ReactorFsim Reference Manual

0_0_4

Generated by Doxygen 1.2.14

Mon Dec 27 11:10:13 2004

Contents

Chapter 1

ReactorFsim Compound Index

1.1 ReactorFsim Compound List

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

Chapter 2

ReactorFsim File Index

2.1 ReactorFsim File List

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

Chapter 3

ReactorFsim Class Documentation

3.1 MuonPropagator Class Reference

MuonPropogator makes muons for **ReactorEvent** (p. 37).

`#include <MuonPropagator.hh>`

Public Methods

- **MuonPropagator** (int newRmaxgen=k.Rmaxgen, double newDepth=450)

  MuonPropogator constructor.

- ~**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 makes muons for **ReactorEvent** (p. 37).

Definition at line 14 of file MuonPropagator.hh.

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=k.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* = k.Isotope)

MyBiSource constructor.

Called with the desired Thallium isotope: 212 or 214. Default is the isotope given in **ReactorConstants** (p. 31), 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;
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
  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=k.Isotope, int newRmaxgen=k.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** (p. 31). 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 Cf252GammaMultiplicityFit 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* = k.Isotope, int *newRmaxgen* = k.Rmaxgen)

MyCfSource constructor.

Called with the desired Cf isotope: 252, which is the default isotope given in **ReactorConstants** (p. 31) 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, Cf252NeutronSpectrum, ReactorConstants::Ndim, and ReactorConstants::pi.

```
                                          {

  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*k.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){
        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 = -1.33 + 119.6*pow(Z,0.3333)/A;
      double phi = 2.51 - 1.13e-5*Z*Z*sqrt(A);
      double 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++){
        len = 1;
        funlux_(gspace,gammaE[n][nn],len);
        //cout << "   252Cf photon energy = " << gammaE[n][nn] << endl;
        tote += gammaE[n][nn];

        len = 2;
        float rv[len];
        ranlux_(rv,len);
        //
```

```
        // 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<k.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<k.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=k.Isotope, int newRmaxgen=k.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* = k.Isotope, int *newRmaxgen* = k.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.

```
                                      {

  Isotope = newIsotope;
  Rmaxgen = newRmaxgen;

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

  // 8He
  if(Isotope == 8){

    // setup the decay parameters (from Jon Link)
    // Nneutron is always 1 and Ngamma always 0
    Nneutron = 1;
    Ngamma = 0;

    // branching fraction (the 1.0000 bin contains all decays
    // which do nothing: neutron E = 0, beta Q = 0)
    float static br[4] = {0.080,0.160,0.169,1.0000};

    // Q-value for beta decay
    float static betaQ[4] = {7.442,5.252,1.852,0.000};

    // Q-value for the total decay chain
    float static totQ = 8.619;

    // neutron energy
    float static neuE[4] = {0.697,2.887,6.727,0.000};

    // prepare the beta decay functions
    bool static first=1;
    float static fspace[3][200];

    if(first){
      float xlo = 0.;
      funlxp_(BetaHighQ,fspace[0],xlo,betaQ[0]);
      funlxp_(BetaMiddleQ,fspace[1],xlo,betaQ[1]);
      funlxp_(BetaLowQ,fspace[2],xlo,betaQ[2]);

      first = 0;
    }
    //
    // let the 8He decay
    //
    int ntrial = 0;
    bool passed = false;
    while(!passed){
      int len=1;
      float rv[len];
      ranlux_(rv,len);
      //
      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
          len = 1;
          funlux_(fspace[i],betaE,len);
          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;

  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=k.Isotope, int newRmaxgen=k.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* = k.Isotope, int *newRmaxgen* = k.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){
      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
            len = 1;
            funlux_(fspace[i],betaE,len);
            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;

     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=k.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* = k.Isotope)

MyTlSource constructor.

Called with the desired Thallium isotope: 208 or 210. Default is the isotope given in **ReactorConstants** (p. 31), 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;
  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
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 ReactorConstants Class Reference

ReactorConstants sets all of the physical constants, unit conversions, and detector parameters used throughout ReactorFsim.

`#include <ReactorConstants.hh>`

Static Public Attributes

- const int **XCorder** = 1
- const double **Emin** = 1.85
- const double **Emax** = 8.15
- const double **Rmaxgen** = 265
- const int **Isotope** = 252
- const int **Ndim** = 200
- const double **R0** = 175
- const double **R1** = 200
- const double **R2** = 275
- const bool **FastNeutronOption** = 0
- const double **SlowNeutronStep** = 0.1
- const double **Temperature** = 300
- const double **Kboltzman** = 8.617342e-11
- const double **pi** = 3.1415928
- const double **Mproton** = 938.27200
- const double **Delta** = 1.2933318
- const double **Melectron** = 0.510998902
- const double **Mmuon** = 105.6583692
- const double **alphainv** = 137.03599976
- const double **Gfermi** = 1.16639e-11
- const double **Navogadro** = 6.02214199e23
- const double **Relectron** = 2.817940285e-13
- const double **XcMeVtoCmsqrd** = 0.389379292e-21
- const double **c** = 29.9792458
- const double **density** = 0.85
- const double **A0** = 1
- const double **A1** = 12
- const double **A2** = 155
- const double **A3** = 157
- const double **Z0** = 1
- const double **Z1** = 6
- const double **Z2** = 64
- const double **Z3** = 64
- const double **f0** = 0.122
- const double **f1** = 0.844
- const double **f2** = 0.148
- const double **f3** = 0.156
- const double **f4** = 0.133
- const double **f5** = 0.855
- const double **I0** = 19.2e-6
- const double **I1** = 78.0e-6

- const double **refracGd** = 1.472
- const double **refracSc** = 1.476
- const double **GdConcentration** = 0.1
- const double **GdConcentrationRef** = 0.1
- const double **mfpGd** = 6
- const double **mfpSc** = 40
- const double **MeanNeuDispl** = 1.7
- const double **tGd** = 24.7e3
- const double **tSc** = 145.8e3
- const double **GdCaptureFraction** = 0.841
- const double **Gd155frac** = 0.1848
- const double **Gd155peak** = 8.536
- const double **Gd157frac** = 0.8152
- const double **Gd157peak** = 7.937
- const double **GammasPerGd** = 3
- const double **Hpeak** = 2.2246
- const double **Cpeak** = 4.9461
- const double **attenlGd** = 400
- const double **attenlSc** = 1000
- const double **PhotonsPerMeV** = 5300
- const double **EScale** = 0.91675
- const double **ScintDecayTime** = 7
- const double **ShortPosFrac0** = 0.30
- const double **ShortPosFrac1** = 0.21
- const double **LongPosFrac** = 0.49
- const double **ShortPosTime0** = 0.19
- const double **ShortPosTime1** = 0.48
- const double **LongPosTime** = 3.41
- const double **PMTcoverage** = 0.2
- const double **PMTdiameter** = 20.16
- const double **PMTqe** = 0.2

3.7.1 Detailed Description

ReactorConstants sets all of the physical constants, unit conversions, and detector parameters used throughout ReactorFsim.

Definition at line 13 of file ReactorConstants.hh.

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

- **ReactorConstants.hh**

3.8 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)
- 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 **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)
- 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_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)

3.8.1 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 21 of file ReactorDetector.hh.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 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 24 of file ReactorDetector.cpp.

```
24                               {
25   //
26   // initialize PMT arrays
27   //
28   Xpmt = new double[0];
29   Ypmt = new double[0];
30   Zpmt = new double[0];
31   PMTdata = new PMT[0];
32   PMTelectron = new PMT[0];
33   PMTpositron = new PMT[0];
34   PMTneutron = new PMT[0];
35   PMTthallium = new PMT[0];
36   PMTbismuth = new PMT[0];
37   PMTcalifornium = new PMT[0];
38   PMTmuon = new PMT[0];
39   //
40   // 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;
}
```

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

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

3.9 ReactorEvent Class Reference

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

`#include <ReactorEvent.hh>`

Public Methods

- **ReactorEvent** (int newNevents, double newRmaxgen=k.Rmaxgen, double newEmin=k.Emin, double newEmax=k.Emax, int newXCorder=k.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**

Static Public Attributes

- const int **Pdim** = 7
- const int **Rdim** = 7

3.9.1 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.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 ReactorEvent::ReactorEvent (int *newNevents*, double *newRmaxgen* = k.Rmaxgen, double *newEmin* = k.Emin, double *newEmax* = k.Emax, int *newXCorder* = k.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 20 of file ReactorEvent.cpp.

References ReactorConstants::Delta, 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, MuonPropagator::GetWeight, ReactorConstants::Melectron, ReactorConstants::Mproton, ReactorConstants::Ndim, and ReactorConstants::pi.

```
                                         {

Nevents = newNevents;
Rmaxgen = newRmaxgen;
Emin = newEmin;
Emax = newEmax;
XCorder = newXCorder;

/*
cout << "ReactorEvent::ReactorEvent(" << Nevents << ","
     << Rmaxgen << "," << Emin << "," << Emax << ","
     << XCorder << ")" << endl;
*/

int nmax = k.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 int    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"){
        set = true;
      }
      else{
        cout << "  SETUP ERROR: unrecognized event type!" << 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, 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() == '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"){
    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,-1);
```

```
    double FluxMax = RMPflux(Emin);

    bool passed=0;

    while(!passed){
      int len = 7;
      float r[len];
      ranlux_(r,len);
      Enu = Emin+(Emax-Emin)*r[0];
      z = -1+2*r[1];
      phi = 2*k.pi*r[2];

      R = Rmaxgen*exp(log(r[3])/3);
      zR = -1+2*r[4];
      phiR = 2*k.pi*r[5];

      float XCtest = XCmax*FluxMax*r[6];
      XCWeight = InverseBeta(Enu,z);
      FluxWeight=RMPflux(Enu);
      passed= XCWeight*FluxWeight>XCtest;
    }

    double E0 = Enu - k.Delta;
    double p0 = sqrt(E0*E0-k.Melectron*k.Melectron);
    double v0 = p0/E0;
    double Ysquared = (k.Delta*k.Delta-k.Melectron*k.Melectron)/2;
    double E1 = E0*(1-Enu/k.Mproton*(1-v0*z))-Ysquared/k.Mproton;
    double p1 = sqrt(E1*E1-k.Melectron*k.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-k.Melectron; ppos++;
    *ppos = p1; ppos++;
    *ppos = 1.; ppos++;

    *pneu = Enu+k.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-k.Delta; pneu++;
    *pneu = sqrt((Enu+k.Mproton-E1)*(Enu+k.Mproton-E1)-
                     (k.Mproton+k.Delta)*(k.Mproton+k.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
  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 = k.Mproton+k.Delta;
        double neutronE = neutronKE+Mneutron;

```

```
      // pick a direction for neutron momentum
      int len = 2;
      float r[len];
      ranlux_(r,len);
      z = -1+2*r[0];
      phi = 2*k.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 photons
      //
      int len = 5;
      float rv[len];
      ranlux_(rv,len);
      double cosg = -1+2*rv[0];
      double phig = 2*k.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)
        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*k.pi;
      //
      // pick a direction for gamma momentum
      //
      float r[len];
      ranlux_(r,len);
      z = -1+2*rv[3];
      phi = 2*k.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
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
if(eventType == "muon"){

  //cout << eventType << endl;
  MuonPropagator Muon(Rmaxgen,Depth);

  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 = k.Mproton+k.Delta;
    float neutronE = neutronKE+Mneutron;

    // pick a direction for neutron momentum
    int len = 2;
    float rv[len];
    ranlux_(rv,len);
    z = -1+2*rv[0];
    phi = 2*k.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*k.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 Melectron = k.Melectron;
    float electronE = electronKE+Melectron;
    //cout << "  electron " << n << " energy = " << electronE << endl;

    // pick a direction for electron momentum
    int len = 2;
    float r[len];
    ranlux_(r,len);
    z = -1+2*r[0];
    phi = 2*k.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*k.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
      int len = 2;
      float r[len];
      ranlux_(r,len);
      z = -1+2*r[0];
      phi = 2*k.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*k.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++;
    }
  }
  //
  // 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;
}
```

3.9.3 Member Function Documentation

3.9.3.1 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 99 of file ReactorEvent.hh.

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

3.9.3.2 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 106 of file ReactorEvent.hh.

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

3.9.3.3 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 88 of file ReactorEvent.hh.

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

3.9.3.4 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 141 of file ReactorEvent.hh.

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

3.9.3.5 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 146 of file ReactorEvent.hh.

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

3.9.3.6 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 127 of file ReactorEvent.hh.

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

3.9.3.7 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 134 of file ReactorEvent.hh.

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

3.9.3.8 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 93 of file ReactorEvent.hh.

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

3.9.3.9 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 113 of file ReactorEvent.hh.

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

3.9.3.10 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 120 of file ReactorEvent.hh.

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

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

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

3.10 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)

3.10.1 Detailed Description

ReactorNtuple creates and fills the ntuple for the ReactorFsim simulation.

Definition at line 12 of file ReactorNtuple.hh.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 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 = 8;
  int ntVarsMax = 127;
  int ntVars = 0;
  char ntTags[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",10,0,10,0,
        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**

3.11 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 event_readme.txt File Reference

Contains the documentation for the event setup text files.

4.2.1 Detailed Description

Contains the documentation for the event setup text files.

Author: Matthew Worcester

Definition in file **event_readme.txt**.

4.3 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)

4.3.1 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**.

4.4 MuonPropagator.cpp File Reference

MuonPropagator (p. 5) C++ file.

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

4.4.1 Detailed Description

MuonPropagator (p. 5) C++ file.

Author: Matthew Worcester

Definition in file **MuonPropagator.cpp**.

4.5 MyBiSource.cpp File Reference

MyBiSource (p. 7) 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"
```

4.5.1 Detailed Description

MyBiSource (p. 7) C++ file.

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

Author: Matthew Worcester

Definition in file **MyBiSource.cpp**.

4.6 MyBiSource.hh File Reference

MyBiSource (p. 7) header file.

`#include <math.h>`

`#include <iostream.h>`

`#include "ReactorConstants.hh"`

Compounds

- class **MyBiSource**
  MyBiSource generates photons from radioactive Thallium decays.

4.6.1 Detailed Description

MyBiSource (p. 7) header file.

Author: Matthew Worcester

Definition in file **MyBiSource.hh**.

4.7 MyCfSource.cpp File Reference

MyCfSource (p. 12) 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)

4.7.1 Detailed Description

MyCfSource (p. 12) C++ file.

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

Author: Matthew Worcester

Definition in file **MyCfSource.cpp**.

4.8 MyCfSource.hh File Reference

MyCfSource (p. 12) 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.

4.8.1 Detailed Description

MyCfSource (p. 12) header file.

Author: Matthew Worcester

Definition in file **MyCfSource.hh**.

4.9 MyHeSource.cpp File Reference

MyHeSource (p. 20) 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"
```

4.9.1 Detailed Description

MyHeSource (p. 20) C++ file.

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

Author: Matthew Worcester

Definition in file **MyHeSource.cpp**.

4.10 MyHeSource.hh File Reference

MyHeSource (p. 20) header file.

`#include <math.h>`

`#include <iostream.h>`

`#include "ReactorConstants.hh"`

Compounds

- class **MyHeSource**

  MyHeSource generates delayed neutrons from radioactive He decays.

4.10.1 Detailed Description

MyHeSource (p. 20) header file.

Author: Matthew Worcester

Definition in file **MyHeSource.hh**.

4.11 MyLiSource.cpp File Reference

MyLiSource (p. 23) 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"
```

4.11.1 Detailed Description

MyLiSource (p. 23) C++ file.

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

Author: Matthew Worcester

Definition in file **MyLiSource.cpp**.

4.12 MyLiSource.hh File Reference

MyLiSource (p. 23) header file.

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

Compounds

- class **MyLiSource**

  MyLiSource generates delayed neutrons from radioactive Li decays.

4.12.1 Detailed Description

MyLiSource (p. 23) header file.

Author: Matthew Worcester

Definition in file **MyLiSource.hh**.

4.13 MyTlSource.cpp File Reference

MyTlSource (p. 26) 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"
```

4.13.1 Detailed Description

MyTlSource (p. 26) C++ file.

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

Author: Matthew Worcester

Definition in file **MyTlSource.cpp**.

4.14 MyTlSource.hh File Reference

MyTlSource (p. 26) header file.

`#include <math.h>`

`#include <iostream.h>`

`#include "ReactorConstants.hh"`

Compounds

- class **MyTlSource**
  MyTlSource generates photons from radioactive Thallium decays.

4.14.1 Detailed Description

MyTlSource (p. 26) header file.

Author: Matthew Worcester

Definition in file **MyTlSource.hh**.

4.15 ReactorConstants.hh File Reference

ReactorConstants (p. 31) header file.

Compounds

- class **ReactorConstants**

  ReactorConstants sets all of the physical constants, unit conversions, and detector parameters used throughout ReactorFsim.

4.15.1 Detailed Description

ReactorConstants (p. 31) header file.

Author: Tim Bolton

Definition in file **ReactorConstants.hh**.

4.16 ReactorDetector.cpp File Reference

ReactorDetector (p. 33) C++ file.

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

4.16.1 Detailed Description

ReactorDetector (p. 33) 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**.

4.17 ReactorDetector.hh File Reference

ReactorDetector (p. 33) header file.

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

Compounds

- class **ReactorDetector::PMT**
- class **ReactorDetector**

  ReactorDetector handles the detector implementation in the ReactorFsim simulation.

4.17.1 Detailed Description

ReactorDetector (p. 33) header file.

Author: Tim Bolton

Definition in file **ReactorDetector.hh**.

4.18 ReactorEvent.cpp File Reference

ReactorEvent (p. 37) C++ file.

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

4.18.1 Detailed Description

ReactorEvent (p. 37) C++ file.

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

Author: Tim Bolton, Matthew Worcester

Definition in file **ReactorEvent.cpp**.

4.19 ReactorEvent.hh File Reference

ReactorEvent (p. 37) 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.

4.19.1 Detailed Description

ReactorEvent (p. 37) header file.

Author: Tim Bolton, Matthew Worcester

Definition in file **ReactorEvent.hh**.

4.20 ReactorFortran.hh File Reference

Defines the Fortran prototypes for calling from C++.

`#include <complex>`

Functions

- void **hbook_init__** ()
- 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 **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 *)

4.20.1 Detailed Description

Defines the Fortran prototypes for calling from C++.

Necessary for the Hbook interface and Cernlib Fortran routines.

Definition in file **ReactorFortran.hh**.

4.21 ReactorFsim.cpp File Reference

ReactorFsim simulation for MWtheta13.

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

Functions

- int **main** ()

  Top level function of ReactorFsim.

4.21.1 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, Matthew Worcester

Definition in file **ReactorFsim.cpp**.

4.21.2 Function Documentation

4.21.2.1 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 30 of file ReactorFsim.cpp.

References ReactorNtuple::Fill, ReactorDetector::LightsOut, ReactorDetector::SetParam_FastNeutronOption, ReactorDetector::SetParam_R0, ReactorDetector::SetParam_R1, ReactorDetector::SetParam_R2, and ReactorTrigger::Trigger.

```
        {
//  ReactorNeutrons Neutrons;
//
//  Set number of events, paw file name, and ntuple name
//
// defaults
int nevents = 100;
double r0=260;
double r1=260;
double rmaxgen=270;
double r2=350;
int fast=0;
int trigger=0;
char* fileName="test.ntuple";
char* ntName="event data";
//
// 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;
std::string fastneutron;
std::string trigsim;
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);

      char newFile[256];
      fin >> newFile;
      fileName = newFile;

       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;
     }

     // '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_4 by T. Bolton" << endl;
  cout << "=============================" << endl;
  cout << "  " << nevents << " events" << endl;
  cout << "  Output file: " << fileName << 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 << "=============================" << endl;
  //
  // init random number seed
  //
  int luxury = 2;
  int seed = 8796896;
  int k1=0;
  int k2=0;
  rluxgo_(luxury,seed,k1,k2);
  //
  // Paw setup.
  //
  hbook_init__();
  //
  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_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);
    ModelA.Fill(Event,DetectorA);
    if(trigger) TriggerA.Trigger(Event,DetectorA);
  }
  //
  // close up paw
  //
  hldir_(" "," ",strlen(" "),strlen(" "));
  char dirName2[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));
}
```

4.22 ReactorNtuple.cpp File Reference

ReactorNtuple (p. 55) 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"
```

4.22.1 Detailed Description

ReactorNtuple (p. 55) 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**.

4.23 ReactorNtuple.hh File Reference

ReactorNtuple (p. 55) header file.

Compounds

- class **ReactorNtuple**

  ReactorNtuple creates and fills the ntuple for the ReactorFsim simulation.

4.23.1 Detailed Description

ReactorNtuple (p. 55) header file.

Author: Tim Bolton

Definition in file **ReactorNtuple.hh**.

4.24 ReactorTrigger.cpp File Reference

ReactorTrigger (p. 61) 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"
```

4.24.1 Detailed Description

ReactorTrigger (p. 61) 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**.

4.25 ReactorTrigger.hh File Reference

ReactorTrigger (p. 61) header file.

Compounds

- class **ReactorTrigger**

  ReactorTrigger performs a simple trigger simulation for the ReactorFsim simulation.

4.25.1 Detailed Description

ReactorTrigger (p. 61) header file.

Author: Matthew Worcester

Definition in file **ReactorTrigger.hh**.

4.26 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** (**ReactorConstants** &, 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)

4.26.1 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**.

4.26.2 Function Documentation

4.26.2.1 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 ReactorConstants k;
  static double pi = k.pi;
  static double m = k.Melectron;
  static double alpha = 1/k.alphainv;
  static double Units = k.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 density = k.density;
  static double Na = k.Navogadro;
  static double A[2] = {k.A0,k.A1};
  static double Z[2] = {k.Z0,k.Z1};
  static double f[2];
  if(material == "scintillator"){
    f[0] = k.f0;
    f[1] = k.f1;
  }
  else if(material == "oil"){
    f[0] = k.f4;
    f[1] = k.f5;
  }
  else{
    cout << "GammaComptonRange: " << material
         << " is an unknown material!  Assuming scintillator." << endl;
    f[0] = k.f0;
    f[1] = k.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);
}
```

Index