy(n,nn);
```

```
        //cout << "   Cf252 source " << n << ", neutron " << nn
        //      << " energy = " << neutronKE << endl;

        double Tneu = CfSource.GetCfNeutronTime(n,nn);
        double Mneutron = MPROTON+DELTA;
        double neutronE = neutronKE+Mneutron;

        // pick a direction for neutron momentum
        const int len = 2;
        float r[len];
        ranlux_(r,len);
        z = -1+2*r[0];
        phi = 2*PI*r[1];
        double x = sqrt(1-z*z)*cos(phi);
        double y = sqrt(1-z*z)*sin(phi);

        *pneu = neutronE; pneu++;
        *pneu = sqrt((neutronE*neutronE)-(Mneutron*Mneutron))*x; pneu++;
        *pneu = sqrt((neutronE*neutronE)-(Mneutron*Mneutron))*y; pneu++;
        *pneu = sqrt((neutronE*neutronE)-(Mneutron*Mneutron))*z; pneu++;
        *pneu = neutronKE; pneu++;
        *pneu = sqrt(neutronE*neutronE-Mneutron*Mneutron); pneu++;
        *pneu = 2.; pneu++; // cf252 source process Id = 2

        *rneu = Tneu; rneu++;
        *rneu = R*sqrt(1-zR*zR)*cos(phiR); rneu++;
        *rneu = R*sqrt(1-zR*zR)*sin(phiR); rneu++;
        *rneu = R*zR; rneu++;
        *rneu = R; rneu++;
        *rneu = zR; rneu++;
        *rneu = phiR; rneu++;

        *rgn = Tneu; rgn++;
        *rgn = R*sqrt(1-zR*zR)*cos(phiR); rgn++;
        *rgn = R*sqrt(1-zR*zR)*sin(phiR); rgn++;
        *rgn = R*zR; rgn++;
        *rgn = R; rgn++;
        *rgn = zR; rgn++;
        *rgn = phiR; rgn++;
      }
      // count the neutrons
      nneu += CfSource.GetNumNeutron(n);

      // loop over the prompt gammas (these are real photons and
      // need to be compton scattered in the scint)
      //
      for(int nn=0;nn<CfSource.GetNumGamma(n);nn++){

        double gammaKE = CfSource.GetCfGammaEnergy(n,nn);
        //cout << "   Cf252 source " << n << ", gamma " << nn
        //      << " energy = " << gammaKE << endl;

        double Tgam = CfSource.GetCfGammaTime(n,nn);

        // compton scatter the gammas
        //
        const int len = 5;
        float rv[len];
        ranlux_(rv,len);
        double cosg = -1+2*rv[0];
        double phig = 2*PI*rv[1];
        double t[3];
        t[0] = cos(phig)*sqrt(1-cosg*cosg);
        t[1] = sin(phig)*sqrt(1-cosg*cosg);
        t[2] = cosg;
```

```
        //for(int i=0;i<3;i++) cout << "t["<<i<<"] " << t[i] << " ";
        //cout << endl;

        double range;
        std::string material;
        if(R < Rmaxgen_RE)
          material = "scintillator";
        else
          material = "oil";
        range = GammaComptonRange(material,gammaKE);
        range*=log(1/rv[2]);
        //cout << "range " << range << endl;
        //
        // get the new x,y,z positions from source R,zR,phiR
        //
        double xpos = R*sqrt(1-zR*zR)*cos(phiR) + range*t[0];
        double ypos = R*sqrt(1-zR*zR)*sin(phiR) + range*t[1];
        double zpos = R*zR + range*t[2];
        //
        // now convert back to R,zR,phiR
        //
        R = sqrt(xpos*xpos + ypos*ypos + zpos*zpos);
        zR = zpos/R;
        phiR = atan2(ypos,xpos);
        if(phiR < 0) phiR += 2*PI;
        //
        // pick a direction for gamma momentum
        //
        float r[len];
        ranlux_(r,len);
        z = -1+2*rv[3];
        phi = 2*PI*rv[4];
        double x = sqrt(1-z*z)*cos(phi);
        double y = sqrt(1-z*z)*sin(phi);

        *pgam = gammaKE; pgam++;
        *pgam = gammaKE*x; pgam++;
        *pgam = gammaKE*y; pgam++;
        *pgam = gammaKE*z; pgam++;
        *pgam = gammaKE; pgam++;
        *pgam = gammaKE; pgam++;
        *pgam = 2.; pgam++; // cf252 source process Id = 2

        *rgam = Tgam; rgam++;
        *rgam = xpos; rgam++;
        *rgam = ypos; rgam++;
        *rgam = zpos; rgam++;
        *rgam = R; rgam++;
        *rgam = zR; rgam++;
        *rgam = phiR; rgam++;
      }
      //
      ngam += CfSource.GetNumGamma(n);

    } // done looping over sources

  } // done with cf source neutron production

  // PMT radiation
  else if(eventType == "pmtrad"){

    XCWeight = 1;
    FluxWeight = 1;

    // nothing happens in these events but photons, generated in
```

```
    // ReactorDetector::GenerateGammas with pmtRad on
    if(pmtRad == "off"){
      cout << "SETUP ERROR: PMT radiation set " << pmtRad << " in a "
           << eventType << " type event" << endl;
      pmtRad = "on";
      cout << "Forcing PMT radiation " << pmtRad << endl;
    }
  }

  // muons
  else if(eventType == "muon"){

    //cout << eventType << endl;

    // create the energy and cosine of zenith angle (theta) arrays
    bool static first = true;
    double static Energy[Mdim],CosTheta[Mdim];
    double dummy;

    if(first){

      // read in the muon energy in GeV and angle
      if(Depth == 300.){
        fstream fin("data/Totb300a.dat", ios::in);
        cout << "setup file: data/Totb300a.dat" << endl;

        for(int i=0;i<Mdim;i++){
          fin >> dummy >> Energy[i] >> dummy >> dummy >> dummy >> CosTheta[i];
          //cout << i << "  " << Energy[i] << "  " << CosTheta[i] << endl;
        }
        printf("Read %d input records\n",Mdim);
      }
      else if(Depth == 450.){
        fstream fin("data/Totb450a.dat", ios::in);
        cout << "setup file: data/Totb450a.dat" << endl;

        for(int i=0;i<Mdim;i++){
          fin >> dummy >> Energy[i] >> dummy >> dummy >> dummy >> CosTheta[i];
          //cout << i << "  " << Energy[i] << "  " << CosTheta[i] << endl;
        }
        printf("Read %d input records\n",Mdim);
      }
      else if(Depth == 500.){
        fstream fin("data/Totb500a.dat", ios::in);
        cout << "setup file: data/Totb500a.dat" << endl;

        for(int i=0;i<Mdim;i++){
          fin >> dummy >> Energy[i] >> dummy >> dummy >> dummy >> CosTheta[i];
          //cout << i << "  " << Energy[i] << "  " << CosTheta[i] << endl;
        }
        printf("Read %d input records\n",Mdim);
      }
      else{
        cout << "ERROR: no muon data for depth of "
             << Depth << " mwe" << endl;
        return;
      }

      first = false;
    }

    /* get the muon energy and cosine of the zenith angle
       the spectra in the Tot dat files are random so start
       with the first entry and cycle through the file
    */
```

```
    static int index = 0;
    double energy = 0.;
    double cosg = 0.;
    //
    if(index < Mdim){
      //
      // muon energy in MeV
      energy = Energy[index]*1000.;
      cosg = CosTheta[index];
      //cout << " muon energy " << energy
      //   << " MeV, cos(theta) " << cosg << endl;
      index++;
    }
    else{
      cout << "  WARNING: end of muon input file reached at index " << index
           << ".  Starting over." << endl;
      index = 0;
    }
    //
    // call the propagator, which returns particle information
    //
    MuonPropagator Muon(RMAXGEN,Depth,energy,cosg);

    XCWeight = Muon.GetWeight();
    FluxWeight = 1;
    //
    // loop over neutrons
    //
    nneu = Muon.GetNumNeutron();
    for(int n=0;n<nneu;n++){

      float neutronKE = Muon.GetNeutronKE(n);
      //cout << "  neutron " << n << " energy = " << neutronKE << endl;

      float Mneutron = MPROTON+DELTA;
      float neutronE = neutronKE+Mneutron;

      // pick a direction for neutron momentum
      const int len = 2;
      float rv[len];
      ranlux_(rv,len);
      z = -1+2*rv[0];
      phi = 2*PI*rv[1];
      double x = sqrt(1-z*z)*cos(phi);
      double y = sqrt(1-z*z)*sin(phi);

      *pneu = neutronE; pneu++;
      *pneu = sqrt((neutronE*neutronE)-(Mneutron*Mneutron))*x; pneu++;
      *pneu = sqrt((neutronE*neutronE)-(Mneutron*Mneutron))*y; pneu++;
      *pneu = sqrt((neutronE*neutronE)-(Mneutron*Mneutron))*z; pneu++;
      *pneu = neutronKE; pneu++;
      *pneu = sqrt(neutronE*neutronE-Mneutron*Mneutron); pneu++;
      *pneu = 3.; pneu++; // muon process Id = 3

      double neuxyz[3];
      for(int i=0;i<3;i++){
        neuxyz[i] = *(Muon.GetNeutronVertexXYZ()+(i+n));
        //cout << "  neuxyz[" << i << "] " << neuxyz[i] << endl;
      }
      double rr = sqrt(neuxyz[0]*neuxyz[0] +
                       neuxyz[1]*neuxyz[1] +
                       neuxyz[2]*neuxyz[2]);
      double rphi = atan2(neuxyz[1],neuxyz[0]);
      if(rphi<0) rphi += 2*PI;

```

```
      double time = Muon.GetNeutronTime(n);
      //cout << " n time " << time << endl;

      *rneu = time; rneu++;
      *rneu = neuxyz[0]; rneu++;
      *rneu = neuxyz[1]; rneu++;
      *rneu = neuxyz[2]; rneu++;
      *rneu = rr; rneu++;
      *rneu = neuxyz[2]/rr; rneu++;
      *rneu = rphi; rneu++;

      *rgn = time; rgn++;
      *rgn = neuxyz[0]; rgn++;
      *rgn = neuxyz[1]; rgn++;
      *rgn = neuxyz[2]; rgn++;
      *rgn = rr; rgn++;
      *rgn = neuxyz[2]/rr; rgn++;
      *rgn = rphi; rgn++;
    }
    //
    // loop over electrons
    //
    nele = Muon.GetNumElectron();
    for(int n=0;n<nele;n++){

      float electronKE = Muon.GetElectronKE(n);
      //cout << "  electron " << n << " KE = " << electronKE << endl;

      float electronE = electronKE+MELECTRON;
      //cout << "  electron " << n << " energy = " << electronE << endl;

      // pick a direction for electron momentum
      const int len = 2;
      float r[len];
      ranlux_(r,len);
      z = -1+2*r[0];
      phi = 2*PI*r[1];
      double x = sqrt(1-z*z)*cos(phi);
      double y = sqrt(1-z*z)*sin(phi);

      *pele = electronE; pele++;
      *pele = sqrt((electronE*electronE)-(MELECTRON*MELECTRON))*x; pele++;
      *pele = sqrt((electronE*electronE)-(MELECTRON*MELECTRON))*y; pele++;
      *pele = sqrt((electronE*electronE)-(MELECTRON*MELECTRON))*z; pele++;
      *pele = electronKE; pele++;
      *pele = sqrt(electronE*electronE-MELECTRON*MELECTRON); pele++;
      *pele = 3.; pele++; // muon process Id = 3

      double elexyz[3];
      for(int i=0;i<3;i++){
        elexyz[i] = *(Muon.GetElectronVertexXYZ()+(i+3*n));
        //cout << "  elexyz[" << i << "] " << elexyz[i] << endl;
      }
      double rr = sqrt(elexyz[0]*elexyz[0] +
                       elexyz[1]*elexyz[1] +
                       elexyz[2]*elexyz[2]);
      double rphi = atan2(elexyz[1],elexyz[0]);
      if(rphi<0) rphi += 2*PI;

      double time = Muon.GetElectronTime(n);
      //cout << " e time " << time << endl;

      *rele = time; rele++;
      *rele = elexyz[0]; rele++;
      *rele = elexyz[1]; rele++;
```

```
      *rele = elexyz[2]; rele++;
      *rele = rr; rele++;
      *rele = elexyz[2]/rr; rele++;
      *rele = rphi; rele++;

      *rge = time; rge++;
      *rge = elexyz[0]; rge++;
      *rge = elexyz[1]; rge++;
      *rge = elexyz[2]; rge++;
      *rge = rr; rge++;
      *rge = elexyz[2]/rr; rge++;
      *rge = rphi; rge++;
    }
    //
    // loop over photons (these are 'virtual' to contain the energy
    // loss of the muon and therefore are not compton scattered)
    //
    ngam = Muon.GetNumGamma();
    for(int n=0;n<ngam;n++){

      float gammaKE = Muon.GetGammaKE(n);
      //cout << "  gamma " << n << " energy = " << gammaKE << endl;

      // pick a direction for gamma momentum
      const int len = 2;
      float r[len];
      ranlux_(r,len);
      z = -1+2*r[0];
      phi = 2*PI*r[1];
      double x = sqrt(1-z*z)*cos(phi);
      double y = sqrt(1-z*z)*sin(phi);

      *pgam = gammaKE; pgam++;
      *pgam = gammaKE*x; pgam++;
      *pgam = gammaKE*y; pgam++;
      *pgam = gammaKE*z; pgam++;
      *pgam = gammaKE; pgam++;
      *pgam = gammaKE; pgam++;
      *pgam = 3.; pgam++; // muon process Id = 3

      double gamxyz[3];
      for(int i=0;i<3;i++){
        gamxyz[i] = *(Muon.GetGammaVertexXYZ()+(i+3*n));
        //cout << "  gamxyz[" << i << "] " << gamxyz[i] << endl;
      }
      double rr = sqrt(gamxyz[0]*gamxyz[0] +
                       gamxyz[1]*gamxyz[1] +
                       gamxyz[2]*gamxyz[2]);
      double rphi = atan2(gamxyz[1],gamxyz[0]);
      if(rphi<0) rphi += 2*PI;

      double time = Muon.GetGammaTime(n);

      *rgam = time; rgam++;
      *rgam = gamxyz[0]; rgam++;
      *rgam = gamxyz[1]; rgam++;
      *rgam = gamxyz[2]; rgam++;
      *rgam = rr; rgam++;
      *rgam = gamxyz[2]/rr; rgam++;
      *rgam = rphi; rgam++;
    }
  }

  // veto
  else if(eventType == "veto"){
```

```
  
  //cout << eventType << endl;
  
  // read in the veto file
  fstream fveto("data/testveto.ascii", ios::in);
  //cout << "setup file: data/testveto.ascii" << endl;
  
  // convert the event number to a character string
  int localeventsize = 0;
  if(Nevent < 10)
    localeventsize = 1;
  else if(Nevent > 9 && Nevent < 100)
    localeventsize = 2;
  else if(Nevent > 99 && Nevent < 1000)
    localeventsize = 3;
  else if(Nevent > 999 && Nevent < 10000)
    localeventsize = 4;
  else if(Nevent > 9999 && Nevent < 100000)
    localeventsize = 5;
  else if(Nevent > 99999 && Nevent < 1000000)
    localeventsize = 6;
  else if(Nevent > 999999 && Nevent < 10000000)
    localeventsize = 7;
  else
    localeventsize = 0;
  
  char *localevent = new char[localeventsize];
//    char localevent[localeventsize];
  sprintf(localevent,"%i",Nevent);
  //cout << "localevent " << localevent << endl;
  
  // counter
  int Npart = 0;
  // number of variables per particle from veto: id,x,y,z,vx,vy,vz,ke
  int Vdim = 8;
  
  double* VetoData = new double[Ndim*Vdim];
  
  bool edone = false;
  while(!edone){
    //
    // get the veto event with the matching event number
    //
    std::string vetoevent;
    getline(fveto,vetoevent);
    //cout << " vetoevent " << vetoevent << endl;
    //
    if(vetoevent == localevent){
      
      bool pdone = false;
      while(!pdone){
        //
        // get the particle(s) information
        //
        if(fveto.peek() == '0'){
          //
          // end of particle list
          //
          edone = true;
          pdone = true;
        }
        else if(fveto.peek() == ' '){
          //
          // a particle, info is: id,x,y,z,vx,vy,vz,ke
          //
```

```
          for(int i=0;i<Vdim;i++) fveto >> VetoData[i+Vdim*Npart];
          //
          // close out any remaining whitespace
          //
          std::string dummy;
          getline(fveto,dummy);
          //
          Npart++;
        }
      } // close loop over particles in veto event
    }
  } // close loop over veto events

  /*
  cout << Npart << " veto particle(s)" << endl;
  for(int n=0;n<Npart*Vdim;n++){
    cout << " VetoData[" << n << "] " << VetoData[n] << endl;
  }
  */

  //
  // sort particles into R and P arrays
  //
  for(int n=0;n<Npart;n++){
    //
    // PID follows the PDG Monte Carlo numbering scheme
    //
    int particleid = int(VetoData[n*Vdim]);
    //
    if(particleid == 11){
      //
      // electron
      //
      nele++;

      // electron KE and total E in MeV
      double electronKE = VetoData[n*Vdim+7]*1000.;
      double electronE = electronKE+MELECTRON;

      *pele = electronE; pele++;
      *pele = VetoData[n*Vdim+4]*1000.; pele++;
      *pele = VetoData[n*Vdim+5]*1000.; pele++;
      *pele = VetoData[n*Vdim+6]*1000.; pele++;
      *pele = electronKE; pele++;
      *pele = sqrt(electronE*electronE-MELECTRON*MELECTRON); pele++;
      *pele = 4.; pele++; // veto process Id = 4

      double elexyz[3];
      for(int i=0;i<3;i++){
        elexyz[i] = VetoData[n*Vdim+(i+1)];
        //cout << "  elexyz[" << i << "] " << elexyz[i] << endl;
      }
      double rr = sqrt(elexyz[0]*elexyz[0] +
                       elexyz[1]*elexyz[1] +
                       elexyz[2]*elexyz[2]);
      double rphi = atan2(elexyz[1],elexyz[0]);
      if(rphi<0) rphi += 2*PI;

      double time = 0.;

      *rele = time; rele++;
      *rele = elexyz[0]; rele++;
      *rele = elexyz[1]; rele++;
      *rele = elexyz[2]; rele++;
      *rele = rr; rele++;
```

```
      *rele = elexyz[2]/rr; rele++;
      *rele = rphi; rele++;

      *rge = time; rge++;
      *rge = elexyz[0]; rge++;
      *rge = elexyz[1]; rge++;
      *rge = elexyz[2]; rge++;
      *rge = rr; rge++;
      *rge = elexyz[2]/rr; rge++;
      *rge = rphi; rge++;
    }
    else if(particleid == -11){
      //
      // positron
      //
      npos++;

      // positron KE and total E in MeV
      double positronKE = VetoData[n*Vdim+7]*1000.;
      double positronE = positronKE+MELECTRON;

      *ppos = positronE; ppos++;
      *ppos = VetoData[n*Vdim+4]*1000.; ppos++;
      *ppos = VetoData[n*Vdim+5]*1000.; ppos++;
      *ppos = VetoData[n*Vdim+6]*1000.; ppos++;
      *ppos = positronKE; ppos++;
      *ppos = sqrt(positronE*positronE-MELECTRON*MELECTRON); ppos++;
      *ppos = 4.; ppos++; // veto process Id = 4

      double posxyz[3];
      for(int i=0;i<3;i++){
        posxyz[i] = VetoData[n*Vdim+(i+1)];
        //cout << "  posxyz[" << i << "] " << posxyz[i] << endl;
      }
      double rr = sqrt(posxyz[0]*posxyz[0] +
                       posxyz[1]*posxyz[1] +
                       posxyz[2]*posxyz[2]);
      double rphi = atan2(posxyz[1],posxyz[0]);
      if(rphi<0) rphi += 2*PI;

      double time = 0.;

      *rpos = time; rpos++;
      *rpos = posxyz[0]; rpos++;
      *rpos = posxyz[1]; rpos++;
      *rpos = posxyz[2]; rpos++;
      *rpos = rr; rpos++;
      *rpos = posxyz[2]/rr; rpos++;
      *rpos = rphi; rpos++;

      *rgp = time; rgp++;
      *rgp = posxyz[0]; rgp++;
      *rgp = posxyz[1]; rgp++;
      *rgp = posxyz[2]; rgp++;
      *rgp = rr; rgp++;
      *rgp = posxyz[2]/rr; rgp++;
      *rgp = rphi; rgp++;
    }
    else if(particleid == 13){
      //
      // muon
      //

      // muon KE in MeV
      double muonKE = VetoData[n*Vdim+7]*1000.;
```

```
        double x = VetoData[n*Vdim+1];
        double y = VetoData[n*Vdim+2];
        double z = VetoData[n*Vdim+3];
        double px = VetoData[n*Vdim+4];
        double py = VetoData[n*Vdim+5];
        double pz = VetoData[n*Vdim+6];

        MuonPropagator Muon(0,Rmaxgen_RE,muonKE,x,y,z,px,py,pz);

        // loop over photons (these are 'virtual' to contain the energy
        // loss of the muon and therefore are not compton scattered)
        //
        ngam = Muon.GetNumGamma();
        for(int n=0;n<ngam;n++){

          float gammaKE = Muon.GetGammaKE(n);
          //cout << "  gamma " << n << " energy = " << gammaKE << endl;

          // pick a direction for gamma momentum
          const int len = 2;
          float r[len];
          ranlux_(r,len);
          z = -1+2*r[0];
          phi = 2*PI*r[1];
          double x = sqrt(1-z*z)*cos(phi);
          double y = sqrt(1-z*z)*sin(phi);

          *pgam = gammaKE; pgam++;
          *pgam = gammaKE*x; pgam++;
          *pgam = gammaKE*y; pgam++;
          *pgam = gammaKE*z; pgam++;
          *pgam = gammaKE; pgam++;
          *pgam = gammaKE; pgam++;
          *pgam = 4.; pgam++; // veto process Id = 4

          double gamxyz[3];
          for(int i=0;i<3;i++){
            gamxyz[i] = *(Muon.GetGammaVertexXYZ()+(i+3*n));
            //cout << "  gamxyz[" << i << "] " << gamxyz[i] << endl;
          }
          double rr = sqrt(gamxyz[0]*gamxyz[0] +
                           gamxyz[1]*gamxyz[1] +
                           gamxyz[2]*gamxyz[2]);
          double rphi = atan2(gamxyz[1],gamxyz[0]);
          if(rphi<0) rphi += 2*PI;

          double time = Muon.GetGammaTime(n);

          *rgam = time; rgam++;
          *rgam = gamxyz[0]; rgam++;
          *rgam = gamxyz[1]; rgam++;
          *rgam = gamxyz[2]; rgam++;
          *rgam = rr; rgam++;
          *rgam = gamxyz[2]/rr; rgam++;
          *rgam = rphi; rgam++;
        }
      }
      else if(particleid == 2112){
        //
        // neutron
        //
        nneu++;

        // neutron KE and total E in MeV
        double neutronKE = VetoData[n*Vdim+7]*1000.;
```

```
      double Mneutron = MPROTON+DELTA;
      double neutronE = neutronKE+Mneutron;

      *pneu = neutronE; pneu++;
      *pneu = VetoData[n*Vdim+4]*1000.; pneu++;
      *pneu = VetoData[n*Vdim+5]*1000.; pneu++;
      *pneu = VetoData[n*Vdim+6]*1000.; pneu++;
      *pneu = neutronKE; pneu++;
      *pneu = sqrt(neutronE*neutronE-Mneutron*Mneutron); pneu++;
      *pneu = 4.; pneu++; // veto process Id = 4

      double neuxyz[3];
      for(int i=0;i<3;i++){
        neuxyz[i] = VetoData[n*Vdim+(i+1)];
        //cout << "  neuxyz[" << i << "] " << neuxyz[i] << endl;
      }
      double rr = sqrt(neuxyz[0]*neuxyz[0] +
                       neuxyz[1]*neuxyz[1] +
                       neuxyz[2]*neuxyz[2]);
      double rphi = atan2(neuxyz[1],neuxyz[0]);
      if(rphi<0) rphi += 2*PI;

      double time = 0.;

      *rneu = time; rneu++;
      *rneu = neuxyz[0]; rneu++;
      *rneu = neuxyz[1]; rneu++;
      *rneu = neuxyz[2]; rneu++;
      *rneu = rr; rneu++;
      *rneu = neuxyz[2]/rr; rneu++;
      *rneu = rphi; rneu++;

      *rgn = time; rgn++;
      *rgn = neuxyz[0]; rgn++;
      *rgn = neuxyz[1]; rgn++;
      *rgn = neuxyz[2]; rgn++;
      *rgn = rr; rgn++;
      *rgn = neuxyz[2]/rr; rgn++;
      *rgn = rphi; rgn++;
    }
    else{
      cout << "ERROR: unknown particle ID from veto: " << particleid << endl;
    }
  } // end loop over particles
  delete[] localevent;
}
else{
  cout << "ERROR: unknown event type: " << eventType << endl;
}
//
// transfer temp data to storage
//
Nnubar    = nnu;
Nelectron = nele;
Npositron = npos;
Nneutron  = nneu;
Ngamma    = ngam;

Pnubar    = new double[Pdim*nnu];
Pelectron = new double[Pdim*nele];
Ppositron = new double[Pdim*npos];
Pneutron  = new double[Pdim*nneu];
Pgamma    = new double[Pdim*ngam];

Rnubar    = new double[Rdim*nnu];
```

```
Relectron = new double[Rdim*nele];
Rgenele   = new double[Rdim*nele];
Rpositron = new double[Rdim*npos];
Rgenpos   = new double[Rdim*npos];
Rneutron  = new double[Rdim*nneu];
Rgenneu   = new double[Rdim*nneu];
Rgamma    = new double[Rdim*ngam];

double* ptr;

ptr = Pnubar+Pdim*Nnubar;
while(ptr>Pnubar)*--ptr=*--pnu;

ptr = Pelectron+Pdim*Nelectron;
while(ptr>Pelectron)*--ptr=*--pele;

ptr = Ppositron+Pdim*Npositron;
while(ptr>Ppositron)*--ptr=*--ppos;

ptr = Pneutron+Pdim*Nneutron;
while(ptr>Pneutron)*--ptr=*--pneu;

ptr = Pgamma+Pdim*Ngamma;
while(ptr>Pgamma)*--ptr=*--pgam;

ptr = Rnubar+Rdim*Nnubar;
while(ptr>Rnubar)*--ptr=*--rnu;

ptr = Relectron+Rdim*Nelectron;
while(ptr>Relectron)*--ptr=*--rele;

ptr = Rgenele+Rdim*Nelectron;
while(ptr>Rgenele)*--ptr=*--rge;

ptr = Rpositron+Rdim*Npositron;
while(ptr>Rpositron)*--ptr=*--rpos;

ptr = Rgenpos+Rdim*Npositron;
while(ptr>Rgenpos)*--ptr=*--rgp;

ptr = Rneutron+Rdim*Nneutron;
while(ptr>Rneutron)*--ptr=*--rneu;

ptr = Rgenneu+Rdim*Nneutron;
while(ptr>Rgenneu)*--ptr=*--rgn;

ptr = Rgamma+Rdim*Ngamma;
while(ptr>Rgamma)*--ptr=*--rgam;

/*
cout << Nnubar << " nubar(s):" << endl;
for(int i=0;i<Nnubar*Pdim;i++){
  cout << " Pnubar[" << i << "] " << Pnubar[i] << endl;
}
for(int i=0;i<Nnubar*Rdim;i++){
  cout << "  Rnubar[" << i << "] " << Rnubar[i] << endl;
}

cout << Nelectron << " electron(s):" << endl;
for(int i=0;i<Nelectron*Pdim;i++){
  cout << " Pelectron[" << i << "] " << Pelectron[i] << endl;
}
for(int i=0;i<Nelectron*Rdim;i++){
  cout << "  Relectron[" << i << "] " << Relectron[i] << endl;
}
```

```
  for(int i=0;i<Nelectron*Rdim;i++){
    cout << "  Rgenele[" << i << "] " << Rgenele[i] << endl;
  }

  cout << Npositron << " positron(s):" << endl;
  for(int i=0;i<Npositron*Pdim;i++){
    cout << " Ppositron[" << i << "] " << Ppositron[i] << endl;
  }
  for(int i=0;i<Npositron*Rdim;i++){
    cout << "  Rpositron[" << i << "] " << Rpositron[i] << endl;
  }
  for(int i=0;i<Npositron*Rdim;i++){
    cout << "  Rgenpos[" << i << "] " << Rgenpos[i] << endl;
  }

  cout << Nneutron << " neutron(s):" << endl;
  for(int i=0;i<Nneutron*Pdim;i++){
    cout << " Pneutron[" << i << "] " << Pneutron[i] << endl;
  }
  for(int i=0;i<Nneutron*Rdim;i++){
    cout << "  Rneutron[" << i << "] " << Rneutron[i] << endl;
  }
  for(int i=0;i<Nneutron*Rdim;i++){
    cout << "  Rgenneu[" << i << "] " << Rgenneu[i] << endl;
  }

  cout << Ngamma << " photon(s):" << endl;
  for(int i=0;i<Ngamma*Pdim;i++){
    cout << " Pgamma[" << i << "] " << Pgamma[i] << endl;
  }
  for(int i=0;i<Ngamma*Rdim;i++){
    cout << "  Rgamma[" << i << "] " << Rgamma[i] << endl;
  }
  */

  // clean up
  delete[] pnubar;
  delete[] pelectron;
  delete[] ppositron;
  delete[] pneutron;
  delete[] pgamma;
  delete[] rnubar;
  delete[] relectron;
  delete[] rgenele;
  delete[] rpositron;
  delete[] rgenpos;
  delete[] rneutron;
  delete[] rgenneu;
  delete[] rgamma;
}
```

3.8.3 Member Function Documentation

3.8.3.1 double ReactorEvent::GetElectronKE (int & *n*) const `[inline]`

Returns the energy of each electron produced in the event.

Called with the integer index for each electron in the Pelectron array, from 0 to the total number of electrons.

Definition at line 101 of file ReactorEvent.hh.

```
101 {return Pelectron[4+n*Pdim];}
```

3.8.3.2 double ReactorEvent::GetElectronParentProcess (int & n) const [inline]

Returns the parent process used to generate each electron in the event.

Called with the integer index for each electron in the Pelectron array, from 0 to the total number of electrons.

Definition at line 108 of file ReactorEvent.hh.

```
108 {return Pelectron[6+n*Pdim];}
```

3.8.3.3 double ReactorEvent::GetEnu (int & n) const [inline]

Returns the energy of each anti-neutrino produced in the event.

Called with the integer index for each anti-neutrino in the Pnubar array, from 0 to the total number of anti-neutrinos.

Definition at line 90 of file ReactorEvent.hh.

```
90 {return Pnubar[0+n*Pdim];}
```

3.8.3.4 double ReactorEvent::GetGammaKE (int & n) const [inline]

Returns the energy of each photon produced in the event.

Called with the integer index for each photon in the Pgamma array, from 0 to the total number of photons.

Definition at line 143 of file ReactorEvent.hh.

```
143 {return Pgamma[0+n*Pdim];}
```

3.8.3.5 double ReactorEvent::GetGammaParentProcess (int & n) const [inline]

Returns the parent process used to generate each photon in the event.

Called with the integer index for each photon in the Pgamma array, from 0 to the total number of photons.

Definition at line 148 of file ReactorEvent.hh.

```
148 {return Pgamma[6+n*Pdim];}
```

3.8.3.6 double ReactorEvent::GetNeutronKE (int & n) const [inline]

Returns the energy of each neutron produced in the event.

Called with the integer index for each neutron in the Pneutron array, from 0 to the total number of neutrons.

Definition at line 129 of file ReactorEvent.hh.

```
129 {return Pneutron[4+n*Pdim];}
```

3.8.3.7 double ReactorEvent::GetNeutronParentProcess (int & n) const [inline]

Returns the parent process used to generate each neutron in the event.

Called with the integer index for each neutron in the Pneutron array, from 0 to the total number of neutrons.

Definition at line 136 of file ReactorEvent.hh.

```
136 {return Pneutron[6+n*Pdim];}
```

3.8.3.8 double ReactorEvent::GetNubarParentProcess (int & n) const [inline]

Returns the parent process used to generate each anti-neutrino in the event.

Called with the integer index for each anti-neutrino in the Pnubar array, from 0 to the total number of anti-neutrinos.

Definition at line 95 of file ReactorEvent.hh.

```
95 {return Pnubar[6+n*Pdim];}
```

3.8.3.9 double ReactorEvent::GetPositronKE (int & n) const [inline]

Returns the energy of each positron produced in the event.

Called with the integer index for each positron in the Ppositron array, from 0 to the total number of positrons.

Definition at line 115 of file ReactorEvent.hh.

```
115 {return Ppositron[4+n*Pdim];}
```

3.8.3.10 double ReactorEvent::GetPositronParentProcess (int & n) const [inline]

Returns the parent process used to generate each positron in the event.

Called with the integer index for each positron in the Ppositron array, from 0 to the total number of positrons.

Definition at line 122 of file ReactorEvent.hh.

```
122 {return Ppositron[6+n*Pdim];}
```

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

- **ReactorEvent.hh**
- **ReactorEvent.cpp**

3.9 ReactorNtuple Class Reference

ReactorNtuple creates and fills the ntuple for the ReactorFsim simulation.

#include <ReactorNtuple.hh>

Public Methods

- **ReactorNtuple** (int ntId, const char *ntName, char *dirName)
  ReactorNtuple constructor.
- ~**ReactorNtuple** ()
  ReactorNtuple destructor.
- **ReactorNtuple** (const ReactorNtuple &Ntuple)
  ReactorNtuple copy constructor.
- ReactorNtuple & **operator=** (const ReactorNtuple &rhs)
  ReactorNtuple overloaded = operator.
- void **Fill** (**ReactorEvent** &Event, **ReactorDetector** &Detector)

3.9.1 Detailed Description

ReactorNtuple creates and fills the ntuple for the ReactorFsim simulation.

Definition at line 12 of file ReactorNtuple.hh.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 ReactorNtuple::ReactorNtuple (int *ntId*, const char * *ntName*, char * *dirName*)

ReactorNtuple constructor.

Called by ReactorFsim with an ntuple Id, name and directory name. Creates all the ntuple arrays and histograms as set by the user.

Definition at line 32 of file ReactorNtuple.cpp.

```
                                          {

// define ntuple tags here
// to add vars, increment ntVars, and add tag to end of list
// don't change wordsize
//
int wordsize = 10;
int ntVarsMax = 150;
int ntVars = 0;
char *ntTags = new char[ntVarsMax*wordsize];
//
// book the general event veriables
//
strcpy(ntTags+ntVars++*wordsize,"evtnum    ");
strcpy(ntTags+ntVars++*wordsize,"wtc       ");
```

```
strcpy(ntTags+ntVars++*wordsize,"wtf      ");
//
// book the nubar event variables
//
strcpy(ntTags+ntVars++*wordsize,"enu      ");
strcpy(ntTags+ntVars++*wordsize,"vnu1r    ");
strcpy(ntTags+ntVars++*wordsize,"vnu1cos  ");
strcpy(ntTags+ntVars++*wordsize,"vnu1phi  ");
strcpy(ntTags+ntVars++*wordsize,"vnu1x    ");
strcpy(ntTags+ntVars++*wordsize,"vnu1y    ");
strcpy(ntTags+ntVars++*wordsize,"vnu1z    ");
//
// book the electron event variables
//
strcpy(ntTags+ntVars++*wordsize,"nele     ");
strcpy(ntTags+ntVars++*wordsize,"KEele1   ");
strcpy(ntTags+ntVars++*wordsize,"ele1cos  ");
strcpy(ntTags+ntVars++*wordsize,"vele1r   ");
strcpy(ntTags+ntVars++*wordsize,"vele1cos ");
strcpy(ntTags+ntVars++*wordsize,"vele1phi ");
strcpy(ntTags+ntVars++*wordsize,"vele1x   ");
strcpy(ntTags+ntVars++*wordsize,"vele1y   ");
strcpy(ntTags+ntVars++*wordsize,"vele1z   ");
strcpy(ntTags+ntVars++*wordsize,"tele1    ");
//
// book the positron event variables
//
strcpy(ntTags+ntVars++*wordsize,"npos     ");
strcpy(ntTags+ntVars++*wordsize,"KEpos1   ");
strcpy(ntTags+ntVars++*wordsize,"pos1cos  ");
strcpy(ntTags+ntVars++*wordsize,"vpos1r   ");
strcpy(ntTags+ntVars++*wordsize,"vpos1cos ");
strcpy(ntTags+ntVars++*wordsize,"vpos1phi ");
strcpy(ntTags+ntVars++*wordsize,"vpos1x   ");
strcpy(ntTags+ntVars++*wordsize,"vpos1y   ");
strcpy(ntTags+ntVars++*wordsize,"vpos1z   ");
strcpy(ntTags+ntVars++*wordsize,"tpos1    ");
//
// book the neutron event variables
//
strcpy(ntTags+ntVars++*wordsize,"nneu     ");
strcpy(ntTags+ntVars++*wordsize,"KEneu1   ");
strcpy(ntTags+ntVars++*wordsize,"neu1cos  ");
strcpy(ntTags+ntVars++*wordsize,"neuproc1 ");
strcpy(ntTags+ntVars++*wordsize,"vneu1r   ");
strcpy(ntTags+ntVars++*wordsize,"vneu1cos ");
strcpy(ntTags+ntVars++*wordsize,"vneu1phi ");
strcpy(ntTags+ntVars++*wordsize,"vneu1x   ");
strcpy(ntTags+ntVars++*wordsize,"vneu1y   ");
strcpy(ntTags+ntVars++*wordsize,"vneu1z   ");
strcpy(ntTags+ntVars++*wordsize,"tneu1    ");
strcpy(ntTags+ntVars++*wordsize,"nsteps1  ");
strcpy(ntTags+ntVars++*wordsize,"absorb1  ");

strcpy(ntTags+ntVars++*wordsize,"KEneu2   ");
strcpy(ntTags+ntVars++*wordsize,"neu2cos  ");
strcpy(ntTags+ntVars++*wordsize,"neuproc2 ");
strcpy(ntTags+ntVars++*wordsize,"vneu2r   ");
strcpy(ntTags+ntVars++*wordsize,"vneu2cos ");
strcpy(ntTags+ntVars++*wordsize,"vneu2phi ");
strcpy(ntTags+ntVars++*wordsize,"vneu2x   ");
strcpy(ntTags+ntVars++*wordsize,"vneu2y   ");
strcpy(ntTags+ntVars++*wordsize,"vneu2z   ");
strcpy(ntTags+ntVars++*wordsize,"tneu2    ");
strcpy(ntTags+ntVars++*wordsize,"nsteps2  ");
```

```
strcpy(ntTags+ntVars++*wordsize,"absorb2  ");

strcpy(ntTags+ntVars++*wordsize,"KEneu3   ");
strcpy(ntTags+ntVars++*wordsize,"neu3cos  ");
strcpy(ntTags+ntVars++*wordsize,"neuproc3 ");
strcpy(ntTags+ntVars++*wordsize,"vneu3r   ");
strcpy(ntTags+ntVars++*wordsize,"vneu3cos ");
strcpy(ntTags+ntVars++*wordsize,"vneu3phi ");
strcpy(ntTags+ntVars++*wordsize,"vneu3x   ");
strcpy(ntTags+ntVars++*wordsize,"vneu3y   ");
strcpy(ntTags+ntVars++*wordsize,"vneu3z   ");
strcpy(ntTags+ntVars++*wordsize,"tneu3    ");
strcpy(ntTags+ntVars++*wordsize,"nsteps3  ");
strcpy(ntTags+ntVars++*wordsize,"absorb3  ");
//
// book the pmt detector variables
//
strcpy(ntTags+ntVars++*wordsize,"ncosbins ");
strcpy(ntTags+ntVars++*wordsize,"nphibins ");
//
strcpy(ntTags+ntVars++*wordsize,"phposvis ");
strcpy(ntTags+ntVars++*wordsize,"phposrec ");
strcpy(ntTags+ntVars++*wordsize,"phneuvis ");
strcpy(ntTags+ntVars++*wordsize,"phneurec ");
strcpy(ntTags+ntVars++*wordsize,"phtlvis  ");
strcpy(ntTags+ntVars++*wordsize,"phtlrec  ");
strcpy(ntTags+ntVars++*wordsize,"phbivis  ");
strcpy(ntTags+ntVars++*wordsize,"phbirec  ");
strcpy(ntTags+ntVars++*wordsize,"phcfvis  ");
strcpy(ntTags+ntVars++*wordsize,"phcfrec  ");
strcpy(ntTags+ntVars++*wordsize,"phmuovis ");
strcpy(ntTags+ntVars++*wordsize,"phmuorec ");
strcpy(ntTags+ntVars++*wordsize,"phelevis ");
strcpy(ntTags+ntVars++*wordsize,"phelerec ");
//
strcpy(ntTags+ntVars++*wordsize,"npepos   ");
strcpy(ntTags+ntVars++*wordsize,"Qpos     ");
strcpy(ntTags+ntVars++*wordsize,"attenpos ");
strcpy(ntTags+ntVars++*wordsize,"sfactpos ");
strcpy(ntTags+ntVars++*wordsize,"npeneu   ");
strcpy(ntTags+ntVars++*wordsize,"Qneu     ");
strcpy(ntTags+ntVars++*wordsize,"attenneu ");
strcpy(ntTags+ntVars++*wordsize,"sfactneu ");
strcpy(ntTags+ntVars++*wordsize,"npetl    ");
strcpy(ntTags+ntVars++*wordsize,"Qtl      ");
strcpy(ntTags+ntVars++*wordsize,"attentl  ");
strcpy(ntTags+ntVars++*wordsize,"sfacttl  ");
strcpy(ntTags+ntVars++*wordsize,"npebi    ");
strcpy(ntTags+ntVars++*wordsize,"Qbi      ");
strcpy(ntTags+ntVars++*wordsize,"attenbi  ");
strcpy(ntTags+ntVars++*wordsize,"sfactbi  ");
strcpy(ntTags+ntVars++*wordsize,"npecf    ");
strcpy(ntTags+ntVars++*wordsize,"Qcf      ");
strcpy(ntTags+ntVars++*wordsize,"attencf  ");
strcpy(ntTags+ntVars++*wordsize,"sfactcf  ");
strcpy(ntTags+ntVars++*wordsize,"npemuo   ");
strcpy(ntTags+ntVars++*wordsize,"Qmuo     ");
strcpy(ntTags+ntVars++*wordsize,"attenmuo ");
strcpy(ntTags+ntVars++*wordsize,"sfactmuo ");
strcpy(ntTags+ntVars++*wordsize,"npeele   ");
strcpy(ntTags+ntVars++*wordsize,"Qele     ");
strcpy(ntTags+ntVars++*wordsize,"attenele ");
strcpy(ntTags+ntVars++*wordsize,"sfactele ");
//
strcpy(ntTags+ntVars++*wordsize,"ngampos  ");
```

```
strcpy(ntTags+ntVars++*wordsize,"ngamneu  ");
strcpy(ntTags+ntVars++*wordsize,"ngamtl   ");
strcpy(ntTags+ntVars++*wordsize,"ngambi   ");
strcpy(ntTags+ntVars++*wordsize,"ngamcf   ");
strcpy(ntTags+ntVars++*wordsize,"ngammuo  ");
strcpy(ntTags+ntVars++*wordsize,"ngamele  ");
//
strcpy(ntTags+ntVars++*wordsize,"fpmtpos  ");
strcpy(ntTags+ntVars++*wordsize,"fpmtneu  ");
strcpy(ntTags+ntVars++*wordsize,"fpmttl   ");
strcpy(ntTags+ntVars++*wordsize,"fpmtbi   ");
strcpy(ntTags+ntVars++*wordsize,"fpmtcf   ");
strcpy(ntTags+ntVars++*wordsize,"fpmtmuo  ");
strcpy(ntTags+ntVars++*wordsize,"fpmtele  ");
//
strcpy(ntTags+ntVars++*wordsize,"hitpmtpo ");
strcpy(ntTags+ntVars++*wordsize,"hitpmtne ");
strcpy(ntTags+ntVars++*wordsize,"hitpmtth ");
strcpy(ntTags+ntVars++*wordsize,"hitpmtbi ");
strcpy(ntTags+ntVars++*wordsize,"hitpmtca ");
strcpy(ntTags+ntVars++*wordsize,"hitpmtmu ");
strcpy(ntTags+ntVars++*wordsize,"hitpmtel ");
//
hbookn_(ntId,ntName,ntVars,dirName,1000,ntTags,
        strlen(ntName),strlen(dirName),wordsize,ntVars);
//
// define some histograms
// Last argument is need for c++ ==> f strings
//
Nvar = ntVars;
Id = ntId;
Offset = 10000*(ntId-1);

hbook1_(Offset+101,"electron PH /PMT/gamma/electron",200,0,0.1,0,
        strlen("electron PH /PMT/gamma/electron"));
hbook1_(Offset+102,"positron PH /PMT/gamma/positron",200,0,0.1,0,
        strlen("positron PH /PMT/gamma/positron"));
hbook1_(Offset+103,"neutron PH /PMT/gamma/neutron",200,0,0.1,0,
        strlen("neutron PH /PMT/gamma/neutron"));
hbook1_(Offset+104,"thallium PH /PMT/gamma/decay",200,0,0.1,0,
        strlen("thallium PH /PMT/gamma/decay"));
hbook1_(Offset+105,"bismuth PH /PMT/gamma/decay",200,0,0.1,0,
        strlen("bismuth PH /PMT/gamma/decay"));
hbook1_(Offset+106,"cf prompt gamma PH /PMT/gamma/decay",200,0,0.1,0,
        strlen("cf prompt gamma PH /PMT/gamma/decay"));
hbook1_(Offset+107,"muon PH /PMT/gamma/muon",200,0,0.1,0,
        strlen("muon PH /PMT/gamma/muon"));
//
hbook1_(Offset+201,"electron time /PMT/gamma/electron",250,0,5e8,0,
        strlen("electron time /PMT/gamma/electron"));
hbook1_(Offset+202,"positron time /PMT/gamma/positron",250,0,50,0,
        strlen("positron time /PMT/gamma/positron"));
hbook1_(Offset+203,"neutron time /PMT/gamma/neutron",250,0,5e5,0,
        strlen("neutron time /PMT/gamma/neutron"));
hbook1_(Offset+204,"thallium time /PMT/gamma/decay",250,0,1e6,0,
        strlen("thallium time /PMT/gamma/decay"));
hbook1_(Offset+205,"bismuth time /PMT/gamma/decay",250,0,1e6,0,
        strlen("bismuth time /PMT/gamma/decay"));
hbook1_(Offset+206,"cf prompt gamma time /PMT/gamma/decay",250,0,50,0,
        strlen("cf prompt gamma time /PMT/gamma/decay"));
hbook1_(Offset+207,"muon time /PMT/gamma/muon",300,0,60,0,
        strlen("muon time /PMT/gamma/muon"));
//
hbook1_(Offset+301,"number of gamma/electron",10,0,10,0,
        strlen("number of gamma/electron"));
```

```
hbook1_(Offset+302,"number of gamma/positron",30,0,30,0,
        strlen("number of gamma/positron"));
hbook1_(Offset+303,"number of gamma/neutron",20,0,20,0,
        strlen("number of gamma/neutron"));
hbook1_(Offset+306,"number of prompt Cf gamma/decay",20,0,20,0,
        strlen("number of prompt CF gamma/decay"));
//
hbook1_(Offset+401,"electron gamma energy",100,0,10,0,
        strlen("electron gamma energy"));
hbook1_(Offset+402,"positron gamma energy",100,0,10,0,
        strlen("positron gamma energy"));
hbook1_(Offset+403,"neutron gamma energy",100,0,10,0,
        strlen("neutron gamma energy"));
hbook1_(Offset+406,"prompt Cf gamma energy",100,0,10,0,
        strlen("prompt Cf gamma energy"));
//
hbook1_(Offset+501,"electron gamma time",250,0,5e8,0,
        strlen("electron gamma time"));
hbook1_(Offset+502,"positron gamma time",250,0,5,0,
        strlen("positron gamma time"));
hbook1_(Offset+503,"neutron gamma time",250,0,5e5,0,
        strlen("neutron gamma time"));
hbook1_(Offset+506,"prompt Cf gamma time",250,0,5,0,
        strlen("prompt Cf gamma time"));
//
hbook1_(Offset+601,"electron gamma radius",400,0,400,0,
        strlen("electron gamma radius"));
hbook1_(Offset+602,"positron gamma radius",400,0,400,0,
        strlen("positron gamma radius"));
hbook1_(Offset+603,"neutron gamma radius",400,0,400,0,
        strlen("neutron gamma radius"));
hbook1_(Offset+606,"prompt Cf gamma radius",400,0,400,0,
        strlen("prompt Cf gamma radius"));
//
hbook1_(Offset+701,"electron gamma cos(theta)",20,-1.1,1.1,0,
        strlen("electron gamma cos(theta)"));
hbook1_(Offset+702,"positron gamma cos(theta)",20,-1.1,1.1,0,
        strlen("positron gamma cos(theta)"));
hbook1_(Offset+703,"neutron gamma cos(theta)",20,-1.1,1.1,0,
        strlen("neutron gamma cos(theta)"));
hbook1_(Offset+706,"prompt Cf gamma cos(theta)",20,-1.1,1.1,0,
        strlen("prompt Cf gamma cos(theta)"));
//
hbook1_(Offset+1001,"npe in each PMT from each electron",200,-0.5,199.5,0,
        strlen("npe in each PMT from each electron"));
hbook1_(Offset+1002,"npe in each PMT from each positron",200,-0.5,199.5,0,
        strlen("npe in each PMT from each positron"));
hbook1_(Offset+1003,"npe in each PMT from each neutron",200,-0.5,199.5,0,
        strlen("npe in each PMT from each neutron"));
hbook1_(Offset+1004,"npe in each PMT from each thallium",200,-0.5,199.5,0,
        strlen("npe in each PMT from each thallium"));
hbook1_(Offset+1005,"npe in each PMT from each bismuth",200,-0.5,199.5,0,
        strlen("npe in each PMT from each bismuth"));
hbook1_(Offset+1006,"npe in each PMT from each calif",200,-0.5,199.5,0,
        strlen("npe in each PMT from each calif"));
hbook1_(Offset+1007,"npe in each PMT from each muon",200,-0.5,199.5,0,
        strlen("npe in each PMT from each muon"));
//
hbook1_(Offset+2001,"total npe in each PMT from electrons",
        200,-0.5,199.5,0,
        strlen("total npe in each PMT from electrons"));
hbook1_(Offset+2002,"total npe in each PMT from positrons",
        200,-0.5,199.5,0,
        strlen("total npe in each PMT from positrons"));
hbook1_(Offset+2003,"total npe in each PMT from neutrons",
```

```
        200,-0.5,199.5,0,
        strlen("total npe in each PMT from neutrons"));
hbook1_(Offset+2004,"total npe in each PMT from thallium",
        200,-0.5,199.5,0,
        strlen("total npe in each PMT from thallium"));
hbook1_(Offset+2005,"total npe in each PMT from bismuth",
        200,-0.5,199.5,0,
        strlen("total npe in each PMT from bismuth"));
hbook1_(Offset+2006,"total npe in each PMT from calif",
        200,-0.5,199.5,0,
        strlen("total npe in each PMT from calif"));
hbook1_(Offset+2007,"total npe in each PMT from muons",
        800,-0.5,799.5,0,
        strlen("total npe in each PMT from muons"));
}
```

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

- **ReactorNtuple.hh**
- **ReactorNtuple.cpp**

3.10 ReactorTree Class Reference

ReactorTreee creates and fills the root tree for the ReactorFsim simulation.

`#include <ReactorTree.hh>`

Public Methods

- **ReactorTree** ()

  ReactorTree constructor.

- **ReactorTree** (const char *TreeName)
- ~**ReactorTree** ()

  ReactorTree destructor.

- void **Fill** (**ReactorEvent** &Event, **ReactorDetector** &Detector)
- void **CloseOut** ()

3.10.1 Detailed Description

ReactorTreee creates and fills the root tree for the ReactorFsim simulation.

Definition at line 14 of file ReactorTree.hh.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 ReactorTree::ReactorTree ()

ReactorTree constructor.

Called by ReactorFsim with a tree name .

Definition at line 20 of file ReactorTree.cpp.

```
20                            {
21 }
```

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

- **ReactorTree.hh**
- **ReactorTree.cpp**

3.11 ReactorTrigger Class Reference

ReactorTrigger performs a simple trigger simulation for the ReactorFsim simulation.

```
#include <ReactorTrigger.hh>
```

Public Methods

- **ReactorTrigger** ()

    ReactorTrigger constructor.

- ~**ReactorTrigger** ()

    ReactorTrigger destructor.

- **ReactorTrigger** (const ReactorTrigger &Trigger)

    ReactorTrigger copy constructor.

- ReactorTrigger & **operator=** (const ReactorTrigger &rhs)

    ReactorTrigger overloaded = operator.

- void **Trigger** (**ReactorEvent** &Event, **ReactorDetector** &Detector)
- bool **PassTrigger** ()

3.11.1 Detailed Description

ReactorTrigger performs a simple trigger simulation for the ReactorFsim simulation.

Definition at line 12 of file ReactorTrigger.hh.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 ReactorTrigger::ReactorTrigger ()

ReactorTrigger constructor.

Called by ReactorFsim.

Definition at line 31 of file ReactorTrigger.cpp.

```
31                               {
32
33   Pass = false;
34 }
```

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

- **ReactorTrigger.hh**
- **ReactorTrigger.cpp**

Chapter 4

ReactorFsim File Documentation

4.1 cf252_readme.txt File Reference

Contains the documentation for the Cf252 source text files.

4.1.1 Detailed Description

Contains the documentation for the Cf252 source text files.

Definition in file **cf252_readme.txt**.

4.2 DetectorDefaults.hh File Reference

DetectorDefaults header file.

4.2.1 Detailed Description

DetectorDefaults header file.

Author: Tim Bolton

Definition in file **DetectorDefaults.hh**.

4.3 event_readme.txt File Reference

Contains the documentation for the event setup text files.

4.3.1 Detailed Description

Contains the documentation for the event setup text files.

Author: Matthew Worcester

Definition in file **event_readme.txt**.

4.4 GammaComptonRange.cpp File Reference

GammaComptonRange C++ file.

```
#include <math.h>
#include <iostream.h>
#include <string>
#include "ReactorConstants.hh"
```

Functions

- double **GammaComptonRange** (std::string material, double egamma)

4.4.1 Detailed Description

GammaComptonRange C++ file.

Implements the GammaComptonRange function, which returns the mean free path of a photon propogating through a given material.

Author: Tim Bolton

Definition in file **GammaComptonRange.cpp**.

4.5 MuonPropagator.cpp File Reference

MuonPropagator (p. 5) C++ file.

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <string>
#include <stdio.h>
#include "ReactorConstants.hh"
#include "ReactorFortran.hh"
#include "MuonPropagator.hh"
#include "MyHeSource.hh"
#include "MyLiSource.hh"
```

4.5.1 Detailed Description

MuonPropagator (p. 5) C++ file.

Author: Matthew Worcester

Definition in file **MuonPropagator.cpp**.

4.6 MyBiSource.cpp File Reference

MyBiSource (p. 17) C++ file.

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <string>
#include "ReactorDetector.hh"
#include "ReactorConstants.hh"
#include "ReactorFortran.hh"
#include "MyBiSource.hh"
```

4.6.1 Detailed Description

MyBiSource (p. 17) C++ file.

Implements the constuctor, copy constructor, and overloaded = operator.

Author: Matthew Worcester

Definition in file **MyBiSource.cpp**.

4.7 MyBiSource.hh File Reference

MyBiSource (p. 17) header file.

```
#include <math.h>
#include <iostream.h>
#include "ReactorConstants.hh"
```

Compounds

- class **MyBiSource**

  MyBiSource generates photons from radioactive Thallium decays.

4.7.1 Detailed Description

MyBiSource (p. 17) header file.

Author: Matthew Worcester

Definition in file **MyBiSource.hh**.

4.8 MyCfSource.cpp File Reference

MyCfSource (p. 22) C++ file.

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <string>
#include "ReactorConstants.hh"
#include "ReactorFortran.hh"
#include "MyCfSource.hh"
```

Functions

- long long **factorial** (int n)

4.8.1 Detailed Description

MyCfSource (p. 22) C++ file.

Implements the constructor, copy constructor, and overloaded = operator and defines the Cf252NeutronSpectrum function.

Author: Matthew Worcester

Definition in file **MyCfSource.cpp**.

4.9 MyCfSource.hh File Reference

MyCfSource (p. 22) header file.

```
#include <math.h>
#include <iostream.h>
#include "ReactorConstants.hh"
```

Compounds

- class **MyCfSource**

  MyCfSource generates prompt photons and delayed neutrons from radioactive Californium decays.

4.9.1 Detailed Description

MyCfSource (p. 22) header file.

Author: Matthew Worcester

Definition in file **MyCfSource.hh**.

4.10 MyHeSource.cpp File Reference

MyHeSource (p. 30) C++ file.

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <string>
#include "ReactorConstants.hh"
#include "ReactorFortran.hh"
#include "MyHeSource.hh"
```

4.10.1 Detailed Description

MyHeSource (p. 30) C++ file.

Implements the constructor, copy constructor, and overloaded = operator and defines the He8NeutronSpectrum function.

Author: Matthew Worcester

Definition in file **MyHeSource.cpp**.

4.11 MyHeSource.hh File Reference

MyHeSource (p. 30) header file.

```
#include <math.h>
#include <iostream.h>
#include "ReactorConstants.hh"
```

Compounds

- class **MyHeSource**

  MyHeSource generates delayed neutrons from radioactive He decays.

4.11.1 Detailed Description

MyHeSource (p. 30) header file.

Author: Matthew Worcester

Definition in file **MyHeSource.hh**.

4.12 MyLiSource.cpp File Reference

MyLiSource (p. 33) C++ file.

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <complex>
#include <string>
#include "ReactorConstants.hh"
#include "ReactorFortran.hh"
#include "MyLiSource.hh"
```

4.12.1 Detailed Description

MyLiSource (p. 33) C++ file.

Implements the constructor, copy constructor, and overloaded = operator and defines the Li8NeutronSpectrum function.

Author: Matthew Worcester

Definition in file **MyLiSource.cpp**.

4.13 MyLiSource.hh File Reference

MyLiSource (p. 33) header file.

`#include <math.h>`

`#include <iostream.h>`

`#include "ReactorConstants.hh"`

Compounds

- class **MyLiSource**

  MyLiSource generates delayed neutrons from radioactive Li decays.

4.13.1 Detailed Description

MyLiSource (p. 33) header file.

Author: Matthew Worcester

Definition in file **MyLiSource.hh**.

4.14 MyTlSource.cpp File Reference

MyTlSource (p. 36) C++ file.

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <string>
#include "ReactorDetector.hh"
#include "ReactorConstants.hh"
#include "ReactorFortran.hh"
#include "MyTlSource.hh"
```

4.14.1 Detailed Description

MyTlSource (p. 36) C++ file.

Implements the constuctor, copy constructor, and overloaded = operator.

Author: Matthew Worcester

Definition in file **MyTlSource.cpp**.

4.15 MyTlSource.hh File Reference

MyTlSource (p. 36) header file.

`#include <math.h>`

`#include <iostream.h>`

`#include "ReactorConstants.hh"`

Compounds

- class **MyTlSource**

  MyTlSource generates photons from radioactive Thallium decays.

4.15.1 Detailed Description

MyTlSource (p. 36) header file.

Author: Matthew Worcester

Definition in file **MyTlSource.hh**.

4.16 ReactorConstants.hh File Reference

ReactorConstants header file.

4.16.1 Detailed Description

ReactorConstants header file.

Author: Tim Bolton

Definition in file **ReactorConstants.hh**.

4.17 ReactorDetector.cpp File Reference

ReactorDetector (p. 41) C++ file.

```
#include <math.h>
#include <iostream.h>
#include <string.h>
#include "ReactorEvent.hh"
#include "ReactorFortran.hh"
#include "ReactorConstants.hh"
#include "DetectorDefaults.hh"
#include "ReactorXC.hh"
#include "ReactorDetector.hh"
#include "ReactorScint.hh"
#include "MyTlSource.hh"
#include "MyBiSource.hh"
```

4.17.1 Detailed Description

ReactorDetector (p. 41) C++ file.

Implements the constuctor, destructor, copy constructor, and overloaded = operator. Implements the constructors, destructor, operators, and functions in the ReactorDetector::PMT sub-class. Contains the detector simulation function, LightsOut, and its necessary subroutines.

Author: Tim Bolton

Definition in file **ReactorDetector.cpp**.

4.18 ReactorDetector.hh File Reference

ReactorDetector (p. 41) header file.

```
#include <math.h>
#include <iostream.h>
#include <string.h>
#include <vector>
#include <fstream>
#include "ReactorEvent.hh"
#include "ReactorFortran.hh"
#include "ReactorConstants.hh"
#include "ReactorXC.hh"
```

Compounds

- struct **PMTinfo**
- class **PMToutput**
- class **ReactorDetector::PMT**
- class **ReactorDetector**

  ReactorDetector handles the detector implementation in the ReactorFsim simulation.

4.18.1 Detailed Description

ReactorDetector (p. 41) header file.

Author: Tim Bolton

Definition in file **ReactorDetector.hh**.

4.19 ReactorEvent.cpp File Reference

ReactorEvent (p. 45) C++ file.

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <string>
#include <stdio.h>
#include "ReactorConstants.hh"
#include "ReactorFortran.hh"
#include "ReactorXC.hh"
#include "ReactorEvent.hh"
#include "MyCfSource.hh"
#include "MuonPropagator.hh"
```

4.19.1 Detailed Description

ReactorEvent (p. 45) C++ file.

Implements the constuctor, destructor, copy constructor, and overloaded = operator.

Author: Tim Bolton, Matthew Worcester

Definition in file **ReactorEvent.cpp**.

4.20 ReactorEvent.hh File Reference

ReactorEvent (p. 45) header file.

```
#include <math.h>
#include <iostream.h>
#include <string>
#include "ReactorConstants.hh"
```

Compounds

- class **ReactorEvent**

  ReactorEvent handles all of the event generation in the ReactorFsim simulation.

4.20.1 Detailed Description

ReactorEvent (p. 45) header file.

Author: Tim Bolton, Matthew Worcester

Definition in file **ReactorEvent.hh**.

4.21 ReactorFortran.hh File Reference

Defines the Fortran prototypes for calling from C++.

`#include <complex>`

Namespaces

- namespace **std**

Functions

- void **hropen_** (const int &, const char *, const char *, const char *, int &, int &, int lchar1, int lchar2, int lchar3)
- void **hbookn_** (const int &, const char *, const int &, const char *, const int &, const char *ntTags, int lchar1, int lchar2, int lchar3, int lchar4)
- void **hbook1_** (const int &, const char *, const int &, const float &, const float &, const float &, const int title_length)
- void **hfill_** (const int &, const float &, const float &y=0, const float &w=1)
- void **hfn_** (const int &, const float *)
- void **hlimit_** (const int &)
- void **hldir_** (const char *, const char *, int lchar1, int lchar2)
- void **hcdir_** (const char *, const char *, int lchar1, int lchar2)
- void **hrout_** (const int &, int &, const char *, int lchar1)
- void **hrend_** (const char *, int lchar1)
- void **ranlux_** (float *, const int &)
- void **rluxgo_** (int &lux, int &init, int &k1, int &k2)
- void **rnpssn_** (const float &Mean, int &Value, int &ErrorCode)
- void **rnormx_** (float *, int &, void(float *, const int &))
- void **rnhpre_** (float *, const int &)
- void **rnhran_** (float *, const int &, const float &, const float &, float &)
- void **funlxp_** (float(const float &), float *, const float &, const float &)
- void **funlux_** (float *, float &, const int &)
- float **divdif_** (float *, float *, const int &, const float &, const int &)
- complex< double > **wgamma_** (complex< double > &)
- float **gamma_** (float *)

4.21.1 Detailed Description

Defines the Fortran prototypes for calling from C++.

Necessary for the Hbook interface and Cernlib Fortran routines.

Definition in file **ReactorFortran.hh**.

4.22 ReactorFsim.cpp File Reference

ReactorFsim simulation for MWtheta13.

```
#include "ReactorFortran.hh"
#include "cfortran.h"
#include "ReactorConstants.hh"
#include "ReactorEvent.hh"
#include "ReactorDetector.hh"
#include "ReactorXC.hh"
#include "TFile.h"
#include "ReactorNtuple.hh"
#include "ReactorTree.hh"
#include "ReactorTrigger.hh"
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <math.h>
#include <string.h>
#include <cstdio>
```

Compounds

- struct **PAWC_DEF**

Defines

- #define **PAWC_SIZE** 50000
- #define **PAWC** COMMON_BLOCK(PAWC,pawc)

Functions

- **COMMON_BLOCK_DEF** (PAWC_DEF, PAWC)
- int **main** ()

  Top level function of ReactorFsim.

Variables

- PAWC_DEF **PAWC**

4.22.1 Detailed Description

ReactorFsim simulation for MWtheta13.

ReactorFsim simulates any number of spherical, three-zone liquid scintillator neutrino detectors.

Main author: Tim Bolton

Contributing authors: Erin Abouzaid, Josh Klein, Matthew Worcester

Definition in file **ReactorFsim.cpp**.

4.22.2 Function Documentation

4.22.2.1 int main ()

Top level function of ReactorFsim.

First sets the number of events, initializes paw, and defines each detector. The event loop creates an event and runs it through each detector added to the loop, so that each detector "sees" the exact same event each time. Finally, paw is closed and the hbook file created.

Definition at line 42 of file ReactorFsim.cpp.

References ReactorTree::CloseOut, ReactorTree::Fill, ReactorNtuple::Fill, ReactorDetector::LightsOut, ReactorDetector::SetParam_FastNeutronOption, ReactorDetector::SetParam_PMTDebugOption, ReactorDetector::SetParam_PMTMergeTime, ReactorDetector::SetParam_R0, ReactorDetector::SetParam_R1, ReactorDetector::SetParam_R2, and ReactorTrigger::Trigger.

```
42         {
43
44 //  ReactorNeutrons Neutrons;
45 //
46 //  Set number of events, paw file name, and ntuple name
47 //
48 // defaults
49 int nevents = 100;
50 double r0=260;
51 double r1=260;
52 double rmaxgen=290;
53 double r2=350;
54 int fast=0;
55 int PMTdebug = 0;
56 double MergeTime = 1;
57 int trigger=0;
58 char* fileName="test.ntuple";
59 char* rootName="test.root";
60 char* ntName="event data";
61 //
62 // setup the event
63 //
64 fstream fin("src/event_setup.txt", ios::in);
65 //cout << "setup file: src/event_setup.txt" << endl;
66
67 char paramLine[256];
68 std::string dummy;
69 bool done = false;
70 std::string fastneutron;
71 std::string trigsim;
72 std::string PMTdebugoption;
73 char pawFile[256];
```

```
char rootFile[256];

while(!done){

  // '#' specifies a comment line -- ignore it
  if(fin.peek() == '#'){

    fin.getline(paramLine, 256);
    continue;
  }

  // 'E'ND or 'e'nd specifies end of file -- set done flag to true
  if(fin.peek() == 'e' || fin.peek() == 'E'){

    done = true;
    continue;
  }

  // 'N'EVENT or 'n'event specifies the number of events
  if(fin.peek() == 'n' || fin.peek() == 'N'){

    fin >> dummy;
    while(fin.peek() == ' ' || fin.peek() == '=')
      fin.ignore(1);

    fin >> nevents;

     if(nevents <= 0){
       cout << "  SETUP ERROR: number of events <= 0!" << endl;
       nevents = 1;
     }

     continue;
   }

   // 'O'UTPUTFILE or 'o'utputfile specifies the output filename
   if(fin.peek() == 'o' || fin.peek() == 'O'){

     fin >> dummy;
     while(fin.peek() == ' ' || fin.peek() == '=')
       fin.ignore(1);

     fin >> pawFile;
     fileName = pawFile;

     continue;
   }

   // 'A'ROOTFILE or 'a'rootfile specifies the output filename
   if(fin.peek() == 'a' || fin.peek() == 'A'){

     fin >> dummy;
     while(fin.peek() == ' ' || fin.peek() == '=')
       fin.ignore(1);

     fin >> rootFile;
     rootName = rootFile;

     continue;
   }

   // 'R'ADII or 'r'adii specifies the three detector radii: R0 R1 R2
   if(fin.peek() == 'r' || fin.peek() == 'R'){

     fin >> dummy;
```

```
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> r0 >> r1 >> r2 >> rmaxgen;

      continue;
    }

    // 'F'ASTNEUTRON or 'f'astneutron turns on/off fast neutrons
    if(fin.peek() == 'f' || fin.peek() == 'F'){

      fin >> dummy;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> fastneutron;

      if(fastneutron == "off"){
        fast = 0;
      }
      else{
        fast = 1;
      }

      continue;
    }

    // 'B'PMTOUTPUTFILE 'b'PMTOUTPUTFILE turns on/off the PMT Debug file
    if(fin.peek() == 'b' || fin.peek() == 'B'){

      fin >> dummy;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> PMTdebugoption;

      if(PMTdebugoption == "on"){
        PMTdebug = 1;
      }
      else{
        PMTdebug = 0;
      }
      continue;
    }

    // 'M'ERGETIME or 'm'ergetime specifies the PMTOutput Resolution (MergeTime)
    if(fin.peek() == 'm' || fin.peek() == 'M'){

      fin >> dummy;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> MergeTime;

      continue;
    }

    // 'T'RIGGER or 't'rigger turns on/off trigger simulation
    if(fin.peek() == 't' || fin.peek() == 'T'){

      fin >> dummy;
      while(fin.peek() == ' ' || fin.peek() == '=')
        fin.ignore(1);

      fin >> trigsim;
```

```

    if(trigsim == "off"){
      trigger = 0;
    }
    else{
      trigger = 1;
    }

    continue;
  }

  // skip K, P, and D
  if(fin.peek() == 'K' || fin.peek() == 'k' ||
     fin.peek() == 'P' || fin.peek() == 'p' ||
     fin.peek() == 'D' || fin.peek() == 'd'){

    fin.getline(paramLine, 256);
    continue;
  }

  // ignore extra whitespace
  if(fin.peek() == ' ' || fin.peek() == '\n'){

    fin.ignore(1);
    continue;
  }
}
//
cout << "ReactorFsim v0_0_5 by T. Bolton" << endl;
cout << "==============================" << endl;
cout << "  " << nevents << " events" << endl;
cout << "  Paw output file: " << fileName << endl;
cout << "  Root output file: " << rootName << endl;
cout << "  R0 = " << r0 << ", R1 = " << r1 << ", R2 = " << r2
     << " (Rmaxgen = " << rmaxgen << ") cm" << endl;
cout << "  Fast neutrons are " << fastneutron << endl;
cout << "  Trigger simulation is " << trigsim << endl;
cout << "  PMT output debugging is " << PMTdebugoption << endl;
cout << "  PMT MergeTime is " << MergeTime << endl;
cout << "==============================" << endl;
//
// init random number seed
//
int luxury = 2;
int seed = 8796896;
int k1=0;
int k2=0;
rluxgo_(luxury,seed,k1,k2);
//
// setup root output file and create tree
//
TFile *rfile= new TFile(rootName,"RECREATE","ReactorTree");
ReactorTree rtree("RTree");
//
// Paw setup.
//
hlimit_(PAWC_SIZE);
int iostat;
int ntLun=99;
int ntBuf=1024;
char* dirName = "TEST";
char* chopt = "N";
hropen_(ntLun,dirName,fileName,chopt,ntBuf,iostat,
        strlen(dirName),strlen(fileName),strlen(chopt));
//
```

```
  // Set up simple detector models
  // Set up simple Ntuple and histogranms
  // See ReactorNtuple.hh for ntuple/histogram details.
  //

  ReactorDetector DetectorA;
  DetectorA.SetParam_FastNeutronOption(fast);
  DetectorA.SetParam_PMTDebugOption(PMTdebug);
  DetectorA.SetParam_PMTMergeTime(MergeTime);
  DetectorA.SetParam_R0(r0);
  DetectorA.SetParam_R1(r1);
  DetectorA.SetParam_R2(r2);
  ReactorNtuple ModelA(1,"model A",dirName);
  ReactorTrigger TriggerA;
  //
  // event loop
  //
  for (int events = 0;events<nevents;events++){
    //
    // create neutrino event
    //
    ReactorEvent Event(nevents,rmaxgen);
    //
    // create fast detector simulation
    //
    DetectorA.LightsOut(Event, events);
    ModelA.Fill(Event,DetectorA);
    rtree.Fill(Event,DetectorA);
    if(trigger) TriggerA.Trigger(Event,DetectorA);
  }
  // close up root
  //
     rfile->Write();
     rtree.CloseOut();
  // close up paw
  //
  hldir_(" "," ",strlen(" "),strlen(" "));
  char *dirName2 = new char[strlen(dirName)+2];
  *dirName2='\0';
  strcat(dirName2,"//");
  strcat(dirName2,dirName);
  hcdir_(dirName2," ",strlen(dirName2),strlen(" "));
  int outopt = 0;
  int icycle;
  hrout_(outopt,icycle," ",strlen(" "));
  hrend_(dirName,strlen(dirName));
}
```

4.23 ReactorNtuple.cpp File Reference

ReactorNtuple (p. 69) C++ file.

```
#include <iostream.h>
#include <math.h>
#include <string.h>
#include <cstdio>
#include "ReactorEvent.hh"
#include "ReactorDetector.hh"
#include "ReactorFortran.hh"
#include "ReactorNtuple.hh"
```

4.23.1 Detailed Description

ReactorNtuple (p. 69) C++ file.

Implements the constuctor, destructor, copy constructor, and overloaded = operator. Also implements the ntuple filling with the Fill routine.

Author: Tim Bolton, Matthew Worcester

Definition in file **ReactorNtuple.cpp**.

4.24 ReactorNtuple.hh File Reference

ReactorNtuple (p. 69) header file.

Compounds

- class **ReactorNtuple**

  ReactorNtuple creates and fills the ntuple for the ReactorFsim simulation.

4.24.1 Detailed Description

ReactorNtuple (p. 69) header file.

Author: Tim Bolton

Definition in file **ReactorNtuple.hh**.

4.25 ReactorTree.cpp File Reference

ReactorTree (p. 75) C++ file.

```
#include <iostream.h>
#include <math.h>
#include <string.h>
#include <cstdio>
#include "ReactorEvent.hh"
#include "ReactorDetector.hh"
#include "ReactorFortran.hh"
#include "ReactorTree.hh"
#include "RMCEvent.hh"
#include "TTree.h"
#include "TBranch.h"
```

4.25.1 Detailed Description

ReactorTree (p. 75) C++ file.

Implements the constuctor, destructor, copy constructor, and overloaded = operator. Also implements tree filling with the Fill routine.

Author: Tim Bolton, Matthew Worcester

Definition in file **ReactorTree.cpp**.

4.26 ReactorTree.hh File Reference

ReactorTree (p. 75) header file.

```
#include "TTree.h"
#include "RMCEvent.hh"
```

Compounds

- class **ReactorTree**

  ReactorTreee creates and fills the root tree for the ReactorFsim simulation.

4.26.1 Detailed Description

ReactorTree (p. 75) header file.

Author: Tim Bolton

Definition in file **ReactorTree.hh**.

4.27 ReactorTrigger.cpp File Reference

ReactorTrigger (p. 76) C++ file.

```
#include <iostream.h>
#include <math.h>
#include <string.h>
#include <cstdio>
#include "ReactorEvent.hh"
#include "ReactorDetector.hh"
#include "ReactorFortran.hh"
#include "ReactorTrigger.hh"
```

4.27.1 Detailed Description

ReactorTrigger (p. 76) C++ file.

Implements the constuctor, destructor, copy constructor, and overloaded = operator. Also implements the trigger simulation with the Trigger routine.

Author: Matthew Worcester

Definition in file **ReactorTrigger.cpp**.

4.28 ReactorTrigger.hh File Reference

ReactorTrigger (p. 76) header file.

Compounds

- class **ReactorTrigger**

  ReactorTrigger performs a simple trigger simulation for the ReactorFsim simulation.

4.28.1 Detailed Description

ReactorTrigger (p. 76) header file.

Author: Matthew Worcester

Definition in file **ReactorTrigger.hh**.

4.29 ReactorXC.hh File Reference

ReactorXC defines several useful functions for ReactorFsim.

`#include "ReactorConstants.hh"`

`#include <math.h>`

`#include <iostream.h>`

`#include <string>`

Functions

- double **U235flux** (double)
- double **RMPflux** (double)
- double **InverseBeta** (double, double, int order=1)
- void **PositronRange** (double, double &, double &)
- double **GammaComptonRange** (std::string, double)

    Returns the mean free path of photons through a given material.

- void **GammaCompton** (double *)
- void **MyGdDecayModel** (double, int &, double *)
- void **NeutronPropagator** (double R0, double R2, double &ke, double &A, double &pass, double *pn, double *rn, double &tn, double Temperature, double *a, double *z, double *nn)

4.29.1 Detailed Description

ReactorXC defines several useful functions for ReactorFsim.

The individual functions are implimented in their respective cpp files.

Author: Tim Bolton

Definition in file **ReactorXC.hh**.

4.29.2 Function Documentation

4.29.2.1 double GammaComptonRange (std::string, double)

Returns the mean free path of photons through a given material.

The function GammaComptonRange is implimented in **GammaComptonRange.cpp**. The first input is a string defining the material through which the photon propogates. The supported materials are "scintillator" and "oil" which are defined by **ReactorConstants.hh**. The second input is the energy of the photon entering the material. The function returns a double equal to the mean free path calculated with the Klein-Nishina cross section.

Definition at line 13 of file GammaComptonRange.cpp.

```
13                                                      {
14
15   //cout << "gamma propogating through " << material << endl;
16   //
17   // Compute the mean free path for Compton scattering.
18   //
```

```
  static double m = MELECTRON;
  static double alpha = 1/ALPHAINV;
  static double Units = XcMeVtoCmsqrd;
  //
  // Overall cross section scale
  //
  static double Thomson = 8*PI*alpha*alpha/3/m/m*Units;
  double w = egamma/m;
  //
  // Klein-Nishina cross section
  //
  double XC = 3*Thomson/4*
    ((1+w)/w/w/w*(2*w*(1+w)/(1+2*w)-log(1+2*w))
     +log(1+2*w)/2/w-(1+3*w)/(1+2*w)/(1+2*w));
  //
  // Need electron density to get mfp. Get H and C contributions.
  //
  static double Na = NAVOGADRO;
  static double A[2] = {A0,A1};
  static double Z[2] = {Z0,Z1};
  static double f[2];
  if(material == "scintillator"){
    f[0] = f0;
    f[1] = f1;
  }
  else if(material == "oil"){
    f[0] = f4;
    f[1] = f5;
  }
  else{
    cout << "GammaComptonRange: " << material
         << " is an unknown material!  Assuming scintillator." << endl;
    f[0] = f0;
    f[1] = f1;
  }
  //for(int i=0;i<2;i++) cout << "f["<<i<<"] " << f[i] << endl;
  static double nel[2];
  static bool first = 1;
  for(int i=0;i<2;i++){
    nel[i]=Na*density*f[i]*Z[i]/A[i];
    first = 0;
  }
  //
  //  Compute mean free path.
  //
  return 1/((nel[0]+nel[1])*XC);
}
```

4.30 RMCEvent.hh File Reference

RMCEvent header file.

`#include <math.h>`

`#include <iostream.h>`

`#include <string.h>`

`#include "ReactorEvent.hh"`

`#include "ReactorDetector.hh"`

`#include "RMCSecondary.hh"`

`#include "TObject.h"`

`#include "TClonesArray.h"`

`#include "TObjArray.h"`

`#include "TRef.h"`

`#include "TTree.h"`

`#include "TChain.h"`

`#include "TBranch.h"`

`#include "TFile.h"`

`#include "TNamed.h"`

`#include "TBuffer.h"`

`#include "TControlBar.h"`

`#include "TCut.h"`

`#include "TEventList.h"`

`#include "TString.h"`

`#include "TH1.h"`

`#include "TButton.h"`

`#include "TArrayI.h"`

`#include "TList.h"`

`#include "TTreeFormula.h"`

`#include "TProofServ.h"`

`#include "TSystem.h"`

Compounds

- class **RMCEvent**

4.30.1 Detailed Description

RMCEvent header file.

Author: Josh Klein

Definition in file **RMCEvent.hh**.

4.31 RMCPMT.hh File Reference

RMCPMT header file.

```
#include "TObject.h"
#include "iostream.h"
#include "TTree.h"
#include "TChain.h"
#include "TBranch.h"
#include "TFile.h"
#include "TNamed.h"
#include "TBuffer.h"
#include "TControlBar.h"
#include "TCut.h"
#include "TEventList.h"
#include "TString.h"
#include "TH1.h"
#include "TButton.h"
#include "TArrayI.h"
#include "TList.h"
#include "TTreeFormula.h"
#include "TProofServ.h"
#include "TSystem.h"
```

Compounds

- class **RMCPMT**

4.31.1 Detailed Description

RMCPMT header file.

Author: Josh Klein

Definition in file **RMCPMT.hh**.

4.32 RMCSecondary.hh File Reference

RMCSecondary header file.

```
#include "TObject.h"
#include "TClonesArray.h"
#include "iostream.h"
#include "TTree.h"
#include "TChain.h"
#include "TBranch.h"
#include "TFile.h"
#include "TNamed.h"
#include "TBuffer.h"
#include "TControlBar.h"
#include "TCut.h"
#include "TEventList.h"
#include "TString.h"
#include "TH1.h"
#include "TButton.h"
#include "TArrayI.h"
#include "TList.h"
#include "ReactorDetector.hh"
#include "RMCPMT.hh"
#include "TTreeFormula.h"
#include "TProofServ.h"
#include "TSystem.h"
```

Compounds

- class **RMCSecondary**

4.32.1 Detailed Description

RMCSecondary header file.

Author: Josh Klein

Definition in file **RMCSecondary.hh**.

Index

~MuonPropagator
  MuonPropagator, 5
~MyBiSource
  MyBiSource, 17
~MyCfSource
  MyCfSource, 22
~MyHeSource
  MyHeSource, 30
~MyLiSource
  MyLiSource, 33
~MyTlSource
  MyTlSource, 36
~ReactorDetector
  ReactorDetector, 41
~ReactorEvent
  ReactorEvent, 45
~ReactorNtuple
  ReactorNtuple, 69
~ReactorTree
  ReactorTree, 75
~ReactorTrigger
  ReactorTrigger, 76

BetaHighestQ
  MyLiSource, 33
BetaHighQ
  MyHeSource, 30
  MyLiSource, 33
BetaLowQ
  MyHeSource, 30
  MyLiSource, 33
BetaMiddleQ
  MyHeSource, 30
  MyLiSource, 33

cf252_readme.txt, 77
Cf252GammaMultiplicity
  MyCfSource, 23
Cf252GammaMultiplicityFit
  MyCfSource, 23
Cf252GammaSpectrum
  MyCfSource, 23
Cf252NeutronSpectrum
  MyCfSource, 23
CleanPMTOutputArray
  ReactorDetector, 41
CloseOut
  ReactorTree, 75
COMMON_BLOCK_DEF
  ReactorFsim.cpp, 98

DetectorDefaults.hh, 78
divdif_
  ReactorFortran.hh, 97

event_readme.txt, 79
eventType
  ReactorEvent, 47

factorial
  MyCfSource.cpp, 84
Fermi
  MyHeSource, 30
  MyLiSource, 33
Fill
  ReactorNtuple, 69
  ReactorTree, 75
funlux_
  ReactorFortran.hh, 97
funlxp_
  ReactorFortran.hh, 97

gamma_
  ReactorFortran.hh, 97
GammaCompton
  ReactorXC.hh, 110
GammaComptonRange
  GammaComptonRange.cpp, 80
  ReactorXC.hh, 110
GammaComptonRange.cpp
  GammaComptonRange, 80
GammaComptonRange.cpp, 80
GetAbsorber
  ReactorDetector, 42
GetBiGammaEnergy
  MyBiSource, 21
GetCfGammaEnergy
  MyCfSource, 28
GetCfGammaTime
  MyCfSource, 22

GetCfNeutronEnergy
MyCfSource, 28
GetCfNeutronTime
MyCfSource, 22
GetCfVertexCosTheta
MyCfSource, 28
GetCfVertexPhi
MyCfSource, 28
GetCfVertexR
MyCfSource, 28
GetEgamma
ReactorDetector, 42
GetElectronCosTheta
ReactorEvent, 46
GetElectronKE
MuonPropagator, 6
ReactorEvent, 66
GetElectronParentProcess
ReactorEvent, 67
GetElectronPhi
ReactorEvent, 46
GetElectronPxyz
ReactorEvent, 46
GetElectronTime
MuonPropagator, 6
ReactorEvent, 46
GetElectronVertexCosTheta
ReactorEvent, 46
GetElectronVertexPhi
ReactorEvent, 46
GetElectronVertexR
ReactorEvent, 46
GetElectronVertexXYZ
MuonPropagator, 6
ReactorEvent, 46
GetEnu
ReactorEvent, 67
GetFluxWeight
ReactorEvent, 45
GetGamma
ReactorDetector, 42
GetGammaKE
MuonPropagator, 6
ReactorEvent, 67
GetGammaParentProcess
ReactorEvent, 67
GetGammaTime
MuonPropagator, 6
ReactorEvent, 46
GetGammaVertexCosTheta
ReactorEvent, 47
GetGammaVertexPhi
ReactorEvent, 47
GetGammaVertexR
ReactorEvent, 47
GetGammaVertexXYZ
MuonPropagator, 6
ReactorEvent, 47
GetGeneratedElectronTime
ReactorEvent, 47
GetGeneratedElectronVertexCosTheta
ReactorEvent, 47
GetGeneratedElectronVertexPhi
ReactorEvent, 47
GetGeneratedElectronVertexR
ReactorEvent, 47
GetGeneratedElectronVertexXYZ
ReactorEvent, 47
GetGeneratedNeutronTime
ReactorEvent, 47
GetGeneratedNeutronVertexCosTheta
ReactorEvent, 47
GetGeneratedNeutronVertexPhi
ReactorEvent, 47
GetGeneratedNeutronVertexR
ReactorEvent, 47
GetGeneratedNeutronVertexXYZ
ReactorEvent, 47
GetGeneratedPositronTime
ReactorEvent, 47
GetGeneratedPositronVertexCosTheta
ReactorEvent, 47
GetGeneratedPositronVertexPhi
ReactorEvent, 47
GetGeneratedPositronVertexR
ReactorEvent, 47
GetGeneratedPositronVertexXYZ
ReactorEvent, 47
GetHeBetaEnergy
MyHeSource, 30
GetHeDecayTime
MyHeSource, 30
GetHeGammaEnergy
MyHeSource, 30
GetHeNeutronEnergy
MyHeSource, 30
GetHeWeight
MyHeSource, 30
GetHitFracBi
ReactorDetector, 42
GetHitFracCf
ReactorDetector, 42
GetHitFracEle
ReactorDetector, 42
GetHitFracMuon
ReactorDetector, 42
GetHitFracNeu
ReactorDetector, 42

GetHitFracPos
ReactorDetector, 42
GetHitFracTl
ReactorDetector, 42
GetLiBetaEnergy
MyLiSource, 33
GetLiDecayTime
MyLiSource, 33
GetLiGammaEnergy
MyLiSource, 33
GetLiNeutronEnergy
MyLiSource, 33
GetLiWeight
MyLiSource, 33
GetMeanAttenBi
ReactorDetector, 42
GetMeanAttenCf
ReactorDetector, 42
GetMeanAttenEle
ReactorDetector, 42
GetMeanAttenMuon
ReactorDetector, 42
GetMeanAttenNeu
ReactorDetector, 41
GetMeanAttenPos
ReactorDetector, 41
GetMeanAttenTl
ReactorDetector, 41
GetMeanNpeBi
ReactorDetector, 41
GetMeanNpeCf
ReactorDetector, 42
GetMeanNpeEle
ReactorDetector, 42
GetMeanNpeMuon
ReactorDetector, 42
GetMeanNpeNeu
ReactorDetector, 41
GetMeanNpePos
ReactorDetector, 41
GetMeanNpeTl
ReactorDetector, 41
GetMeanQBi
ReactorDetector, 41
GetMeanQCf
ReactorDetector, 42
GetMeanQEle
ReactorDetector, 42
GetMeanQMuon
ReactorDetector, 42
GetMeanQNeu
ReactorDetector, 41
GetMeanQPos
ReactorDetector, 41
GetMeanQTl
ReactorDetector, 41
GetMeanSfactorBi
ReactorDetector, 42
GetMeanSfactorCf
ReactorDetector, 42
GetMeanSfactorEle
ReactorDetector, 42
GetMeanSfactorMuon
ReactorDetector, 42
GetMeanSfactorNeu
ReactorDetector, 41
GetMeanSfactorPos
ReactorDetector, 41
GetMeanSfactorTl
ReactorDetector, 41
GetMuonCosTheta
MuonPropagator, 5
GetMuonE
MuonPropagator, 5
GetMuonPhi
MuonPropagator, 5
GetNcosbins
ReactorDetector, 42
GetNeutronCosTheta
ReactorEvent, 46
GetNeutronDipole
ReactorDetector, 42
GetNeutronKE
MuonPropagator, 5
ReactorEvent, 67
GetNeutronParentProcess
ReactorEvent, 67
GetNeutronPhi
ReactorEvent, 46
GetNeutronPxyz
ReactorEvent, 46
GetNeutronSteps
ReactorDetector, 42
GetNeutronTime
MuonPropagator, 6
ReactorEvent, 46
GetNeutronVertexCosTheta
ReactorEvent, 47
GetNeutronVertexPhi
ReactorEvent, 47
GetNeutronVertexR
ReactorEvent, 47
GetNeutronVertexXYZ
MuonPropagator, 6
ReactorEvent, 47
GetNevent
ReactorEvent, 45
GetNgamma

ReactorDetector, 42
GetNphibins
ReactorDetector, 42
GetNpmt
ReactorDetector, 42
GetNubarParentProcess
ReactorEvent, 68
GetNubarTime
ReactorEvent, 46
GetNubarVertexCosTheta
ReactorEvent, 46
GetNubarVertexPhi
ReactorEvent, 46
GetNubarVertexR
ReactorEvent, 46
GetNubarVertexXYZ
ReactorEvent, 46
GetNumBiGammas
MyBiSource, 17
GetNumCfSources
MyCfSource, 22
GetNumElectron
MuonPropagator, 6
ReactorEvent, 45
GetNumGamma
MuonPropagator, 6
MyCfSource, 22
ReactorEvent, 45
GetNumNeutron
MuonPropagator, 5
MyCfSource, 22
ReactorEvent, 45
GetNumNubar
ReactorEvent, 45
GetNumPositron
ReactorEvent, 45
GetNumTlGammas
MyTlSource, 36
GetPHbismuth
ReactorDetector, 41
GetPHbisReco
ReactorDetector, 41
GetPHcalifornium
ReactorDetector, 41
GetPHcalReco
ReactorDetector, 41
GetPHelectron
ReactorDetector, 41
GetPHeleReco
ReactorDetector, 41
GetPHmuon
ReactorDetector, 41
GetPHmuoReco
ReactorDetector, 41
GetPHneuReco
ReactorDetector, 41
GetPHneutron
ReactorDetector, 41
GetPHpositron
ReactorDetector, 41
GetPHposReco
ReactorDetector, 41
GetPHthallium
ReactorDetector, 41
GetPHthaReco
ReactorDetector, 41
GetPmtRad
ReactorEvent, 45
GetPositronCosTheta
ReactorEvent, 46
GetPositronDipole
ReactorDetector, 42
GetPositronKE
ReactorEvent, 68
GetPositronParentProcess
ReactorEvent, 68
GetPositronPhi
ReactorEvent, 46
GetPositronPxyz
ReactorEvent, 46
GetPositronTime
ReactorEvent, 46
GetPositronVertexCosTheta
ReactorEvent, 47
GetPositronVertexPhi
ReactorEvent, 47
GetPositronVertexR
ReactorEvent, 46
GetPositronVertexXYZ
ReactorEvent, 47
GetRefracSc
ReactorDetector, 42
GetTgamma
ReactorDetector, 42
GetTlGammaEnergy
MyTlSource, 40
GetWeight
MuonPropagator, 6
GetXCWeight
ReactorEvent, 45

hbook1_
ReactorFortran.hh, 97
hbookn_
ReactorFortran.hh, 97
hcdir_
ReactorFortran.hh, 97
hfill_

ReactorFortran.hh, 97
hfn_
ReactorFortran.hh, 97
hldir_
ReactorFortran.hh, 97
hlimit_
ReactorFortran.hh, 97
hrend_
ReactorFortran.hh, 97
hropen_
ReactorFortran.hh, 97
hrout_
ReactorFortran.hh, 97

Intersection
MuonPropagator, 5
InverseBeta
ReactorXC.hh, 110
IsotopeDistance
MuonPropagator, 6

LightsOut
ReactorDetector, 41

main
ReactorFsim.cpp, 99
MuonPropagator
~MuonPropagator, 5
GetElectronKE, 6
GetElectronTime, 6
GetElectronVertexXYZ, 6
GetGammaKE, 6
GetGammaTime, 6
GetGammaVertexXYZ, 6
GetMuonCosTheta, 5
GetMuonE, 5
GetMuonPhi, 5
GetNeutronKE, 5
GetNeutronTime, 6
GetNeutronVertexXYZ, 6
GetNumElectron, 6
GetNumGamma, 6
GetNumNeutron, 5
GetWeight, 6
Intersection, 5
IsotopeDistance, 6
MuonPropagator, 5, 6
operator=, 5
RadCorr, 5
Rotate, 5
MuonPropagator, 5
MuonPropagator, 6
MuonPropagator.cpp, 81
MyBiSource
~MyBiSource, 17
GetNumBiGammas, 17
MyBiSource, 17
operator=, 17
MyBiSource, 17
GetBiGammaEnergy, 21
MyBiSource, 17
MyBiSource.cpp, 82
MyBiSource.hh, 83
MyCfSource
~MyCfSource, 22
Cf252GammaMultiplicity, 23
Cf252GammaMultiplicityFit, 23
Cf252GammaSpectrum, 23
Cf252NeutronSpectrum, 23
GetCfGammaTime, 22
GetCfNeutronTime, 22
GetNumCfSources, 22
GetNumGamma, 22
GetNumNeutron, 22
MyCfSource, 22, 23
operator=, 22
MyCfSource, 22
GetCfGammaEnergy, 28
GetCfNeutronEnergy, 28
GetCfVertexCosTheta, 28
GetCfVertexPhi, 28
GetCfVertexR, 28
MyCfSource, 23
MyCfSource.cpp
factorial, 84
MyCfSource.cpp, 84
MyCfSource.hh, 85
MyGdDecayModel
ReactorXC.hh, 110
MyHeSource
~MyHeSource, 30
BetaHighQ, 30
BetaLowQ, 30
BetaMiddleQ, 30
Fermi, 30
GetHeBetaEnergy, 30
GetHeDecayTime, 30
GetHeGammaEnergy, 30
GetHeNeutronEnergy, 30
GetHeWeight, 30
MyHeSource, 30, 31
operator=, 30
MyHeSource, 30
MyHeSource, 31
MyHeSource.cpp, 86
MyHeSource.hh, 87
MyLiSource
~MyLiSource, 33

BetaHighestQ, 33
BetaHighQ, 33
BetaLowQ, 33
BetaMiddleQ, 33
Fermi, 33
GetLiBetaEnergy, 33
GetLiDecayTime, 33
GetLiGammaEnergy, 33
GetLiNeutronEnergy, 33
GetLiWeight, 33
MyLiSource, 33, 34
operator=, 33

MyLiSource, 33
MyLiSource, 34

MyLiSource.cpp, 88

MyLiSource.hh, 89

MyTlSource
~MyTlSource, 36
GetNumTlGammas, 36
MyTlSource, 36
operator=, 36

MyTlSource, 36
GetTlGammaEnergy, 40
MyTlSource, 36

MyTlSource.cpp, 90

MyTlSource.hh, 91

NeutronPropagator
ReactorXC.hh, 110

operator=
MuonPropagator, 5
MyBiSource, 17
MyCfSource, 22
MyHeSource, 30
MyLiSource, 33
MyTlSource, 36
ReactorDetector, 41
ReactorEvent, 45
ReactorNtuple, 69
ReactorTrigger, 76

PassTrigger
ReactorTrigger, 76

PAWC
ReactorFsim.cpp, 98

PAWC_SIZE
ReactorFsim.cpp, 98

Pdim
ReactorEvent, 47

PMToutput
ReactorDetector, 43

pmtRad
ReactorEvent, 47

PositronRange
ReactorXC.hh, 110

RadCorr
MuonPropagator, 5

ranlux_
ReactorFortran.hh, 97

Rdim
ReactorEvent, 47

ReactorConstants.hh, 92

ReactorDetector
~ReactorDetector, 41
CleanPMTOutputArray, 41
GetAbsorber, 42
GetEgamma, 42
GetGamma, 42
GetHitFracBi, 42
GetHitFracCf, 42
GetHitFracEle, 42
GetHitFracMuon, 42
GetHitFracNeu, 42
GetHitFracPos, 42
GetHitFracTl, 42
GetMeanAttenBi, 42
GetMeanAttenCf, 42
GetMeanAttenEle, 42
GetMeanAttenMuon, 42
GetMeanAttenNeu, 41
GetMeanAttenPos, 41
GetMeanAttenTl, 41
GetMeanNpeBi, 41
GetMeanNpeCf, 42
GetMeanNpeEle, 42
GetMeanNpeMuon, 42
GetMeanNpeNeu, 41
GetMeanNpePos, 41
GetMeanNpeTl, 41
GetMeanQBi, 41
GetMeanQCf, 42
GetMeanQEle, 42
GetMeanQMuon, 42
GetMeanQNeu, 41
GetMeanQPos, 41
GetMeanQTl, 41
GetMeanSfactorBi, 42
GetMeanSfactorCf, 42
GetMeanSfactorEle, 42
GetMeanSfactorMuon, 42
GetMeanSfactorNeu, 41
GetMeanSfactorPos, 41
GetMeanSfactorTl, 41
GetNcosbins, 42
GetNeutronDipole, 42
GetNeutronSteps, 42

GetNgamma, 42
GetNphibins, 42
GetNpmt, 42
GetPHbismuth, 41
GetPHbisReco, 41
GetPHcalifornium, 41
GetPHcalReco, 41
GetPHelectron, 41
GetPHeleReco, 41
GetPHmuon, 41
GetPHmuoReco, 41
GetPHneuReco, 41
GetPHneutron, 41
GetPHpositron, 41
GetPHposReco, 41
GetPHthallium, 41
GetPHthaReco, 41
GetPositronDipole, 42
GetRefracSc, 42
GetTgamma, 42
LightsOut, 41
operator=, 41
PMToutput, 43
ReactorDetector, 41, 43
ReactorDetector::GetHitPMTsBis, 42
ReactorDetector::GetHitPMTsCal, 42
ReactorDetector::GetHitPMTsEle, 42
ReactorDetector::GetHitPMTsMuo, 42
ReactorDetector::GetHitPMTsNeu, 42
ReactorDetector::GetHitPMTsPos, 42
ReactorDetector::GetHitPMTsTha, 42
ReactorDetector::GetPMTdata, 42
SetParam_attenlGd, 43
SetParam_attenlSc, 43
SetParam_FastNeutronOption, 43
SetParam_GammasPerGd, 43
SetParam_GdCaptureFraction, 43
SetParam_GdConcentration, 43
SetParam_MeanNeuDispl, 43
SetParam_mfpGd, 43
SetParam_mfpSc, 43
SetParam_PhotonsPerMeV, 43
SetParam_PMTcoverage, 43
SetParam_PMTDebugOption, 43
SetParam_PMTdiameter, 43
SetParam_PMTMergeTime, 43
SetParam_PMTqe, 43
SetParam_R0, 42
SetParam_R1, 43
SetParam_R2, 43
SetParam_refracGd, 43
SetParam_refracSc, 43
SetParam_ScintDecayTime, 43
SetParam_Temperature, 43
SetParam_tGd, 43
SetParam_tSc, 43
TranslateRadialDistance, 43
ReactorDetector, 41
ReactorDetector, 43
ReactorDetector.cpp, 93
ReactorDetector.hh, 94
ReactorDetector::GetHitPMTsBis
ReactorDetector, 42
ReactorDetector::GetHitPMTsCal
ReactorDetector, 42
ReactorDetector::GetHitPMTsEle
ReactorDetector, 42
ReactorDetector::GetHitPMTsMuo
ReactorDetector, 42
ReactorDetector::GetHitPMTsNeu
ReactorDetector, 42
ReactorDetector::GetHitPMTsPos
ReactorDetector, 42
ReactorDetector::GetHitPMTsTha
ReactorDetector, 42
ReactorDetector::GetPMTdata
ReactorDetector, 42
ReactorEvent
~ReactorEvent, 45
eventType, 47
GetElectronCosTheta, 46
GetElectronPhi, 46
GetElectronPxyz, 46
GetElectronTime, 46
GetElectronVertexCosTheta, 46
GetElectronVertexPhi, 46
GetElectronVertexR, 46
GetElectronVertexXYZ, 46
GetFluxWeight, 45
GetGammaTime, 46
GetGammaVertexCosTheta, 47
GetGammaVertexPhi, 47
GetGammaVertexR, 47
GetGammaVertexXYZ, 47
GetGeneratedElectronTime, 47
GetGeneratedElectronVertexCosTheta, 47
GetGeneratedElectronVertexPhi, 47
GetGeneratedElectronVertexR, 47
GetGeneratedElectronVertexXYZ, 47
GetGeneratedNeutronTime, 47
GetGeneratedNeutronVertexCosTheta, 47
GetGeneratedNeutronVertexPhi, 47
GetGeneratedNeutronVertexR, 47
GetGeneratedNeutronVertexXYZ, 47
GetGeneratedPositronTime, 47

GetGeneratedPositronVertexCosTheta, 47
GetGeneratedPositronVertexPhi, 47
GetGeneratedPositronVertexR, 47
GetGeneratedPositronVertexXYZ, 47
GetNeutronCosTheta, 46
GetNeutronPhi, 46
GetNeutronPxyz, 46
GetNeutronTime, 46
GetNeutronVertexCosTheta, 47
GetNeutronVertexPhi, 47
GetNeutronVertexR, 47
GetNeutronVertexXYZ, 47
GetNevent, 45
GetNubarTime, 46
GetNubarVertexCosTheta, 46
GetNubarVertexPhi, 46
GetNubarVertexR, 46
GetNubarVertexXYZ, 46
GetNumElectron, 45
GetNumGamma, 45
GetNumNeutron, 45
GetNumNubar, 45
GetNumPositron, 45
GetPmtRad, 45
GetPositronCosTheta, 46
GetPositronPhi, 46
GetPositronPxyz, 46
GetPositronTime, 46
GetPositronVertexCosTheta, 47
GetPositronVertexPhi, 47
GetPositronVertexR, 46
GetPositronVertexXYZ, 47
GetXCWeight, 45
operator=, 45
Pdim, 47
pmtRad, 47
Rdim, 47
ReactorEvent, 45, 48
SetNeutronVertex, 47
SetPositronVertex, 47
XCReweight, 45

ReactorEvent, 45
GetElectronKE, 66
GetElectronParentProcess, 67
GetEnu, 67
GetGammaKE, 67
GetGammaParentProcess, 67
GetNeutronKE, 67
GetNeutronParentProcess, 67
GetNubarParentProcess, 68
GetPositronKE, 68
GetPositronParentProcess, 68
ReactorEvent, 48

ReactorEvent.cpp, 95
ReactorEvent.hh, 96
ReactorFortran.hh
divdif_, 97
funlux_, 97
funlxp_, 97
gamma_, 97
hbook1_, 97
hbookn_, 97
hcdir_, 97
hfill_, 97
hfn_, 97
hldir_, 97
hlimit_, 97
hrend_, 97
hropen_, 97
hrout_, 97
ranlux_, 97
rluxgo_, 97
rnhpre_, 97
rnhran_, 97
rnormx_, 97
rnpssn_, 97
wgamma_, 97

ReactorFortran.hh, 97
ReactorFsim.cpp
COMMON_BLOCK_DEF, 98
PAWC, 98
PAWC_SIZE, 98

ReactorFsim.cpp, 98
main, 99

ReactorNtuple
~ReactorNtuple, 69
Fill, 69
operator=, 69
ReactorNtuple, 69

ReactorNtuple, 69
ReactorNtuple, 69

ReactorNtuple.cpp, 104
ReactorNtuple.hh, 105
ReactorTree
~ReactorTree, 75
CloseOut, 75
Fill, 75
ReactorTree, 75

ReactorTree, 75
ReactorTree, 75

ReactorTree.cpp, 106
ReactorTree.hh, 107
ReactorTrigger
~ReactorTrigger, 76
operator=, 76
PassTrigger, 76
ReactorTrigger, 76

Trigger, 76
ReactorTrigger, 76
ReactorTrigger, 76
ReactorTrigger.cpp, 108
ReactorTrigger.hh, 109
ReactorXC.hh
GammaCompton, 110
InverseBeta, 110
MyGdDecayModel, 110
NeutronPropagator, 110
PositronRange, 110
RMPflux, 110
U235flux, 110
ReactorXC.hh, 110
GammaComptonRange, 110
rluxgo_
ReactorFortran.hh, 97
RMCEvent.hh, 112
RMCPMT.hh, 114
RMCSecondary.hh, 115
RMPflux
ReactorXC.hh, 110
rnhpre_
ReactorFortran.hh, 97
rnhran_
ReactorFortran.hh, 97
rnormx_
ReactorFortran.hh, 97
rnpssn_
ReactorFortran.hh, 97
Rotate
MuonPropagator, 5

SetNeutronVertex
ReactorEvent, 47
SetParam_attenlGd
ReactorDetector, 43
SetParam_attenlSc
ReactorDetector, 43
SetParam_FastNeutronOption
ReactorDetector, 43
SetParam_GammasPerGd
ReactorDetector, 43
SetParam_GdCaptureFraction
ReactorDetector, 43
SetParam_GdConcentration
ReactorDetector, 43
SetParam_MeanNeuDispl
ReactorDetector, 43
SetParam_mfpGd
ReactorDetector, 43
SetParam_mfpSc
ReactorDetector, 43
SetParam_PhotonsPerMeV
ReactorDetector, 43
SetParam_PMTcoverage
ReactorDetector, 43
SetParam_PMTDebugOption
ReactorDetector, 43
SetParam_PMTdiameter
ReactorDetector, 43
SetParam_PMTMergeTime
ReactorDetector, 43
SetParam_PMTqe
ReactorDetector, 43
SetParam_R0
ReactorDetector, 42
SetParam_R1
ReactorDetector, 43
SetParam_R2
ReactorDetector, 43
SetParam_refracGd
ReactorDetector, 43
SetParam_refracSc
ReactorDetector, 43
SetParam_ScintDecayTime
ReactorDetector, 43
SetParam_Temperature
ReactorDetector, 43
SetParam_tGd
ReactorDetector, 43
SetParam_tSc
ReactorDetector, 43
SetPositronVertex
ReactorEvent, 47

TranslateRadialDistance
ReactorDetector, 43
Trigger
ReactorTrigger, 76

U235flux
ReactorXC.hh, 110

wgamma_
ReactorFortran.hh, 97

XCReweight
ReactorEvent, 45