# ReactorFsim Reference Manual
0_0_2

Generated by Doxygen 1.2.14

Thu Jul 29 14:27:03 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 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.*
- double **GetCfNeutronEnergy** (int &n) const
  *Returns the energy of the neutron produced by each Cf decay.*
- 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 delayed neutrons from the Cf decay as a function of neutron energy.*

### 3.1.1 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. 16). The neutrons are generated by the Cf252NeutronSpectrum function, which the probability density of the produced neutrons as a function of neutron energy. One neutron is produced per Cf source. The photons are generated by a gaussian peak centered around XX MeV.

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 25 of file MyCfSource.hh.

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 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. 16) 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 17 of file MyCfSource.cpp.

References 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];

    if(first){
      float xlo = 0.;
      float xhi = 50.;
      funlxp_(Cf252NeutronSpectrum,fspace,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;
  }

  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 energy
    int len = 1;
    funlux_(fspace,neutronE[n],len);
    //cout << "   Cf Neutron Energy = " << neutronE[n] << endl;

  } // done looping over sources

  // zero out remaining energy and position arrays
  for(int n=Nsources;n<k.Ndim;n++){
    neutronE[n] = 0.;
    R[n] = 0.;
    zR[n] = 0.;
    phiR[n] = 0.;
  }

} // done with Cf 252

if(Isotope == 255){

  Nsources = 0;
```

```
    for(int n=Nsources;n<k.Ndim;n++){

      neutronE[n] = 0.;
      R[n] = 0.;
      zR[n] = 0.;
      phiR[n] = 0.;
    }
  } // done with Cf 255

}
```

### 3.1.3 Member Function Documentation

#### 3.1.3.1 double MyCfSource::GetCfNeutronEnergy (int & *n*) const [inline]

Returns the energy of the neutron 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.

Definition at line 54 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

```
54 {return neutronE[n];}
```

#### 3.1.3.2 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 63 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

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

#### 3.1.3.3 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 67 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

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

#### 3.1.3.4 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 59 of file MyCfSource.hh.

Referenced by ReactorEvent::ReactorEvent.

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

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

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

## 3.2 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.*

### 3.2.1 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.

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 MyTlSource::MyTlSource (int *newIsotope* = k.Isotope)

MyTlSource constructor.

Called with the desired Thallium isotope: 208 or 210. Default is the isotope given in **Reactor-Constants** (p. 16), 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[100];
int Ngamma[100];
float gammaE[100][6];

// 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;
          */
        }
        done = true;
        continue;

        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;
     }
   }

   // 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];

       passed = true;
     }
   }

 } // 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;
            */
          }
          done = true;
          continue;

          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

}
```

### 3.2.3 Member Function Documentation

#### 3.2.3.1 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**

## 3.3 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** = 200
- const int **Isotope** = 252
- const int **Ndim** = 100
- 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 **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 **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

### 3.3.1 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**

## 3.4 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 **GetPHpositron** () const
- double **GetPHneutron** () const
- double **GetPHposReco** () const
- double **GetPHneuReco** () const
- double **GetMeanNpePos** () const
- double **GetMeanAttenPos** () const
- double **GetMeanSfactorPos** () const
- double **GetMeanNpeNeu** () const
- double **GetMeanAttenNeu** () const
- double **GetMeanSfactorNeu** () const
- double **GetHitFracPos** ()
- double **GetHitFracNeu** ()
- 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** (char parent)
- void **GetGamma** (char parent, double *xyz)
- void **GetGamma** (char 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 &npos, double *PHpos, double *Tpos, int &nneu, double *PHneu, double *Tneu)
- 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)

### 3.4.1 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.

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 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 23 of file ReactorDetector.cpp.

```
23                              {
24   //
25   // initialize PMT arrays
26   //
27   Xpmt = new double[0];
28   Ypmt = new double[0];
29   Zpmt = new double[0];
30   PMTdata = new PMT[0];
31   PMTpositron = new PMT[0];
32   PMTneutron = new PMT[0];
33   //
34   // Set parameters
35   //
```

```
SetNominalParameters();
SetPMT();
//
// initialize gamma arrays
//
Ngamma = 0;
Rgamma = new double[Ngamma];
Egamma = new double[Ngamma];
Tgamma = new double[Ngamma];
Ogamma = new char[Ngamma];

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**

## 3.5 ReactorEvent Class Reference

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

`#include <ReactorEvent.hh>`

### Public Methods

- **ReactorEvent** (double newNevents=0, double newRmaxgen=k.Rmaxgen, double new-Emin=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 **GetNevents** () const

  *Returns the number of events.*

- int **GetNumNubar** () const

  *Returns the number of anti-neutrinos.*

- int **GetNumPositron** () const

  *Returns the number of positrons.*

- int **GetNumNeutron** () const

  *Returns the number of neutrons.*

- 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 **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 **GetNubarTime** (int &n) const
- double **GetPositronTime** (int &n) const
- double **GetNeutronTime** (int &n) const
- double **GetNubarVertexR** (int &n) const
- double **GetNubarVertexCosTheta** (int &n) const
- double **GetNubarVertexPhi** (int &n) const
- double *const **GetNubarVertexXYZ** ()
- 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 **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

### 3.5.1 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.

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 ReactorEvent::ReactorEvent (double *newNevents* = 0, 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 18 of file ReactorEvent.cpp.

References ReactorConstants::Delta, MyCfSource::GetCfNeutronEnergy, MyCfSource::GetCfVertexCosTheta, MyCfSource::GetCfVertexPhi, MyCfSource::GetCfVertexR, MyCfSource::GetNumCfSources, ReactorConstants::Melectron, ReactorConstants::Mproton, ReactorConstants::Ndim, and ReactorConstants::pi.

```
22                                     {
23   Nevents = newNevents;
24   Rmaxgen = newRmaxgen;
25   Emin = newEmin;
26   Emax = newEmax;
27   XCorder = newXCorder;
28
29   int nmax = k.Ndim;
30   int nnu  = 0;
31   int npos = 0;
32   int nneu = 0;
33
34   double* pnubar = new double[nmax];
35   double* ppositron = new double[nmax];
36   double* pneutron = new double[nmax];
37
38   double* pnu  = pnubar;
39   double* ppos = ppositron;
40   double* pneu = pneutron;
41
```

```
double* rnubar = new double[nmax];
double* rpositron = new double[nmax];
double* rgenpos = new double[nmax];
double* rneutron = new double[nmax];
double* rgenneu = new double[nmax];

double* rnu  = rnubar;
double* rpos = rpositron;
double* rgp = rgenpos;
double* rneu = rneutron;
double* rgn = rgenneu;

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];
string dummy;
bool done = false;
bool set = false;

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;
  }

  // 'T'YPE or 't'ype specifies the event type
  if(fin.peek() == 't' || fin.peek() == 'T'){

    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 == "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;
    }

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

      fin.ignore(1);
      continue;
    }
  }

  // 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"){

    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++){
      float neutronKE = CfSource.GetCfNeutronEnergy(n);
      //cout << "   Cf252 source " << n << " energy = "
      //           << neutronKE << endl;

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

      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

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

      *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 neutrons
    nneu += Nsources;

  } // done with cf source neutron production

  //
  // transfer temp data to storage
  //
  Nnubar    = nnu;
  Npositron = npos;
  Nneutron  = nneu;

  Pnubar    = new double[Pdim*nnu];
  Ppositron = new double[Pdim*npos];
  Pneutron  = new double[Pdim*nneu];

  Rnubar    = new double[Rdim*nnu];
  Rpositron = new double[Rdim*npos];
  Rgenpos   = new double[Rdim*npos];
  Rneutron  = new double[Rdim*nneu];
  Rgenneu   = new double[Rdim*nneu];

  double* ptr;

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

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

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

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

  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;

  /*
  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 << 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;
  }
  */

}
```

### 3.5.3 Member Function Documentation

#### 3.5.3.1 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 83 of file ReactorEvent.hh.

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

#### 3.5.3.2 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 108 of file ReactorEvent.hh.

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

#### 3.5.3.3 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 115 of file ReactorEvent.hh.

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

#### 3.5.3.4 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 88 of file ReactorEvent.hh.

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

#### 3.5.3.5 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 94 of file ReactorEvent.hh.

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

#### 3.5.3.6 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 101 of file ReactorEvent.hh.

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

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

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

# 3.6 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)

### 3.6.1 Detailed Description

ReactorNtuple creates and fills the ntuple for the ReactorFsim simulation.

Definition at line 12 of file ReactorNtuple.hh.

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 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 53 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 = 100;
  char ntTags[ntVarsMax*wordsize];
  //
  int ntVars = 0;
  int id = 1;
  char Name[100];
  //
  // book the general event veriables, nt=1001
```

```
//
strcpy(ntTags+ntVars++*wordsize,"nevents  ");
strcpy(ntTags+ntVars++*wordsize,"wtc      ");
strcpy(ntTags+ntVars++*wordsize,"wtf      ");
//
IdGen = 1000*ntId+id++;
sprintf(Name,"%s%s",ntName,", general");
NvarGen = ntVars;
//
hbookn_(IdGen,Name,NvarGen,dirName,1000,ntTags,
        strlen(Name),strlen(dirName),wordsize,NvarGen);
//
// reset
ntVars = 0;
//
// book the nubar event variables, nt=1002
//
// up to 5 nubar per event
strcpy(ntTags+ntVars++*wordsize,"nnu      ");

strcpy(ntTags+ntVars++*wordsize,"enu1     ");
strcpy(ntTags+ntVars++*wordsize,"nuproc1  ");
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    ");

strcpy(ntTags+ntVars++*wordsize,"enu2     ");
strcpy(ntTags+ntVars++*wordsize,"nuproc2  ");
strcpy(ntTags+ntVars++*wordsize,"vnu2r    ");
strcpy(ntTags+ntVars++*wordsize,"vnu2cos  ");
strcpy(ntTags+ntVars++*wordsize,"vnu2phi  ");
strcpy(ntTags+ntVars++*wordsize,"vnu2x    ");
strcpy(ntTags+ntVars++*wordsize,"vnu2y    ");
strcpy(ntTags+ntVars++*wordsize,"vnu2z    ");

strcpy(ntTags+ntVars++*wordsize,"enu3     ");
strcpy(ntTags+ntVars++*wordsize,"nuproc3  ");
strcpy(ntTags+ntVars++*wordsize,"vnu3r    ");
strcpy(ntTags+ntVars++*wordsize,"vnu3cos  ");
strcpy(ntTags+ntVars++*wordsize,"vnu3phi  ");
strcpy(ntTags+ntVars++*wordsize,"vnu3x    ");
strcpy(ntTags+ntVars++*wordsize,"vnu3y    ");
strcpy(ntTags+ntVars++*wordsize,"vnu3z    ");

strcpy(ntTags+ntVars++*wordsize,"enu4     ");
strcpy(ntTags+ntVars++*wordsize,"nuproc4  ");
strcpy(ntTags+ntVars++*wordsize,"vnu4r    ");
strcpy(ntTags+ntVars++*wordsize,"vnu4cos  ");
strcpy(ntTags+ntVars++*wordsize,"vnu4phi  ");
strcpy(ntTags+ntVars++*wordsize,"vnu4x    ");
strcpy(ntTags+ntVars++*wordsize,"vnu4y    ");
strcpy(ntTags+ntVars++*wordsize,"vnu4z    ");

strcpy(ntTags+ntVars++*wordsize,"enu5     ");
strcpy(ntTags+ntVars++*wordsize,"nuproc5  ");
strcpy(ntTags+ntVars++*wordsize,"vnu5r    ");
strcpy(ntTags+ntVars++*wordsize,"vnu5cos  ");
strcpy(ntTags+ntVars++*wordsize,"vnu5phi  ");
strcpy(ntTags+ntVars++*wordsize,"vnu5x    ");
strcpy(ntTags+ntVars++*wordsize,"vnu5y    ");
strcpy(ntTags+ntVars++*wordsize,"vnu5z    ");
//
```

```
IdNu = 1000*ntId+id++;
sprintf(Name,"%s%s",ntName,", nubar");
NvarNu = ntVars;
//
hbookn_(IdNu,Name,NvarNu,dirName,1000,ntTags,
        strlen(Name),strlen(dirName),wordsize,NvarNu);
//
// reset
ntVars = 0;
//
// book the positron event variables, nt=1003
//
// up to 5 positron per event
strcpy(ntTags+ntVars++*wordsize,"npos     ");

strcpy(ntTags+ntVars++*wordsize,"KEpos1   ");
strcpy(ntTags+ntVars++*wordsize,"pos1cos  ");
strcpy(ntTags+ntVars++*wordsize,"posproc1 ");
strcpy(ntTags+ntVars++*wordsize,"gpos1r   ");
strcpy(ntTags+ntVars++*wordsize,"gpos1cos ");
strcpy(ntTags+ntVars++*wordsize,"gpos1phi ");
strcpy(ntTags+ntVars++*wordsize,"gpos1x   ");
strcpy(ntTags+ntVars++*wordsize,"gpos1y   ");
strcpy(ntTags+ntVars++*wordsize,"gpos1z   ");
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    ");

strcpy(ntTags+ntVars++*wordsize,"KEpos2   ");
strcpy(ntTags+ntVars++*wordsize,"pos2cos  ");
strcpy(ntTags+ntVars++*wordsize,"posproc2 ");
strcpy(ntTags+ntVars++*wordsize,"gpos2r   ");
strcpy(ntTags+ntVars++*wordsize,"gpos2cos ");
strcpy(ntTags+ntVars++*wordsize,"gpos2phi ");
strcpy(ntTags+ntVars++*wordsize,"gpos2x   ");
strcpy(ntTags+ntVars++*wordsize,"gpos2y   ");
strcpy(ntTags+ntVars++*wordsize,"gpos2z   ");
strcpy(ntTags+ntVars++*wordsize,"vpos2r   ");
strcpy(ntTags+ntVars++*wordsize,"vpos2cos ");
strcpy(ntTags+ntVars++*wordsize,"vpos2phi ");
strcpy(ntTags+ntVars++*wordsize,"vpos2x   ");
strcpy(ntTags+ntVars++*wordsize,"vpos2y   ");
strcpy(ntTags+ntVars++*wordsize,"vpos2z   ");
strcpy(ntTags+ntVars++*wordsize,"tpos2    ");

strcpy(ntTags+ntVars++*wordsize,"KEpos3   ");
strcpy(ntTags+ntVars++*wordsize,"pos3cos  ");
strcpy(ntTags+ntVars++*wordsize,"posproc3 ");
strcpy(ntTags+ntVars++*wordsize,"gpos3r   ");
strcpy(ntTags+ntVars++*wordsize,"gpos3cos ");
strcpy(ntTags+ntVars++*wordsize,"gpos3phi ");
strcpy(ntTags+ntVars++*wordsize,"gpos3x   ");
strcpy(ntTags+ntVars++*wordsize,"gpos3y   ");
strcpy(ntTags+ntVars++*wordsize,"gpos3z   ");
strcpy(ntTags+ntVars++*wordsize,"vpos3r   ");
strcpy(ntTags+ntVars++*wordsize,"vpos3cos ");
strcpy(ntTags+ntVars++*wordsize,"vpos3phi ");
strcpy(ntTags+ntVars++*wordsize,"vpos3x   ");
strcpy(ntTags+ntVars++*wordsize,"vpos3y   ");
strcpy(ntTags+ntVars++*wordsize,"vpos3z   ");
strcpy(ntTags+ntVars++*wordsize,"tpos3    ");
```

```

strcpy(ntTags+ntVars++*wordsize,"KEpos4   ");
strcpy(ntTags+ntVars++*wordsize,"pos4cos  ");
strcpy(ntTags+ntVars++*wordsize,"posproc4 ");
strcpy(ntTags+ntVars++*wordsize,"gpos4r   ");
strcpy(ntTags+ntVars++*wordsize,"gpos4cos ");
strcpy(ntTags+ntVars++*wordsize,"gpos4phi ");
strcpy(ntTags+ntVars++*wordsize,"gpos4x   ");
strcpy(ntTags+ntVars++*wordsize,"gpos4y   ");
strcpy(ntTags+ntVars++*wordsize,"gpos4z   ");
strcpy(ntTags+ntVars++*wordsize,"vpos4r   ");
strcpy(ntTags+ntVars++*wordsize,"vpos4cos ");
strcpy(ntTags+ntVars++*wordsize,"vpos4phi ");
strcpy(ntTags+ntVars++*wordsize,"vpos4x   ");
strcpy(ntTags+ntVars++*wordsize,"vpos4y   ");
strcpy(ntTags+ntVars++*wordsize,"vpos4z   ");
strcpy(ntTags+ntVars++*wordsize,"tpos4    ");

strcpy(ntTags+ntVars++*wordsize,"KEpos5   ");
strcpy(ntTags+ntVars++*wordsize,"pos5cos  ");
strcpy(ntTags+ntVars++*wordsize,"posproc5 ");
strcpy(ntTags+ntVars++*wordsize,"gpos5r   ");
strcpy(ntTags+ntVars++*wordsize,"gpos5cos ");
strcpy(ntTags+ntVars++*wordsize,"gpos5phi ");
strcpy(ntTags+ntVars++*wordsize,"gpos5x   ");
strcpy(ntTags+ntVars++*wordsize,"gpos5y   ");
strcpy(ntTags+ntVars++*wordsize,"gpos5z   ");
strcpy(ntTags+ntVars++*wordsize,"vpos5r   ");
strcpy(ntTags+ntVars++*wordsize,"vpos5cos ");
strcpy(ntTags+ntVars++*wordsize,"vpos5phi ");
strcpy(ntTags+ntVars++*wordsize,"vpos5x   ");
strcpy(ntTags+ntVars++*wordsize,"vpos5y   ");
strcpy(ntTags+ntVars++*wordsize,"vpos5z   ");
strcpy(ntTags+ntVars++*wordsize,"tpos5    ");
//
IdPos = 1000*ntId+id++;
sprintf(Name,"%s%s",ntName,", positron");
NvarPos = ntVars;
//
hbookn_(IdPos,Name,NvarPos,dirName,1000,ntTags,
        strlen(Name),strlen(dirName),wordsize,NvarPos);
//
// reset
ntVars = 0;
//
// book the neutron event variables, nt=1004
//
// up to 5 neutron per event
strcpy(ntTags+ntVars++*wordsize,"nneu     ");

strcpy(ntTags+ntVars++*wordsize,"KEneu1   ");
strcpy(ntTags+ntVars++*wordsize,"neu1cos  ");
strcpy(ntTags+ntVars++*wordsize,"neuproc1 ");
strcpy(ntTags+ntVars++*wordsize,"gneu1r   ");
strcpy(ntTags+ntVars++*wordsize,"gneu1cos ");
strcpy(ntTags+ntVars++*wordsize,"gneu1phi ");
strcpy(ntTags+ntVars++*wordsize,"gneu1x   ");
strcpy(ntTags+ntVars++*wordsize,"gneu1y   ");
strcpy(ntTags+ntVars++*wordsize,"gneu1z   ");
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,"gneu2r   ");
strcpy(ntTags+ntVars++*wordsize,"gneu2cos ");
strcpy(ntTags+ntVars++*wordsize,"gneu2phi ");
strcpy(ntTags+ntVars++*wordsize,"gneu2x   ");
strcpy(ntTags+ntVars++*wordsize,"gneu2y   ");
strcpy(ntTags+ntVars++*wordsize,"gneu2z   ");
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,"gneu3r   ");
strcpy(ntTags+ntVars++*wordsize,"gneu3cos ");
strcpy(ntTags+ntVars++*wordsize,"gneu3phi ");
strcpy(ntTags+ntVars++*wordsize,"gneu3x   ");
strcpy(ntTags+ntVars++*wordsize,"gneu3y   ");
strcpy(ntTags+ntVars++*wordsize,"gneu3z   ");
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  ");

strcpy(ntTags+ntVars++*wordsize,"KEneu4   ");
strcpy(ntTags+ntVars++*wordsize,"neu4cos  ");
strcpy(ntTags+ntVars++*wordsize,"neuproc4 ");
strcpy(ntTags+ntVars++*wordsize,"gneu4r   ");
strcpy(ntTags+ntVars++*wordsize,"gneu4cos ");
strcpy(ntTags+ntVars++*wordsize,"gneu4phi ");
strcpy(ntTags+ntVars++*wordsize,"gneu4x   ");
strcpy(ntTags+ntVars++*wordsize,"gneu4y   ");
strcpy(ntTags+ntVars++*wordsize,"gneu4z   ");
strcpy(ntTags+ntVars++*wordsize,"vneu4r   ");
strcpy(ntTags+ntVars++*wordsize,"vneu4cos ");
strcpy(ntTags+ntVars++*wordsize,"vneu4phi ");
strcpy(ntTags+ntVars++*wordsize,"vneu4x   ");
strcpy(ntTags+ntVars++*wordsize,"vneu4y   ");
strcpy(ntTags+ntVars++*wordsize,"vneu4z   ");
strcpy(ntTags+ntVars++*wordsize,"tneu4    ");
strcpy(ntTags+ntVars++*wordsize,"nsteps4  ");
strcpy(ntTags+ntVars++*wordsize,"absorb4  ");

strcpy(ntTags+ntVars++*wordsize,"KEneu5   ");
strcpy(ntTags+ntVars++*wordsize,"neu5cos  ");
strcpy(ntTags+ntVars++*wordsize,"neuproc5 ");
strcpy(ntTags+ntVars++*wordsize,"gneu5r   ");
```

```
strcpy(ntTags+ntVars++*wordsize,"gneu5cos ");
strcpy(ntTags+ntVars++*wordsize,"gneu5phi ");
strcpy(ntTags+ntVars++*wordsize,"gneu5x   ");
strcpy(ntTags+ntVars++*wordsize,"gneu5y   ");
strcpy(ntTags+ntVars++*wordsize,"gneu5z   ");
strcpy(ntTags+ntVars++*wordsize,"vneu5r   ");
strcpy(ntTags+ntVars++*wordsize,"vneu5cos ");
strcpy(ntTags+ntVars++*wordsize,"vneu5phi ");
strcpy(ntTags+ntVars++*wordsize,"vneu5x   ");
strcpy(ntTags+ntVars++*wordsize,"vneu5y   ");
strcpy(ntTags+ntVars++*wordsize,"vneu5z   ");
strcpy(ntTags+ntVars++*wordsize,"tneu5    ");
strcpy(ntTags+ntVars++*wordsize,"nsteps5  ");
strcpy(ntTags+ntVars++*wordsize,"absorb5  ");
//
IdNeu = 1000*ntId+id++;
sprintf(Name,"%s%s",ntName,", neutron");
NvarNeu = ntVars;
//
hbookn_(IdNeu,Name,NvarNeu,dirName,1000,ntTags,
        strlen(Name),strlen(dirName),wordsize,NvarNeu);
//
// reset
ntVars = 0;
//
// book the pmt detector variables, nt=1005
//
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,"npepos   ");
strcpy(ntTags+ntVars++*wordsize,"attenpos ");
strcpy(ntTags+ntVars++*wordsize,"sfactpos ");
strcpy(ntTags+ntVars++*wordsize,"npeneu   ");
strcpy(ntTags+ntVars++*wordsize,"attenneu ");
strcpy(ntTags+ntVars++*wordsize,"sfactneu ");
strcpy(ntTags+ntVars++*wordsize,"ngampos  ");
strcpy(ntTags+ntVars++*wordsize,"ngamneu  ");
strcpy(ntTags+ntVars++*wordsize,"ngamtha  ");
strcpy(ntTags+ntVars++*wordsize,"fpmtpos  ");
strcpy(ntTags+ntVars++*wordsize,"fpmtneu  ");

strcpy(ntTags+ntVars++*wordsize,"dposx    ");
strcpy(ntTags+ntVars++*wordsize,"dposy    ");
strcpy(ntTags+ntVars++*wordsize,"dposz    ");
strcpy(ntTags+ntVars++*wordsize,"dneux    ");
strcpy(ntTags+ntVars++*wordsize,"dneuy    ");
strcpy(ntTags+ntVars++*wordsize,"dneuz    ");
//
IdDet = 1000*ntId+id++;
sprintf(Name,"%s%s",ntName,", detector");
NvarDet = ntVars;
//
hbookn_(IdDet,Name,NvarDet,dirName,1000,ntTags,
        strlen(Name),strlen(dirName),wordsize,NvarDet);
//
// define some histograms
// Last argument is need for c++ ==> f strings
//
Offset = 10000*(ntId-1);
```

```
/*
hbook1_(Offset+101,"phpos",200,0,0.1,0,strlen("phpos"));
hbook1_(Offset+102,"phneu",200,0,0.1,0,strlen("phneu"));
//
hbook1_(Offset+201,"tpos",250,0,25,0,strlen("tpos"));
hbook1_(Offset+202,"tneu",250,0,500000,0,strlen("tneu"));
//
hbook1_(Offset+301,"ngpos",10,0,10,0,strlen("ngpos"));
hbook1_(Offset+302,"ngneu",20,0,20,0,strlen("ngneu"));
//
hbook1_(Offset+401,"egpos",200,0,10,0,strlen("egpos"));
hbook1_(Offset+402,"egneu",200,0,20,0,strlen("egneu"));
//
hbook1_(Offset+501,"rgp-rpos",200,0,100,0,strlen("rgp-rpos"));
hbook1_(Offset+502,"rgn-rneu",200,0,100,0,strlen("rgn-rneu"));
//
hbook1_(Offset+601,"rgp-rnu",200,0,100,0,strlen("rgp-rnu"));
hbook1_(Offset+602,"rgn-rnu",200,0,100,0,strlen("rgn-rnu"));
//
hbook1_(Offset+701,"tgp-tpos",250,0,5,0,strlen("tgp-tpos"));
hbook1_(Offset+702,"tgn-tneu",250,0,5,0,strlen("tgn-tneu"));
//
hbook1_(Offset+801,"tgp-tnu",250,0,25,0,strlen("tgp-tnu"));
hbook1_(Offset+802,"tgn-tnu",250,0,1000000,0,strlen("tgn-tnu"));
*/
}
```

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

- **ReactorNtuple.hh**
- **ReactorNtuple.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 MyCfSource.cpp File Reference

**MyCfSource** (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 "MyCfSource.hh"
```

### 4.4.1 Detailed Description

**MyCfSource** (p. 5) C++ file.

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

Author: Matthew Worcester

Definition in file **MyCfSource.cpp**.

## 4.5 MyCfSource.hh File Reference

**MyCfSource** (p. 5) 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.*

### 4.5.1 Detailed Description

**MyCfSource** (p. 5) header file.

Author: Matthew Worcester

Definition in file **MyCfSource.hh**.

## 4.6 MyTlSource.cpp File Reference

**MyTlSource** (p. 11) 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"`

### 4.6.1 Detailed Description

**MyTlSource** (p. 11) C++ file.

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

Author: Matthew Worcester

Definition in file **MyTlSource.cpp**.

## 4.7 MyTlSource.hh File Reference

**MyTlSource** (p. 11) header file.

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

### Compounds

- class **MyTlSource**

  *MyTlSource generates photons from radioactive Thallium decays.*

### 4.7.1 Detailed Description

**MyTlSource** (p. 11) header file.

Author: Matthew Worcester

Definition in file **MyTlSource.hh**.

## 4.8 ReactorConstants.hh File Reference

**ReactorConstants** (p. 16) header file.

### Compounds

- class **ReactorConstants**

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

### 4.8.1 Detailed Description

**ReactorConstants** (p. 16) header file.

Author: Tim Bolton

Definition in file **ReactorConstants.hh**.

## 4.9 ReactorDetector.cpp File Reference

**ReactorDetector** (p. 18) 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"`

### 4.9.1 Detailed Description

**ReactorDetector** (p. 18) 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**.

## 4.10 ReactorDetector.hh File Reference

**ReactorDetector** (p. 18) 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.*

### 4.10.1 Detailed Description

**ReactorDetector** (p. 18) header file.

Author: Tim Bolton

Definition in file **ReactorDetector.hh**.

## 4.11 ReactorEvent.cpp File Reference

**ReactorEvent** (p. 21) 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"
```

### 4.11.1 Detailed Description

**ReactorEvent** (p. 21) C++ file.

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

Author: Tim Bolton, Matthew Worcester

Definition in file **ReactorEvent.cpp**.

## 4.12 ReactorEvent.hh File Reference

**ReactorEvent** (p. 21) 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.*

### 4.12.1 Detailed Description

**ReactorEvent** (p. 21) header file.

Author: Tim Bolton, Matthew Worcester

Definition in file **ReactorEvent.hh**.

## 4.13 ReactorFortran.hh File Reference

Defines the Fortran prototypes for calling from C++.

### 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 &)

### 4.13.1 Detailed Description

Defines the Fortran prototypes for calling from C++.

Necessary for the Hbook interface and Cernlib Fortran routines.

Definition in file **ReactorFortran.hh**.

## 4.14 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 <iostream.h>
#include <math.h>
#include <string.h>
```

### Functions

- int **main** ()
  *Top level function of ReactorFsim.*

### 4.14.1 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**.

### 4.14.2 Function Documentation

#### 4.14.2.1 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 26 of file ReactorFsim.cpp.

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

```
26          {
27
28  //  ReactorNeutrons Neutrons;
29  //
```

```
//  Set number of events, paw file name, and ntuple name
//
int nevents = 100000;
double rmaxgen=225;
int fast=0;
char* fileName="/local/data14/reactorfsim/cf252_pmtradon_100k.ntuple";
char* ntName="event data";
//
// 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(175);
DetectorA.SetParam_R1(200);
DetectorA.SetParam_R2(275);
ReactorNtuple ModelA(1,"model A",dirName);
//
// 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);
  //
}
//
// 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));
}
```

## 4.15 ReactorNtuple.cpp File Reference

**ReactorNtuple** (p. 31) 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"
```

### 4.15.1 Detailed Description

**ReactorNtuple** (p. 31) 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**.

## 4.16 ReactorNtuple.hh File Reference

**ReactorNtuple** (p. 31) header file.

### Compounds

- class **ReactorNtuple**

  *ReactorNtuple creates and fills the ntuple for the ReactorFsim simulation.*

### 4.16.1 Detailed Description

**ReactorNtuple** (p. 31) header file.

Author: Tim Bolton

Definition in file **ReactorNtuple.hh**.

# 4.17 ReactorXC.hh File Reference

ReactorXC defines several useful functions for ReactorFsim.

`#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 **MyGdDecayModel** (double, int &, double *)
- void **NeutronPropagator** (double R0, double R2, double &ke, double &a, double &pass, double *pn, double *rn, double &tn)

## 4.17.1 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**.

## 4.17.2 Function Documentation

### 4.17.2.1 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   //
19   static ReactorConstants k;
20   static double pi = k.pi;
21   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