“Survival”

Generated by Doxygen 1.8.11

Contents

Chapter 1

Main Page

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2015

1.1 Usage

1.1.1 Unix and Unix-like systems.

To execute the program the user has to call from the command line:

```
1 $ source setenv.sh
2 $ ./survival -SIMULATION_OPTION CHOSEN_VALUES ...
```

Then the user has the possibility to set a number of physical (and not only physical) parameters by using the syntax: `-PARAMETER_NAME PARAMETER_VALUE`

Here is the complete list of parameters and their meaning:

- `-projectName` It's a string representing the prefix to give at any file and directories that will be created in the simulation. The default value is "NewProject".
- `-output` The user has the possibility to choose between three kinds of output (and all possible combination between them):
    1. "LQ_pars" Then a file will be created, named "PROJECTNAME_LQparameters_MKM.csv", containing the information about the parameters chosen for the simulation and the values of the simulated LQ α and β parameters (a new line for each energy evaluated).
    2. "meanValues" Then a file will be created, named "PROJECTNAME_survival_MKM.csv", containing the information about the parameters chosen for the simulation and the values of doses delivered and survival observed (a new line for each energy or dose evaluated).

3. "cellValues" This kind of output is supported only by the MonteCarlo calculusType. It is a way to store the values of dose and survival obtained for each single cell irradiated during the monte carlo simulation. Then a directory will be created, named "PROJECTNAME_survival_data". In the directory the user will find a description file named "000_MonteCarlo_parameters.csv", listing the parameters used in the simulation, and a directory with the same name containing the corresponding data. In particular in the subdirectory some file will be created (a file for each level of dose imposed), each one containing two column with the dose delivered and the survival observed for each cell irradiated. When a new simulation is lauched with the same project name, the program will do a check over all the description files in the directory, if the parameters of the simulation are the same of another one already done, then it will enter the related subdirectory and append data there, if not a new description file (with progressive number) and corresponding subdirectory will be created.

   Warning

   Storing these informations could occupy a lot of memory in the computer, depending on the parameter set for the simulation. Use with caution!

   Note

   This parameter has to be specified. No default values are set.

- `-precision` Supported only by the MonteCarlo calculusType, it's a `double` identifying the precision to be reached in the calculation. Two possibilities are provided, as the user can indicate:
  1. A positive integer: it will be taken as the number of cell to irradiate for each level of nominal dose imposed.
  2. A double between 0 and 1: it indicates the relative error on the simulated survival to be reached before stopping the calculation.
- `-parallelismType` A positive integer indicating the level of parallelism to be used in the simulation. The user has the possibility to specify the number of threads to dedicate at the calculation, in particular:
  1. 1: Only 1 thread = parallelism disabled.
  2. N>1: The number of threads to be used.
  3. 0: Then the program will define a number of thread corresponding to the number of core of the computer executing the program. This is set as default value.
- `-model` It's a string representing the model to use in the simulation. Five models are supported:
  1. "LEMI" Is the first formulation of the Local Effect Model, the published reference is:
     M. Scholz and G. Kraft, "Track structure and the calculation of biological effects of heavy charged particles", *Advances in Space Research* **18**, 5-14 (1996)
  2. "LEMII" A reformulation of the Local Effect Model, as described in:
     T. Elsässer and M. Scholz, "Cluster effects within the local effect model", *Radiation Research* **167**, 319-329 (2007)
  3. "LEMIII" A third formulation of the Local Effect Model, well described in the paper:
     T. Elsässer, M. Krämer and M. Scholz, "Accuracy of the local effect model for the prediction of biologic effects of carbon ion beams \em in \em vitro and \em in \em vivo", *International Journal of Radiation Oncology-Biology-Physics **71**, 866-872* (2008)
  4. "MKM" The Microdosimetric Kinetic Model, in the formulation of Hawkins, with the approach of Kase who suggest to use the Kiefer-Chatterjee amorphous track model. Some published references:
     R.B. Hawkins, "A Statistical Theory of Cell Killing by Radiation of Varying Linear Energy Transfer", *Radiation Research* **140**, 366-374 (1994) – And subsequent references
     Y. Kase, T. Kanai, N. Matsufuji, Y. Furusawa, T. Elsasser, and M. Scholz, "Biophysical calculation of cell survival probabilities using amorphous track structure models for heavy-ion irradiation", *Physics in Medicine and Biology* **53**, 37-59 (2008)
  5. "tMKM_Manganaro2017" A monte carlo reformulation of the MKM, extended to the temporal dimension to evaluate the effect of the time structure of the irradiation on the LQ parameters, as described in:
     Manganaro, L., Russo, G., Cirio, R., Dalmasso, F., Giordanengo, S., Monaco, V., ... Attili, A. (2017). A Monte Carlo approach to the microdosimetric kinetic model to account for dose rate time structure effects in ion beam therapy with application in treatment planning simulations. *Medical Physics*, 44(4),

1577–1589.
The default value for this option is "MKM"

- `-calculusType` A string identifying the type of calculus to be done. Some possibilities are available:

  1. "rapidLEM_Scholz2006" It's an implementation of the method described in:
     M. Krämer and M. Scholz, "Rapid calculation of biological effects in ion radiotherapy", *Physics in medicine and biology* **51**, 1959-1970 (2006)
     It's compatible only with LEMI-LEMII-LEMIII models. See Survival::Calculus::rapidLEM_Scholz2006() for details.
  2. "rapidLEM_Russo2011" A new rapid method for LEM, proved to be more accurate, described in:
     G. Russo, "Develpment of a radiobiological database for carbon ion Treatment Planning Systems - Modelling and simulating the irradiation process", *PhD Thesis*, Università degli studi di Torino (2011)
     Also this method is compatible only with LEMI-LEMII-LEMIII models. See Survival::Calculus::rapidLEM_Russo2011() for details.
  3. "rapidMKM_Kase2008" A fast implementation of the MKM calculation as described in:
     Kase, Y., Kanai, T., Matsufuji, N., Furusawa, Y., Elsässer, T., & Scholz, M. (2008). Biophysical calculation of cell survival probabilities using amorphous track structure models for heavy-ion irradiation. *Physics in Medicine and Biology*, 53(1), 37–59 It's compatible only with the MKM model and it's the default value for this option. See Survival::Calculus::rapidMKM_Kase2008() for details.
  4. "rapidMKM_Attili2013" A fast original implementation of the MKM model, combining the methods described in:
     Hawkins_2003 Hawkins, R. B. (2003). A microdosimetric-kinetic model for the effect of non-Poisson distribution of lethal lesions on the variation of RBE with LET. *Radiation Research*, 160(1), 61–69, and Kase, Y., Kanai, T., Matsufuji, N., Furusawa, Y., Elsässer, T., & Scholz, M. (2008). Biophysical calculation of cell survival probabilities using amorphous track structure models for heavy-ion irradiation. *Physics in Medicine and Biology*, 53(1), 37–59. See Survival::Calculus::rapidMKM_Attili2013() for details.
  5. "MonteCarlo" Compatible with all models implemented, performs a monte carlo simulation of the irradiation process to get the LQ parameters.

- `-cellType` A string identifying the name of the cell lline used in the calculation. The default value is "Cell1"

  Note

  The cell line in reality is completely determined by the model parameters chosen. This is only a tag to indicate the cell but it isn't used in the simulation.

- `-MKM_alpha0` A double representing IDEALLY the linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}. The simulated α parameter will tend to this value for low LET. This option is compatible only with the MKM (and tMKM) model, it won't be use if a different model is chosen. The default value is 0.312 Gy^{-1}, a tipical value representing the Human Salivary Gland (HSG) cell line.

- `-MKM_beta0` A double representing IDEALLY the linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}. The simulated β parameter will tend to this value for low LET. This option is compatible only with the MKM (and tMKM) model, it won't be use if a different model is chosen. The default value is 0.073 Gy^{-2}, a tipical value representing the Human Salivary Gland (HSG) cell line.

- `-MKM_rNucleus` The radius of the cell expressed in μm. This option is compatible only with the MKM (and tMKM) model, it won't be use if a different model is chosen. The default value is 4.611 μm, a tipical value representing the Human Salivary Gland (HSG) cell line.

- `-MKM_rDomain` The radius of domains wich constitute the MKM nucleus expressed in μm. This option is compatible only with the MKM (and tMKM) model, it won't be use if a different model is chosen. The default value is 0.365 μm, a tipical value representing the Human Salivary Gland (HSG) cell line.

- `-MKM_timeConst` The time constant associated to the repair kinetics of the nucleus.

- `-LEM_alpha0` A double representing IDEALLY the linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}. The simulated α parameter will tend to this value for low LET. This option is compatible only with the LEM (I, II and III) model, it won't be use if a different model is chosen. The default value is 0.312 Gy^{-1}, a tipical value representing the Human Salivary Gland (HSG) cell line.

- `-LEM_beta0` A double representing IDEALLY the linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}. The simulated β parameter will tend to this value for low LET. This option is compatible only with the LEM (I, II and III) model, it won't be use if a different model is chosen. The default value is 0.073 Gy^{-2}, a tipical value representing the Human Salivary Gland (HSG) cell line.
- `-LEM_rNucleus` The radius of the cell expressed in μm. This option is compatible only with the LEM (I, II and III) model, it won't be use if a different model is chosen. The default value is 4.611 μm, a tipical value representing the Human Salivary Gland (HSG) cell line.
- `-LEM_Dt` The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy. This option is compatible only with the LEM (I, II and III) model, it won't be use if a different model is chosen. The default value is 30 Gy.
- `-ion` A string identifying the chemical symbol of the element, without mass number specifications. Ions from proton to neon are supported.
- `-energies` A sequence of kinetic energies of the primary ions to be evaluated, expressed in MeV. This and the "-lets" option are mutually exclusive, but one of the two has to be specified.
- `-lets` A sequence of LET of the primary ions to be evaluated, expressed in MeV. This and the "-energies" option are mutually exclusive, but one of the two has to be specified.
- `-doses` The sequence of doses to be delivered. The default is 1 to 6 Gy (step 1), in order to construct a survival curve.
- `-nFraction` Supported only by the tMKM model, it require an integer representing the number of fraction in which to divide each nominal dose to be delivered. The default is 1 (single fraction).
- `-timeSpacing` Supported only by the tMKM model, it indicate the time spacing between consecutive fractions, expressed in hours. The default is 0 (no time spacing).
- `-fracDeliveryTime` Supported only by the tMKM model, it indicate the fraction delivery time, expressed in hours. The default is 0 (istantaneous delivering).
- `-spectrum_file` Useful for mixed fields evaluation. This option provide the possibility to irradiate the cellular population with different ions and different energies in a mixed field, described in an external file (see the TEMPLATE for the spectrum_file). The user has to specify the name of the file containing the spectrum complete with its relative or absolute path.

  Warning

  The "mixed fields option" hasn't been deeply tested!

- `-trackMode` In the case of mixed fields this option require a string indicating the way in which to interpretate the spectrum specification. Two possibilities are provided:

  1. "histogram": then the spectrum will be considered as an histogram where each particle is a bin with its "weight" (see the TEMPLATE for the spectrum_file).
  2. "random": then the program will random extract with uniform probability (iteration by iteration) the particle to use.

Typing `-help` a hint will be display, suggesting how to use the program.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 Survival Namespace Reference

Classes

- class BetheBlochTable

  Class used to load precomputed values of LET for different ions of varying kinetic energy, evaluated by means of the Bethe-Bloch formula.
- class Calculus

  It implements some methods to simulate the irradiation process evaluating the cellular survival on a population of cells and extrapolating the radiobiological LQ parameters.
- class CellLine

  Represents the cell line to which the nucleus belongs and hosts the radiobiological and structural characteristics of the cell and the different parametrizations of the models.
- class Nucleus

  Linked to a specific CellLine object, this class defines the structure of the cellular nucleus to be irradiated and evaluates the dose absorbed during the interaction with a track.
- class Nucleus_Integral

  Implements a nucleus as a 2D circular object and provides methods to evaluate the number of lethal events observed.
- class Nucleus_Integral_t

  Implements a nucleus as a 2D circular object and provides methods to evaluate the number of lethal events observed taking into account the time structure of the irradiation.
- class Nucleus_MKM

  Inherited from the Nucleus pure virtual class, it implements the cellular nucleus as defined and used in the MKM model.
- class Nucleus_MonteCarlo

  Inherited from the Nucleus_Pixel class, it performs the integration of the dose deposited by the track via the Monte Carlo importance sampling method.
- class Nucleus_Pixel

  Inherited from the Nucleus pure virtual class, it implements the nucleus structure used in the LEM.
- class Nucleus_tMKM

  Inherited from the Nucleus pure virtual class, it implements the nucleus defined in the Monte Carlo temporal reformulation of the MKM model (MCt-MKM).
- class Particle

  This class defines the object "particle".
- class Particles

This is a container class, used to group together Particle objects. It implements the structure and methods to manage a vector of particles.

- class Pixel

  Implements the Pixel features to be used in the Nucleus_Pixel class.

- class Track

  Constructed on the base of an ion Particle object, this class represents the local dose distribution around that ion.

- class Track_Elsasser2007

  Inherited from the Track class, it implements the LEM II track model.

- class Track_Elsasser2008

  Inherited from the Track class, it implements the LEM III track model.

- class Track_KieferChatterjee

  Inherited from the Track pure virtual class, it implements the Kiefer-Chatterje amorphous track structure, used in the MKM model.

- class Track_Scholz2000

  Inherited from the Track class, it implements the LEM I track model.

- class Tracks

  This is a container class for Track objects; it implements the structure and methods to manage a vector of tracks.

Functions

- int _mkdir (const char *path)

  Portable wrapper for mkdir. Internally used by mkdir().

- double betheBloch_inv_Srim (string ionType, double let_imposed)
- double betheBloch_Srim (string ionType, double e_c_imposed)
- void fit_LQ (vector< double > dose, vector< double > survival, vector< double > survivalUncertainty, double &alpha, double &alphaUncertainty, double &beta, double &betaUncertainty, double &chiSquared, double &incompleteGammaQ)
- bool folder_exists (string foldername)
- int mkdir (const char *path)

  Recursive, portable wrapper for mkdir.

- void parse (int argc, char *argv[ ], string &cellType, string &model, string &trackType, string &parametrizationType, string &calculusType, double &precision, int &parallelismType, vector< double > &doses, vector< string > &parameter_name, double &MKM_alpha0, double &MKM_beta0, double &MKM_rNucleus, double &MKM_rDomain, double &tMKM_ac, double &LEM_alpha0, double &LEM_beta0, double &LEM_rNucleus, double &LEM_Dt, string &ionType, int &particleA, int &particleZ, string &trackMode, string &energyType, vector< double > &energies, int &nFraction, double &timeSpacing, double &fracDeliveryTime, bool &saveAlphaBeta, bool &saveMeans, bool &saveCell, string &projectName, bool &mono, string &spectrum_file)
- void Usage ()

  Display an hint to the user to correctly use the executable.

- double betheBloch_inv_Srim (std::string ionType, double let_imposed)

  Returns the kinetic energy associated to the value of LET imposed for a specified ion.

- double betheBloch_Srim (std::string ionType, double e_c_imposed)
- void fit_LQ (std::vector< double > dose, std::vector< double > survival, std::vector< double > survivalUncertainty, double &alpha, double &alphaUncertainty, double &beta, double &betaUncertainty, double &chiSquared, double &incompleteGammaQ)

  Perform a fit on a data set of doses and associated survival according to the linear quadratic formula.

- bool folder_exists (std::string foldername)

  Checks if a folder exists.

- void parse (int argc, char *argv[ ], std::string &cellType, std::string &model, std::string &trackType, std::string &parametrizationType, std::string &calculusType, double &precision, int &parallelismType, std::vector< double > &doses, std::vector< std::string > &parameter_name, double &MKM_alpha0, double &MKM_beta0, double &MKM_rNucleus, double &MKM_rDomain, double &tMKM_ac, double &LEM_alpha0, double &LEM_beta0, double &LEM_rNucleus, double &LEM_Dt, std::string &ionType, int &particleA, int &particleZ, std::string &trackMode, std::string &energyType, std::vector< double > &energies, int &nFraction, double &timeSpacing, double &fracDeliveryTime, bool &saveAlphaBeta, bool &saveMeans, bool &saveCell, std::string &projectName, bool &mono, std::string &spectrum_file)

  Parses the input arguments in the `main` *to set the calculation parameters.*

6.1.1 Function Documentation

6.1.1.1 int Survival::_mkdir (const char * *path*)

Portable wrapper for mkdir. Internally used by mkdir().

Parameters

path	The full path of the directory to create.

Returns

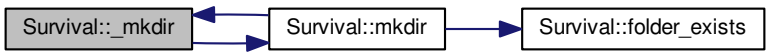
0 on success, otherwise -1.

See also

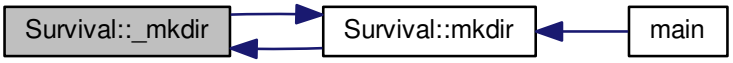
folder_exists() and mkdir()

Definition at line 45 of file usefulFunctions.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.1.2 double Survival::betheBloch_inv_Srim (std::string *ionType,* double *let_imposed*)

Returns the kinetic energy associated to the value of LET imposed for a specified ion.

The method opens and read the data file correspondent to the ion chosen until it finds in the table the nearest neighbors of the LET imposed, then it interpolates these values to get the correct kinetic energy required.

Parameters

ionType	A `string` identifying the ion (chemical symbol).
let_imposed	The LET (in MeV/um) correspondent to the required kinetic energy.

Returns

The kinetic energy in MeV/u associated to the value of LET imposed for a specified ion.

Warning

The execution of the program will be terminated if:

- The data file of the ion chosen doesn't exist.
- The environment hasn't been set correctly.
- The LET imposed is out of the available range for the ion specified.

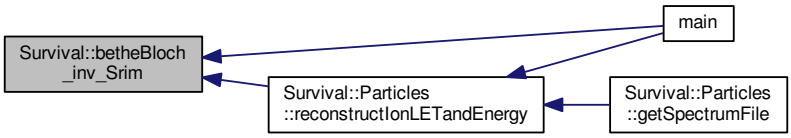
See also

BetheBlochTable and betheBloch_Srim()

6.1.1.3 double Survival::betheBloch_inv_Srim (string *ionType,* double *let_imposed*)

Definition at line 176 of file usefulFunctions.cpp.

Here is the caller graph for this function:



6.1.1.4 double Survival::betheBloch_Srim (std::string *ionType,* double *e_c_imposed*)

Returns the kinetic energy associated to the value of kinetic energy imposed for a specified ion.

The method makes use of a BetheBlochTable object, managed by an STL map (in pairing with a `string` identifying ion type chosen). The idea is to avoid reading the file every time the function BetheBlochTable::GetLet() is called. For this reason the map is defined `static`, and BetheBlochTable() (which reads the correspondent file) is called only when an ion is used for the first time.

Parameters

ionType	A `string` identifying the ion (chemical symbol).
e_c_imposed	The kinetic energy (in MeV/u) correspondent to the required kinetic energy.

Returns

The LET in MeV/um associated to the value of kinetic energy imposed for a specified ion.

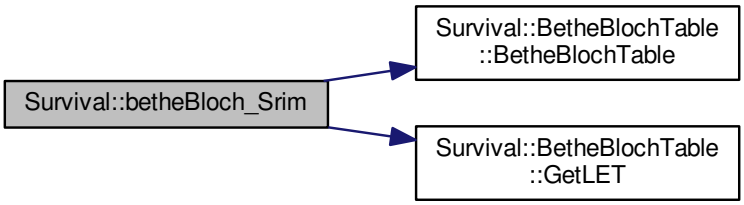
See also

betheBloch_inv_Srim()

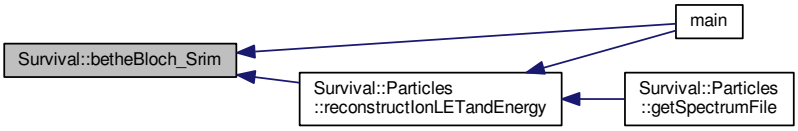
6.1.1.5 double Survival::betheBloch_Srim (string *ionType*, double *e_c_imposed*)

Definition at line 234 of file usefulFunctions.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.1.6 void Survival::fit_LQ (std::vector< double > *dose*, std::vector< double > *survival*, std::vector< double > *survivalUncertainty*, double & *alpha*, double & *alphaUncertainty*, double & *beta*, double & *betaUncertainty*, double & *chiSquared*, double & *incompleteGammaQ*)

Perform a fit on a data set of doses and associated survival according to the linear quadratic formula.

The method perform a fit (making use of the GSL library) on the data set of doses (as x coordinate) and survivals (as y coordinate), weighted on the survival uncertainty, according to the linear quadratic formula:

$$S = \exp(-\alpha D - \beta D^2)$$

The method returns the LQ parameters α and β and some other useful informations on the fit (such as the χ^2 or the uncertainties on the estimated parameters) by overwriting the corresponding variables passed by reference.

Parameters

dose	The vector of doses (in Gy) to be used in the fit.
survival	The vector of survivals to be used in the fit.
survivalUncertainty	The vector of uncertainties associated to the evaluated survival to be used as weights in the fit.
alpha	The linear parameter α estimated, expressed in Gy^{-1}.
alphaUncertainty	The uncertainty associated to the α parameter (in Gy^{-1}).
beta	The quadratic parameter β estimated, expressed in Gy^{-2}.
betaUncertainty	The uncertainty associated to the β parameter (in Gy^{-2}).
chiSquared	The value of the χ^2 obtained from the fit.
incompleteGammaQ	The normalized incomplete Gamma Function $Q(a,x) = \frac{1}{\Gamma(a)} \int_x^{+\infty} t^{a-1}e^{-t}dt$ ($a > 0$; $x >= 0$).

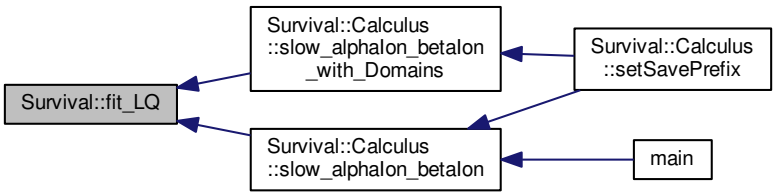
See also

Calculus::slow_meanDose_meanSurvival()

6.1.1.7 void Survival::fit_LQ (vector< double > *dose*, vector< double > *survival*, vector< double > *survivalUncertainty*, double & *alpha*, double & *alphaUncertainty*, double & *beta*, double & *betaUncertainty*, double & *chiSquared*, double & *incompleteGammaQ*)

Definition at line 256 of file usefulFunctions.cpp.

Here is the caller graph for this function:



6.1.1.8 bool Survival::folder_exists (std::string *foldername*)

Checks if a folder exists.

Parameters

foldername	The path to the folder to check.

Returns

`true` if the folder exists, `false` otherwise.

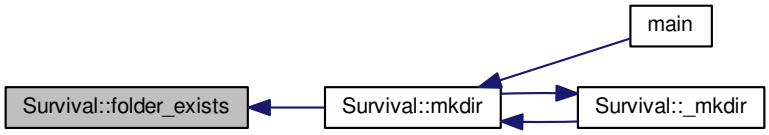
See also

_mkdir() and mkdir()

6.1.1.9 bool Survival::folder_exists (string *foldername*)

Definition at line 306 of file usefulFunctions.cpp.

Here is the caller graph for this function:



6.1.1.10 int Survival::mkdir (const char * *path*)

Recursive, portable wrapper for mkdir.

Parameters

path	The full path of the directory to create.

Returns

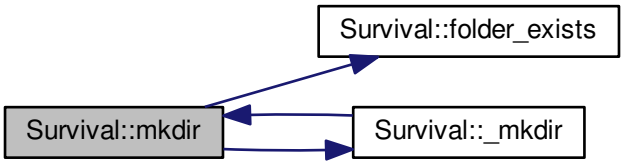
0 on success, otherwise -1.

See also

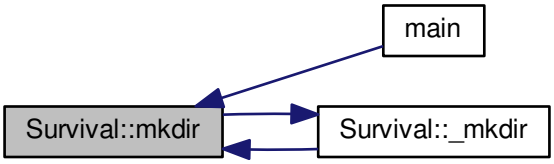
_mkdir() and folder_exists()

Definition at line 315 of file usefulFunctions.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.1.11 **void Survival::parse (int *argc,* char ∗ *argv[ ],* std::string & *cellType,* std::string & *model,* std::string & *trackType,* std::string & *parametrizationType,* std::string & *calculusType,* double & *precision,* int & *parallelismType,* std::vector< double > & *doses,* std::vector< std::string > & *parameter_name,* double & *MKM_alpha0,* double & *MKM_beta0,* double & *MKM_rNucleus,* double & *MKM_rDomain,* double & *tMKM_ac,* double & *LEM_alpha0,* double & *LEM_beta0,* double & *LEM_rNucleus,* double & *LEM_Dt,* std::string & *ionType,* int & *particleA,* int & *particleZ,* std::string & *trackMode,* std::string & *energyType,* std::vector< double > & *energies,* int & *nFraction,* double & *timeSpacing,* double & *fracDeliveryTime,* bool & *saveAlphaBeta,* bool & *saveMeans,* bool & *saveCell,* std::string & *projectName,* bool & *mono,* std::string & *spectrum_file*)**

Parses the input arguments in the `main` to set the calculation parameters.

Parses the input arguments passed by the user in order to set the calculation parameters for the simulation. If an incorrect setting occurs, the program will be terminated and the method display an hint to the user to fix the problem.

Note

All parameters, with the exception of `argc` and `argv`, are passed by reference to be overwritten.

Parameters

argc	The number of input arguments.
argv	A pointer to the array of `chars`, that is the input arguments set by the user.

Parameters

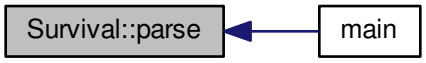
cellType	A `string` identifying the name of the cell line.
model	A `string` identifying the model to be used in the simulation. "LEMI", "LEMII", "LEMIII", "MKM" and "tMKM" supported.
trackType	A `string` defining the type of Track to be used in the simulation. "KieferChatterjee", "Scholz2000", "Elsasser2007" and "Elsasser2008" supported.
parametrizationType	
calculusType	A `string` identifying the way to perform
precision	Fix the ending condition of the Monte Carlo simulation.
parallelismType	The number of threads needed to be used in the simulation. It has to be a ...
doses	A vector of `double` containing MIN, MAX and number of doses to be simulated in order to construct the survival curve and extrapolate the LQ parameters.
parameter_name	A vector of `string` identifying the name of the model parameters correspondent to the model chosen.
MKM_alpha0	The MKM α_0 parameter, expressed in Gy^{-1}.
MKM_beta0	The MKM β_0 parameter, expressed in Gy^{-2}.
MKM_rNucleus	The radius of the MKM nucleus.
MKM_rDomain	The radius of domains in the MKM nucleus.
tMKM_ac	The time constant representing the cellular repair kinetics, expressed in h^{-1}.
LEM_alpha0	The LEM α_0 parameter, expressed in Gy^{-1}.
LEM_beta0	The LEM β_0 parameter, expressed in Gy^{-2}.
LEM_rNucleus	The radius of the LEM nucleus.
LEM_Dt	The transition dose beyond which the standard linear quadratic parametrization is no more valid according to the LEM parametrization.
ionType	A `string` identifying the ion generating the track (Chemical symbol: H, He, Li, ...). Ions until "Ne" are supported.
particleA	The mass number of the ion chosen.
particleZ	The atomic number of the ion chosen.
trackMode	A `string` identifying the modality to pass the vector of particles in the mixed fields case. "histogram" or "random" are supported.
energyType	A `string` identifying if input values are energies or LETs. "LET" or "energy" supported.
energies	A vector of `double` to store energies ot LETs to be used.
nFraction	A vector of `double` containing the number of fractions to be delivered in a fractionated treatment (MIN, MAX and STEP). Compatible with tMKM model.
timeSpacing	A vector of `double` containing the time spacing between fractions of a fractionated treatment (MIN, MAX and STEP). Compatible with tMKM model.
fracDeliveryTime	A vector of `double` containing the fraction delivery time of a fractionated treatment (MIN, MAX and STEP). Compatible with tMKM model.
saveAlphaBeta	A boolean value identifying if LQ data are to be saved.
saveMeans	A boolean value identifying if mean doses and survivals are to be saved.
saveCell	A boolean value identifying if the data of dose absorbed and survival observed by each single cell irradiated in the Monte Carlo simulation are to be saved.
projectName	The name of the project, chosen by the user.
mono	A boolean value identifying if the radiation is monoenergetic or not.
spectrum_file	A `string` identifying the name of the file containing the spectrum of energies to be used in the simulation (complete with its absolute or relative path).

6.1.1.12 void Survival::parse (int *argc,* char * *argv[ ],* string & *cellType,* string & *model,* string & *trackType,* string & *parametrizationType,* string & *calculusType,* double & *precision,* int & *parallelismType,* vector< double > & *doses,* vector< string > & *parameter_name,* double & *MKM_alpha0,* double & *MKM_beta0,* double & *MKM_rNucleus,* double & *MKM_rDomain,* double & *tMKM_ac,* double & *LEM_alpha0,* double & *LEM_beta0,* double & *LEM_rNucleus,* double & *LEM_Dt,* string & *ionType,* int & *particleA,* int & *particleZ,* string & *trackMode,* string & *energyType,* vector< double > & *energies,* int & *nFraction,* double & *timeSpacing,* double & *fracDeliveryTime,* bool & *saveAlphaBeta,* bool & *saveMeans,* bool & *saveCell,* string & *projectName,* bool & *mono,* string & *spectrum_file*)

Definition at line 338 of file usefulFunctions.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

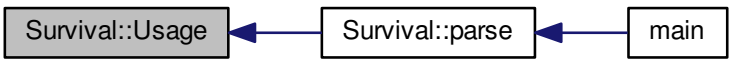


6.1.1.13 void Survival::Usage ( )

Display an hint to the user to correctly use the executable.

Definition at line 1078 of file usefulFunctions.cpp.

Here is the caller graph for this function:



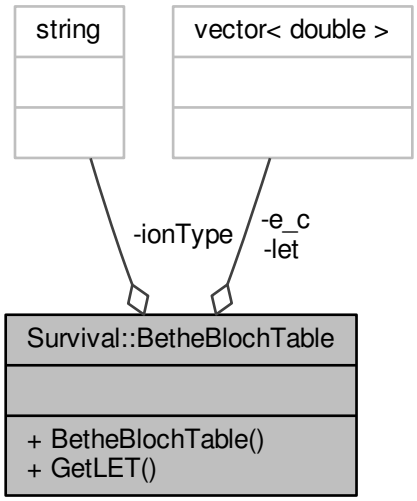
Chapter 7

Class Documentation

7.1 Survival::BetheBlochTable Class Reference

Class used to load precomputed values of LET for different ions of varying kinetic energy, evaluated by means of the Bethe-Bloch formula.

Collaboration diagram for Survival::BetheBlochTable:



Public Member Functions

- BetheBlochTable (const string &ionType_)

  Constructor. Instantiates and sets the object.
- double GetLET (double e_c_imposed) const

  Returns the value of LET corresponding to the kinetic energy imposed.

Private Attributes

- string ionType

  A string defining the name of the ion (chemical symbol).
- vector< double > e_c

  A vector to store the list of kinetic energies, expressed in MeV/u.
- vector< double > let

  A vector to store the precomputed LETs, expressed in MeV/um.

7.1.1 Detailed Description

Class used to load precomputed values of LET for different ions of varying kinetic energy, evaluated by means of the Bethe-Bloch formula.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2008–2011

Depending on the ion type chosen, it loads (in the constructor) the correspondent file from the "data" directory containing precomputed values of LET (linear energy transfer) each one associated to a different kinetic energy, storing them into two vectors (one for the kinetic energies e_c and one for the LETs let). It provides a method (GetLET()) to get the LET of the ions correspondent to a given kinetic energy.

Definition at line 74 of file usefulFunctions.cpp.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Survival::BetheBlochTable::BetheBlochTable (const string & *ionType_*) [inline]

Constructor. Instantiates and sets the object.

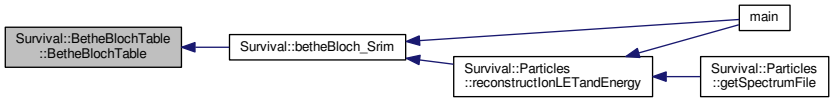
Opens the data file containing the table correspondent to the chosen ion and store the values of kinetic energies and LETs in e_c and let respectively.

Warning

The execution of the program will be terminated if the data file doesn't exist or if the environment hasn't been set correctly.

Definition at line 84 of file usefulFunctions.cpp.

Here is the caller graph for this function:



7.1.3 Member Function Documentation

7.1.3.1 double Survival::BetheBlochTable::GetLET (double *e_c_imposed*) const [inline]

Returns the value of LET corresponding to the kinetic energy imposed.

Find the indexes in the e_c vector of the two nearest neighbors of the energy fixed and interpolates them to get the correct value of LET.

Parameters

e_c_imposed	The kinetic energy (in MeV/u) correspondent to the required LET.

Returns

The value of LET corresponding to the kinetic energy imposed, expressed in MeV/um.

Warning

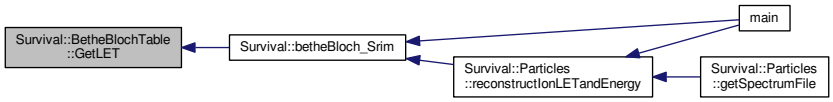
The execution of the program will be terminated if the kinetic energy imposed is out of the available range for the ion specified.

See also

betheBloch_Srim()

Definition at line 131 of file usefulFunctions.cpp.

Here is the caller graph for this function:



7.1.4 Member Data Documentation

7.1.4.1 vector<double> Survival::BetheBlochTable::e_c [private]

A vector to store the list of kinetic energies, expressed in MeV/u.

Definition at line 168 of file usefulFunctions.cpp.

7.1.4.2 string Survival::BetheBlochTable::ionType [private]

A string defining the name of the ion (chemical symbol).

Definition at line 160 of file usefulFunctions.cpp.

7.1.4.3 vector<double> Survival::BetheBlochTable::let [private]

A vector to store the precomputed LETs, expressed in MeV/um.

Definition at line 171 of file usefulFunctions.cpp.

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

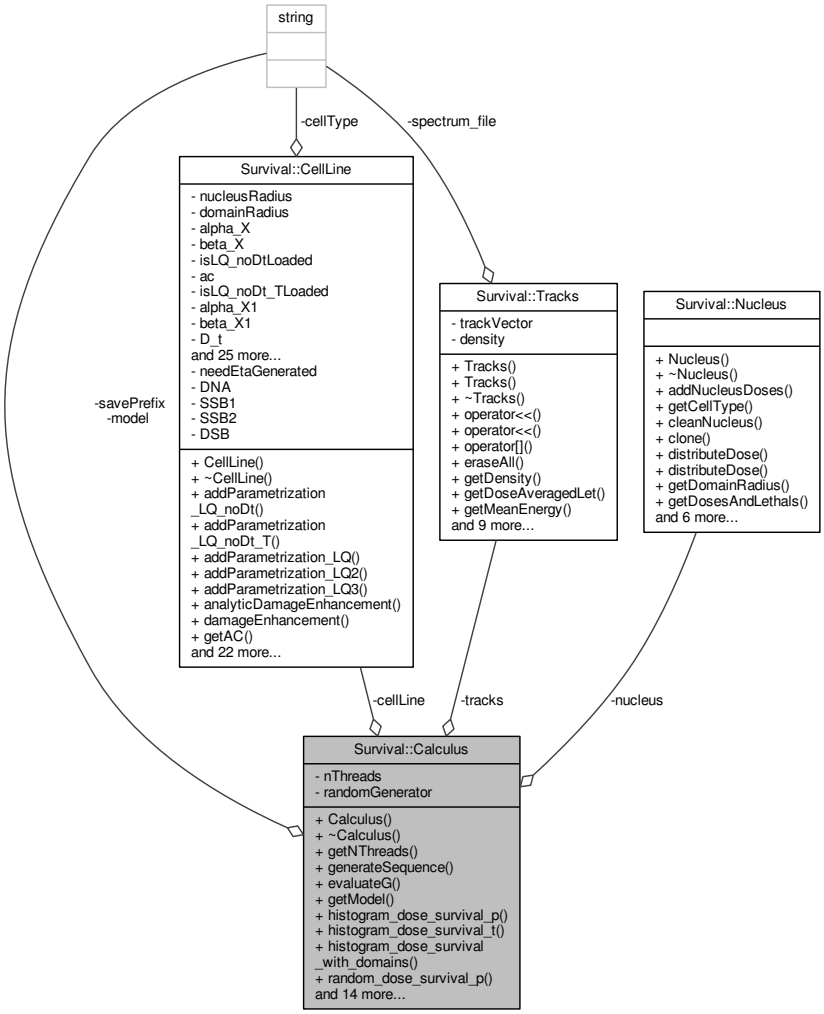
- src/usefulFunctions.cpp

7.2 Survival::Calculus Class Reference

It implements some methods to simulate the irradiation process evaluating the cellular survival on a population of cells and extrapolating the radiobiological LQ parameters.

`#include <Calculus.h>`

Collaboration diagram for Survival::Calculus:



Public Member Functions

- Calculus (const Tracks &tracksRef, const CellLine &cellLineRef, Nucleus &nucleusRef, std::string savePrefix, std::string model, int p_type=1, long randomSeed=0)

  Constructor. Instantiates and sets the object.
- ~Calculus ()

  Destructor.
- int getNThreads (void) const

  Returns the number of threads used in the simulation.
- std::vector< double > generateSequence (int nEv, double begin, double duration)

  Generate an ordered fixed-size sequence of pseudorandom numbers in [begin, begin+duration].
- void evaluateG (const std::string trackMode, double totalDose, int nFrac, double timeSpacing, double fracDeliveryTime, double precision, double alpha, double beta)

  Perform a Monte Carlo simulation delivering a fixed total dose with a defined time structure to a cell population, evaluating the resulting cell survival in order to get the value of the G factor that defines the temporal effect of the irradiation.
- std::string getModel () const

  Returns the model used in the simulation.
- void histogram_dose_survival_p (const double doseImposed, double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty, Nucleus &nuc_cp, bool clean=true)

  Simulates, via the Monte Carlo method, the irradiation process of a nucleus which is part of the cellular population.
- void histogram_dose_survival_t (Nucleus &nuc_cp, const double doseImposed, const double time, const double fracDeliveryTime)

  Simulates via the Monte Carlo method the irradiation process of a nucleus, which is part of the cellular population, in a single fraction with a defined duration.
- void histogram_dose_survival_with_domains (const double doseImposed, double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty, std::vector< double > &doses, std::vector< double > &lethals, std::vector< double > &dosesUncertainty, std::vector< double > &lethalsUncertainty, bool clean=true)

  Simulates, via the Monte Carlo method, the irradiation process of a nucleus which is part of the cellular population. This function was thought to control the dose delivery inside the MKM nucleus. It's similar to histogram_dose_survival() but provide also informations on the microscopic dose deposition pattern in each domain of the nucleus.
- void random_dose_survival_p (const double doseImposed, double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty, Nucleus &nuc_cp, bool clean=true)

  Simulates, via the Monte Carlo method, the irradiation process of a nucleus which is part of the cellular population.
- void rapidINFN_alphaIon_betaIon (double &alphaIon, double &betaIon)

  This method was developed to improve both the α and β estimation with respect to the approach of Scholz. It is currently applicable to the LEM I version only (the possibility of extending the approach to the subsequent versions is under investigation).
- void rapidMKM_Kase2008 (double &alphaIon, double &betaIon)

  Fast implementation of the MKM as described in (Kase_2008)
- void rapidMKM_Kase2008_corrected_beta (double &alphaIon, double &betaIon)

  Extension of the rapidMKM_Kase2008() method with $\beta = \beta(\mathrm{LET})$.
- void rapidMKM_Attili2013 (double &alphaIon, double &betaIon)

  This method provide a fast original implementation of the MKM model, combining the methods described in (Hawkins_2003) and (Kase_2008).
- void rapidMKM_Attili2013_corrected_beta (double &alphaIon, double &betaIon)

  Extension of the rapidMKM_Attili2013() method with $\beta = \beta(\mathrm{LET})$.
- void rapidLEM_Russo2011 (double &alphaIon, double &betaIon)

  This method is based on the approach of Scholz (rapidScholz_alphaIon_betaIon()) but provides a more precise estimation of the α parameter.
- void rapidLEM_Scholz2006 (double &alphaIon, double &betaIon)

  This method provide a faster approximate implementation of the LEM model that avoids the Monte Carlo simulation.
- void setNThreads (int nTh)

Sets the number of threads.

- void setSavePrefix (std::string save_prefix)

  Sets the prefix of the output file name.

- void slow_alphaIon_betaIon (const std::string trackMode, const std::vector< double > parameters, const std::vector< double > dosesImposed, const double precision, double &alphaIon, double &alphaIon↩ Uncertainty, double &betaIon, double &betaIonUncertainty, const int nFraction, const double timeSpacing, const double fracDeliveryTime, const bool saveAlphaBeta, const bool saveMeans, const bool saveCell, const std::string title_means)

  Method called to perform a Monte Carlo simulation to reproduce the irradiation process getting the LQ parameters α and β.

- void slow_alphaIon_betaIon_with_Domains (const std::string trackMode, const double minDose, const double maxDose, const int numberOfDoses, const double precision, double &alphaIon, double &alphaIon↩ Uncertainty, double &betaIon, double &betaIonUncertainty)

  This function was thought to control the dose delivery inside the MKM nucleus. It's similar to slow_alphaIon_betaIon() but provide also informations on the microscopic dose deposition pattern in each domain of the nucleus.

- void slow_meanDose_meanSurvival (const std::string trackMode, const double doseImposed, const double precision, double &meanDose, double &meanDoseUncertainty, double &meanSurvival, double &mean↩ SurvivalUncertainty, const int nFraction, const double timeSpacing, const double fracDeliveryTime, const bool saveCell)

  Perform the Monte Carlo simulation relatively to a single value of dose imposed and returns the mean values of dose absorbed and survival observed in the cell population.

- void slow_meanDose_meanSurvival_with_Domains (const std::string trackMode, const double doseImposed, const double precision, double &meanDose, double &meanDoseUncertainty, double &meanSurvival, double &meanSurvivalUncertainty)

  Perform the Monte Carlo simulation relatively to a single value of dose imposed and returns the mean values of dose absorbed and survival observed in the cell population. This function was thought to control the dose delivery inside the MKM nucleus. It's similar to slow_meanDose_meanSurvival() but provide also informations on the microscopic dose deposition pattern in each domain of the nucleus.

- void verbatim_dose_survival (double &dose, double &doseUncertainty, double &survival, double &survival↩ Uncertainty, bool clean=true)

  Evaluates the dose deposited in the nucleus using directly the tracks vector without modifying it and without random numbers extractions.

Private Attributes

- const Tracks & tracks

  A `const` reference to a Track object corresponding to the Particle interacting with nucleus.

- const CellLine & cellLine

  A `const` reference to a CellLine object corresponding to the cell line to which the nucleus belongs.

- Nucleus & nucleus

  A reference to the cellular nucleus.

- int nThreads

  The number of threads needed to be used in the simulation (if parallelism is supported).

- gsl_rng * randomGenerator

  A pointer to a gsl_rng object, useful in the generation of pseudorandom numbers in the Monte Carlo simulation.

- std::string savePrefix

  The prefix of the output file.

- std::string model

  The model used in the simulation.

7.2.1 Detailed Description

It implements some methods to simulate the irradiation process evaluating the cellular survival on a population of cells and extrapolating the radiobiological LQ parameters.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2011–2018

The class makes use of Track and Nucleus objects to simulate the irradiation process. Different methods are implemented in order to perform Monte Carlo simulations or approximated analytic evaluations of the process.

Definition at line 23 of file Calculus.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Calculus::Calculus (const Tracks & *tracksRef,* const CellLine & *cellLineRef,* Nucleus & *nucleusRef,* std::string *savePrefix,* std::string *model,* int *p_type* = `1`, long *randomSeed* = `0`)

Constructor. Instantiates and sets the object.

It needs references to Tracks, CellLine and Nucleus objects to be used in the simulation. It needs also the prefix of the output file and provides the possibility to fix the seed of the pseudorandom numbers generator and to manage the number of threads (only necessary for methods that implement a multithread system). The class makes use of the GSL library to manage the generation of pseudorandom numbers, hence an instance of the Tausworthe generator is created and the pseudorandom numbers generator is seeded.

Parameters

tracksRef	A reference to a Tracks object to manage the radiation.
cellLineRef	A reference to a CellLine object containing the structural and radiobiological informations.
nucleusRef	A reference to a Nucleus object to be irradiated.
savePrefix	A `string` indicating the prefix of the output file.
model	The model used in the simulation.
p_type	The type of parallelism desired (see nThreads).
randomSeed	The seed to be used in the initialization of the pseudorandom numbers generator.

Definition at line 60 of file Calculus.cpp.

7.2.2.2 Calculus::~Calculus ( )

Destructor.

Deletes the Calculus object and the gsl_rng pointer.

Definition at line 92 of file Calculus.cpp.

7.2.3 Member Function Documentation

7.2.3.1 void Calculus::evaluateG (const std::string *trackMode*, double *totalDose*, int *nFrac*, double *timeSpacing*, double *fracDeliveryTime*, double *precision*, double *alpha*, double *beta*)

Perform a Monte Carlo simulation delivering a fixed total dose with a defined time structure to a cell population, evaluating the resulting cell survival in order to get the value of the G factor that defines the temporal effect of the irradiation.

The user has to fix:

1. The precision required for the simulation, that is the statistics to be reached to terminate the simulation. Two possibilities are supported, so the user can:
   - Fix the number of iterations, hence the `precision` has to be an integer value greater (or at least equal) to 1.
   - Define a constraint on the precision to reach in the simulation in the evaluation of the cell survival (precisely the relative error on the survival), hence the `precision` has to be a `double` in (0, 1).
2. The structure of the irradiation, which is defined by:
   - The total dose to be delivered.
   - The number of fractions of the treatment.
   - The time spacing between two consecutive fractions.
   - The time needed to deliver a single fraction. Once set these features, the irradiation process of the cell population is simulated by calling the histogram_dose_survival_t() method in a parallel loop. Each thread performs the irradiation of a single cell (with its complete time structure) and at the end of each irradiation the method updates the total mean dose and survival observed, with associated uncertainties, and determines if the precision set by the user is reached or not. All the values of dose and survival obtained, cell by cell, are saved in an output file. Once reached the precision required, the method evaluates the G factor using the LQ α and β parameters obtained in the case of acute irradiation (parameters of the function) by means of the relation:

$$G = \frac{-\ln(S) - \alpha D}{\beta D^2}$$

   where D is the total dose delivered and S the resulting survival. It evaluates also the G factor following the analytical expression (valid only in case of fractionated treatment with neglecting the intra-fraction time structure):

$$G = 1 - \frac{2}{D^2} \sum_{i=0}^{N-1} \sum_{j=i+1}^{N-2} \left(1 - \exp[-(a+c)(t_j - t_i)]\right) d^2$$

   where d represents the dose delivered per fraction, N the total number of fractions, $(t_j - t_i)$ represents the temporal distance between two fractions and (a+c) is the time constant (see Nucleus_tMKM::ac). The calculated values, together with the informations on the time structure are saved in another output file.

Note

Survival uncertainty: the estimation of the survival uncertainty is knowingly biased, but extremely efficient and memory-friendly. Note that this quantity is used only to decide when it is possible to stop the simulation hence it doesn't affect the result (and that the bias decrease with increasing iteration).

Parameters

trackMode	Defined for completeness. Only "histogram" is supported.
totalDose	The total dose to be delivered, expressed in Gy.
nFrac	The total number of fractions constituting the treatment.
timeSpacing	The time spacing between two consecutive fractions, expressed in hours.
fracDeliveryTime	The time needed to deliver a single fraction, expressed in hours.
precision	The precision to be reached in the simulation (could be a fixed number of iterations or a constraint on the survival precision).
alpha	LQ α parameter obtained in acute conditions.
beta	LQ β parameter obtained in acute conditions.

Warning

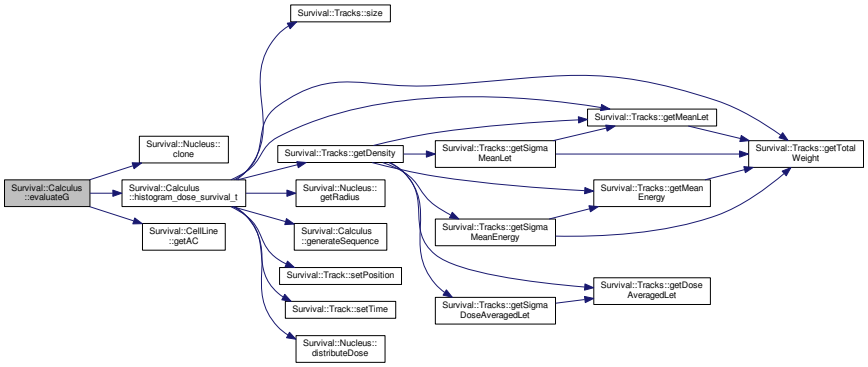
The execution of the program will be terminated if the precision is not set correctly.

See also

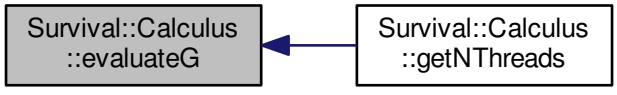
histogram_dose_survival_t() and slow_meanDose_meanSurvival()

Definition at line 99 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.2 vector< double > Calculus::generateSequence (int *nEv*, double *begin*, double *duration*)

Generate an ordered fixed-size sequence of pseudorandom numbers in [begin, begin+duration].

Parameters

nEv	Size of the sequence, that is the number of pseudorandom numbers generated.
begin	The minimum number extractable.
duration	The width of the interval.

Returns

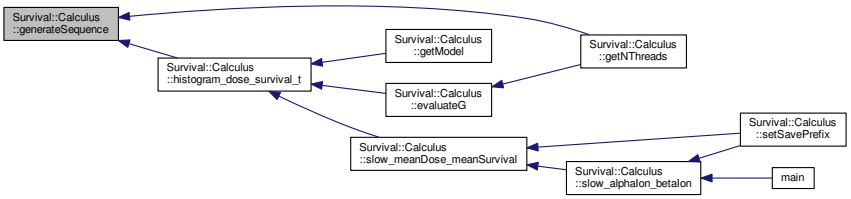
A STL vector containing an ordered sequence of pseudorandom numbers.

Warning

The execution of the program will be terminated if a negative "duration" is chosen.

Definition at line 247 of file Calculus.cpp.

Here is the caller graph for this function:



7.2.3.3 std::string Survival::Calculus::getModel ( ) const [inline]

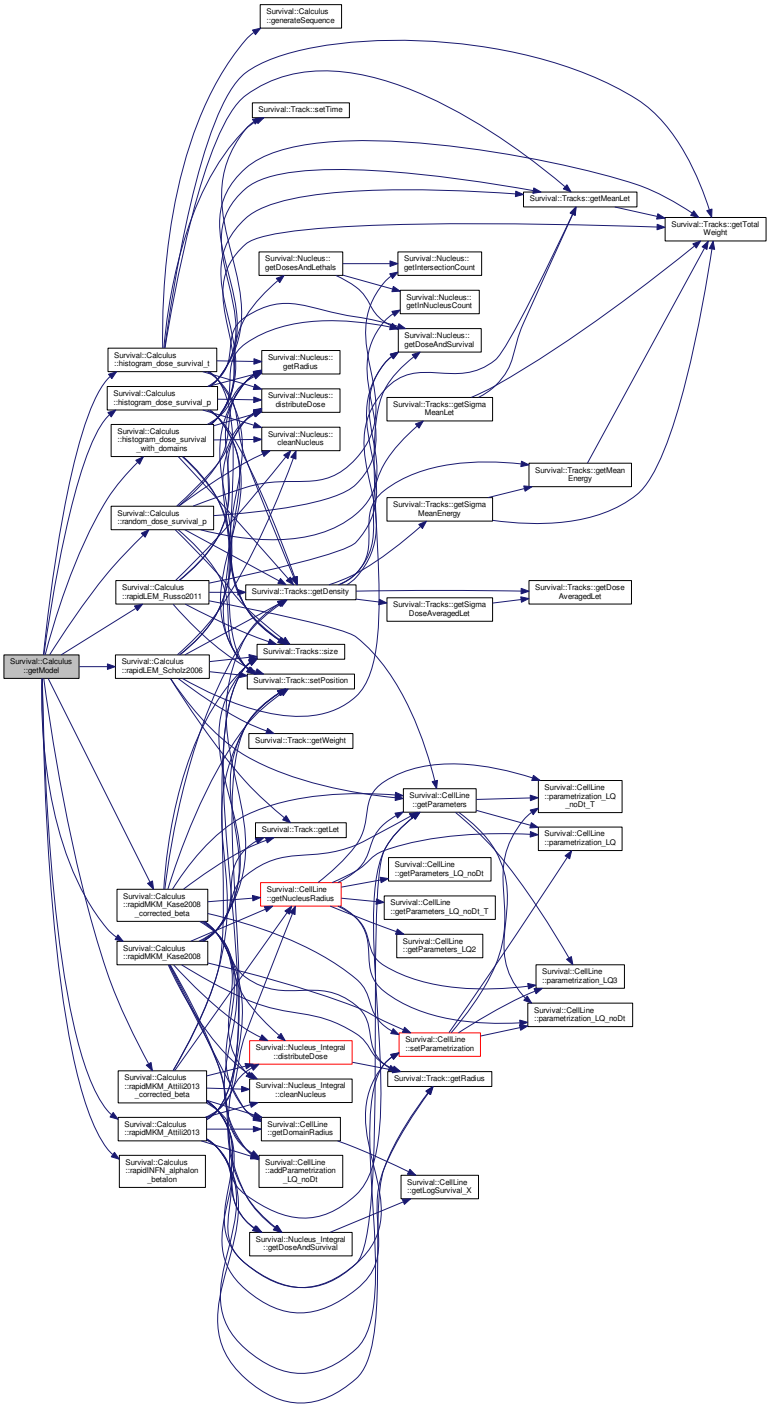
Returns the model used in the simulation.

Returns

A string indicating the model used in the simulation.

Definition at line 128 of file Calculus.h.

Here is the call graph for this function:



7.2.3.4 int Survival::Calculus::getNThreads (void) const [inline]

Returns the number of threads used in the simulation.

Returns

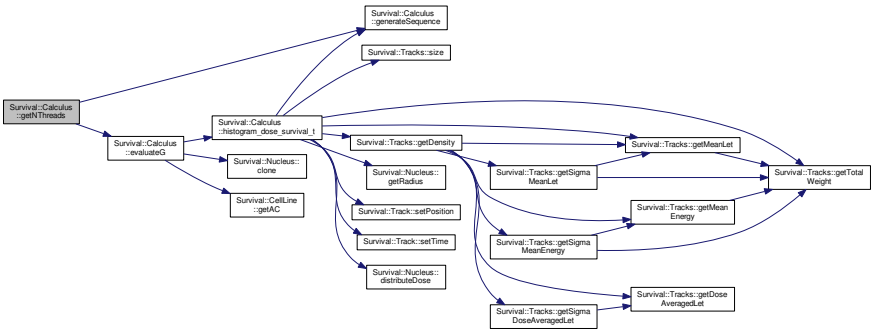
The number of threads used in the simulation.

See also

nThreads

Definition at line 60 of file Calculus.h.

Here is the call graph for this function:



7.2.3.5 void Calculus::histogram_dose_survival_p (const double *doseImposed,* double & *dose,* double & *doseUncertainty,* double & *survival,* double & *survivalUncertainty,* Nucleus & *nuc_cp,* bool *clean* = `true`)

Simulates, via the Monte Carlo method, the irradiation process of a nucleus which is part of the cellular population.

First of all the function calculate the fluence of the beam on the base of the dose imposed, by means of the following relation:

$$\Phi = \rho \frac{D}{K \cdot \langle LET \rangle}$$

where D is the dose imposed, K is a constant to accounts for the different units of measure used, ρ is the density of the target and $\langle LET \rangle$ is the mean value of LET of the radiation (that coincides with the LET of the particle only in the monoenergetic case). The number of particles interacting with the nucleus is randomly extracted with a poisson distribution whose mean value is determined on the basis of the fluence, opportunely normalized accounting also for the weight of the particle used. Then, in two nested `for` loops over the number of particles extracted and over the tracks vector, the dose deposited in the nucleus is evaluated by means of the Nucleus::distributeDose() method. At the end of the loop, dose and survival (with respective uncertainties) are updated by calling the Nucleus::get↩DoseAndSurvival() method.

Note

Parallelism: If the parallelism is disabled, the same nucleus could be irradiated and cleaned recursively in order to save memory; but if the parallelism is used, each thread must work on a different nucleus, hence this method provide the possibility to indicate also the nucleus to be irradiated with the `nuc_cp` parameter. This is the only difference between this method and histogram_dose_survival() which is actually useless. It was not deleted only for a chronological reason, because the parallelism was implemented later, but at present it is not used.

Parameters

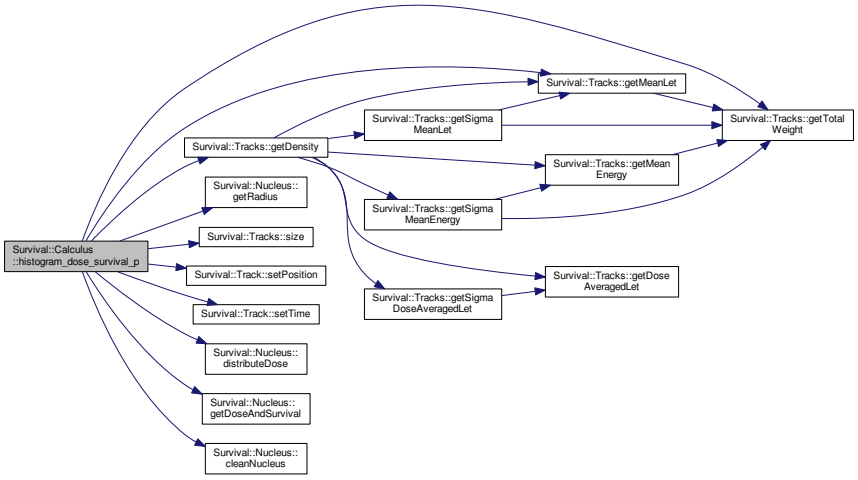
doseImposed	The dose imposed to be delivered to the nucleus, expressed in Gy.
dose	The dose absorbed by the nucleus in the irradiation, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival observed, passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.
nuc_cp	A reference to the nucleus to be irradiated.
clean	A boolean value indicating if the nucleus has to be cleaned at the end of the evaluation.

See also

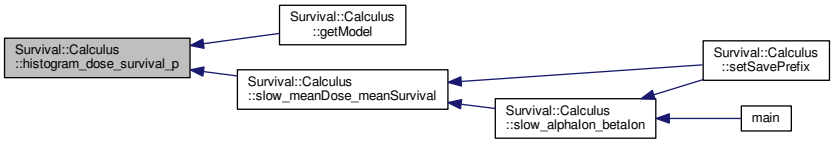
random_dose_survival_p() and histogram_dose_survival_t()

Definition at line 264 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.6 void Calculus::histogram_dose_survival_t (Nucleus & *nuc_cp,* const double *doseImposed,* const double *time,* const double *fracDeliveryTime*)

Simulates via the Monte Carlo method the irradiation process of a nucleus, which is part of the cellular population, in a single fraction with a defined duration.

First of all the function calculate the fluence of the beam on the base of the dose imposed, by means of the following relation:

$$\Phi = \rho \frac{D}{K \cdot \langle LET \rangle}$$

where D is the dose imposed, K is a constant to accounts for the different units of measure used, ρ is the density of the target and $\langle LET \rangle$ is the mean value of LET of the radiation (that coincides with the LET of the particle only in the monoenergetic case). The number of particles interacting with the nucleus is randomly extracted with a poisson distribution whose mean value is determined on the basis of the fluence, opportunely normalized accounting also for the weight of the particle used. Before irradiating, the time structure of the irradiation has to be defined: starting from an instant indicated as parameter of the function (t_0), a sequence of times is randomly extracted with uniform probability in $[t_0,\ t_0 + \Delta t]$ (where Δt indicates the fraction delivery time); each time extracted is associated to a particle as its temporal index. Then, in two nested `for` loops over the number of particles extracted and over the tracks vector, the dose deposited in the nucleus is evaluated by means of the Nucleus::distributeDose() method.

Parameters

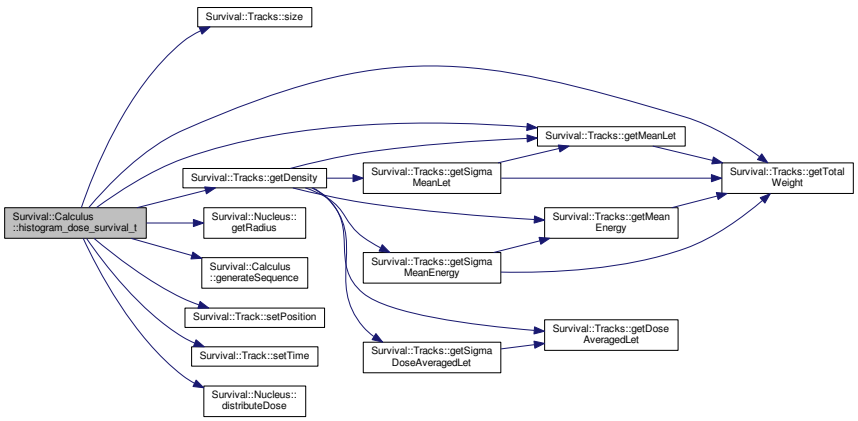
nuc_cp	A reference to the nucleus to be irradiated.
doseImposed	The dose to be delivered in the irradiation in a single fraction, expressed in Gy.
time	The starting point of the temporal sequence, expressed in hours.
fracDeliveryTime	The time needed to deliver a single fraction, expressed in hours.

See also

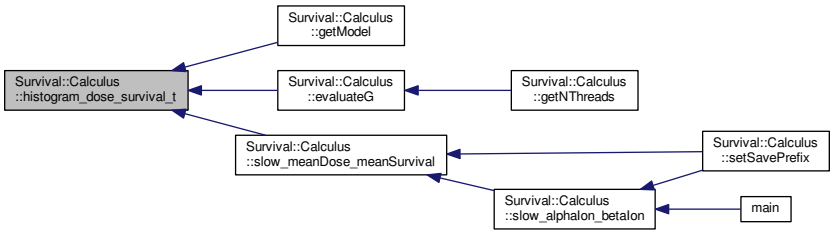
evaluateG() and histogram_dose_survival_p()

Definition at line 315 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.7 void Calculus::histogram_dose_survival_with_domains (const double *doseImposed,* double & *dose,* double & *doseUncertainty,* double & *survival,* double & *survivalUncertainty,* std::vector< double > & *doses,* std::vector< double > & *lethals,* std::vector< double > & *dosesUncertainty,* std::vector< double > & *lethalsUncertainty,* bool *clean* = `true`)

Simulates, via the Monte Carlo method, the irradiation process of a nucleus which is part of the cellular population. This function was thought to control the dose delivery inside the MKM nucleus. It's similar to histogram_dose_survival() but provide also informations on the microscopic dose deposition pattern in each domain of the nucleus.

First of all the function calculate the fluence of the beam on the base of the dose imposed, by means of the following relation:

$$\Phi = \rho \frac{D}{K \cdot \langle LET \rangle}$$

where D is the dose imposed, K is a constant to accounts for the different units of measure used, ρ is the density of the target and $\langle LET \rangle$ is the mean value of LET of the radiation (that coincides with the LET of the particle only in the monoenergetic case). The number of particles interacting with the nucleus is randomly extracted with a poisson distribution whose mean value is determined on the basis of the fluence, opportunely normalized accounting also for the weight of the particle used. Then, in two nested `for` loops over the number of particles extracted and over the tracks vector, the dose deposited in the nucleus is evaluated by means of the Nucleus::distributeDose() method. At the end of the loop, dose and survival (with respective uncertainties) are updated by calling the Nucleus::getDoseAndSurvival() method. It returns also the microscopical informations on doses deposited and lethal events observed in each domain by overwriting the correspondent parameters.

Parameters

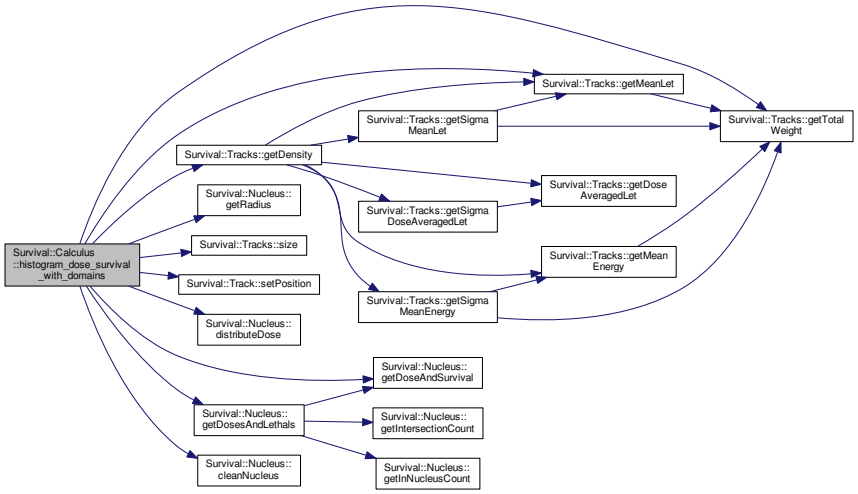
doseImposed	The dose imposed to be delivered to the nucleus, expressed in Gy.
dose	The dose absorbed by the nucleus in the irradiation, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival observed, passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.
doses	The vector of doses absorbed by each domain, expressed in Gy, passed by reference to be overwritten.
lethals	The vector of lethal events observed in each domain, passed by reference to be overwritten.
dosesUncertainty	The uncertainties associated to the doses absorbed, expressed in Gy, passed by reference to be overwritten.
lethalsUncertainty	The uncertainties associated to the lethal events observed, passed by reference to be overwritten.
clean	A boolean value indicating if the nucleus has to be cleaned at the end of the evaluation.

See also

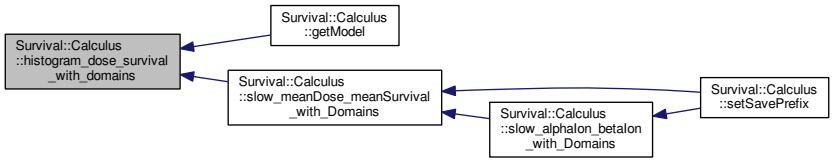
histogram_dose_survival_p()

Definition at line 365 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.8 void Calculus::random_dose_survival_p (const double *doseImposed,* double & *dose,* double & *doseUncertainty,* double & *survival,* double & *survivalUncertainty,* Nucleus & *nuc_cp,* bool *clean* = `true`)

Simulates, via the Monte Carlo method, the irradiation process of a nucleus which is part of the cellular population.

First of all the function calculate the fluence of the beam on the base of the dose imposed, by means of the following relation:

$$\Phi = \rho \frac{D}{K \cdot \langle LET \rangle}$$

where D is the dose imposed, K is a constant to accounts for the different units of measure used, ρ is the density of the target and $\langle LET \rangle$ is the mean value of LET of the radiation (that coincides with the LET of the particle only in the monoenergetic case). The number of particles interacting with the nucleus is randomly extracted with a poisson distribution whose mean value is determined on the basis of the fluence, opportunely normalized. Then, in a `for` loops over the total number of particles extracted, a different particle from the tracks vector is randomly chosen (iteration by iteration) with uniform probability and the dose deposited in the nucleus is evaluated by means of the Nucleus::distributeDose() method. At the end of the loop, dose and survival (with respective uncertainties) are updated by calling the Nucleus::getDoseAndSurvival() method.

Note

Parallelism: If the parallelism is disabled, the same nucleus could be irradiated and cleaned recursively in order to save memory; but if the parallelism is used, each thread must work on a different nucleus, hence this method provide the possibility to indicate also the nucleus to be irradiated with the `nuc_cp` parameter. This is the only difference between this method and random_dose_survival() which is actually useless. It was not deleted only for a chronological reason, because the parallelism was implemented later, but at present it is not used.

Parameters

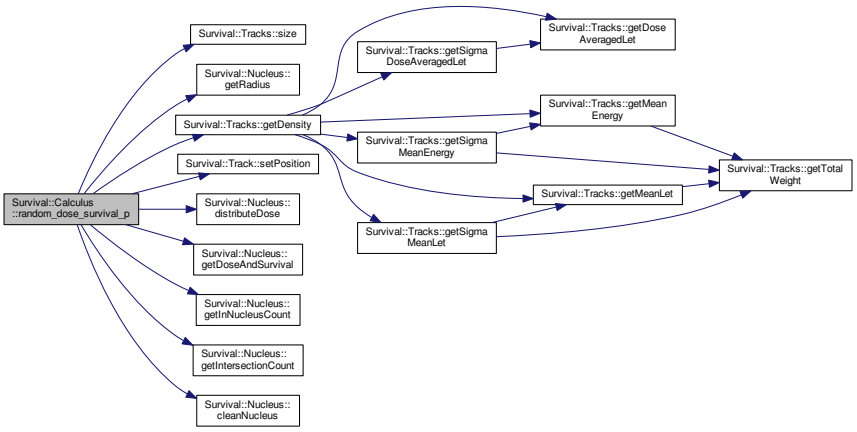
doseImposed	The dose imposed to be delivered to the nucleus, expressed in Gy.
dose	The dose absorbed by the nucleus in the irradiation, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival observed, passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.
nuc_cp	A reference to the nucleus to be irradiated.
clean	A boolean value indicating if the nucleus has to be cleaned at the end of the evaluation.

See also

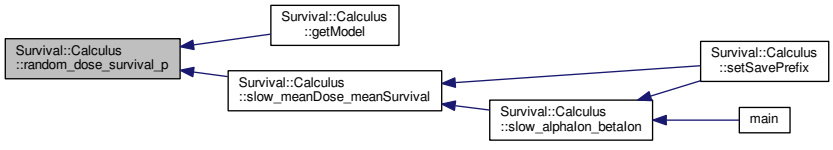
random_dose_survival(), histogram_dose_survival_p() and slow_meanDose_meanSurvival()

Definition at line 419 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.9 void Calculus::rapidINFN_alphaIon_betaIon (double & *alphaIon,* double & *betaIon*)

This method was developed to improve both the α and β estimation with respect to the approach of Scholz. It is currently applicable to the LEM I version only (the possibility of extending the approach to the subsequent versions is under investigation).

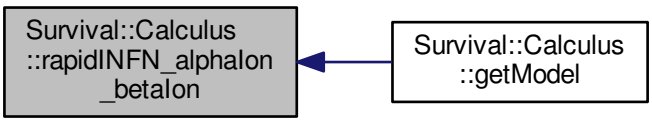
This method is based on a quasi-analytic solution of the LEM for the α_P and β_P parameters.

Warning

It hasn't been implemented yet.

Definition at line 479 of file Calculus.cpp.

Here is the caller graph for this function:



7.2.3.10 void Calculus::rapidLEM_Russo2011 (double & *alphaIon,* double & *betaIon*)

This method is based on the approach of Scholz (rapidScholz_alphaIon_betaIon()) but provides a more precise estimation of the α parameter.

This method was proposed by the INFN in 2011, with the work of Russo (1). To improve the rapidScholz_alpha↩ Ion_betaIon() method it is necessary to include the effect of ions that, passing near the nucleus, irradiate it only through their track penumbra.

Here, the rigorous derivations of the formulas used is omitted, the reader interested look at the work of Russo (1).

The LQ α parameter could be calculated in the monoenergetic case as:

$$\alpha = \alpha_X \left(1 - \langle z \rangle_{dir} \frac{\rho A_{nucl}}{LET}\right) + (1 - \langle S \rangle_{dir}) \frac{\rho A_{nucl}}{LET}$$

where ρ represents the density of the medium, A_{nucl} the area of the nucleus, $\langle z \rangle_{dir}$ and $\langle S \rangle_{dir}$ are the single-event dose and survival corresponding to the ion traversing the nucleus at its center and α_X is the linear quadratic α-parameter characteristic for X-rays. The β parameter, as in the case of the approach of Scholz, is evaluated as

$$\beta = \left(\frac{\alpha}{\alpha_P}\right)^2 \beta_P$$

where, in this case, diversely from the approach of Scholz:

$$\alpha_P = \alpha_X \left(1 - \langle z \rangle_{dir} \frac{\rho A_{nucl}}{LET}\right) - \ln(\langle S \rangle_{dir}) \frac{\rho A_{nucl}}{LET}$$

and

$$\beta_P = \frac{s - \alpha_P}{2D_t}$$

(see the LEM II parametrization for *s* and D_t, CellLine::parametrization_LQ2()). To accounts also for the mixed fields, the method evaluates $\langle S \rangle_{dir}$ for each tracks of the tracks vector estimating its α accounting also for the particle weight (Particle::weight). Then α and β are evaluating according to the theory of dual radiation action (2):

$$\alpha = \frac{\sum_i \alpha_i \, LET_i}{\sum_i LET_i}$$

$$\sqrt{\beta} = \frac{\sum_i \sqrt{\beta_i} \, LET_i}{\sum_i LET_i}$$

The function provide also the possibility to plot the results.

Parameters

alphaIon	The LQ α parameter expressed in Gy^{-1}, passed by reference to be overwritten.
betaIon	The LQ β parameter expressed in Gy^{-2}, passed by reference to be overwritten.

See also

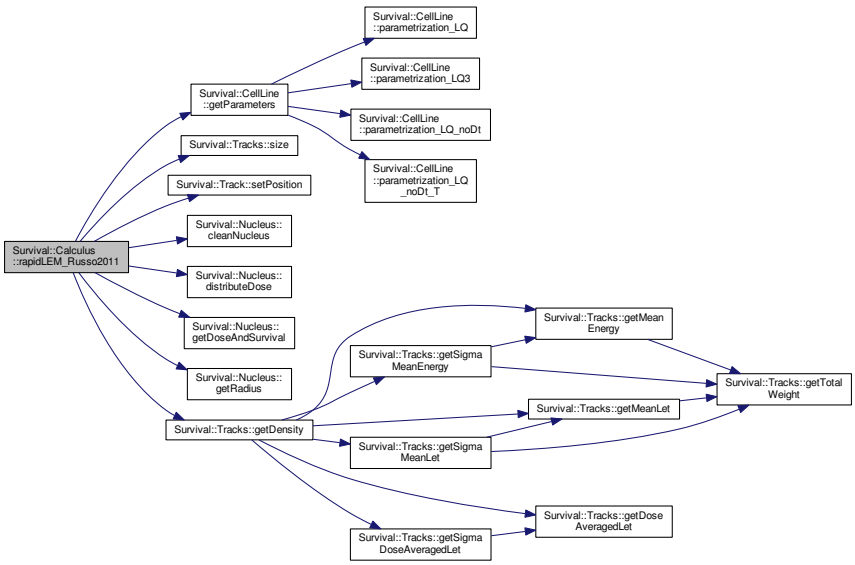
rapidScholz_alphaIon_betaIon() and rapidINFN_alphaIon_betaIon()

1. G. Russo, "Develpment of a radiobiological database for carbon ion Treatment Planning Systems - Modelling and simulating the irradiation process", *PhD Thesis*, Università degli studi di Torino (2011).

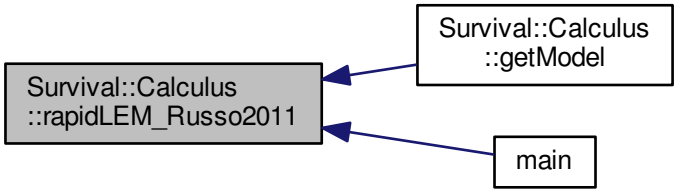
2. M. Zaider and H.H. Rossi, "The synergistic effets of different radiations", *Radiation Research* **160**, 61-69 (2003).

Definition at line 921 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.11 void Calculus::rapidLEM_Scholz2006 (double & *alphalon*, double & *betalon*)

This method provide a faster approximate implementation of the LEM model that avoids the Monte Carlo simulation.

This method was proposed by Krämer and Scholz in 2006 (1) and it is applicable only to the case of monoenergetic irradiation, but the estimation is extended to the mixed-field case exploiting the Theory of Dual Radiation Action (TDRA) by Zaider and Rossi (2). The method assumes that only direct impacts of ions on the cell nucleus are relevant for the evaluation of the α parameter that could be calculated (in the monoenergetic case) as:

$$\alpha = \frac{\rho A_{nucl}}{LET}(1 - \langle S \rangle_{dir})$$

where ρ represents the density of the medium, A_{nucl} the area of the nucleus and $\langle S \rangle_{dir}$ is the single-event survival corresponding to the ion traversing the nucleus at its center, disregarding the small dependence over the ion impact parameter. The β parameter is estimated as:

$$\beta = \left(\frac{\alpha}{\alpha_P}\right)^2 \beta_P$$

where

$$\alpha_P = -\ln(\langle S \rangle_{dir}) \frac{\rho A_{nucl}}{LET}$$

and

$$\beta_P = \frac{s - \alpha_P}{2D_t}$$

(see the LEM II parametrization for *s* and D_t, CellLine::parametrization_LQ2()). To accounts also for the mixed fields, the method evaluates $\langle S \rangle_{dir}$ for each tracks of the tracks vector estimating its α accounting also for the particle weight (Particle::weight). Then α and β are evaluating according to the TDRA:

$$\alpha = \frac{\sum_i \alpha_i \, LET_i}{\sum_i LET_i}$$

$$\sqrt{\beta} = \frac{\sum_i \sqrt{\beta_i} \, LET_i}{\sum_i LET_i}$$

The function provide also the possibility to plot the results.

Note

For a rigorous derivation of the formulas used see the published reference.

Parameters

alphaIon	The LQ α parameter expressed in Gy^{-1}, passed by reference to be overwritten.
betaIon	The LQ β parameter expressed in Gy^{-2}, passed by reference to be overwritten.

See also

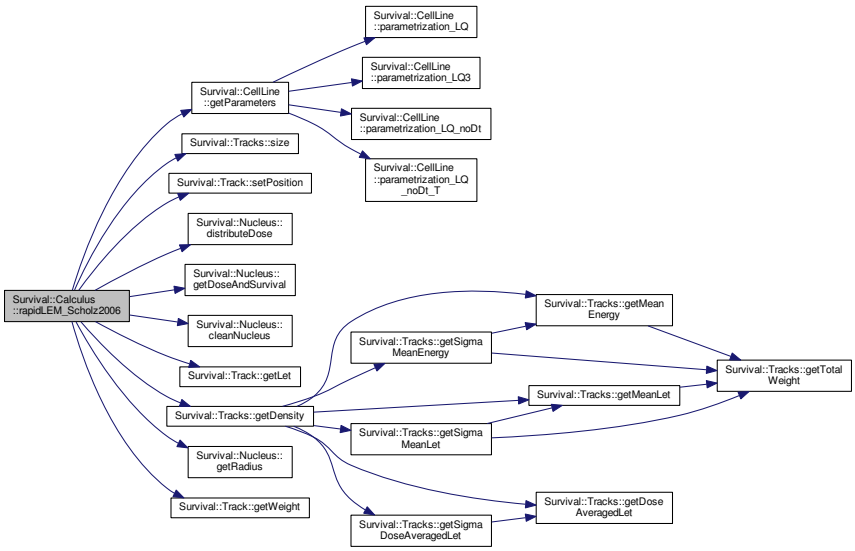
rapidRusso_alphaIon_betaIon() and rapidINFN_alphaIon_betaIon()

1. M. Krämer and M. Scholz, "Rapid calculation of biological effects in ion radiotherapy", *Physics in medicine and biology* **51**, 1959-1970 (2006).

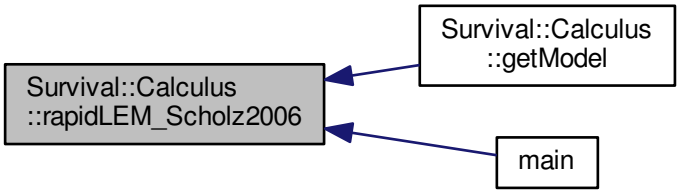
2. M. Zaider and H.H. Rossi, "The synergistic effets of different radiations", *Radiation Research* **160**, 61-69 (2003).

Definition at line 988 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.12 void Calculus::rapidMKM_Attili2013 (double & *alphaIon*, double & *betaIon*)

This method provide a fast original implementation of the MKM model, combining the methods described in (Hawkins_2003) and (Kase_2008).

Whithin this method the non-Poissonian corrective factor introduced in (Hawkins_2003) is exactly evaluated using the track model adopted by (Kase_2008) and performing an explicit integration of the track averaged dose in the cell nucleus.

Within this approach, the estimate for α is obtained as:

$$\alpha = \frac{1 - \exp(-\alpha_P \, \gamma_{nucleus})}{\gamma_{nucleus}}$$

where $\gamma_{nucleus}$ is the dose-weighted average of the specific energy deposited in the nucleus by a single track, evaluated by integrating:

$$\gamma_{nucleus} = \frac{\langle z_n^2 \rangle}{\langle z_n \rangle}$$

where z_n is the specific energy deposited in the single domain in each interaction event, while

$$\alpha_P = \alpha_0 + \beta_0 \, \gamma$$

indicating with α_0 and β_0 the input LQ parameters of the model, which can be identified with the LQ parameter of the reference X-ray irradiation. γ is the dose-weighted average of the dose deposited by a single event in the domain, or:

$$\gamma = \frac{\langle z_d^2 \rangle}{\langle z_d \rangle}$$

where z_d is the specific energy deposited in the single domain in each interaction event. For β no recipe is available. In most of the applications it is assumed to be constant and equal to β_X, even if this contrasts with most of the experimental data. To accounts also for the mixed fields, the method estimates α and β for each tracks of the tracks vector accounting also for the particle weight (Particle::weight). Then α and β are evaluating according to the TDRA:

$$\alpha = \frac{\sum_i \alpha_i \, LET_i}{\sum_i LET_i}$$

$$\sqrt{\beta} = \frac{\sum_i \sqrt{\beta_i} \, LET_i}{\sum_i LET_i}$$

Parameters

alphaIon	The LQ α parameter expressed in Gy^{-1}, passed by reference to be overwritten.
betaIon	The LQ β parameter expressed in Gy^{-2}, passed by reference to be overwritten.

See also

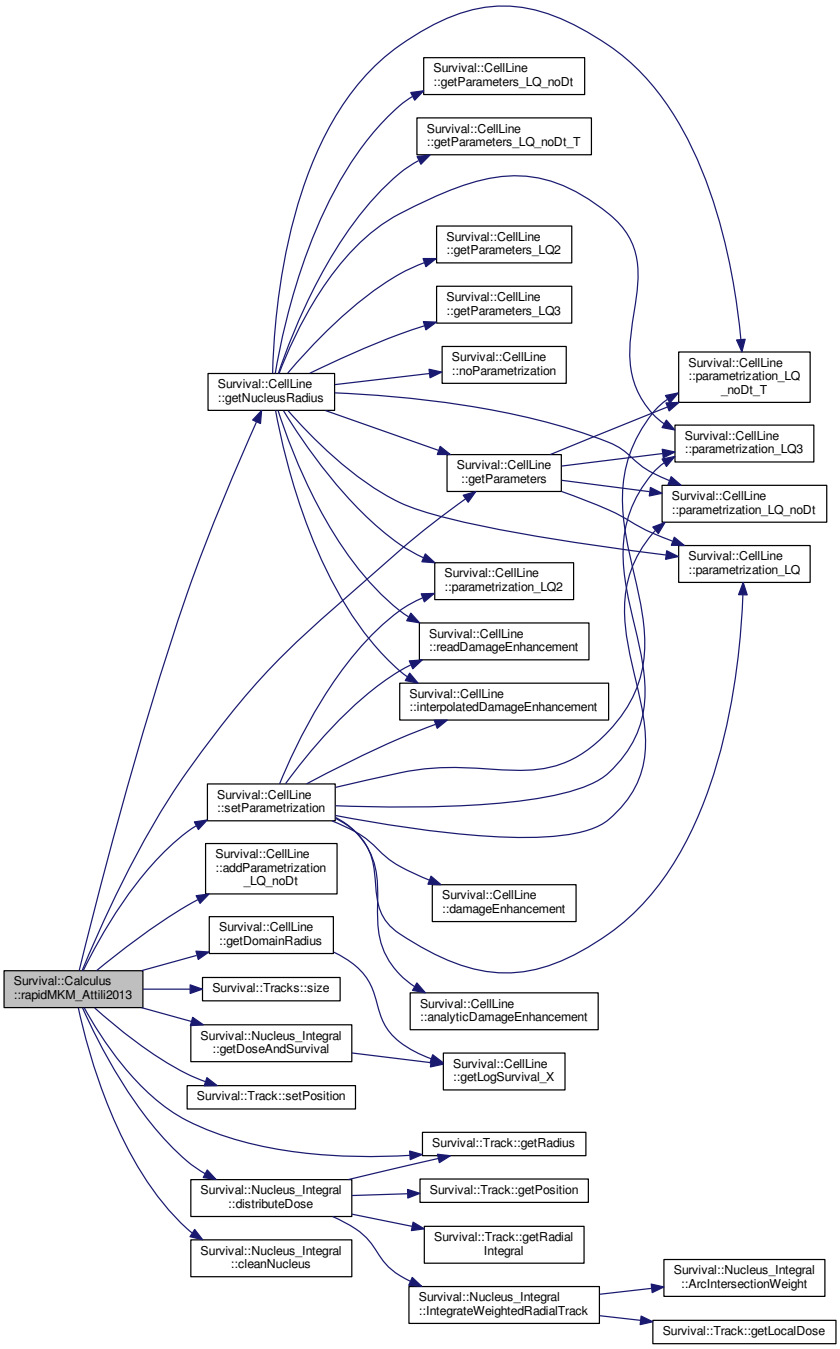
slow_alphaIon_betaIon() and rapidMKM_Kase2008()

Hawkins, R. B. (2003). A microdosimetric-kinetic model for the effect of non-Poisson distribution of lethal lesions on the variation of RBE with LET. *Radiation Research*, 160(1), 61–69.

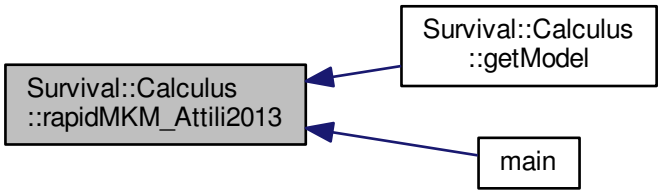
Kase, Y., Kanai, T., Matsufuji, N., Furusawa, Y., Elsässer, T., & Scholz, M. (2008). Biophysical calculation of cell survival probabilities using amorphous track structure models for heavy-ion irradiation. *Physics in Medicine and Biology*, 53(1), 37–59

Definition at line 712 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.13 void Calculus::rapidMKM_Attili2013_corrected_beta (double & *alphalon*, double & *betalon*)

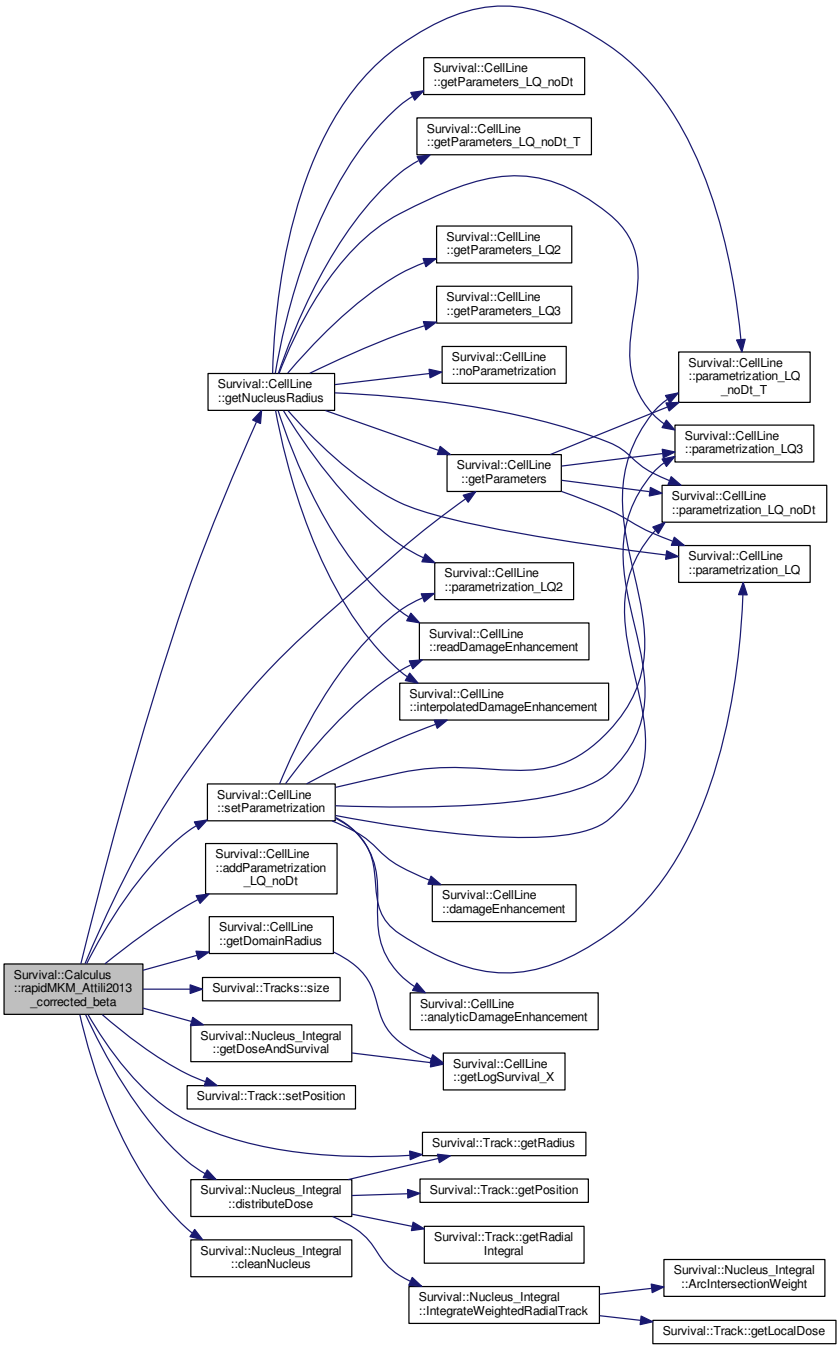
Extension of the rapidMKM_Attili2013() method with $\beta = \beta(\mathrm{LET})$.

This implementation correspond to the implementation of rapidMKM_Attili2013() with the LET-dependent non-poissonian correction factor ($alpha/alpha_P$) applied to the quadratic term (β_0) too:

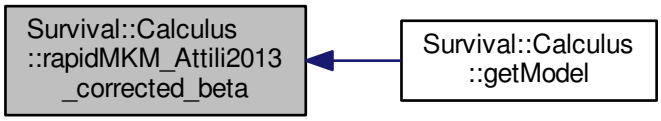
$$\beta = \left(\frac{\alpha}{\alpha_P}\right)^2 \beta_0$$

Definition at line 816 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.14 void Calculus::rapidMKM_Kase2008 (double & *alphalon*, double & *betalon*)

Fast implementation of the MKM as described in (Kase_2008)

Within this approach, the estimate for α is obtained as:

$$\alpha = \frac{1 - \exp(-\alpha_P \, \gamma_{nucleus})}{\gamma_{nucleus}}$$

where $\gamma_{nucleus}$ is the dose-weighted average of the specific energy deposited in the nucleus by a single track, evaluated by means of the approximated formula:

$$\gamma_{nucleus} = \frac{LET}{\rho \, \sigma}$$

while

$$\alpha_P = \alpha_0 + \beta_0 \, \gamma$$

indicating with α_0 and β_0 the input LQ parameters of the MKM, which can be identified with the LQ parameters for X-ray reference irradiation, and with γ the dose-weighted average of the dose deposited by a single event in the domain, or:

$$\gamma = \frac{\langle z_d^2 \rangle}{\langle z_d \rangle}$$

where z_d is the specific energy deposited in the single domain in each interaction event. For β no recipe is available. In this implementation it is assumed to be constant and equal to β_0, even if this contrasts with most of the experimental data. To accounts also for the mixed fields, the method estimates α and β for each tracks of the tracks vector accounting also for the particle weight (Particle::weight). Then α and β are evaluating according to the TDRA:

$$\alpha = \frac{\sum_i \alpha_i \, LET_i}{\sum_i LET_i}$$

$$\sqrt{\beta} = \frac{\sum_i \sqrt{\beta_i} \, LET_i}{\sum_i LET_i}$$

Parameters

alphalon	The LQ α parameter expressed in Gy^{-1}, passed by reference to be overwritten.
betalon	The LQ β parameter expressed in Gy^{-2}, passed by reference to be overwritten.

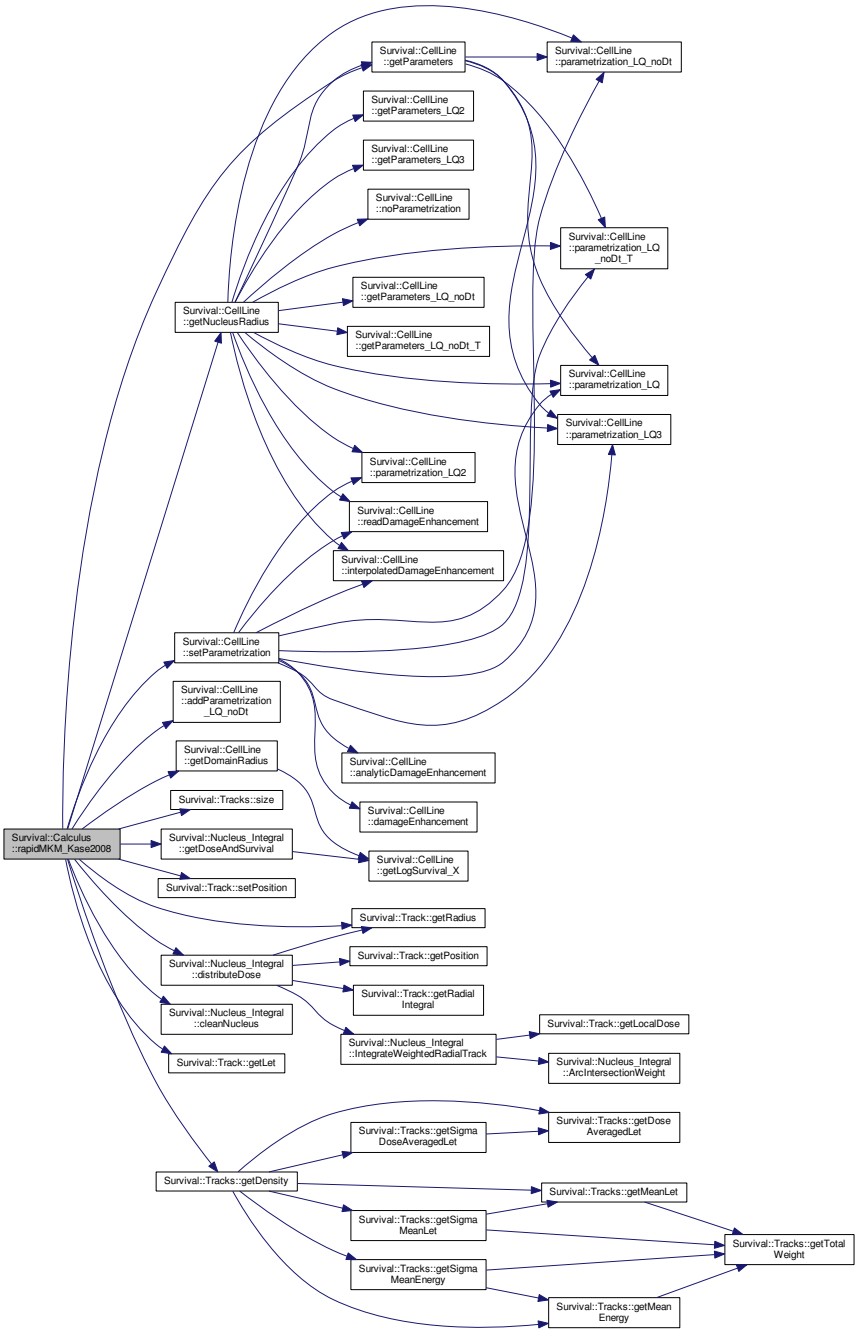
See also

slow_alphaIon_betaIon() and rapidMKM_Attili2013()

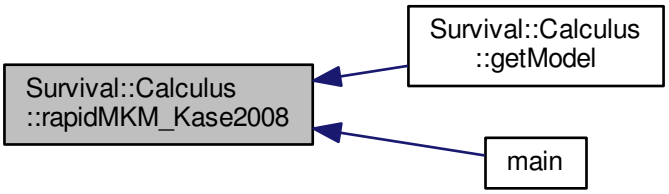
Kase, Y., Kanai, T., Matsufuji, N., Furusawa, Y., Elsässer, T., & Scholz, M. (2008). Biophysical calculation of cell survival probabilities using amorphous track structure models for heavy-ion irradiation. *Physics in Medicine and Biology*, 53(1), 37–59

Definition at line 558 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.15 void Calculus::rapidMKM_Kase2008_corrected_beta (double & *alphalon*, double & *betalon*)

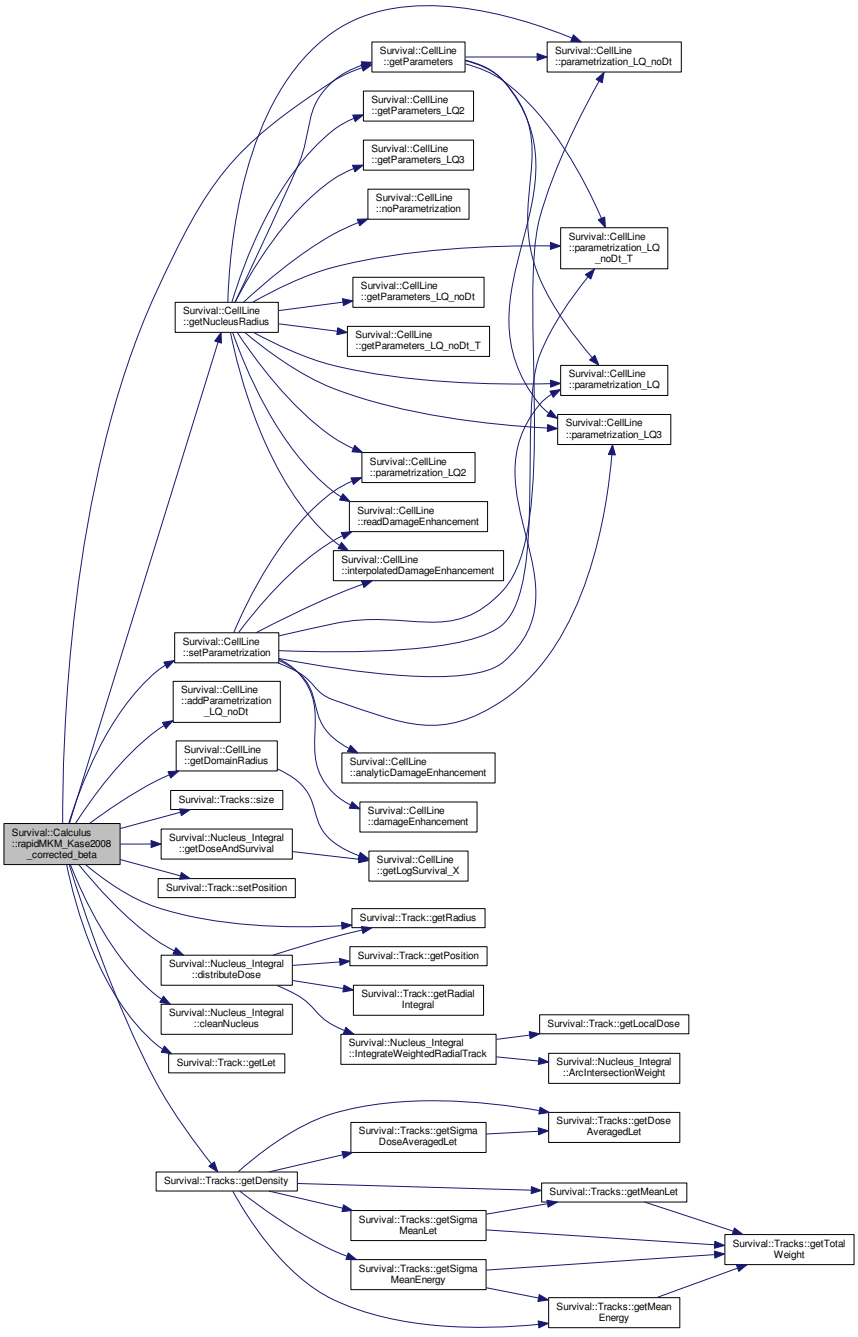
Extension of the rapidMKM_Kase2008() method with $\beta = \beta(\mathrm{LET})$.

This implementation correspond to the implementation of rapidMKM_Kase2008() with the LET-dependent non-poissonian correction factor ($alpha/alpha_P$) applied to the quadratic term (β_0) too:

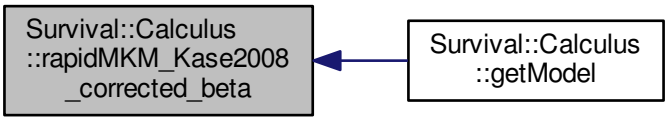
$$\beta = \left(\frac{\alpha}{\alpha_P}\right)^2 \beta_0$$

Definition at line 635 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.16 void Survival::Calculus::setNThreads (int *nTh*) [inline]

Sets the number of threads.

Parameters

nTh	The number of threads to be set.

See also

nThreads

Definition at line 465 of file Calculus.h.

7.2.3.17 void Survival::Calculus::setSavePrefix (std::string *save_prefix*) [inline]

Sets the prefix of the output file name.

Parameters

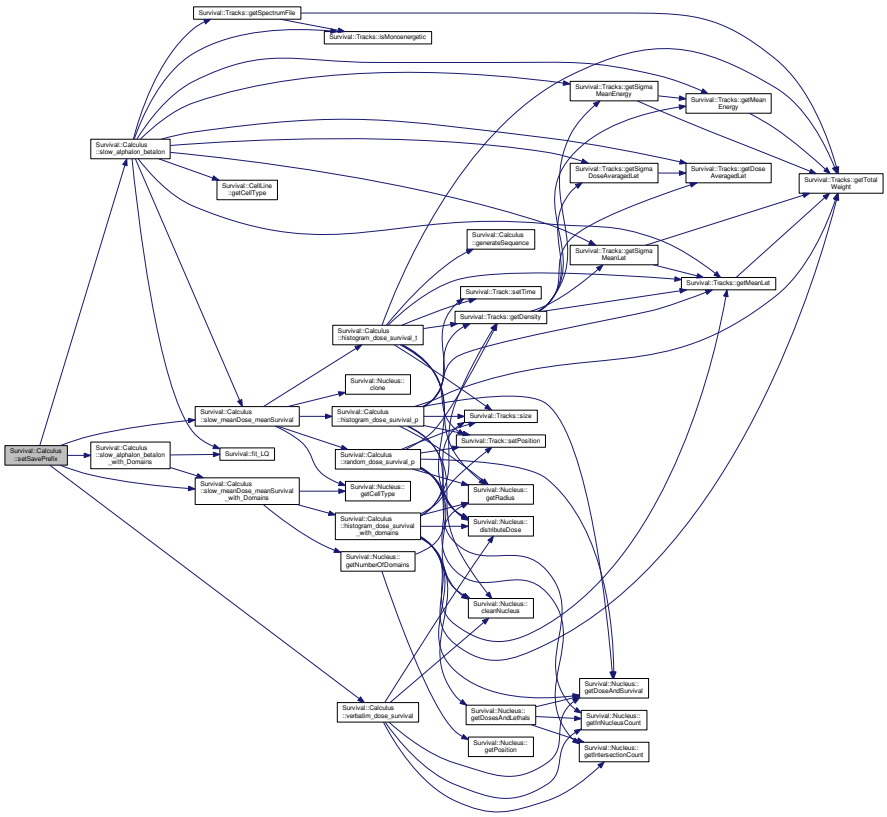
save_prefix	The prefix of the output file name.

See also

savePrefix

Definition at line 473 of file Calculus.h.

Here is the call graph for this function:



7.2.3.18 void Calculus::slow_alphalon_betalon (const std::string *trackMode*, const std::vector< double > *parameters*, const std::vector< double > *dosesImposed*, const double *precision*, double & *alphalon*, double & *alphalonUncertainty*, double & *betalon*, double & *betalonUncertainty*, const int *nFraction*, const double *timeSpacing*, const double *fracDeliveryTime*, const bool *saveAlphaBeta*, const bool *saveMeans*, const bool *saveCell*, const std::string *title_means*)

Method called to perform a Monte Carlo simulation to reproduce the irradiation process getting the LQ parameters α and β.

The idea is to simulate the irradiation process of an entire cell population and to repeat the simulation for different values of dose imposed in order to obtain a complete survival curve. For each value of dose imposed the mean value of dose absorbed and cellular survival observed in the population are obtained by calling the slow_mean↩ Dose_meanSurvival() method. Then the survival curve is fitted by means of the fit_LQ() method in order to get the LQ parameters. Finally the function returns these parameters with the associated uncertainties by overwriting the correspondent variables passed by reference.

Parameters

trackMode	A `string` defining the modality to pass the vector of particles in the mixed fields case. The possibilities are "histogram" or "random".
parameters	The vector containing the model parameters to be used in the simulation (1).
dosesImposed	A vector containing the values of nominal dose to be simulated, expressed in Gy.
precision	Fix the ending condition of the Monte Carlo simulation.
alphaIon	The LQ α parameter expressed in Gy^{-1}, passed by reference to be overwritten.
alphaIonUncertainty	The uncertainty associated to the α parameter (in Gy^{-1}), passed by reference to be overwritten.
betaIon	The LQ β parameter expressed in Gy^{-2}, passed by reference to be overwritten.
betaIonUncertainty	The uncertainty associated to the β parameter (in Gy^{-2}), passed by reference to be overwritten.
nFraction	The total number of fraction, in case of fractionated treatment.
timeSpacing	The time spacing between fractions, expressed in hours.
fracDeliveryTime	The delivery time of each fraction, expressed in hours.
saveAlphaBeta	A boolean parameter indicating if the extrapolated α and β parameters are to be saved.
saveMeans	A boolean parameter indicating if the informations on mean dose and survival observed are to be saved.
saveCell	A boolean parameter indicating if the dose-survival data of each cell irradiated are to be saved.
title_means	The title of the file where the method will save the informations on mean dose and survival resultin from the simulation.

Warning

The execution of the program will be terminated if the minimum dose imposed is greater than the maximum one or if an inexistent track mode is selected.

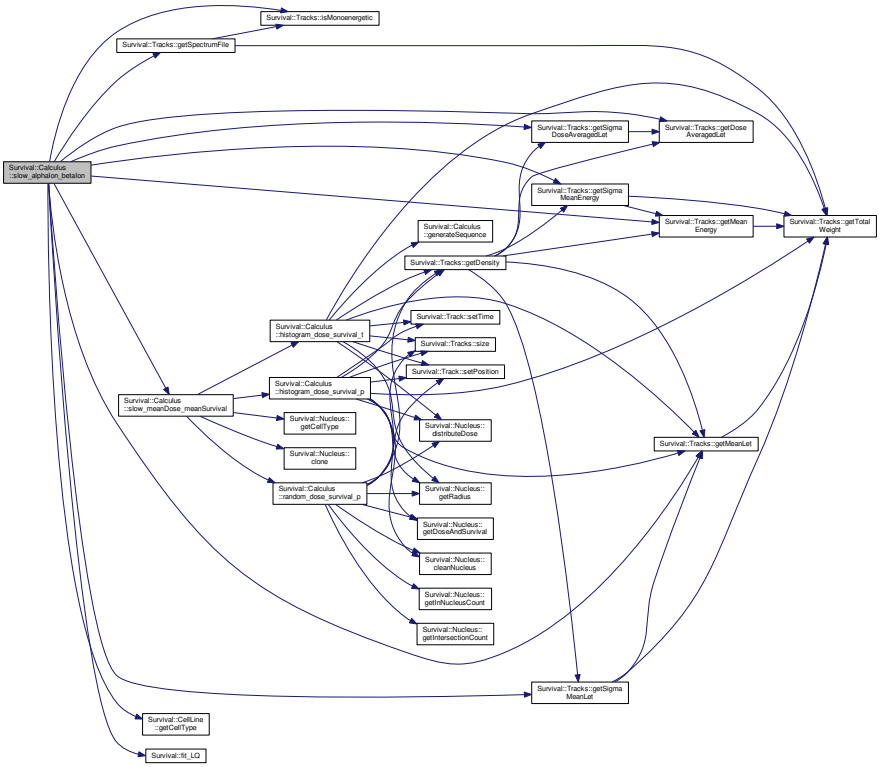
See also

slow_alphaIon_betaIon_with_Domains() and histogram_dose_survival()

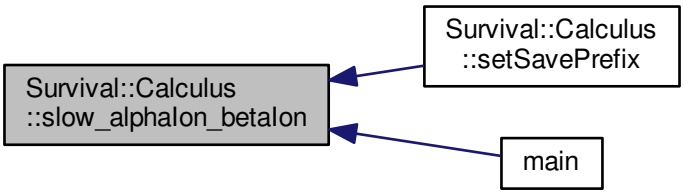
1. These parameters are stored in the CellLine object but the way to get them is a little bit tricky, hence this is an easier and not aestethically perfect way to get these informations. This has to be fixed in the next versions of the program.

Definition at line 1108 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.19 void Calculus::slow_alphaIon_betaIon_with_Domains (const std::string *trackMode*, const double *minDose*, const double *maxDose*, const int *numberOfDoses*, const double *precision*, double & *alphaIon*, double & *alphaIonUncertainty*, double & *betaIon*, double & *betaIonUncertainty*)

This function was thought to control the dose delivery inside the MKM nucleus. It's similar to slow_alphaIon_beta↩Ion() but provide also informations on the microscopic dose deposition pattern in each domain of the nucleus.

The idea is to simulate the irradiation process of an entire cell population and to repeat the simulation for different values of dose imposed in order to obtain a complete survival curve. For each value of dose imposed the mean

value of dose absorbed and cellular survival observed in the population are obtained by calling the slow_mean↩ Dose_meanSurvival_with_Domains() method. Then the survival curve is fitted by means of the fit_LQ() method in order to get the LQ parameters. Finally the function returns these parameters with the associated uncertainties by overwriting the correspondent variables passed by reference.

Parameters

trackMode	Defined for completeness. Only "histogram" is supported.
minDose	The minimum value of dose imposed expressed in Gy.
maxDose	The maximum value of dose imposed expressed in Gy.
numberOfDoses	The total number of doses imposed (together with `minDose` and `maxDose` this allow to define the sequence).
precision	Fix the ending condition of the Monte Carlo simulation.
alphaIon	The LQ α parameter expressed in Gy^{-1}, passed by reference to be overwritten.
alphaIonUncertainty	The uncertainty associated to the α parameter (in Gy^{-1}), passed by reference to be overwritten.
betaIon	The LQ β parameter expressed in Gy^{-2}, passed by reference to be overwritten.
betaIonUncertainty	The uncertainty associated to the β parameter (in Gy^{-2}), passed by reference to be overwritten.

Warning

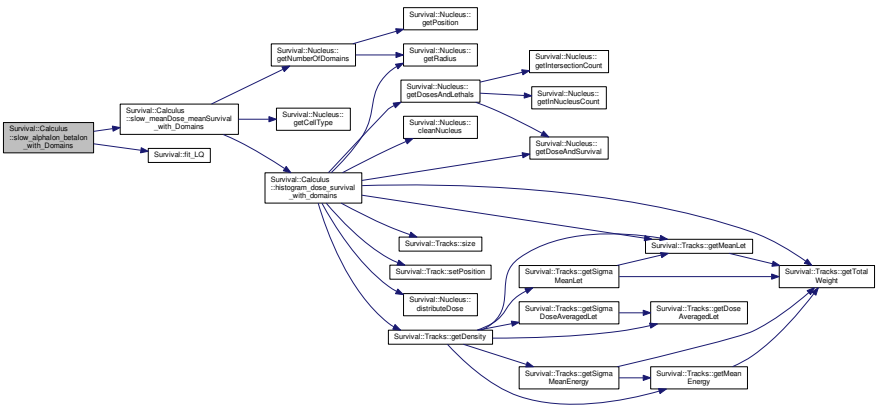
The execution of the program will be terminated if the minimum dose imposed is greater than the maximum one or if an inexistent track mode is selected.
Even if not explicitly required, the nucleus has to be a Nucleus_MKM object.

See also

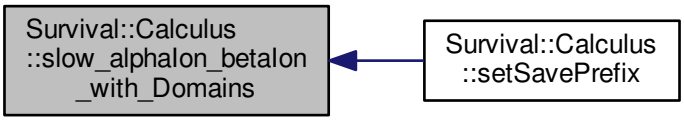
slow_alphaIon_betaIon() and histogram_dose_survival_with_domains()

Definition at line 1197 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.20 void Calculus::slow_meanDose_meanSurvival (const std::string *trackMode,* const double *doseImposed,* const double *precision,* double & *meanDose,* double & *meanDoseUncertainty,* double & *meanSurvival,* double & *meanSurvivalUncertainty,* const int *nFraction,* const double *timeSpacing,* const double *fracDeliveryTime,* const bool *saveCell*)

Perform the Monte Carlo simulation relatively to a single value of dose imposed and returns the mean values of dose absorbed and survival observed in the cell population.

The user has to fix the precision required for the simulation, that is the statistics to be reached to terminate the simulation. Two possibilities are supported, so the user can:

- Fix the number of iterations, hence the `precision` has to be an integer value greater (or at least equal) to 1.
- Define a constraint on the precision to reach in the simulation in the evaluation of the cell survival (precisely the relative error on the survival), hence the `precision` has to be a `double` in (0, 1). Once set the precision desired, the irradiation process of the cell population is simulated by calling the histogram_dose↩ _survival_p() or the random_dose_survival_p() methods in a parallel loop (or histogram_dose_survival_t() in the case of temporal studies), depending on the trackMode selected (see below). Each thread performs the irradiation of a single cell and at the end of each irradiation the method updates the total mean dose and survival observed, with associated uncertainties, and determines if the precision set by the user is reached or not. All the values of dose and survival obtained, cell by cell, are saved in an output file.

Note

Survival uncertainty: the estimation of the survival uncertainty is knowingly biased, but extremely efficient and memory-friendly. Note that this quantity is used only to decide when it is possible to stop the simulation hence it doesn't affect the result (and that the bias decrease with increasing iteration).
track mode: this is useful in the case of mixed fields, that is when the field is constituted by different particles. In that case the there are different ways to extract from the tracks vector the particle to be used event by event:

- If "histogram" is selected then the tracks vector is interpreted as an histogram where each particle has a specific weight, defined by its frequency in the histogram. Different particle, in this case, are not necessary equiprobable.
- If "random" is selected then the different tracks in the tracks vector are considered equiprobable and a random extraction between them is performed event by event.

Parameters

trackMode	A `string` defining the modality to pass the vector of particles in the mixed fields case. The possibilities are "histogram" or "random".
doseImposed	The value of dose to be delivered in the irradiation, expressed in Gy.
precision	The precision to be reached in the simulation (could be a fixed number of iterations or a constraint on the survival precision).
meanDose	The mean value of dose absorbed expressed in Gy, passed by reference to be overwritten.
meanDoseUncertainty	The uncertainty on meanDose, expressed in Gy, passed by reference to be overwritten.
meanSurvival	The mean value of cellular survival observed, passed by reference to be overwritten.
meanSurvivalUncertainty	The uncertainty associated to the mean survival, passed by reference to be overwritten.
nFraction	The number of fraction in the case of fractionated treatment.
timeSpacing	The time spacing between fractions, expressed in hours.
fracDeliveryTime	The delivery time of each fraction, expressed in hours.
saveCell	A boolean parameter indicating if the dose-survival data of each cell irradiated are to be saved or not.

Warning

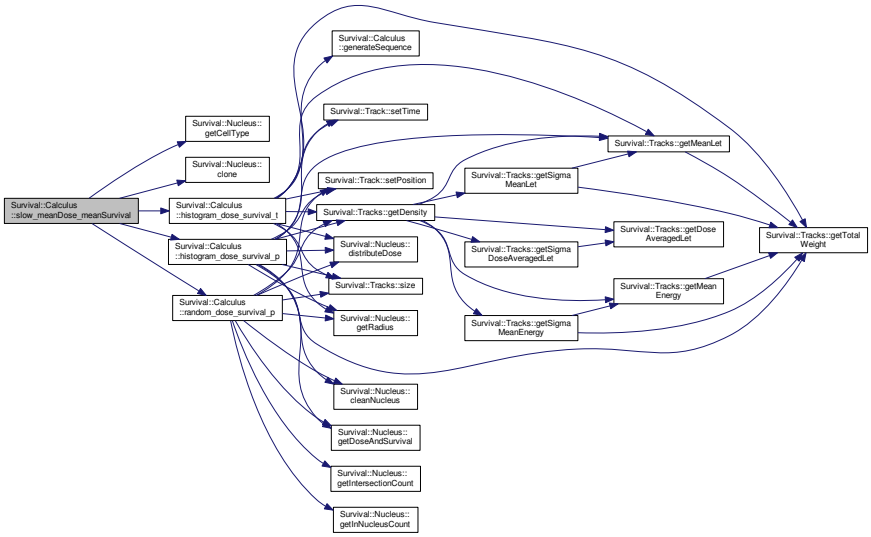
The execution of the program will be terminated if the precision is not set correctly or if an inexistent track mode is selected.

See also

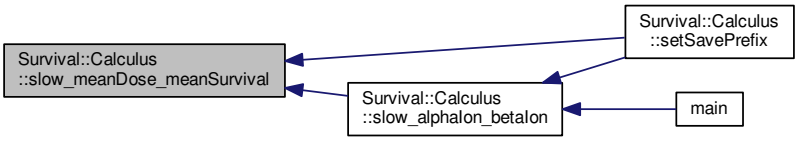
slow_alphaIon_betaIon()

Definition at line 1250 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.21 void Calculus::slow_meanDose_meanSurvival_with_Domains (const std::string *trackMode,* const double *doseImposed,* const double *precision,* double & *meanDose,* double & *meanDoseUncertainty,* double & *meanSurvival,* double & *meanSurvivalUncertainty*)

Perform the Monte Carlo simulation relatively to a single value of dose imposed and returns the mean values of dose absorbed and survival observed in the cell population. This function was thought to control the dose delivery inside the MKM nucleus. It's similar to slow_meanDose_meanSurvival() but provide also informations on the microscopic dose deposition pattern in each domain of the nucleus.

The user has to fix the precision required for the simulation, that is the statistics to be reached to terminate the simulation. Two possibilities are supported, so the user can:

- Fix the number of iterations, hence the `precision` has to be an integer value greater (or at least equal) to 1.
- Define a constraint on the precision to reach in the simulation in the evaluation of the cell survival (precisely the relative error on the survival), hence the `precision` has to be a `double` in (0, 1). Once set the precision desired, the irradiation process of the cell population is simulated by calling the histogram_dose_survival_with_domains() method in a loop that, in each iteration, performs the irradiation of a single cell. At the end of the irradiation, the method updates the total mean dose and survival observed, with associated uncertainties, and determines if the precision set by the user is reached or not. All the values of dose and survival obtained, cell by cell, are saved in an output file together with the total number of lethal events observed in the nucleus.

Note

Survival uncertainty: the estimation of the survival uncertainty is knowingly biased, but extremely efficient and memory-friendly. Note that this quantity is used only to decide when it is possible to stop the simulation hence it doesn't affect the result (and that the bias decrease with increasing iteration).

Parameters

trackMode	Defined for completeness. Only "histogram" is supported.
doseImposed	The value of dose to be delivered in the irradiation, expressed in Gy.
precision	The precision to be reached in the simulation (could be a fixed number of iterations or a constraint on the survival precision).
meanDose	The mean value of dose absorbed expressed in Gy, passed by reference to be overwritten.
meanDoseUncertainty	The uncertainty on meanDose, expressed in Gy, passed by reference to be overwritten.
meanSurvival	The mean value of cellular survival observed, passed by reference to be overwritten.
meanSurvivalUncertainty	The uncertainty associated to the mean survival, passed by reference to be overwritten.

Warning

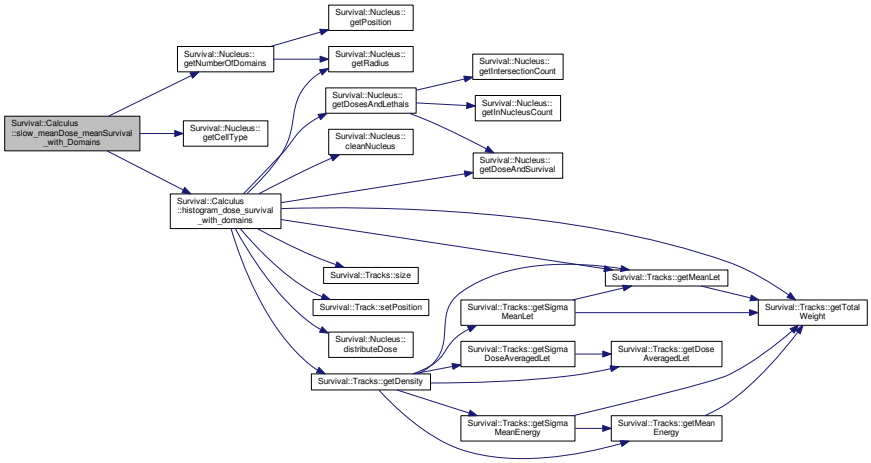
The execution of the program will be terminated if the precision is not set correctly or if an inexistent track mode is selected.

See also

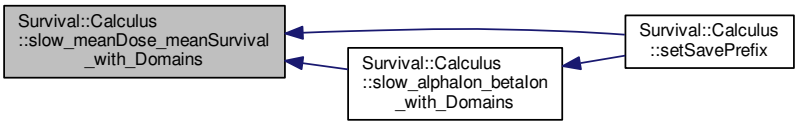
slow_alphaIon_betaIon() and slow_alphaIon_betaIon_with_Domains()

Definition at line 1409 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.22 void Calculus::verbatim_dose_survival (double & *dose*, double & *doseUncertainty*, double & *survival*, double & *survivalUncertainty*, bool *clean* = `true`)

Evaluates the dose deposited in the nucleus using directly the tracks vector without modifying it and without random numbers extractions.

It simply calls the Nucleus::distributeDose(const Tracks) method passing the tracks vector and then it gets informations on dose deposited and survival observed, with associated uncertainties, via the Nucleus::getDoseAnd↩Survival() method.

Parameters

dose	The dose absorbed by the nucleus in the irradiation, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival observed, passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.
clean	A boolean value indicating if the nucleus has to be cleaned at the end of the evaluation.

Note

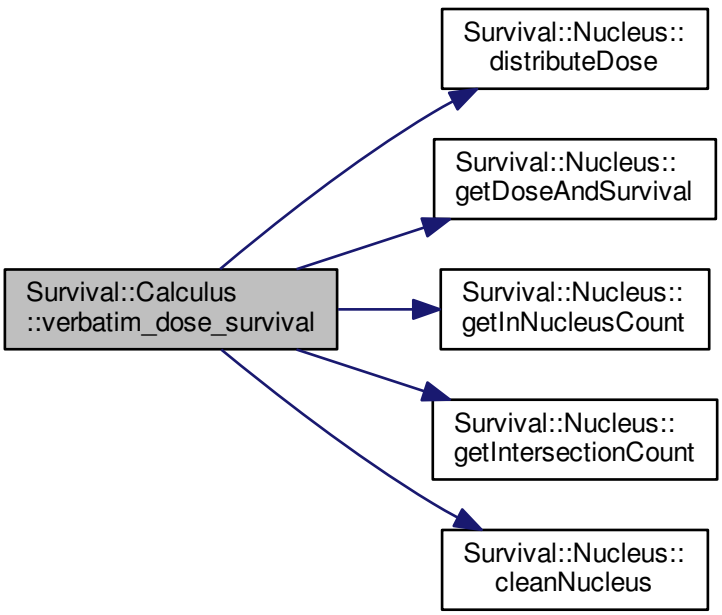
It's actually unused.

See also

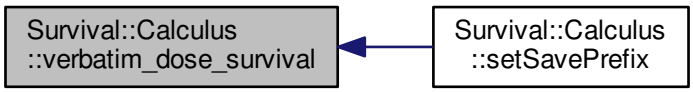
histogram_dose_survival_p() and random_dose_survival_p()

Definition at line 1509 of file Calculus.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.4 Member Data Documentation

7.2.4.1 const CellLine& Survival::Calculus::cellLine [private]

A `const` reference to a CellLine object corresponding to the cell line to which the nucleus belongs.

Definition at line 644 of file Calculus.h.

7.2.4.2 std::string Survival::Calculus::model [private]

The model used in the simulation.

Definition at line 665 of file Calculus.h.

7.2.4.3 int Survival::Calculus::nThreads [private]

The number of threads needed to be used in the simulation (if parallelism is supported).

Possible cases are:

- 0: Uses a number of threads corresponding to the number of core of the machine executing this program.
- 1: Uses 1 threads, i.e. Disabled multithread
- A number greater than 1: Specifies the exact number of threads.

Definition at line 656 of file Calculus.h.

7.2.4.4 Nucleus& Survival::Calculus::nucleus [private]

A reference to the cellular nucleus.

Definition at line 647 of file Calculus.h.

7.2.4.5 gsl_rng* Survival::Calculus::randomGenerator [private]

A pointer to a gsl_rng object, useful in the generation of pseudorandom numbers in the Monte Carlo simulation.

Definition at line 659 of file Calculus.h.

7.2.4.6 std::string Survival::Calculus::savePrefix [private]

The prefix of the output file.

Definition at line 662 of file Calculus.h.

7.2.4.7 const Tracks& Survival::Calculus::tracks [private]

A `const` reference to a Track object corresponding to the Particle interacting with nucleus.

Definition at line 641 of file Calculus.h.

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

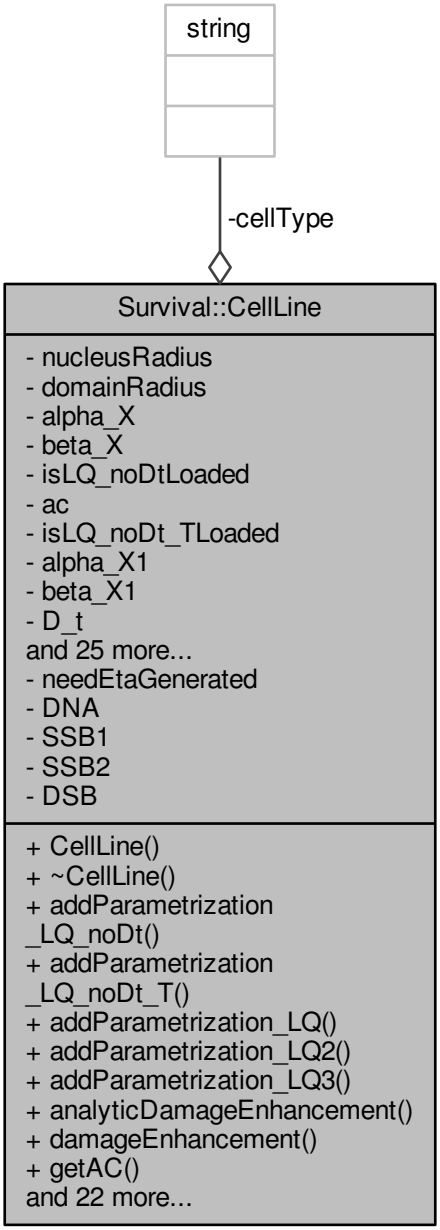
- include/Calculus.h
- src/Calculus.cpp

7.3 Survival::CellLine Class Reference

Represents the cell line to which the nucleus belongs and hosts the radiobiological and structural characteristics of the cell and the different parametrizations of the models.

`#include <CellLine.h>`

Collaboration diagram for Survival::CellLine:



Public Member Functions

- CellLine (const std::string cell_type, const double nucleus_radius=10.0, const double domain_radius=10.0)

  Constructor. Instantiates and sets the object.
- ~CellLine ()

  Destructor.
- void addParametrization_LQ_noDt (const double alphaX, const double betaX)

Adds the LQ_noDt parametrization to the cell line (used in the MKM model) for the evaluation of the cellular survival.

- void addParametrization_LQ_noDt_T (const double alphaX, const double betaX, const double ac_=2.187)

  Adds the LQ_noDt_T parametrization to the cell line (used in the MCt-MKM model) for the evaluation of the cellular survival.
- void addParametrization_LQ (const double alphaX, const double betaX, const double Dt)

  Adds the LEM I parametrization of the survival to the cell line.
- void addParametrization_LQ2 (const double alphaX, const double betaX, const double Dt2, const double genome_Length, const double alphaSSB=1250.0, const double alphaDSB=30.0, long int basePairs=25)

  Adds the LEM II parametrization of the survival to the cell line.
- void addParametrization_LQ3 (const double alphaX, const double betaX, const double Dt3, const double genome_Length, const double alphaSSB=1250.0, const double alphaDSB=30.0, long int basePairs=25)

  Adds the LEM III parametrization of the survival to the cell line.
- double analyticDamageEnhancement (const double dose) const

  Evaluate the damage enhancement factor by means of an analytic approximated expression.
- double damageEnhancement (const double dose) const

  Evaluate the damage enhancement factor corresponding to a certain dose absorbed via a Monte Carlo simulation.
- double getAC () const

  Returns the time constant associated to the repair kinetics of the nucleus, expressed in h^{-1}.
- std::string getCellType () const

  Returns a string identifying the name of the cell line.
- double getDomainRadius () const

  Returns the radius of the domain expressed in um.
- double getLogSurvival_X (const double dose) const

  Returns the natural logarithm of the cellular survival evaluated on the basis of the selected parametrization.
- double getLogSurvival_X (const std::vector< double >doses, const std::vector< double >times) const

  Overload. Returns the natural logarithm of the cellular survival taking into account the time structure of the irradiation.
- double getNucleusRadius () const

  Returns the radius of the nucleus expressed in um.
- void getParameters (double &returnAlpha_X, double &returnBeta_X, double &returnD_t) const

  Returns the linear quadratic α and β parameters and the transition dose, corresponding to the selected parametrization, by overwriting three `double` variables passed by reference.
- void getParameters_LQ_noDt (double &returnAlpha_X, double &returnBeta_X) const

  Returns the linear quadratic α and β parameters corresponding to the "LQ_noDt" parametrization by overwriting two `double` variables passed by reference.
- void getParameters_LQ_noDt_T (double &returnAlpha_X, double &returnBeta_X, double &ac_) const

  Returns the linear quadratic α and β parameters corresponding to the "LQ_noDt" parametrization and the time constant by overwriting three `double` variables passed by reference.
- void getParameters_LQ2 (double &returnAlpha_X2, double &returnBeta_X2, double &returnD_t2, double &returnGenomeLength, double &returnAlpha_SSB, double &returnAlpha_DSB, long int &returnBase_Pairs) const

  Returns the parameters characteristic of the LQ2 parametrization (used in the LEM II formulation) overwriting some variables passed by reference.
- void getParameters_LQ3 (double &returnAlpha_X3, double &returnBeta_X3, double &returnD_t3, double &returnGenomeLength, double &returnAlpha_SSB, double &returnAlpha_DSB, long int &returnBase_Pairs) const

  Returns the parameters characteristic of the LQ3 parametrization (used in the LEM III formulation) overwriting some variables passed by reference.
- double interpolatedDamageEnhancement (const double dose) const

  Get the value of the damage enhancement factor from the precalculated curve (doseForEta, etaPre) interpolating the nearest neighbors of the dose imposed.
- double noParametrization (const double dummy) const

  If called it interrupts the execution of the program, because no parametrizazion is selected.
- double noParametrization (const std::vector< double >v1, const std::vector< double >v2) const

If called it interrupts the execution of the program, because no parametrizazion is selected.

- double parametrization_LQ_noDt (const double dose) const

  Returns the logarithmic cellular survival corresponding to a particular dose absorbed. Implements the parametrization used in the MKM formulation.
- double parametrization_LQ_noDt_T (const std::vector< double > doses, const std::vector< double > times) const

  Returns the logarithmic cellular survival associated to a sequence of doses absorbed with a specific time structure. Implements the parametrization used in the MCt-MKM formulation.
- double parametrization_LQ (const double dose) const

  Returns the logarithmic cellular survival corresponding to a particular dose absorbed. Implements the parametrization used in the LEM I formulation.
- double parametrization_LQ2 (const double dose) const

  Returns the logarithmic cellular survival corresponding to a particular dose absorbed. Implements the parametrization used in the LEM II formulation.
- double parametrization_LQ3 (const double dose) const

  Returns the logarithmic cellular survival corresponding to a particular dose absorbed. Implements the parametrization used in the LEM II formulation.
- double readDamageEnhancement (const double dose) const

  Reads and returns the value of the damage enhancement factor correspondent to the required dose from an external file.
- void setDomainRadius (double domainRadius_)

  Sets the radius of the domain relative to the MKM parametrization of the nucleus.
- void setNucleusRadius (double nucleusRadius_)

  Sets the radius of the nucleus.
- void setParametrization (const std::string parametrization_type)

  Sets an X-ray parametrization for the evaluation of the cellular survival.

Private Attributes

- std::string cellType

  A `string` identifying the name of the cell line.
- double nucleusRadius

  The radius of the nucleus characteristic for the cell line, expressed in um.
- double domainRadius

  The radius of the domain associated to the MKM parametrization of the nucleus, expressed in um.
- double alpha_X

  The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.
- double beta_X

  The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.
- bool isLQ_noDtLoaded

  A boolean value identifying if the "LQ_noDt" parametrization is selected.
- double ac

  The time constant associated to the repair kinetics of the cell, expressed in h^{-1}.
- bool isLQ_noDt_TLoaded

  A boolean value identifying if the "LQ_noDt_T" parametrization is selected.
- double alpha_X1

  The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.
- double beta_X1

  The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.
- double D_t

  The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.

- double s

  The coefficient of the exponential tail: $s = \alpha + 2\beta D_t$.
- double logS_t

  The logarithmic survival associated to a dose absorbed D_t, evaluated according to the standard linear quadratic parametrization.
- bool isLQloaded

  A boolean value identifying if the "LQ" parametrization is selected.
- double alpha_X2

  The linear quadratic α*-parameter characteristic for X-rays, expressed in* Gy^{-1}.
- double beta_X2

  The linear quadratic β*-parameter characteristic for X-rays, expressed in* Gy^{-2}.
- double D_t2

  The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.
- double s2

  The coefficient of the exponential tail: $s = \alpha + 2\beta D_t$.
- double logS_t2

  The logarithmic survival associated to a dose absorbed D_t, evaluated according to the standard linear quadratic parametrization.
- double genomeLength

  The genome length expressed in number of base pairs.
- double alpha_SSB

  The number of SSBs directly produced by the irradiation in the whole genome per unit of dose absorbed.
- double alpha_DSB

  The number of DSBs directly produced by the irradiation in the whole genome per unit of dose absorbed.
- long int base_Pairs

  The distance (in unit of based pairs) between two SSBs resulting in a DSB.
- bool isLQ2loaded

  A boolean value identifying if one the "LQ2" parametrizations is selected.
- double alpha_X3

  The linear quadratic α*-parameter characteristic for X-rays, expressed in* Gy^{-1}.
- double beta_X3

  The linear quadratic β*-parameter characteristic for X-rays, expressed in* Gy^{-2}.
- double D_t3

  The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.
- double s3

  The coefficient of the exponential tail: $s = \alpha + 2\beta D_t$.
- double logS_t3

  The logarithmic survival associated to a dose absorbed D_t, evaluated according to the standard linear quadratic parametrization.
- bool isLQ3loaded

  A boolean value identifying if the "LQ3" parametrization is selected.
- double(CellLine::∗ selectedDamageEnhancement)(const double dose) const

  A pointer to functions that identifies the selected way to evaluate the enhancement factor $\eta(D)$ *in LEM II and III formulations.*
- double(CellLine::∗ selectedEtaGeneration)(const double dose) const

  A pointer to functions that identifies the selected way to evaluate the enhancement factor $\eta(D)$ *in LEM II and III formulations.*
- double(CellLine::∗ selectedParametrization)(const double dose) const

  A pointer to functions that identifies the selected parametrization.
- double(CellLine::∗ selectedParametrizationT)(const std::vector< double >doses, const std::vector< double >times) const

A pointer to functions that identifies the selected parametrization when the temporal effects of the irradiation are taken into account.

- double doseForEta [200]

  An array used to store the values of dose for to precalculate the enhancement factor curve.
- double etaPre [200]

  An array containing the precalculated values of the enhancement factor as a function of the dose absorbed.

Static Private Attributes

- static bool needEtaGenerated = false

  A boolean data member that indicates if the selected parametrization requires the generation of the enhancement factor.
- static int DNA [10000000]

  An array representing the genome of the cell.
- static bool SSB1 [10000000]

  A boolean array representing the single strand breaks (SSB) in the first strand.
- static bool SSB2 [10000000]

  A boolean array representing the single strand breaks (SSB) in the second strand.
- static bool DSB [10000000]

  A boolean array representing the double strand breaks (DSB) in the genome.

7.3.1 Detailed Description

Represents the cell line to which the nucleus belongs and hosts the radiobiological and structural characteristics of the cell and the different parametrizations of the models.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2011–2015

Besides its hosting function, this class computes the local number of lethal events corresponding to a local dose deposition by means of the LEM I, LEM II, LEM III and LQ survival parametrizations for X-rays irradiation. Different parametrizations can be contemporary loaded in a CellLine object; the parametrization in use is specified via the public method setParametrization(). The evaluation of the clustering damage enhancement of LEM II and LEM III can be performed in several ways: it can be fully generated on the fly via Monte Carlo, loaded from an external file or generated using an approximated analytical expression.

Definition at line 19 of file CellLine.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 CellLine::CellLine (const std::string *cell_type*, const double *nucleus_radius* = `10.0`, const double *domain_radius* = `10.0`)

Constructor. Instantiates and sets the object.

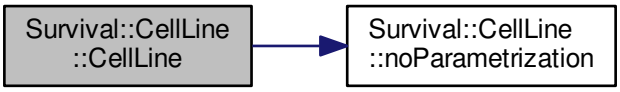
Each numeric data member is set to zero by default, with the exception of the parameters of the constructor. The selected parametrization is set to "noParametrization".

Parameters

cell_type	A `string` identifying the name of the cell line.
nucleus_radius	The radius of the nucleus characteristic for the cell line, expressed in um (default 10 um).
domain_radius	The radius of the domain associated to the MKM parametrization of the nucleus, expressed in um (default 10 um).

Definition at line 31 of file CellLine.cpp.

Here is the call graph for this function:

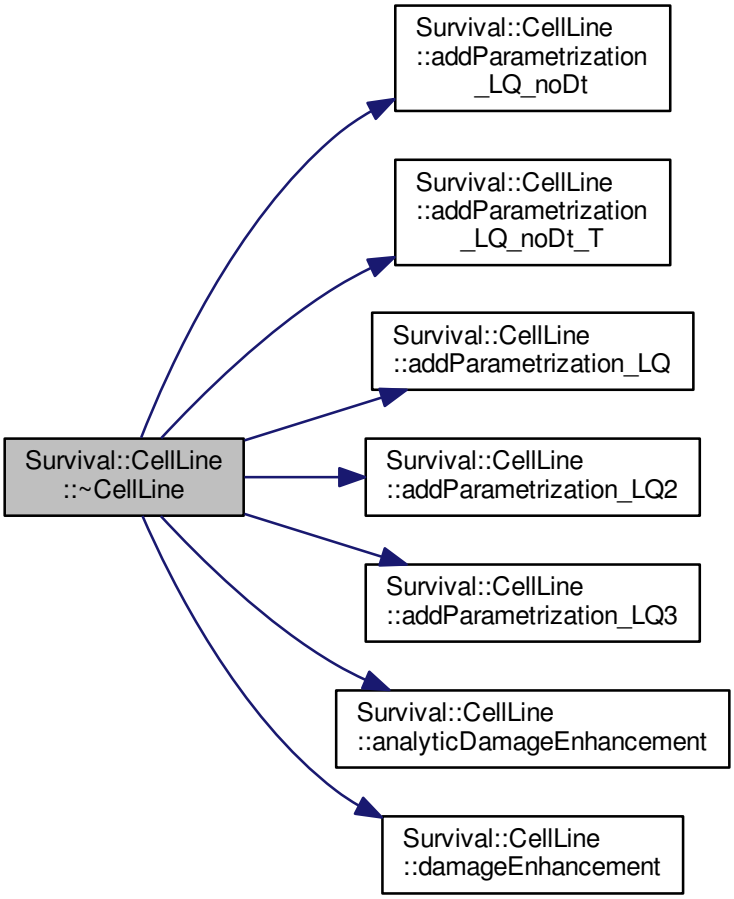


7.3.2.2 Survival::CellLine::~CellLine ( ) [inline]

Destructor.

Definition at line 36 of file CellLine.h.

Here is the call graph for this function:



7.3.3 Member Function Documentation

7.3.3.1 void CellLine::addParametrization_LQ (const double *alphaX,* const double *betaX,* const double *Dt*)

Adds the LEM I parametrization of the survival to the cell line.

Parameters

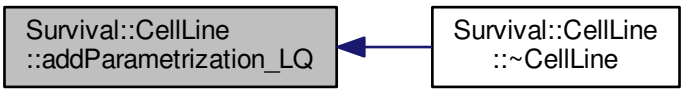
alphaX	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.
betaX	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.
Dt	The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.

See also

parametrization_LQ()

Definition at line 97 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.2 void CellLine::addParametrization_LQ2 (const double *alphaX*, const double *betaX*, const double *Dt2*, const double *genome_Length*, const double *alphaSSB* = 1250.0, const double *alphaDSB* = 30.0, long int *basePairs* = 25)

Adds the LEM II parametrization of the survival to the cell line.

Parameters

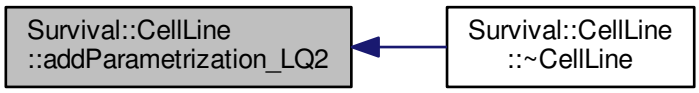
alphaX	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.
betaX	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.
Dt2	The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.
genome_Length	The genome length expressed in unit of base pairs (genomeLength).
alphaSSB	The number of SSBs directly produced by the irradiation in the whole genome per unit of dose absorbed (alpha_SSB).
alphaDSB	The number of DSBs directly produced by the irradiation in the whole genome per unit of dose absorbed (alpha_DSB).
basePairs	The distance (in number of based pairs) between two SSBs resulting in a DSB (base_Pairs).

See also

parametrization_LQ2()

Definition at line 115 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.3 void CellLine::addParametrization_LQ3 (const double *alphaX*, const double *betaX*, const double *Dt3*, const double *genome_Length*, const double *alphaSSB* = 1250.0, const double *alphaDSB* = 30.0, long int *basePairs* = 25)

Adds the LEM III parametrization of the survival to the cell line.

Parameters

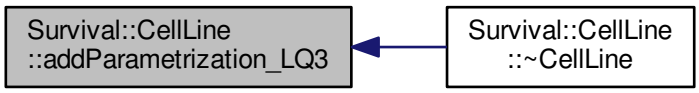
alphaX	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.
betaX	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.
Dt3	The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.
genome_Length	The genome length expressed in unit of base pairs (genomeLength).
alphaSSB	The number of SSBs directly produced by the irradiation in the whole genome per unit of dose absorbed (alpha_SSB).
alphaDSB	The number of DSBs directly produced by the irradiation in the whole genome per unit of dose absorbed (alpha_DSB).
basePairs	The distance (in number of based pairs) between two SSBs resulting in a DSB (base_Pairs).

See also

parametrization_LQ3()

Definition at line 141 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.4 void CellLine::addParametrization_LQ_noDt (const double *alphaX*, const double *betaX*)

Adds the LQ_noDt parametrization to the cell line (used in the MKM model) for the evaluation of the cellular survival.

Sets alpha_X and beta_X to the passed values.

Parameters

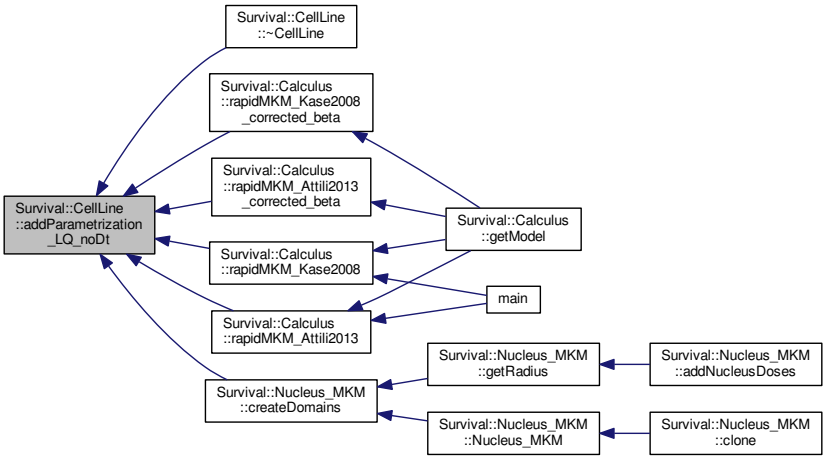
alphaX	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.
betaX	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.

See also

parametrization_LQ_noDt()

Definition at line 71 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.5 void CellLine::addParametrization_LQ_noDt_T (const double *alphaX*, const double *betaX*, const double *ac_* = 2.187)

Adds the LQ_noDt_T parametrization to the cell line (used in the MCt-MKM model) for the evaluation of the cellular survival.

Sets alpha_X, beta_X and ac to the passed values.

Parameters

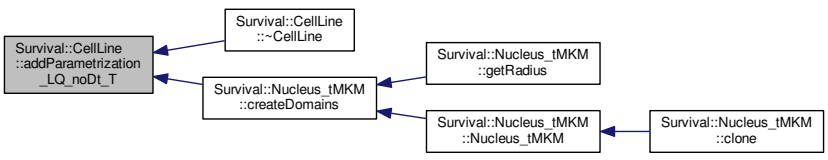
alphaX	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.
betaX	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.
*ac_*	The time constant associated to the repair kinetics of the nucleus, expressed in h^{-1}.

See also

parametrization_LQ_noDt_T()

Definition at line 83 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.6 double CellLine::analyticDamageEnhancement (const double *dose*) const

Evaluate the damage enhancement factor by means of an analytic approximated expression.

The damage enhancement factor is evaluated by means of the expression:

$$\langle \eta(D) \rangle = 1 + \sum_{n=1}^{h} \frac{\exp(-(\tilde{\alpha}_{SSB} + \tilde{\alpha}_{DSB})D)}{\tilde{\alpha}_{DSB}\, D} \frac{(\tilde{\alpha}_{SSB}\, D)^n (1 - 2^{(1-n)})}{n!}$$

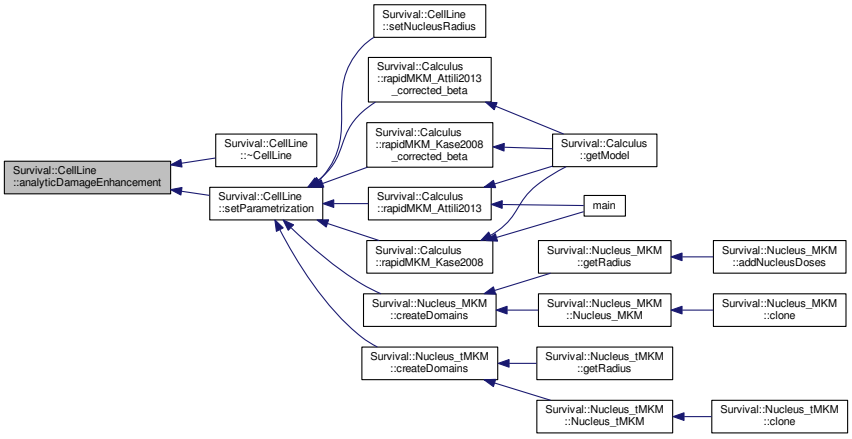
derived from statistical consideration on the probability to observed near SSB on the DNA. In the formula, *h* represents base_Pairs and $\tilde{\alpha}_{SSB}$ and $\tilde{\alpha}_{DSB}$ represent alpha_SSB alpha_DSB respectively multiplied by the ratio between base_Pairs and genomeLength.

Note

The analytic formula underestimates the real value of the damage enhancement factor. The best way to evaluate it is therefore the Monte Carlo simulation performed via the damageEnhancement() factor. The problem is the time necessary to the evaluation; hence for rapid estimates it could be used this method.

Definition at line 167 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.7 double CellLine::damageEnhancement (const double *dose*) const

Evaluate the damage enhancement factor corresponding to a certain dose absorbed via a Monte Carlo simulation.

For the fixed dose:

- It directly generates a number of DSB given by: $N_{DSB}(D) = \alpha_{DSB}\, L_{Genome}\, D$; placed in random position in the genome.
- It directly generates a number of SSB given by: $N_{SSB}(D) = \alpha_{SSB}\, L_{Genome}\, D$; placed in random position on the two strands. SSB near to a DSB (in a window of width base_Pairs centered on the DSB) are excluded from the computation.
- The DNA is read base by base, when a SSB is identified, if it can be found another SSB inside a window of width base_Pairs, a counter is incremented ($N_{2SSB}(D)$).

The value of the resulting damage enhancement factor is evaluated by means of the relation:

$$\eta = \frac{N_{DSB}(D) + N_{2SSB}(D)}{N_{DSB}(D)}$$

Parameters

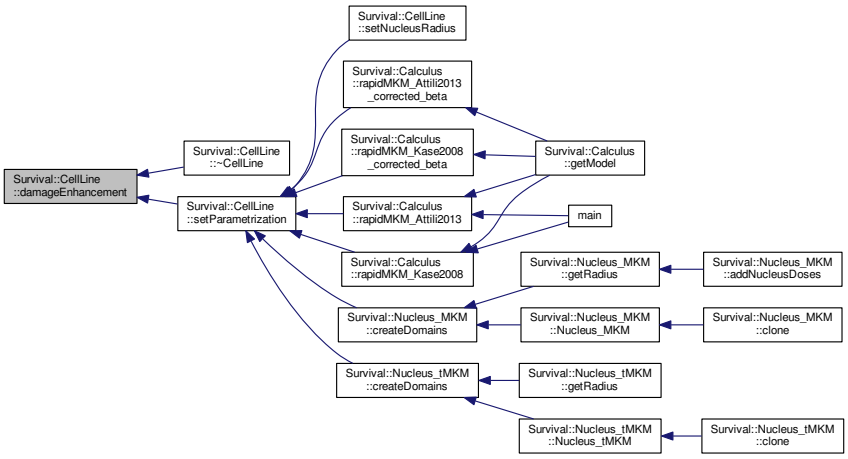
dose	The dose absorbed by the cell, expressed in Gy.

Returns

The value of the damage enhancement factor correspondent to the dose absorbed.

Definition at line 195 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.8 double Survival::CellLine::getAC ( ) const [inline]

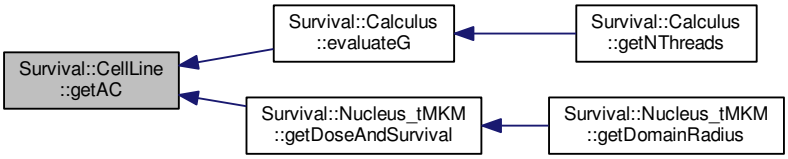
Returns the time constant associated to the repair kinetics of the nucleus, expressed in h^{-1}.

Returns

The time constant associated to the repair kinetics of the nucleus, expressed in h^{-1}.

Definition at line 150 of file CellLine.h.

Here is the caller graph for this function:



7.3.3.9 std::string Survival::CellLine::getCellType ( ) const [inline]

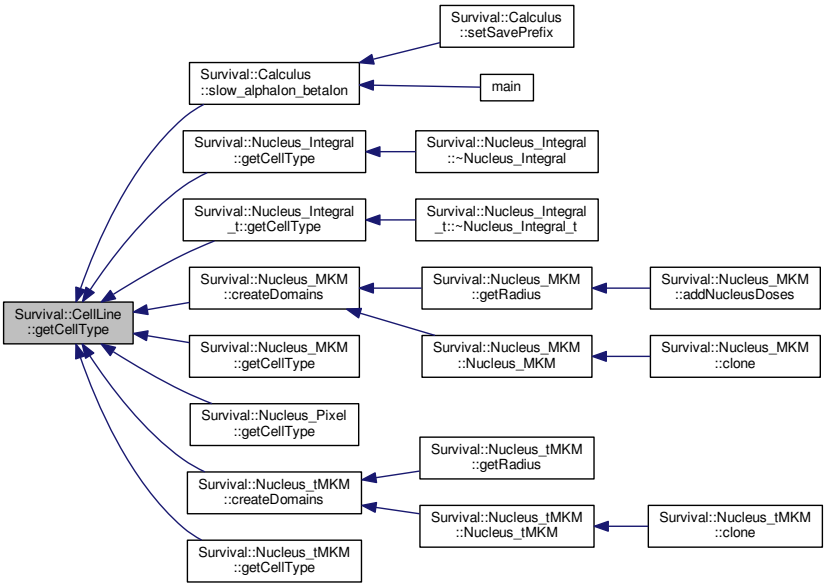
Returns a string identifying the name of the cell line.

Returns

cellType The name of the cell line to which the nucleus belongs.

Definition at line 156 of file CellLine.h.

Here is the caller graph for this function:



7.3.3.10 double Survival::CellLine::getDomainRadius ( ) const [inline]

Returns the radius of the domain expressed in um.

Returns

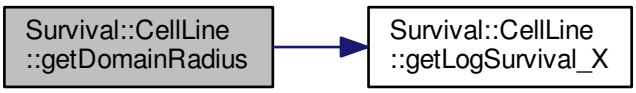
domainRadius The radius of the domain relative to the MKM parametrization of the nucleus, expressed in um.

See also

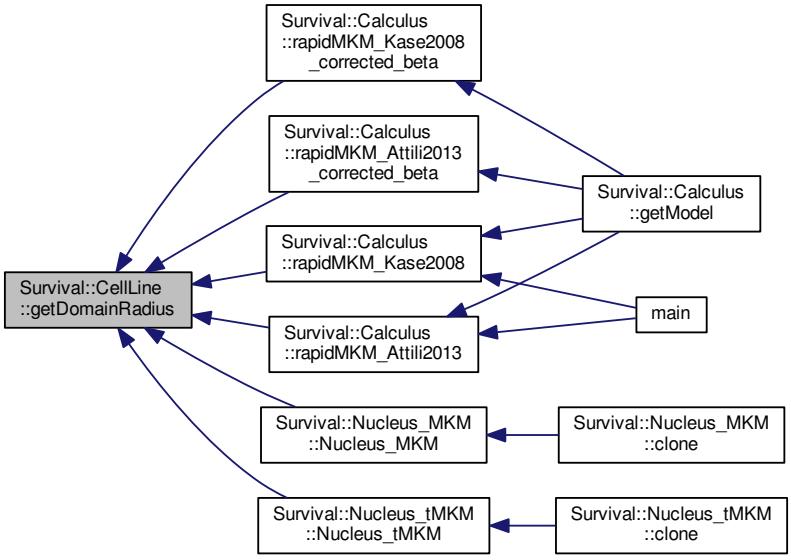
Nucleus_MKM

Definition at line 164 of file CellLine.h.

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.11 double CellLine::getLogSurvival_X (const double *dose*) const

Returns the natural logarithm of the cellular survival evaluated on the basis of the selected parametrization.

The logarithmic survival is evaluated by calling the selected parametrization.

Parameters

dose	The dose absorbed by the nucleus, needed to evaluate the cellular survival, expressed in Gy.

Returns

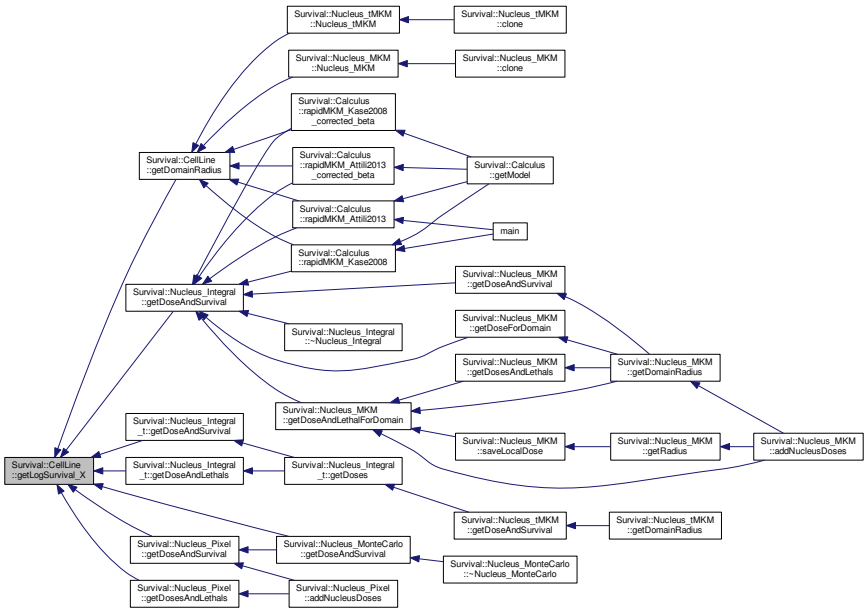
The cellular survival associated to the dose absorbed.

See also

parametrization_LQ(), parametrization_LQ_noDt() and CellLine::getLogSurvival_X(const vector<double>, const vector<double>)

Definition at line 328 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.12 double CellLine::getLogSurvival_X (const std::vector< double > *doses*, const std::vector< double > *times*) const

Overload. Returns the natural logarithm of the cellular survival taking into account the time structure of the irradiation.

The logarithmic survival is evaluated by calling getParameters_LQ_noDt_T() if the corresponding parametrization is selected.

Parameters

doses	A vector containing the sequence of doses deposited in the nucleus, expressed in Gy, each elements is associated to one interaction.
times	A vector containing the sequence of interaction times (expressed in hours), each elements is associated to one interaction.

Returns

The cellular survival associated to the specific structure of doses absorbed.

Warning

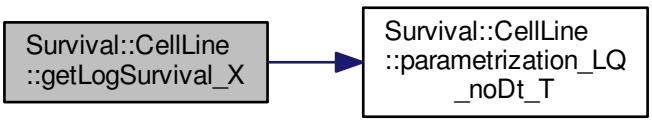
The execution of the program will be terminated if the parametrization selected isn't "parametrization_LQ_↩ noDt_T".

See also

parametrization_LQ(), parametrization_LQ_noDt() and getLogSurvival_X()

Definition at line 335 of file CellLine.cpp.

Here is the call graph for this function:



7.3.3.13 double Survival::CellLine::getNucleusRadius ( ) const [inline]

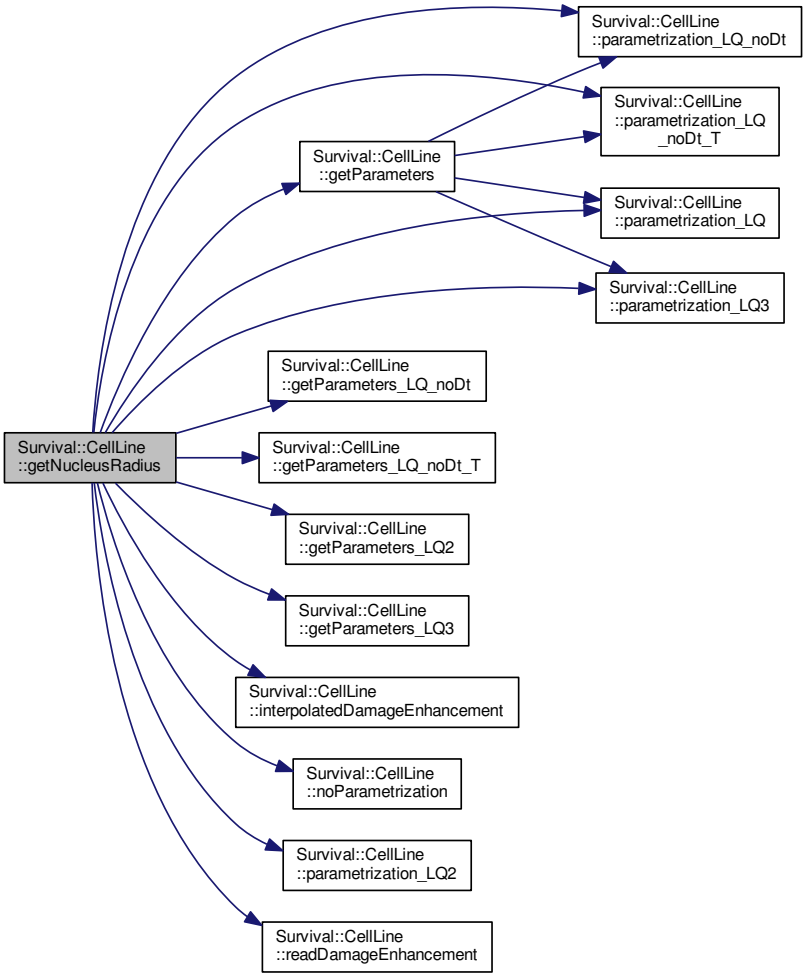
Returns the radius of the nucleus expressed in um.

Returns

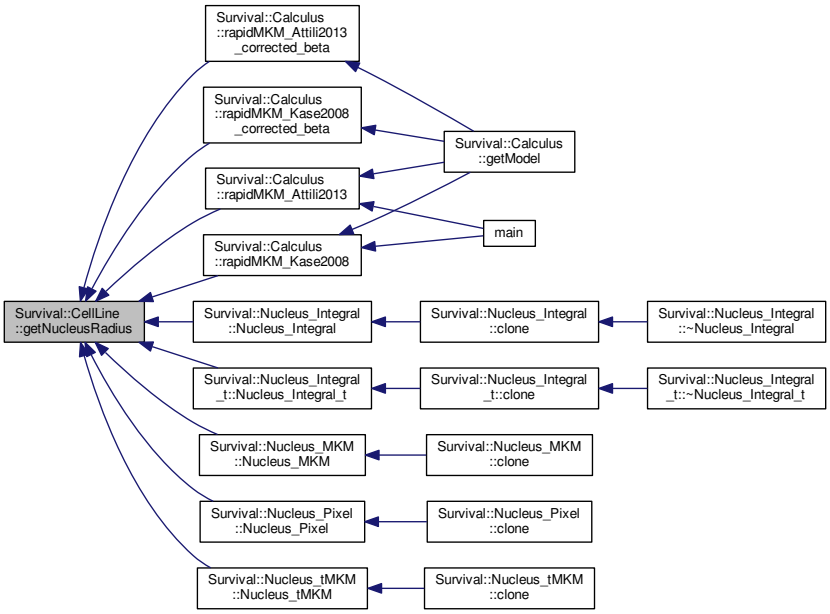
nucleusRadius The radius of the nucleus expressed in um.

Definition at line 198 of file CellLine.h.

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.14 void CellLine::getParameters (double & *returnAlpha_X,* double & *returnBeta_X,* double & *returnD_t*) const

Returns the linear quadratic α and β parameters and the transition dose, corresponding to the selected parametrization, by overwriting three `double` variables passed by reference.

In the case of the "noDt-parametrization", D_t is set to -1.

Parameters

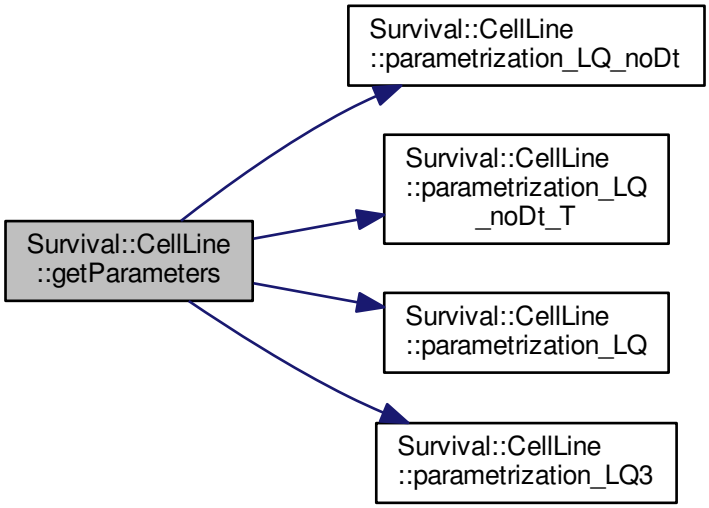
returnAlpha↩ _X	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.
returnBeta_X	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.
returnD_t	The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.

See also

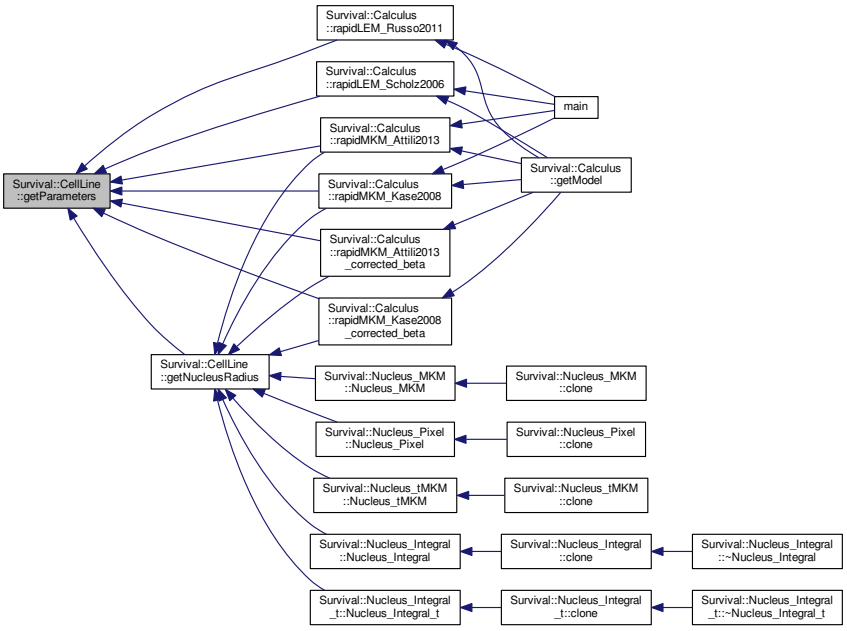
setParametrization()

Definition at line 348 of file CellLine.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3.15 void CellLine::getParameters_LQ2 (double & *returnAlpha_X2,* double & *returnBeta_X2,* double & *returnD_t2,* double & *returnGenomeLength,* double & *returnAlpha_SSB,* double & *returnAlpha_DSB,* long int & *returnBase_Pairs*) const

Returns the parameters characteristic of the LQ2 parametrization (used in the LEM II formulation) overwriting some variables passed by reference.

Parameters

returnAlpha_X2	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}, passed by reference to be overwritten.
returnBeta_X2	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}, passed by reference to be overwritten.
returnD_t2	The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy, passed by reference to be overwritten.
returnGenomeLength	The genome length expressed in unit of base pairs, passed by reference to be overwritten.
returnAlpha_SSB	The number of SSBs directly produced by the irradiation in the whole genome per unit of dose absorbed, passed by reference to be overwritten.
returnAlpha_DSB	The number of DSBs directly produced by the irradiation in the whole genome per unit of dose absorbed, passed by reference to be overwritten.
returnBase_Pairs	The distance (in number of based pairs) between two SSBs resulting in a DSB, passed by reference to be overwritten.

Warning

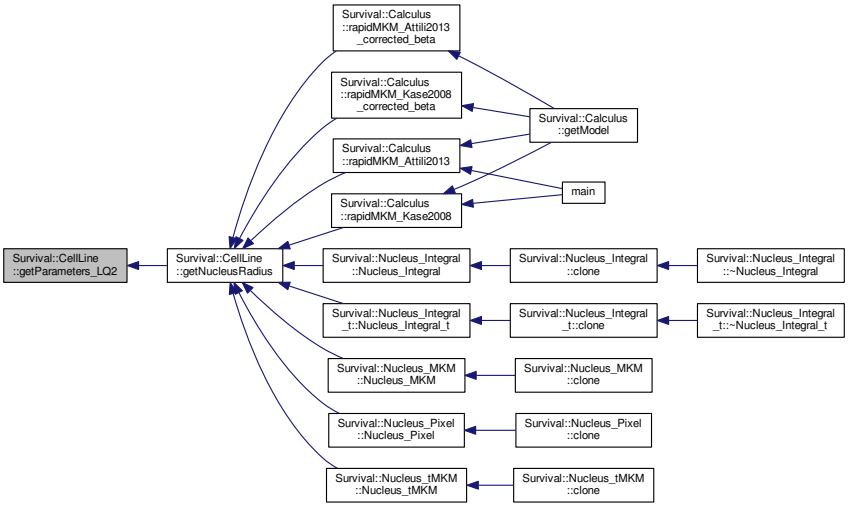
The execution of the program will be terminated if an incorrect parametrization is selected.

See also

setParametrization, addParametrization_LQ2 and parametrization_LQ2

Definition at line 417 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.16 void CellLine::getParameters_LQ3 (double & *returnAlpha_X3,* double & *returnBeta_X3,* double & *returnD_t3,* double & *returnGenomeLength,* double & *returnAlpha_SSB,* double & *returnAlpha_DSB,* long int & *returnBase_Pairs*) const

Returns the parameters characteristic of the LQ3 parametrization (used in the LEM III formulation) overwriting some variables passed by reference.

Parameters

returnAlpha_X3	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}, passed by reference to be overwritten.
returnBeta_X3	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}, passed by reference to be overwritten.
returnD_t3	The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy, passed by reference to be overwritten.
returnGenomeLength	The genome length expressed in unit of base pairs, passed by reference to be overwritten.
returnAlpha_SSB	The number of SSBs directly produced by the irradiation in the whole genome per unit of dose absorbed, passed by reference to be overwritten.
returnAlpha_DSB	The number of DSBs directly produced by the irradiation in the whole genome per unit of dose absorbed, passed by reference to be overwritten.
returnBase_Pairs	The distance (in number of based pairs) between two SSBs resulting in a DSB, passed by reference to be overwritten.

Warning

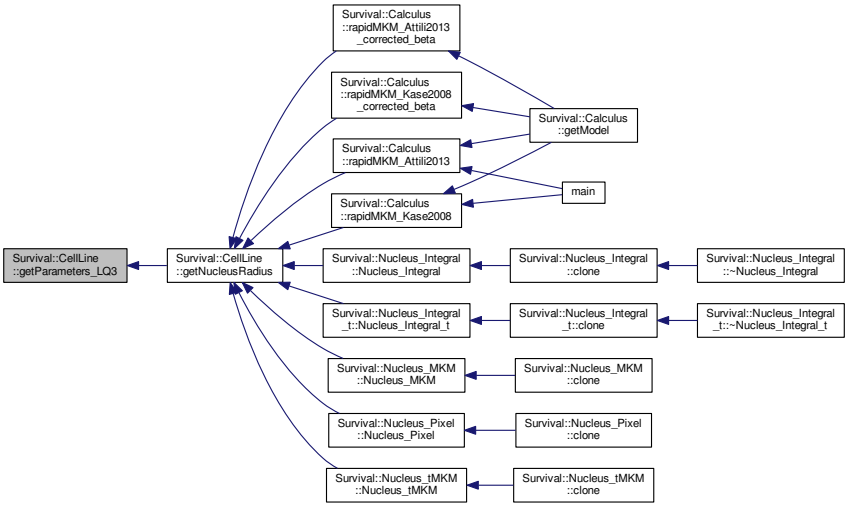
The execution of the program will be terminated if an incorrect parametrization is selected.

See also

setParametrization, addParametrization_LQ3 and parametrization_LQ3

Definition at line 444 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.17 void CellLine::getParameters_LQ_noDt (double & *returnAlpha_X,* double & *returnBeta_X*) const

Returns the linear quadratic α and β parameters corresponding to the "LQ_noDt" parametrization by overwriting two `double` variables passed by reference.

Parameters

returnAlpha_X	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}, passed by reference to be overwritten.
returnBeta_X	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}, passed by reference to be overwritten.

Warning

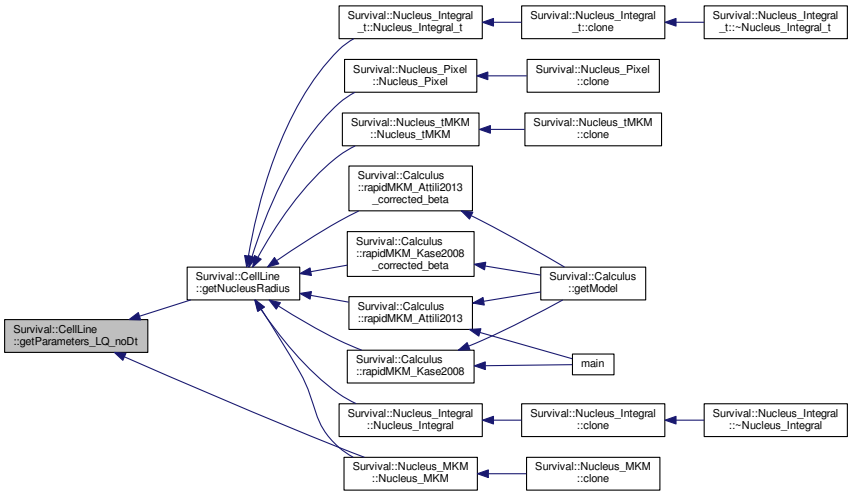
The execution of the program will be terminated if an incorrect parametrization is selected.

See also

setParametrization() and parametrization_LQ_noDt()

Definition at line 381 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.18 void CellLine::getParameters_LQ_noDt_T (double & *returnAlpha_X,* double & *returnBeta_X,* double & *ac_*) const

Returns the linear quadratic α and β parameters corresponding to the "LQ_noDt" parametrization and the time constant by overwriting three `double` variables passed by reference.

Parameters

returnAlpha↩ _X	The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}, passed by reference to be overwritten.
returnBeta_X	The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}, passed by reference to be overwritten.
*ac_*	The time constant associated to the repair kinetics of the nucleus, expressed in h^{-1}, passed by reference to be overwritten.

Warning

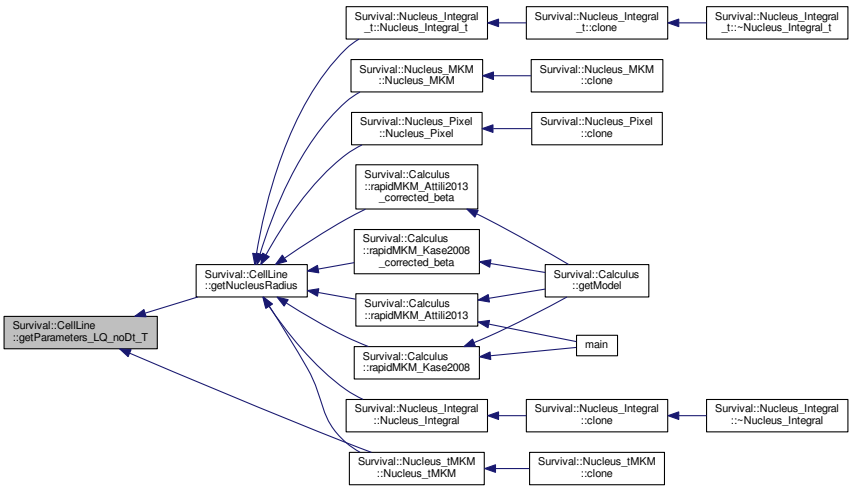
The execution of the program will be terminated if an incorrect parametrization is selected.

See also

setParametrization() and parametrization_LQ_noDt_T()

Definition at line 398 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.19 double CellLine::interpolatedDamageEnhancement (const double *dose*) const

Get the value of the damage enhancement factor from the precalculated curve (doseForEta, etaPre) interpolating the nearest neighbors of the dose imposed.

Parameters

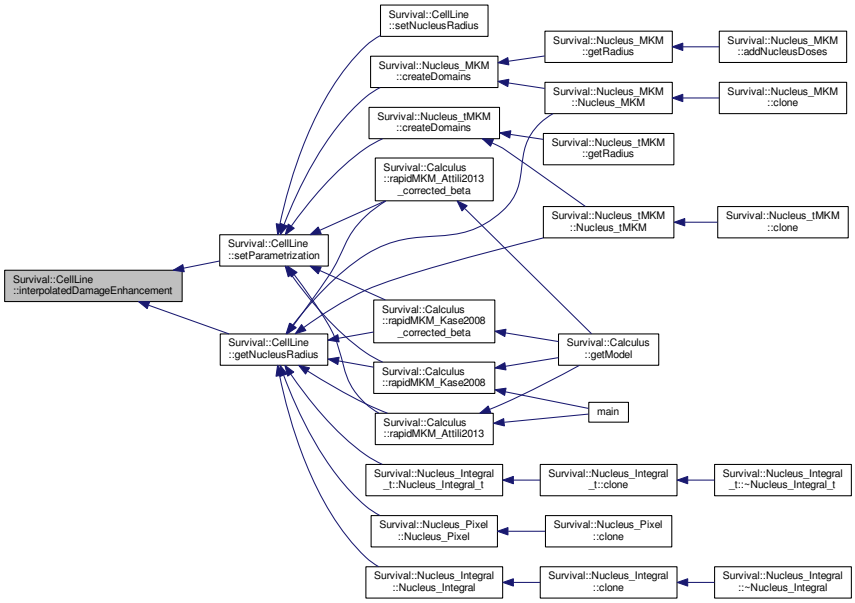
dose	The dose absorbed by the cell, expressed in Gy.

Returns

The value of the damage enhancement factor correspondent to the dose absorbed.

Definition at line 471 of file CellLine.cpp.

Here is the caller graph for this function:

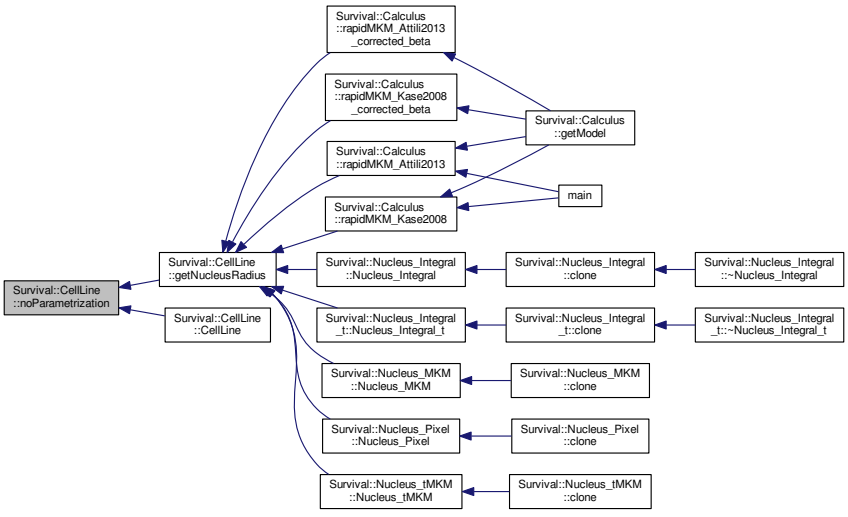


7.3.3.20 double CellLine::noParametrization (const double *dummy*) const

If called it interrupts the execution of the program, because no parametrizazion is selected.

Definition at line 491 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.21 double Survival::CellLine::noParametrization (const std::vector< double > *v1*, const std::vector< double > *v2*) const

If called it interrupts the execution of the program, because no parametrizazion is selected.

7.3.3.22 double CellLine::parametrization_LQ (const double *dose*) const

Returns the logarithmic cellular survival corresponding to a particular dose absorbed. Implements the parametrization used in the LEM I formulation.

It implements the LEM I parametrization of the cellular survival based on the assumption that beyond a dose D_t the standard parametrization of the LQ model is no more valid as it was observed "an exponential tail" (function of the dose absorbed).

$$S = \exp(-\alpha\, D - beta\, D^2) \qquad D < D_t$$

$$S = \exp(-\alpha\, D_t - beta\, D_t^2)\exp(-s(D - D_t)) = S_t \exp(-s(D - D_t)) \qquad D >= D_t$$

where $s = \alpha + 2\,\beta D_t$.

Parameters

dose	The dose absorbed expressed in Gy.

Returns

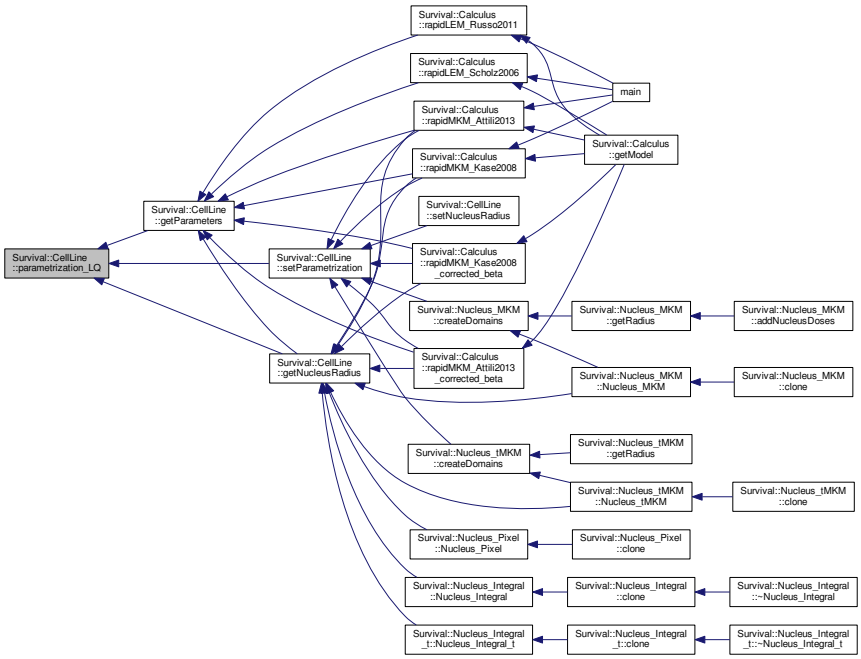
The logarithmic cellular survival corresponding to a particular dose absorbed.

See also

setParametrization(), parametrization_LQ_noDt_T() and parametrization_LQ()

Definition at line 538 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.23 double CellLine::parametrization_LQ2 (const double *dose*) const

Returns the logarithmic cellular survival corresponding to a particular dose absorbed. Implements the parametrization used in the LEM II formulation.

The LEM II formulation adopts the same parametrization used in the LEM I (see parametrization_LQ()) but, to accounts for the so called *clustering effect*, an enhancement factor η is added which multiplies the dose absorbed when $D > D_t$. The resulting parametrization can be written as:

$$S = \exp(-\alpha\, D - beta\, D^2) \qquad D < D_t$$

$$S = \exp(-\alpha\, D_t - beta\, D_t^2) \exp\left[-s(\eta(D)D - D_t)\right] = S_t \exp\left[-s(\eta(D)D - D_t)\right] \qquad D >= D_t$$

where $s = \alpha + 2\,\beta D_t$.

η is a function of the dose absorbed, and there are several ways to generate it:

- It can be generated via a Monte Carlo Simulation (damageEnhancement())
- It can be generated via an analytic approximation (analyticDamageEnhancement())
- It can be read from an external file (readDamageEnhancement())

Parameters

dose	The dose absorbed expressed in Gy.

Returns

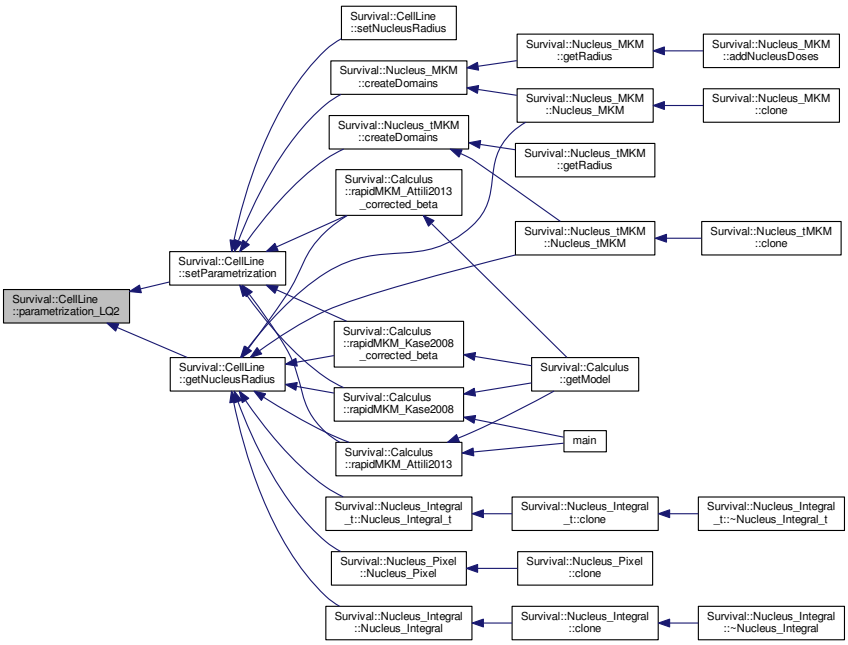
The logarithmic cellular survival corresponding to a particular dose absorbed.

See also

setParametrization(), parametrization_LQ_noDt_T() and parametrization_LQ()

Definition at line 548 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.24 double CellLine::parametrization_LQ3 (const double *dose*) const

Returns the logarithmic cellular survival corresponding to a particular dose absorbed. Implements the parametrization used in the LEM II formulation.

The parametrization used is identical to the one defined in the LEM II formulation (see parametrization_LQ2()). Briefly, the survival is evaluated as:

$$S = \exp(-\alpha\, D - beta\, D^2) \qquad D < D_t$$

$$S = \exp(-\alpha\, D_t - beta\, D_t^2) \exp\left[-s(\eta(D)D - D_t)\right] = S_t \exp\left[-s(\eta(D)D - D_t)\right] \qquad D >= D_t$$

Parameters

dose	The dose absorbed expressed in Gy.

Returns

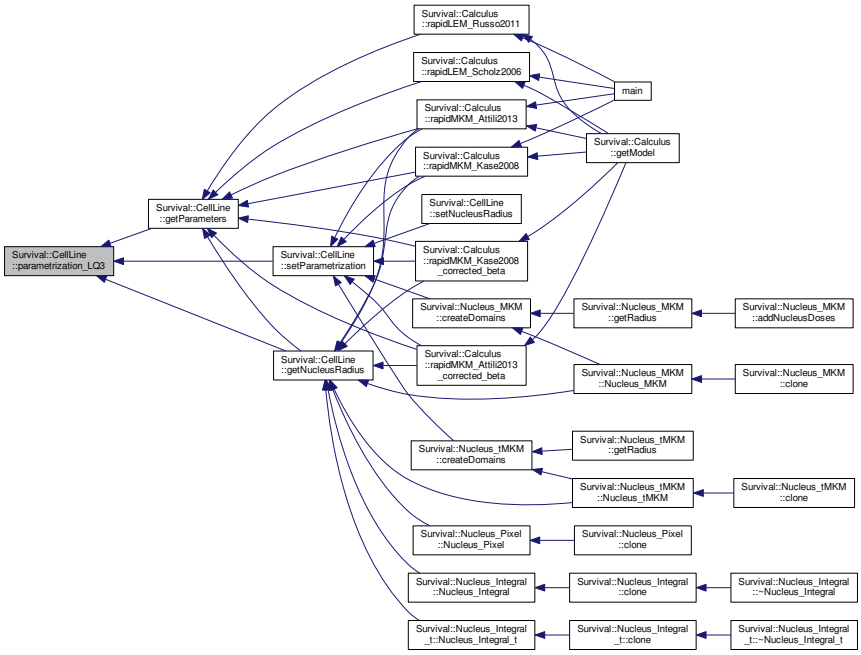
The logarithmic cellular survival corresponding to a particular dose absorbed.

See also

setParametrization(), parametrization_LQ_noDt_T() and parametrization_LQ()

Definition at line 560 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.25 double CellLine::parametrization_LQ_noDt (const double *dose*) const

Returns the logarithmic cellular survival corresponding to a particular dose absorbed. Implements the parametrization used in the MKM formulation.

The survival is evaluated by means of the standard linear quadratic relation:

$$S = \exp(-\alpha\, D - beta\, D^2)$$

Note

The function returns the natural logarithm of the survival, NOT the survival.

Parameters

dose	The dose absorbed expressed in Gy.

Returns

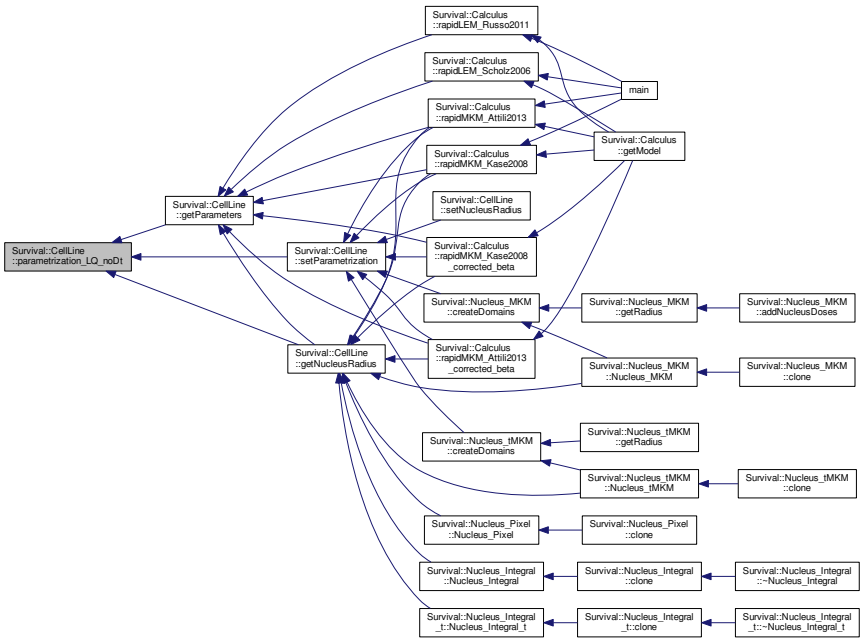
The logarithmic cellular survival corresponding to a particular dose absorbed.

See also

setParametrization(), parametrization_LQ_noDt_T() and parametrization_LQ()

Definition at line 512 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.26 double CellLine::parametrization_LQ_noDt_T (const std::vector< double > *doses*, const std::vector< double > *times*) const

Returns the logarithmic cellular survival associated to a sequence of doses absorbed with a specific time structure. Implements the parametrization used in the MCt-MKM formulation.

At present, this parametrization is associated to the tMKM model (1). The logarithmic survival (L) is evaluated by means of the following relation:

$$L = -\alpha_d \left(\sum_{i=1}^{N} z_i \right) - \beta_d \left(\sum_{i=1}^{N} \right)^2 - 2\beta \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[1 - \exp\left(-(a+c)(t_j - t_i) \right) \right] z_i \, z_j$$

where N represent the length of the vector of doses (or times), the sum (a+c) represents ac and z_i and t_i represent the i-th element of the vectors of times and doses absorbed respectively.

Parameters

doses	The vector representing the sequence of doses absorbed, expressed in Gy.
times	The vector representing the sequence of interaction times, expressed in hours.

Returns

The logarithmic cellular survival associated to a sequence of doses absorbed with a specific time structure.

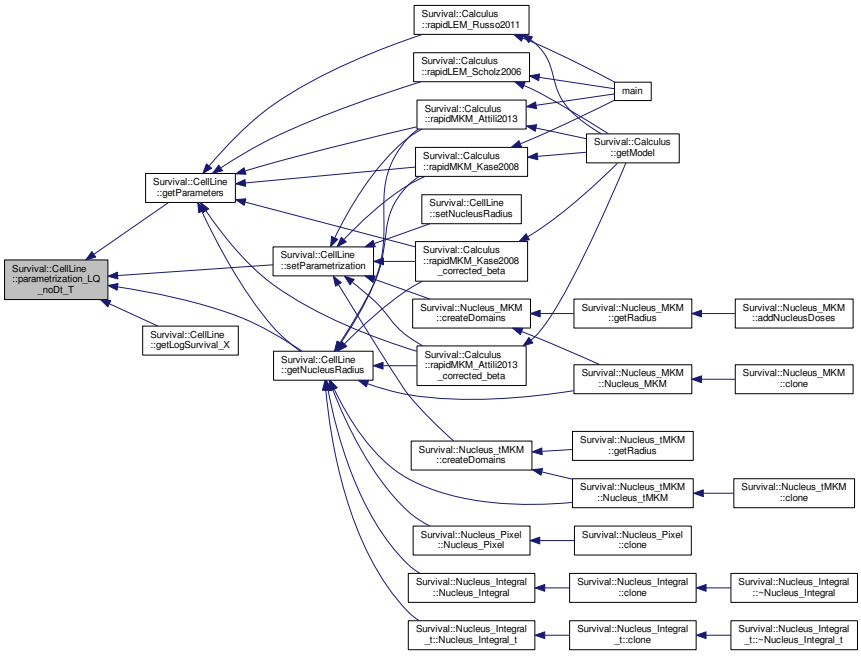
See also

setParametrization(), parametrization_LQ_noDt_T(), parametrization_LQ() and Nucleus_tMKM

1. L. Manganaro, ..., A. Attili, ...

Definition at line 519 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.27 double CellLine::readDamageEnhancement (const double *dose*) const

Reads and returns the value of the damage enhancement factor correspondent to the required dose from an external file.

Parameters

dose	The dose absorbed, expressed in Gy, to calculate the correspondent damaga enhancement factor.

Returns

The value of the damage enhancement factor.

Note

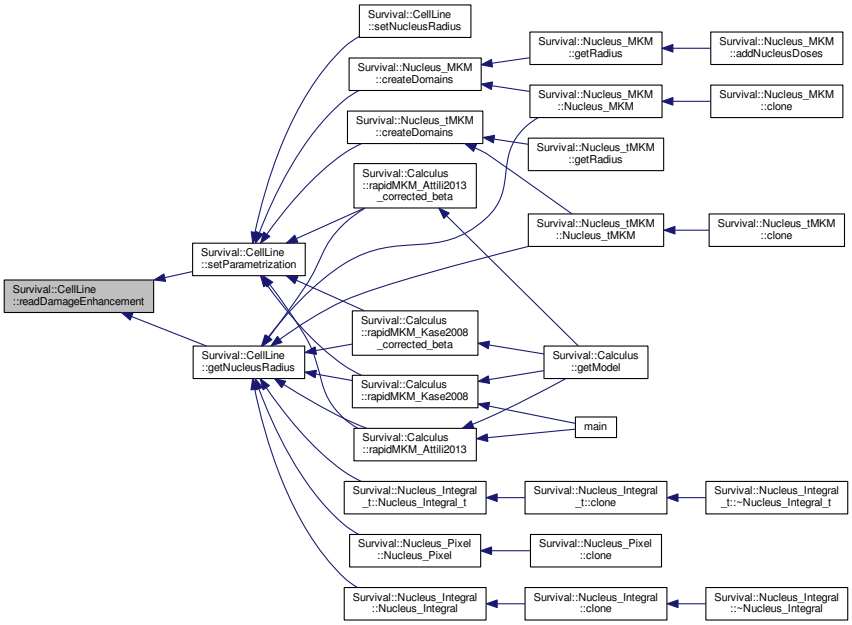
This method was thought to import the exact curve published in the LEM description.

Warning

The execution of the program will be terminated if the file doesn't exist.

Definition at line 572 of file CellLine.cpp.

Here is the caller graph for this function:



7.3.3.28 void Survival::CellLine::setDomainRadius (double *domainRadius_*) [inline]

Sets the radius of the domain relative to the MKM parametrization of the nucleus.

Parameters

*domain↩ Radius_*	The radius of the domain expressed in um.

See also

domainRadius, Nucleus_MKM

Definition at line 421 of file CellLine.h.

7.3.3.29 void Survival::CellLine::setNucleusRadius (double *nucleusRadius_*) [inline]

Sets the radius of the nucleus.

Parameters

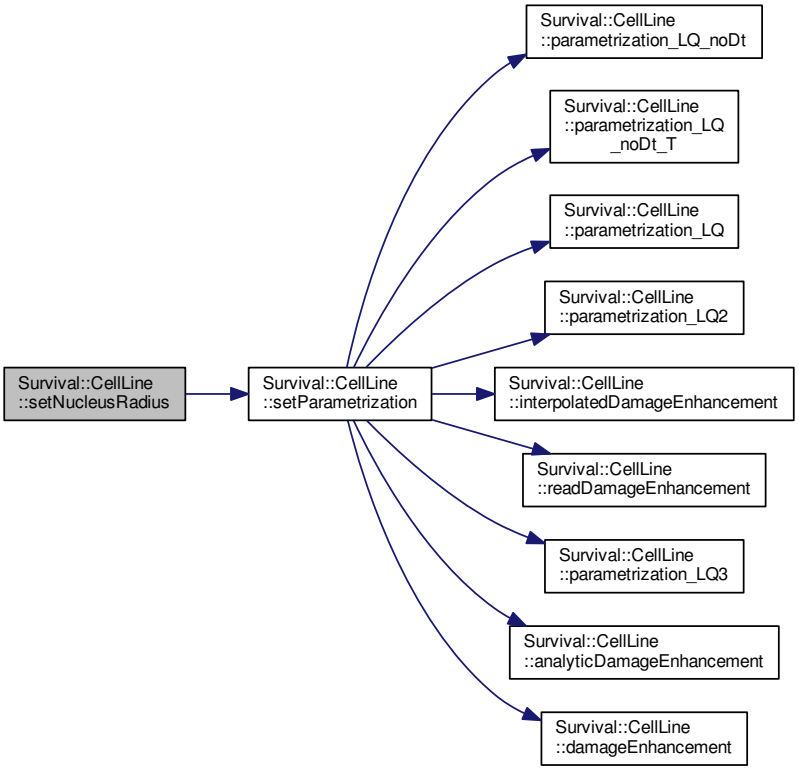
*nucleus↩ Radius_*	The radius of the nucleus expressed in um.

See also

nucleusRadius

Definition at line 429 of file CellLine.h.

Here is the call graph for this function:



7.3.3.30 void CellLine::setParametrization (const std::string *parametrization_type*)

Sets an X-ray parametrization for the evaluation of the cellular survival.

Parameters

parametrization_type	A string indicating the name of the desired parametrization.

The possible choices for the parametrization and the correspondents options set by the function are listed in the

following table:

Parametrization	Model	selected↩ Parametrization	selectedDamage↩ Enhancement	selectedEta↩ Generation
LQ	LEM I	parametrization_LQ()	None	None
LQ2	LEM II	parametrization_L↩ Q2()	interpolated↩ Damage↩ Enhancement()	readDamage↩ Enhancement()
LQ3	LEM III	parametrization_L↩ Q3()	interpolated↩ Damage↩ Enhancement()	readDamage↩ Enhancement()
LQ_noDt	MKM	parametrization_LQ↩ _noDt()	None	None
LQ_noDt_T	MCt-MKM	parametrization_LQ↩ _noDt_T()	None	None
LQ2_readfile	LEM II	parametrization_L↩ Q2()	interpolated↩ Damage↩ Enhancement()	readDamage↩ Enhancement()
LQ2_interpolated_↩ MC	LEM II	parametrization_L↩ Q2()	interpolated↩ Damage↩ Enhancement()	damage↩ Enhancement()
LQ2_interpolated_↩ analytic	LEM II	parametrization_L↩ Q2()	interpolated↩ Damage↩ Enhancement()	analyticDamage↩ Enhancement()
LQ2_punctual_↩ analytic	LEM II	parametrization_L↩ Q2()	analyticDamage↩ Enhancement()	analyticDamage↩ Enhancement()
LQ2_punctual_MC	LEM II	parametrization_L↩ Q2()	damage↩ Enhancement()	damage↩ Enhancement()

If the parametrization selected needs an eta generated, then the function generates it by (recursively) using the selectedEtaGeneration pointer, storing the calculated values of dose and η in doseForEta and etaPre respectively.

Note

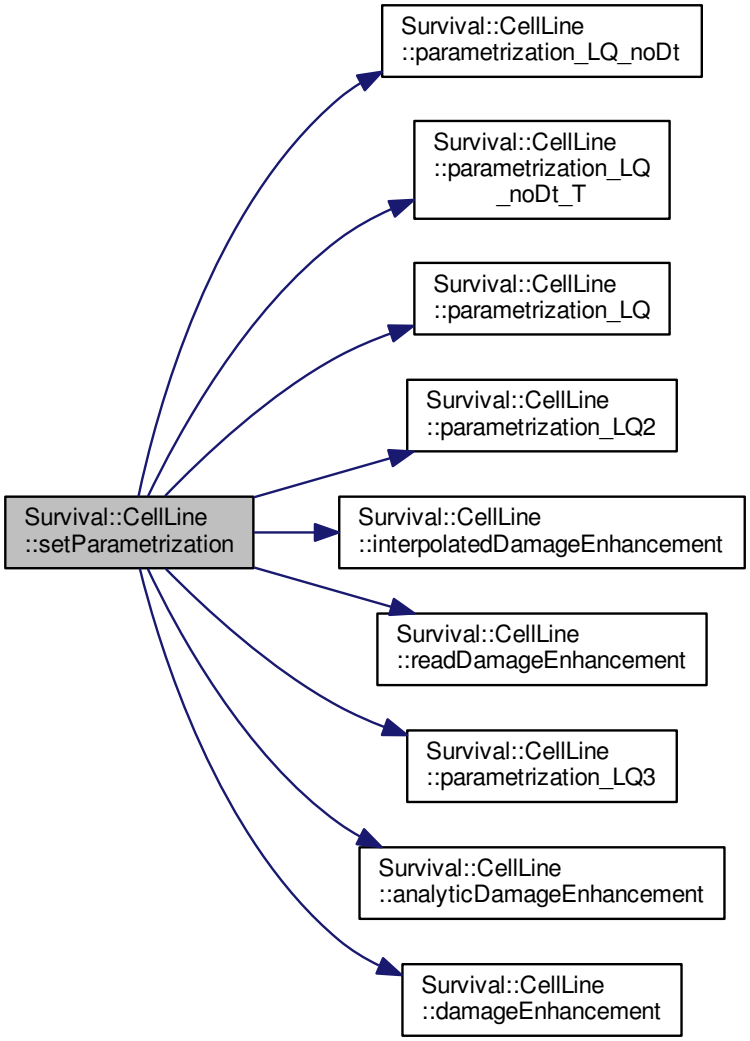
`selectedParametrization`, `selectedDamageEnhancement` and `selectedEtaGeneration` are pointers to functions.

Warning

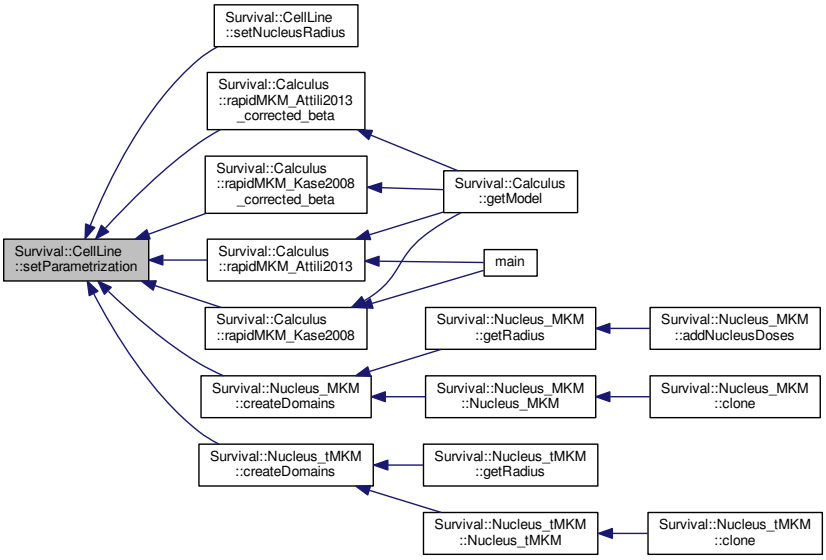
The execution of the program will be terminated if an inexistent parametrization is selected.

Definition at line 605 of file CellLine.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.4 Member Data Documentation

7.3.4.1 double Survival::CellLine::ac [private]

The time constant associated to the repair kinetics of the cell, expressed in h^{-1}.

Definition at line 482 of file CellLine.h.

7.3.4.2 double Survival::CellLine::alpha_DSB [private]

The number of DSBs directly produced by the irradiation in the whole genome per unit of dose absorbed.

Definition at line 529 of file CellLine.h.

7.3.4.3 double Survival::CellLine::alpha_SSB [private]

The number of SSBs directly produced by the irradiation in the whole genome per unit of dose absorbed.

Definition at line 526 of file CellLine.h.

7.3.4.4 double Survival::CellLine::alpha_X [private]

The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.

Definition at line 472 of file CellLine.h.

7.3.4.5 double Survival::CellLine::alpha_X1 [private]

The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.

Definition at line 489 of file CellLine.h.

7.3.4.6 double Survival::CellLine::alpha_X2 [private]

The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.

Definition at line 508 of file CellLine.h.

7.3.4.7 double Survival::CellLine::alpha_X3 [private]

The linear quadratic α-parameter characteristic for X-rays, expressed in Gy^{-1}.

Definition at line 548 of file CellLine.h.

7.3.4.8 long int Survival::CellLine::base_Pairs [private]

The distance (in unit of based pairs) between two SSBs resulting in a DSB.

Definition at line 532 of file CellLine.h.

7.3.4.9 double Survival::CellLine::beta_X [private]

The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.

Definition at line 475 of file CellLine.h.

7.3.4.10 double Survival::CellLine::beta_X1 [private]

The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.

Definition at line 492 of file CellLine.h.

7.3.4.11 double Survival::CellLine::beta_X2 [private]

The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.

Definition at line 511 of file CellLine.h.

7.3.4.12 double Survival::CellLine::beta_X3 [private]

The linear quadratic β-parameter characteristic for X-rays, expressed in Gy^{-2}.

Definition at line 551 of file CellLine.h.

7.3.4.13 std::string Survival::CellLine::cellType [private]

A `string` identifying the name of the cell line.

Definition at line 462 of file CellLine.h.

7.3.4.14 double Survival::CellLine::D_t [private]

The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.

Definition at line 495 of file CellLine.h.

7.3.4.15 double Survival::CellLine::D_t2 [private]

The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.

Definition at line 514 of file CellLine.h.

7.3.4.16 double Survival::CellLine::D_t3 [private]

The transition dose beyond which the standard linear quadratic parametrization is no more valid, expressed in Gy.

Definition at line 554 of file CellLine.h.

7.3.4.17 int CellLine::DNA [static], [private]

An array representing the genome of the cell.

Definition at line 627 of file CellLine.h.

7.3.4.18 double Survival::CellLine::domainRadius [private]

The radius of the domain associated to the MKM parametrization of the nucleus, expressed in um.

Definition at line 468 of file CellLine.h.

7.3.4.19 double Survival::CellLine::doseForEta[200] [private]

An array used to store the values of dose for to precalculate the enhancement factor curve.

It is constituted by 200 values logarithmically spaced in $[100, 5 \cdot 10^6]$.

Definition at line 615 of file CellLine.h.

7.3.4.20 bool CellLine::DSB [static], [private]

A boolean array representing the double strand breaks (DSB) in the genome.

Definition at line 636 of file CellLine.h.

7.3.4.21 double Survival::CellLine::etaPre[200] [private]

An array containing the precalculated values of the enhancement factor as a function of the dose absorbed.

It is constituted by 200 values indicated the enhancement factor for each value of the doseForEta array.

Definition at line 621 of file CellLine.h.

7.3.4.22 double Survival::CellLine::genomeLength [private]

The genome length expressed in number of base pairs.

Definition at line 523 of file CellLine.h.

7.3.4.23 bool Survival::CellLine::isLQ2loaded [private]

A boolean value identifying if one the "LQ2" parametrizations is selected.

Possible cases are:

- "LQ2"
- "LQ2_interpolated_analytic"
- "LQ2_interpolated_MC"
- "LQ2_interpolated_readfile"
- "LQ2_punctual_analytic"
- "LQ2_punctual_MC"

Definition at line 544 of file CellLine.h.

7.3.4.24 bool Survival::CellLine::isLQ3loaded [private]

A boolean value identifying if the "LQ3" parametrization is selected.

Definition at line 563 of file CellLine.h.

7.3.4.25 bool Survival::CellLine::isLQ_noDt_TLoaded [private]

A boolean value identifying if the "LQ_noDt_T" parametrization is selected.

Definition at line 485 of file CellLine.h.

7.3.4.26 bool Survival::CellLine::isLQ_noDtLoaded [private]

A boolean value identifying if the "LQ_noDt" parametrization is selected.

Definition at line 478 of file CellLine.h.

7.3.4.27 bool Survival::CellLine::isLQloaded [private]

A boolean value identifying if the "LQ" parametrization is selected.

Definition at line 504 of file CellLine.h.

7.3.4.28 double Survival::CellLine::logS_t [private]

The logarithmic survival associated to a dose absorbed D_t, evaluated according to the standard linear quadratic parametrization.

Definition at line 501 of file CellLine.h.

7.3.4.29 double Survival::CellLine::logS_t2 [private]

The logarithmic survival associated to a dose absorbed D_t, evaluated according to the standard linear quadratic parametrization.

Definition at line 520 of file CellLine.h.

7.3.4.30 double Survival::CellLine::logS_t3 [private]

The logarithmic survival associated to a dose absorbed D_t, evaluated according to the standard linear quadratic parametrization.

Definition at line 560 of file CellLine.h.

7.3.4.31 bool CellLine::needEtaGenerated = false [static], [private]

A boolean data member that indicates if the selected parametrization requires the generation of the enhancement factor.

Definition at line 624 of file CellLine.h.

7.3.4.32 double Survival::CellLine::nucleusRadius [private]

The radius of the nucleus characteristic for the cell line, expressed in um.

Definition at line 465 of file CellLine.h.

7.3.4.33 double Survival::CellLine::s [private]

The coefficient of the exponential tail: $s = \alpha + 2\beta D_t$.

Definition at line 498 of file CellLine.h.

7.3.4.34 double Survival::CellLine::s2 [private]

The coefficient of the exponential tail: $s = \alpha + 2\beta D_t$.

Definition at line 517 of file CellLine.h.

7.3.4.35 double Survival::CellLine::s3 [private]

The coefficient of the exponential tail: $s = \alpha + 2\beta D_t$.

Definition at line 557 of file CellLine.h.

7.3.4.36 double(CellLine::* Survival::CellLine::selectedDamageEnhancement) (const double dose) const [private]

A pointer to functions that identifies the selected way to evaluate the enhancement factor $\eta(D)$ in LEM II and III formulations.

Possible choices are:

- interpolatedDamageEnhancement()
- analyticDamageEnhancement()
- damageEnhancement()

It's used in parametrization_LQ2() and parametrization_LQ3() methods.

Definition at line 575 of file CellLine.h.

7.3.4.37 double(CellLine::* Survival::CellLine::selectedEtaGeneration) (const double dose) const [private]

A pointer to functions that identifies the selected way to evaluate the enhancement factor $\eta(D)$ in LEM II and III formulations.

Possible choices are:

- readDamageEnhancement()
- analyticDamageEnhancement()
- damageEnhancement()

It's used in setParametrization() to calculate a curve for η as a function of the dose absorbed to be stored in doseForEta and etaPre arrays.

See also

parametrization_LQ2() and parametrization_LQ3()

Definition at line 588 of file CellLine.h.

7.3.4.38 double(CellLine::* Survival::CellLine::selectedParametrization) (const double dose) const [private]

A pointer to functions that identifies the selected parametrization.

Possible parametrizations are:

- noParametrization()
- parametrization_LQ_noDt() (MKM)
- parametrization_LQ() (LEM I)
- parametrization_LQ2() (LEM II)
- parametrization_LQ3() (LEM III)

Definition at line 599 of file CellLine.h.

7.3.4.39 double(CellLine::* Survival::CellLine::selectedParametrizationT) (const std::vector< double >doses, const std::vector< double >times) const [private]

A pointer to functions that identifies the selected parametrization when the temporal effects of the irradiation are taken into account.

Possible parametrizations are:

- noParametrization(const std::vector<double>, const std::vector<double>)
- parametrization_LQ_noDt_T() (MCt-MKM)

Definition at line 607 of file CellLine.h.

7.3.4.40 bool CellLine::SSB1 [static], [private]

A boolean array representing the single strand breaks (SSB) in the first strand.

Definition at line 630 of file CellLine.h.

7.3.4.41 bool CellLine::SSB2 [static], [private]

A boolean array representing the single strand breaks (SSB) in the second strand.

Definition at line 633 of file CellLine.h.

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

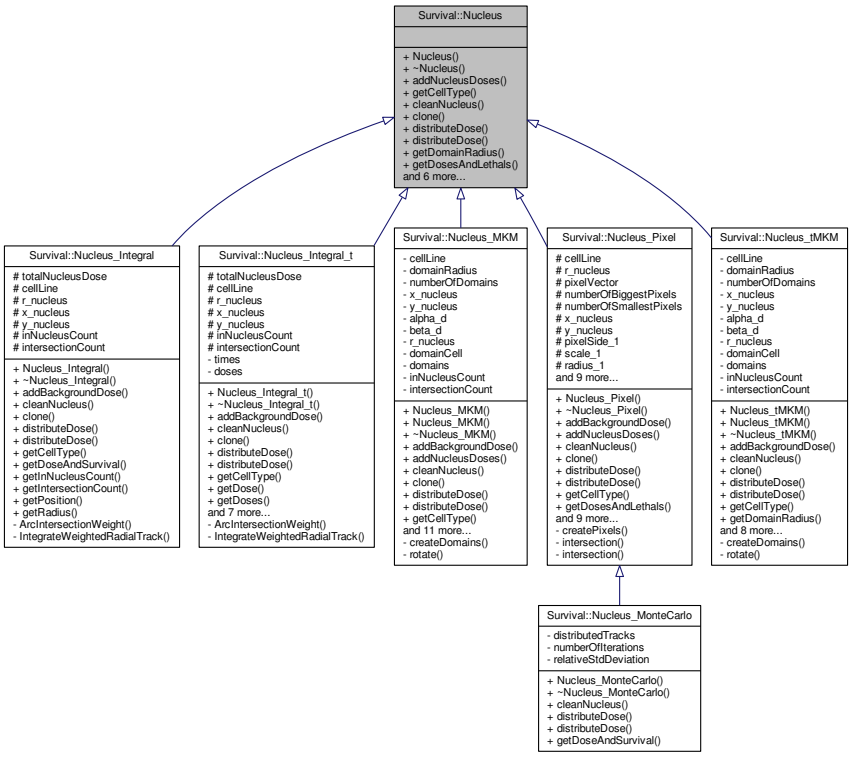
- include/CellLine.h
- src/CellLine.cpp

7.4 Survival::Nucleus Class Reference

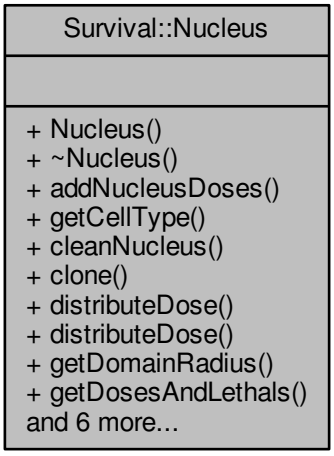
Linked to a specific CellLine object, this class defines the structure of the cellular nucleus to be irradiated and evaluates the dose absorbed during the interaction with a track.

`#include <Nucleus.h>`

Inheritance diagram for Survival::Nucleus:



Collaboration diagram for Survival::Nucleus:



Public Member Functions

- Nucleus ()

  Constructor of a pure virtual class (empty).
- virtual ∼Nucleus ()

  Destructor of a pure virtual class (empty).
- virtual void addNucleusDoses (Nucleus &)

  Declaration of the virtual function getNucleusDoses (for a more detailed description see the derived classes).
- virtual std::string getCellType () const =0

  Declaration of the pure virtual function getCellType (for a more detailed description see the derived classes).
- virtual void cleanNucleus ()=0

  Declaration of the pure virtual function cleanNucleus (for a more detailed description see the derived classes).
- virtual Nucleus ∗ clone (const CellLine &)=0

  Declaration of the pure virtual function clone (for a more detailed description see the derived classes).
- virtual void distributeDose (const Track &track)=0

  Declaration of the pure virtual function distributeDose (for a more detailed description see the derived classes).
- virtual void distributeDose (const Tracks &tracks)=0

  Declaration of the pure virtual function distributeDose (for a more detailed description see the derived classes).
- virtual double getDomainRadius ()

  Declaration of the virtual function getAC (for a more detailed description see the derived classes).
- virtual void getDosesAndLethals (std::vector< double > &, std::vector< double > &, std::vector< double > &, std::vector< double > &)

  Declaration of the virtual function getDosesandLethals (for a more detailed description see the derived classes).
- virtual void getDoseAndSurvival (double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty) const =0

  Declaration of the pure virtual function getDoseAndSurvival (for a more detailed description see the derived classes).
- virtual int getInNucleusCount () const =0

  Declaration of the pure virtual function getInNucleusCount (for a more detailed description see the derived classes).
- virtual int getIntersectionCount () const =0

Declaration of the pure virtual function getIntersectionCount (for a more detailed description see the derived classes).

- virtual int getNumberOfDomains ()

  Declaration of the virtual function getNumberOfDomains (for a more detailed description see the derived classes).
- virtual void getPosition (double &returnX, double &returnY) const =0

  Declaration of the pure virtual function getPosition (for a more detailed description see the derived classes).
- virtual double getRadius () const =0

  Declaration of the pure virtual function getRadius (for a more detailed description see the derived classes).

7.4.1 Detailed Description

Linked to a specific CellLine object, this class defines the structure of the cellular nucleus to be irradiated and evaluates the dose absorbed during the interaction with a track.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2007–2015

The idea is that this class receives a set of tracks contained in a given Tracks object and corresponding to a certain spatial configuration of ion transversals, with the objective of:

- superimposing the tracks in order to compute the composite local dose distribution;
- transforming local doses in local number of lethal events by queries to the CellLine object;
- integrating the local dose and the local number of lethal events
- giving back the mean dose and mean survival estimation

Since these tasks can be in principle accomplished in several ways, this class has been declared pure virtual. The present derived implementation are Nucleus_MKM, Nucleus_tMKM, Nucleus_Integral, Nucleus_Integral_t, Nucleus_Pixel and Nucleus_MonteCarlo classes which implements the structure of the nucleus defined in the LEM I, II, III and MKM models.

For a more detailed description see the derived classes.

See also

Nucleus_MKM, Nucleus_tMKM, Nucleus_Integral, Nucleus_Integral_t, Nucleus_Pixel and Nucleus_MonteCarlo

Definition at line 32 of file Nucleus.h.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Survival::Nucleus::Nucleus ( ) [inline]

Constructor of a pure virtual class (empty).

Definition at line 37 of file Nucleus.h.

7.4.2.2 virtual Survival::Nucleus::~Nucleus ( ) [inline], [virtual]

Destructor of a pure virtual class (empty).

Definition at line 40 of file Nucleus.h.

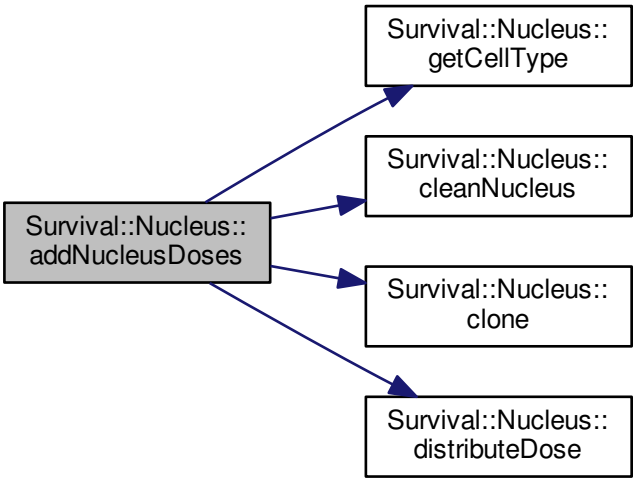
7.4.3 Member Function Documentation

7.4.3.1 virtual void Survival::Nucleus::addNucleusDoses (Nucleus &) [inline], [virtual]

Declaration of the virtual function getNucleusDoses (for a more detailed description see the derived classes).

Definition at line 43 of file Nucleus.h.

Here is the call graph for this function:

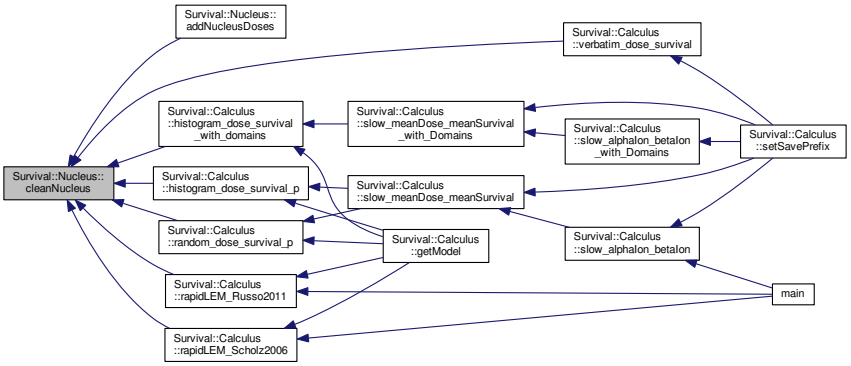


7.4.3.2 virtual void Survival::Nucleus::cleanNucleus ( ) [pure virtual]

Declaration of the pure virtual function cleanNucleus (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_Pixel, Survival::Nucleus_MKM, Survival::Nucleus_tMKM, Survival::Nucleus_↩ MonteCarlo, Survival::Nucleus_Integral_t, and Survival::Nucleus_Integral.

Here is the caller graph for this function:

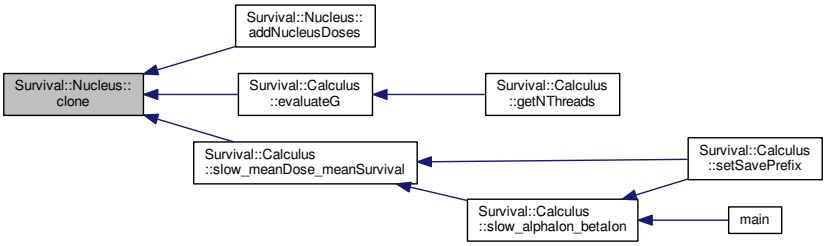


7.4.3.3 virtual Nucleus* Survival::Nucleus::clone (const CellLine &) [pure virtual]

Declaration of the pure virtual function clone (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_Pixel, Survival::Nucleus_MKM, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral_t, and Survival::Nucleus_Integral.

Here is the caller graph for this function:

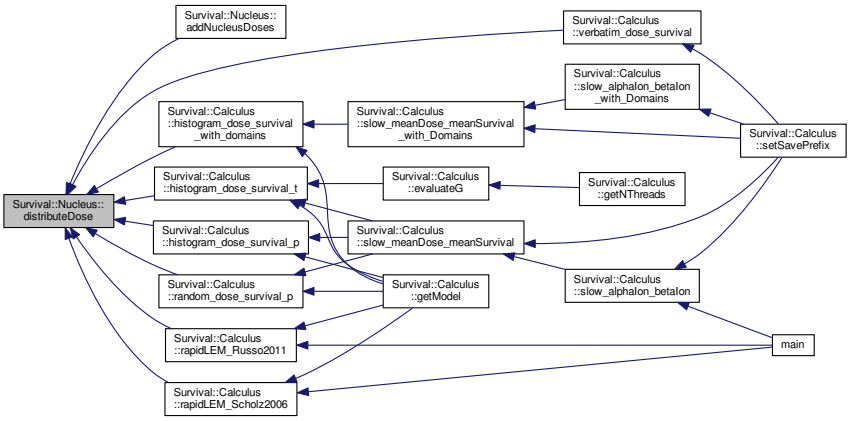


7.4.3.4 virtual void Survival::Nucleus::distributeDose (const Track & *track*) [pure virtual]

Declaration of the pure virtual function distributeDose (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_Pixel, Survival::Nucleus_MKM, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral, Survival::Nucleus_Integral_t, and Survival::Nucleus_MonteCarlo.

Here is the caller graph for this function:



7.4.3.5 virtual void Survival::Nucleus::distributeDose (const Tracks & *tracks*) [pure virtual]

Declaration of the pure virtual function distributeDose (for a more detailed description see the derived classes).

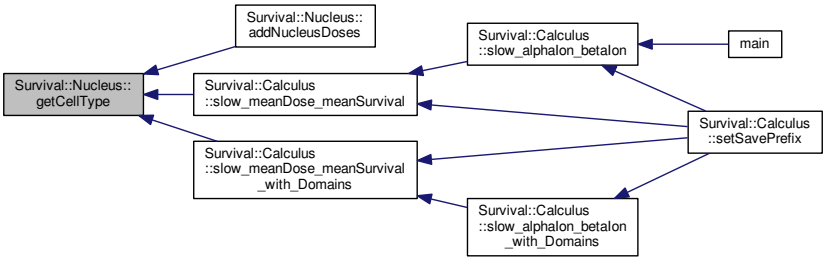
Implemented in Survival::Nucleus_Pixel, Survival::Nucleus_MKM, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral, Survival::Nucleus_Integral_t, and Survival::Nucleus_MonteCarlo.

7.4.3.6 virtual std::string Survival::Nucleus::getCellType ( ) const [pure virtual]

Declaration of the pure virtual function getCellType (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_Pixel, Survival::Nucleus_MKM, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral, and Survival::Nucleus_Integral_t.

Here is the caller graph for this function:



7.4.3.7 virtual double Survival::Nucleus::getDomainRadius ( ) [inline], [virtual]

Declaration of the virtual function getAC (for a more detailed description see the derived classes).

Declaration of the virtual function getDomainRadius (for a more detailed description see the derived classes).

Reimplemented in Survival::Nucleus_MKM, and Survival::Nucleus_tMKM.

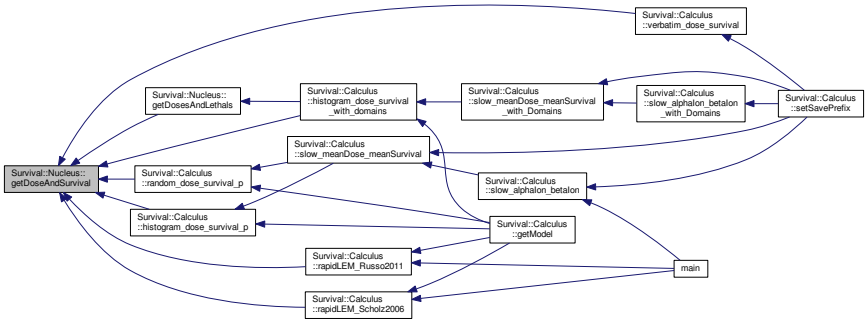
Definition at line 64 of file Nucleus.h.

7.4.3.8 virtual void Survival::Nucleus::getDoseAndSurvival (double & *dose*, double & *doseUncertainty*, double & *survival*, double & *survivalUncertainty*) const [pure virtual]

Declaration of the pure virtual function getDoseAndSurvival (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_MKM, Survival::Nucleus_Pixel, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral_t, and Survival::Nucleus_Integral.

Here is the caller graph for this function:

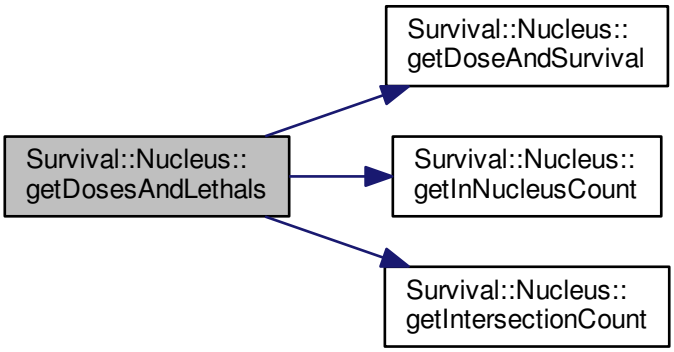


7.4.3.9 virtual void Survival::Nucleus::getDosesAndLethals (std::vector< double > & , std::vector< double > & , std::vector< double > & , std::vector< double > &) [inline], [virtual]

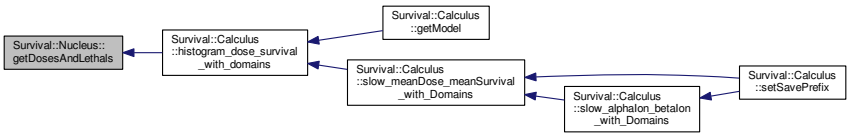
Declaration of the virtual function getDosesandLethals (for a more detailed description see the derived classes).

Definition at line 67 of file Nucleus.h.

Here is the call graph for this function:



Here is the caller graph for this function:

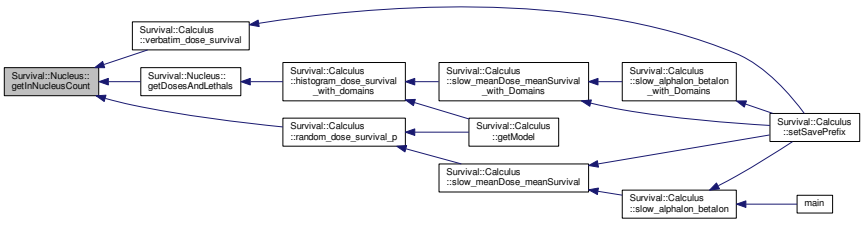


7.4.3.10 virtual int Survival::Nucleus::getInNucleusCount ( ) const [pure virtual]

Declaration of the pure virtual function getInNucleusCount (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_MKM, Survival::Nucleus_Pixel, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral_t, and Survival::Nucleus_Integral.

Here is the caller graph for this function:

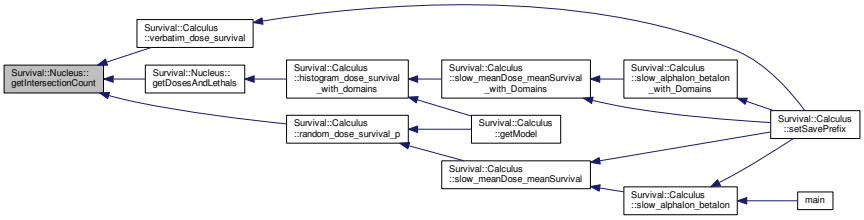


7.4.3.11 virtual int Survival::Nucleus::getIntersectionCount ( ) const [pure virtual]

Declaration of the pure virtual function getIntersectionCount (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_MKM, Survival::Nucleus_Pixel, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral_t, and Survival::Nucleus_Integral.

Here is the caller graph for this function:



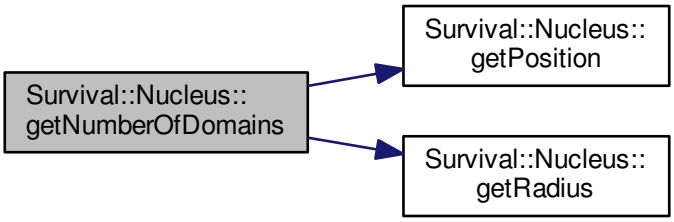
7.4.3.12 virtual int Survival::Nucleus::getNumberOfDomains ( ) [inline], [virtual]

Declaration of the virtual function getNumberOfDomains (for a more detailed description see the derived classes).

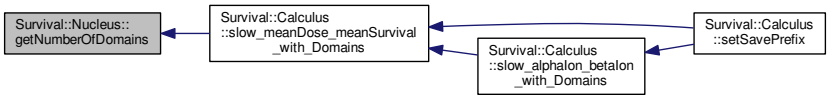
Reimplemented in Survival::Nucleus_MKM, and Survival::Nucleus_tMKM.

Definition at line 82 of file Nucleus.h.

Here is the call graph for this function:



Here is the caller graph for this function:

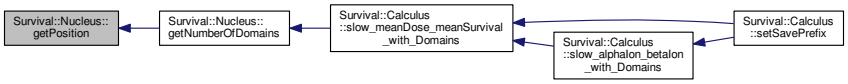


7.4.3.13 virtual void Survival::Nucleus::getPosition (double & *returnX,* double & *returnY*) const [pure virtual]

Declaration of the pure virtual function getPosition (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_MKM, Survival::Nucleus_Pixel, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral_t, and Survival::Nucleus_Integral.

Here is the caller graph for this function:

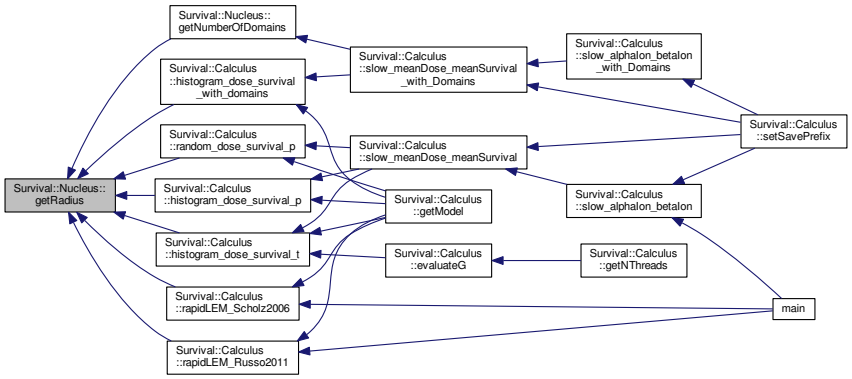


7.4.3.14 virtual double Survival::Nucleus::getRadius ( ) const [pure virtual]

Declaration of the pure virtual function getRadius (for a more detailed description see the derived classes).

Implemented in Survival::Nucleus_MKM, Survival::Nucleus_Pixel, Survival::Nucleus_tMKM, Survival::Nucleus_↩ Integral_t, and Survival::Nucleus_Integral.

Here is the caller graph for this function:



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

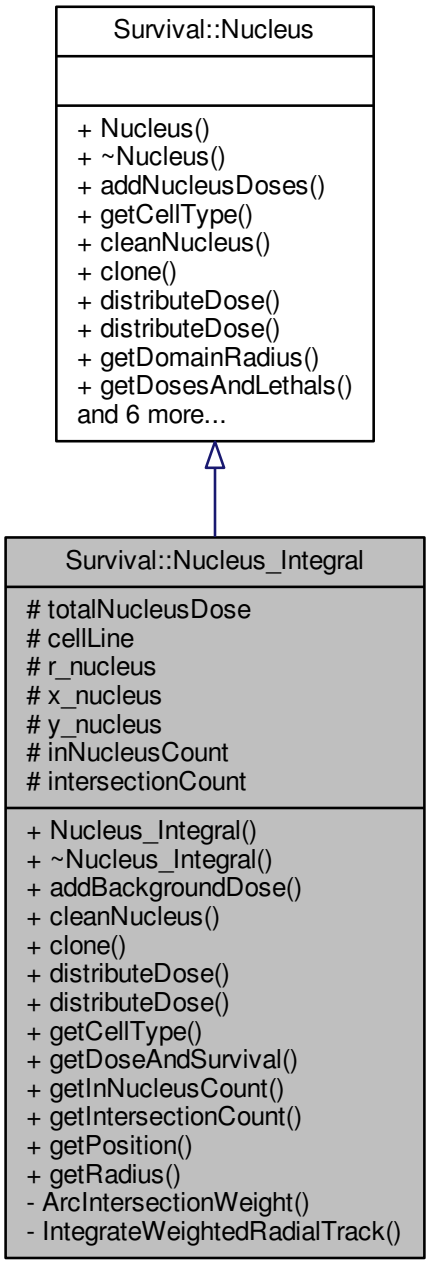
- include/Nucleus.h

7.5 Survival::Nucleus_Integral Class Reference

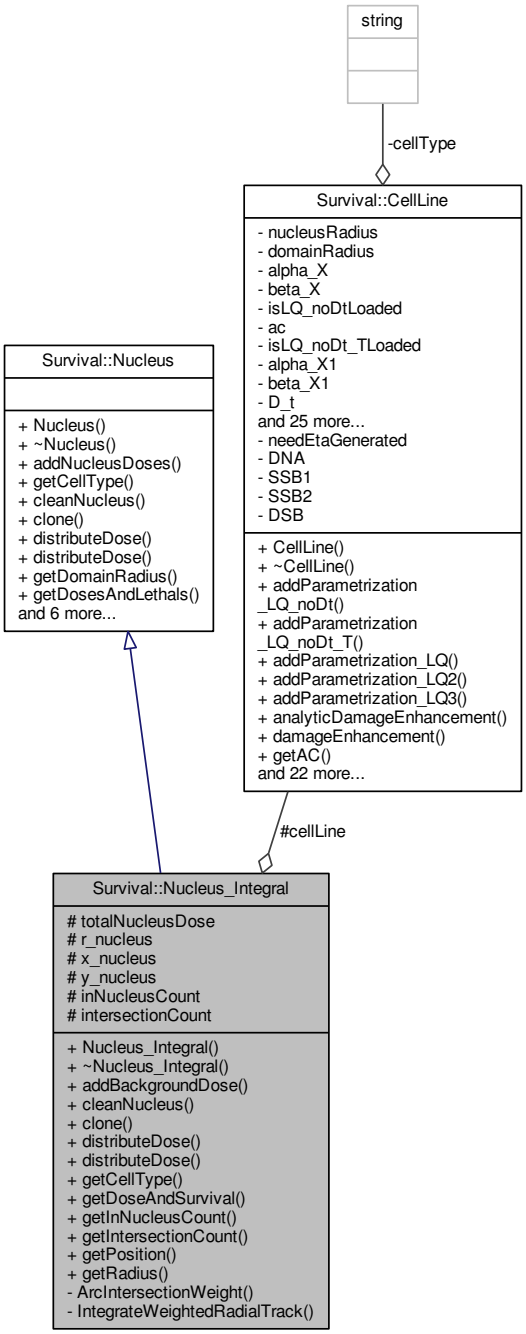
Implements a nucleus as a 2D circular object and provides methods to evaluate the number of lethal events observed.

`#include <Nucleus_Integral.h>`

Inheritance diagram for Survival::Nucleus_Integral:



Collaboration diagram for Survival::Nucleus_Integral:



Public Member Functions

- Nucleus_Integral (const CellLine &cellLineRef, const double xPosition=0.0, const double yPosition=0.0)

  Constructor. Instantiates and sets the object.
- virtual ~Nucleus_Integral ()

  Destructor.
- void addBackgroundDose (const double dose)

Adds a constant value of dose absorbed by the nucleus.

- virtual void cleanNucleus ()

  Resets to zero inNucleusCount and intersectionCount counters and the total dose absorbed (totalNucleusDose).
- virtual Nucleus_Integral * clone (const CellLine &)

  Returns a pointer to a new Nucleus_Integral object. It not really a clone but a new clean object.
- virtual void distributeDose (const Track &track)

  Integrates the radial profile of the track in the intersection area with the nucleus to evaluate the dose deposited.
- virtual void distributeDose (const Tracks &tracks)

  Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for every track of the container.
- virtual std::string getCellType () const

  Returns the name of the cell line to which the nucleus belongs.
- virtual void getDoseAndSurvival (double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty) const

  Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated survival, with respective uncertainties.
- virtual int getInNucleusCount () const

  Returns the number of times that the nucleus has been crossed through by a Particle.
- virtual int getIntersectionCount () const

  Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.
- virtual void getPosition (double &returnX, double &returnY) const

  Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.
- virtual double getRadius () const

  Returns the radius of the nucleus expressed in um.

Protected Attributes

- double totalNucleusDose

  The total dose absorbed by the nucleus, expressed in Gy.
- const CellLine & cellLine

  A reference to a CellLine object where the characteristics of the cell line to which the nucleus belongs are stored.
- double r_nucleus

  The radius of the nucleus, expressed in um.
- const double x_nucleus

  The position of the center of the nucleus (`x` coordinate) referred to the beam axis, expressed in mm.
- const double y_nucleus

  The position of the center of the nucleus (`y` coordinate) referred to the beam axis, expressed in mm.
- int inNucleusCount

  The number of times that the nucleus has been crossed through by a Particle.
- int intersectionCount

  The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

Private Member Functions

- double ArcIntersectionWeight (double r, double b)

  Evaluate the length of the arc of circumference (expressed in radians) derived from the intersection between the nucleus and a circumference whose radius is equal to `r` and whose center is far `b` from the center of the nucleus.
- double IntegrateWeightedRadialTrack (const Track &track, double rMin, double rMax, double b, double &area, double step)

  Performs the integral of the radial profile of the track in the intersection area with the nucleus.

7.5.1 Detailed Description

Implements a nucleus as a 2D circular object and provides methods to evaluate the number of lethal events observed.

Author

Andrea Attili
Lorenzo Manganaro
Lorenzo Marengo
Germano Russo

Date

2011–2015

This class implements a nucleus as a 2D circular object that represents the cross section shown by the cell to the particle. It contains a reference to the cell line to which the nucleus belongs, which is used to get informations such as the radius. It provides method to evaluate the dose deposited by the radiation in the interaction with the nucleus itself and methods to get the dose absorbed and the associated cellular survival.

See also

Nucleus_MKM and Nucleus_Integral_t

Definition at line 20 of file Nucleus_Integral.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Nucleus_Integral::Nucleus_Integral (const CellLine & *cellLineRef*, const double *xPosition* = 0.0, const double *yPosition* = 0.0)

Constructor. Instantiates and sets the object.

Parameters

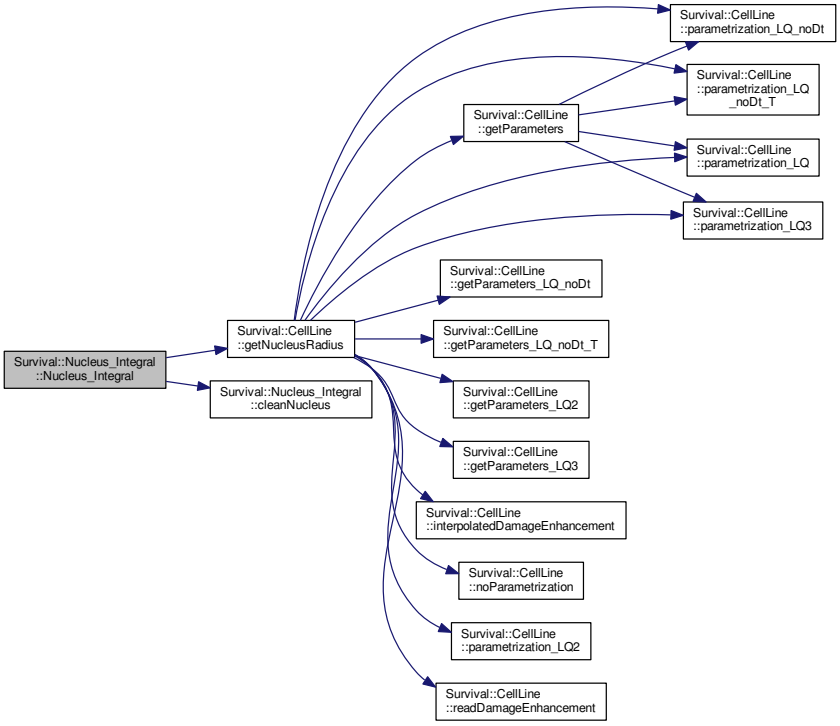
cellLineRef	A reference to the corresponding CellLine.
xPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.
yPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.

See also

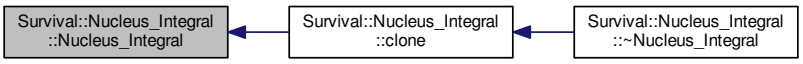
cleanNucleus() and Nucleus_MKM::createDomains()

Definition at line 22 of file Nucleus_Integral.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

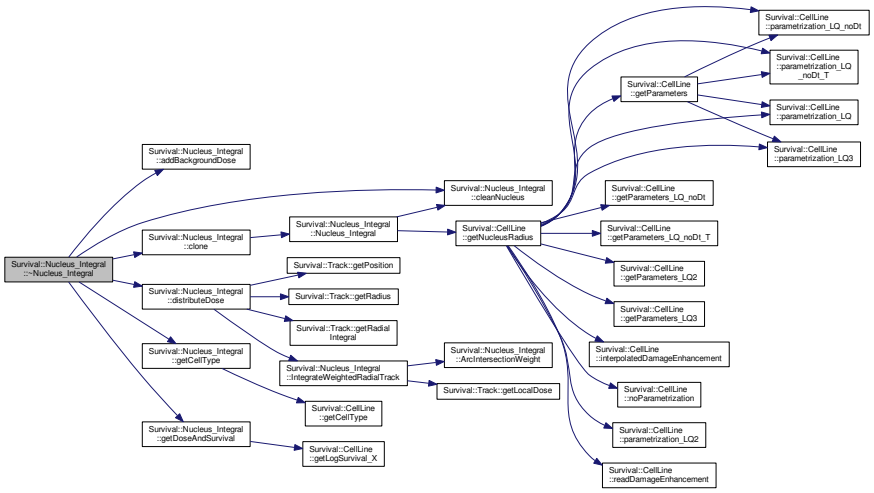


7.5.2.2 virtual Survival::Nucleus_Integral::~Nucleus_Integral ( ) [inline], [virtual]

Destructor.

Definition at line 37 of file Nucleus_Integral.h.

Here is the call graph for this function:



7.5.3 Member Function Documentation

7.5.3.1 void Nucleus_Integral::addBackgroundDose (const double *dose*)

Adds a constant value of dose absorbed by the nucleus.

The method updates the totalNucleusDose value, adding a constant value chosen by the user.

Parameters

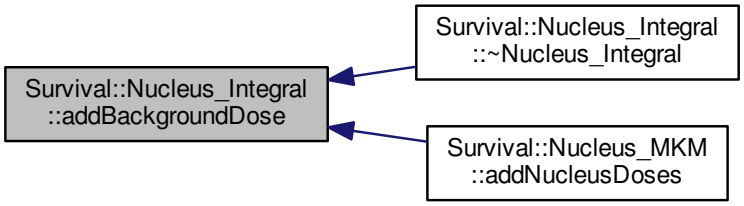
dose	The value of dose absorbed to be added, expressed in Gy.

See also

Nucleus_MKM::addBackgroundDose()

Definition at line 36 of file Nucleus_Integral.cpp.

Here is the caller graph for this function:



7.5.3.2 double Nucleus_Integral::ArcIntersectionWeight (double *r*, double *b*) [private]

Evaluate the length of the arc of circumference (expressed in radians) derived from the intersection between the nucleus and a circumference whose radius is equal to `r` and whose center is far `b` from the center of the nucleus.

The calculus is performed by considering all possible cases and relative subcases:

1. if `b` is smaller than the radius of the nucleus (R_N), i.e. center inside the nucleus, and:
   - if $r < R_N - b$ returns 2π (full circle);
   - else returns the length of the arc, that is $2 \arccos\left(\frac{b}{2r} + \frac{r}{2b} - \frac{R_N^2}{2br}\right)$.
2. if the center is outside the nucleus and:
   - if $r < b - R_N$ no intersection occurs, therefore it returns 0;
   - else if $r < b + R_N$ it returns the length of the arc, that is $2 \arccos\left(\frac{b}{2r} + \frac{r}{2b} - \frac{R_N^2}{2br}\right)$. Else no intersection occurs and it returns 0.

Parameters

r	The radius of the circle, expressed in um.
b	The impact parameter of the track, expressed in um.

Returns

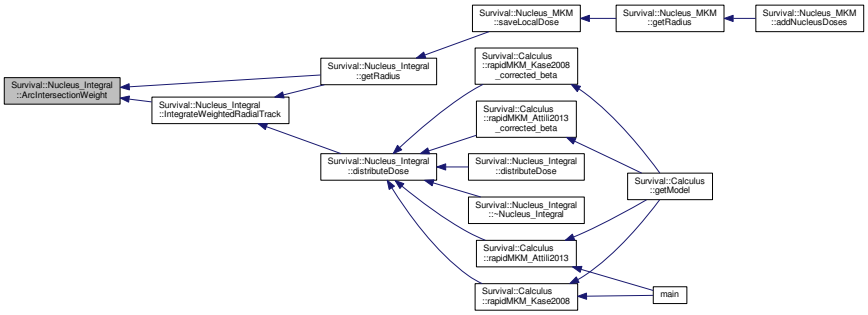
The length of the arc of circumference, expressed in radians.

See also

IntegrateWeightedRadialTrack() and distributeDose()

Definition at line 167 of file Nucleus_Integral.cpp.

Here is the caller graph for this function:



7.5.3.3 void Nucleus_Integral::cleanNucleus ( ) [virtual]

Resets to zero inNucleusCount and intersectionCount counters and the total dose absorbed (totalNucleusDose).

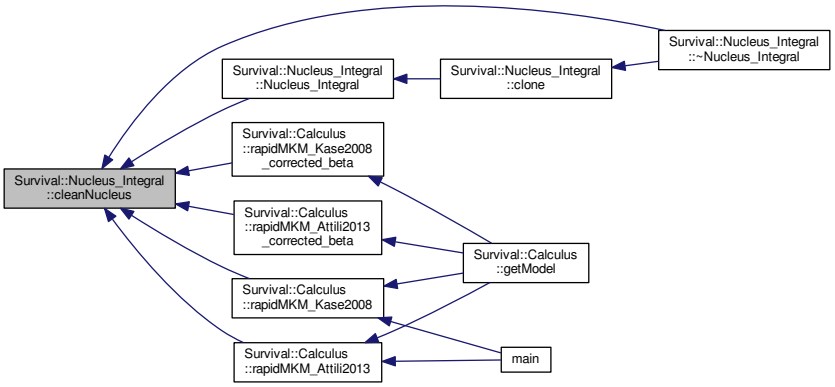
See also

Nucleus_MKM::cleanNucleus()

Implements Survival::Nucleus.

Definition at line 43 of file Nucleus_Integral.cpp.

Here is the caller graph for this function:



7.5.3.4 Nucleus_Integral * Nucleus_Integral::clone (const CellLine & *cellLine*) [virtual]

Returns a pointer to a new Nucleus_Integral object. It not really a clone but a new clean object.

Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

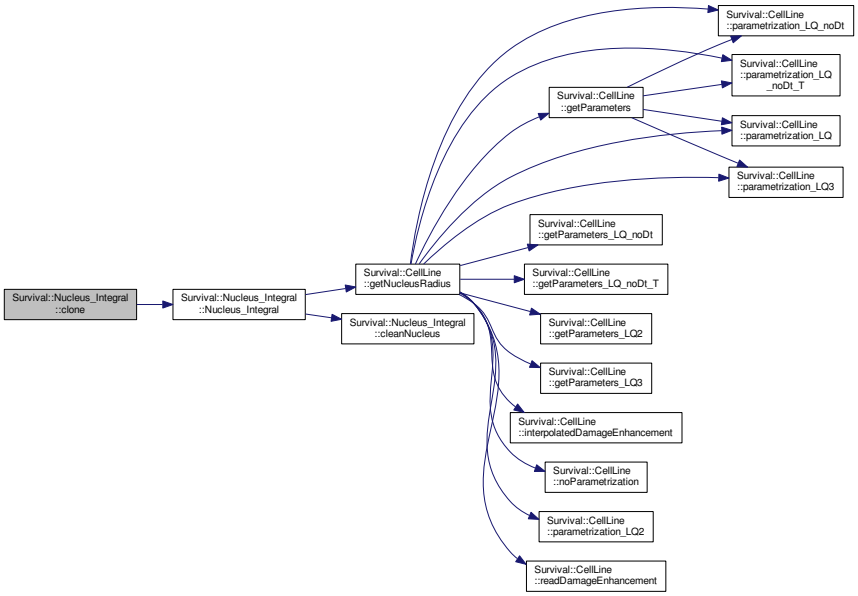
Note

To create a real clone of another nucleus, a better implementation of the copy constructor is needed.

Implements Survival::Nucleus.

Definition at line 52 of file Nucleus_Integral.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

7.5.3.5 void Nucleus_Integral::distributeDose (const Track & *track*) [virtual]

Integrates the radial profile of the track in the intersection area with the nucleus to evaluate the dose deposited.

It's a very tricky and clever method that performs the integral of the radial profile of the track in its common area with the nucleus. First of all it centers the origin of the reference system (RS) on the position of the nucleus, identifying the position of the track in the new RS. Then it examines all possible cases:

- Track inside the nucleus;
- Track outside the nucleus but interacting with it;
- Track non interacting with the nucleus (in this case it trivially returns nothing). In the first two cases the respective counter is incremented (inNucleusCount or intersectionCount), then the method identifies the intersection area and calls IntegrateWeightedRadialTrack() that performs the integral in the area defined and evaluates the area itself (unless the Track is all contained in the nucleus: in that case the function Track::getRadialIntegral() is called, and the value of the area is calculated *manually* as πR^2).

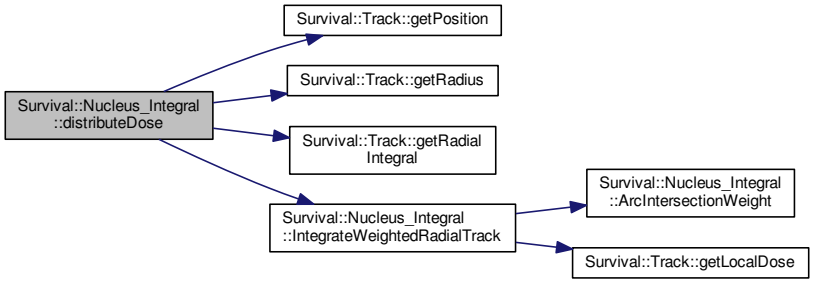
Parameters

track	The Track of the particle interacting with the nucleus.

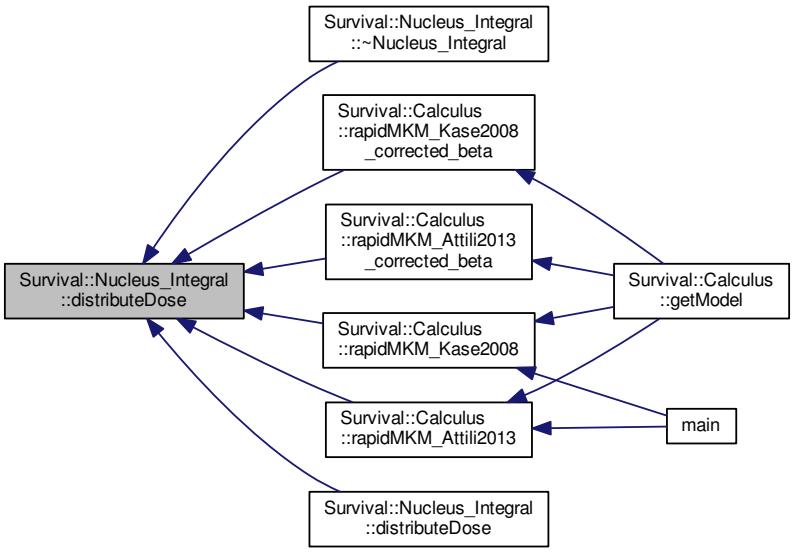
Implements Survival::Nucleus.

Definition at line 67 of file Nucleus_Integral.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.6 void Nucleus_Integral::distributeDose (const Tracks & *tracks*) [virtual]

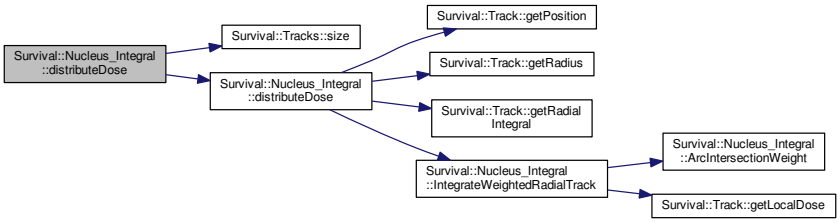
Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for every track of the container.

Since the Tracks class is a container for Track objects, this method calls distributeDose(const Track &track) in a `for` loop over each track contained in the Tracks object.

Implements Survival::Nucleus.

Definition at line 129 of file Nucleus_Integral.cpp.

Here is the call graph for this function:



7.5.3.7 string Nucleus_Integral::getCellType ( ) const [virtual]

Returns the name of the cell line to which the nucleus belongs.

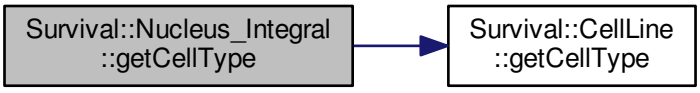
Returns

A `string` corresponding to the name of the cell line to which the nucleus belongs, getting the information by cellLine.

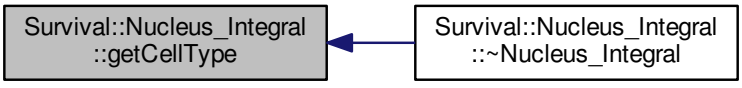
Implements Survival::Nucleus.

Definition at line 137 of file Nucleus_Integral.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.8 void Nucleus_Integral::getDoseAndSurvival (double & *dose,* double & *doseUncertainty,* double & *survival,* double & *survivalUncertainty*) const [virtual]

Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated survival, with respective uncertainties.

The dose absorbed coincides with totalNucleusDose, the survival is evaluated by means of an exponential function of the lethal events observed, evaluated through the CellLine::getLogSurvival_X() method.

Parameters

dose	The total dose absorbed by the nucleus, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival associated to the dose absorbed by the nucleus, passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.

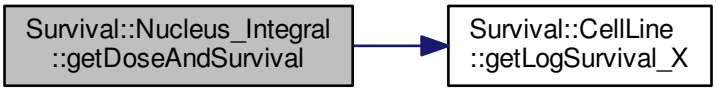
Note

The method was thought to associate also an uncertainty to dose and survival, but this possibility hasn't been implemented yet, therefore actually -1 is assigned to those values.

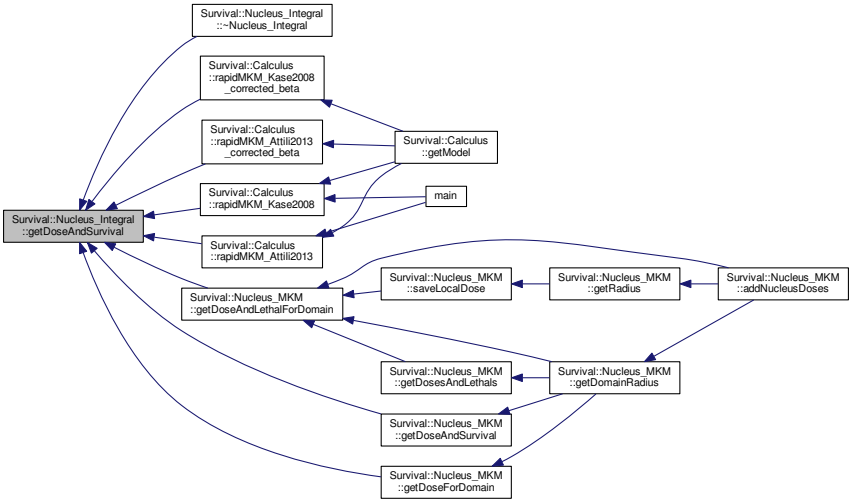
Implements Survival::Nucleus.

Definition at line 144 of file Nucleus_Integral.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.9 virtual int Survival::Nucleus_Integral::getInNucleusCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus has been crossed through by a Particle.

Returns

inNucleusCount The number of times that the nucleus has been crossed through by a Particle.

See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 110 of file Nucleus_Integral.h.

7.5.3.10 virtual int Survival::Nucleus_Integral::getIntersectionCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

Returns

intersectionCount The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

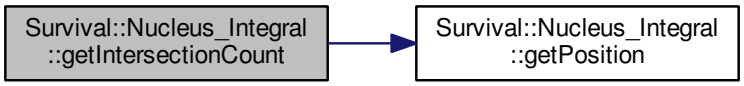
See also

distributeDose()

Implements Survival::Nucleus.

Definition at line 118 of file Nucleus_Integral.h.

Here is the call graph for this function:



7.5.3.11 void Nucleus_Integral::getPosition (double & *returnX*, double & *returnY*) const [virtual]

Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the nucleus referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `y` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.

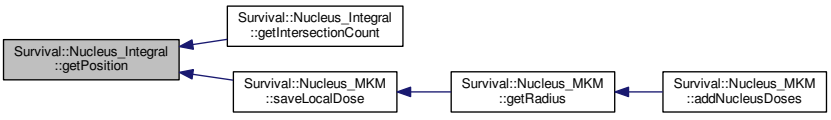
See also

Nucleus_MKM::getPosition()

Implements Survival::Nucleus.

Definition at line 158 of file Nucleus_Integral.cpp.

Here is the caller graph for this function:



7.5.3.12 virtual double Survival::Nucleus_Integral::getRadius ( ) const [inline], [virtual]

Returns the radius of the nucleus expressed in um.

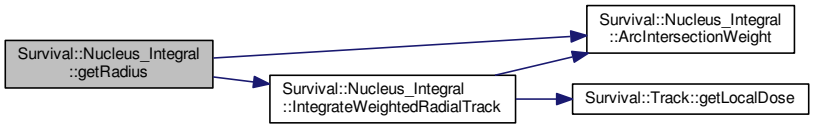
Returns

r_nucleus The radius of the nucleus expressed in um.

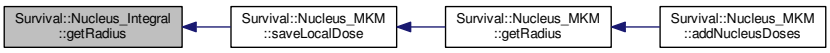
Implements Survival::Nucleus.

Definition at line 136 of file Nucleus_Integral.h.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.3.13 double Nucleus_Integral::IntegrateWeightedRadialTrack (const Track & *track*, double *rMin*, double *rMax*, double *b*, double & *area*, double *step*) [private]

Performs the integral of the radial profile of the track in the intersection area with the nucleus.

The problem is to integrate a function over a complex area originated by the random intersection of two circles. The way this method performs the task is to evaluate it numerically, dividing the area (or the radius to be covered) in a number of finite (small) step and evaluating for each step the length of the arc of circumference by means of the ArcIntersectionWeight() method. The sum of all these lengths is equal to the intersection area and it's overwritten in the correspondent parameter. For each step, defined by a specific radius, the method gets the local dose from the track (Track::getLocalDose()) and the integral is evaluated considering a step function constructed in this way. Finally the value of the integral is normalized over the intersection area.

Parameters

track	A reference to the Track of the particle interacting with the nucleus.
rMin	Minimum radius, lower limit of integration, expressed in um.
rMax	Maximum radius, upper limit of integration, expressed in um.
b	The impact parameters of the track, expressed in um.
area	The intersection area between Track and Nucleus, passed by reference to be overwritten with its correct value.
step	The length of the radial step of integration.

Returns

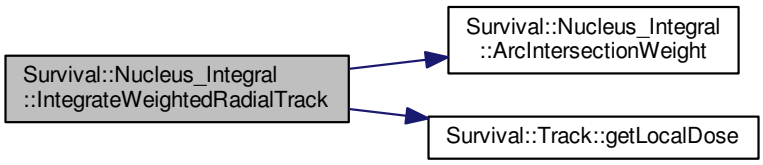
The value of the integral normalized over the intersection area, expressed in Gy.

See also

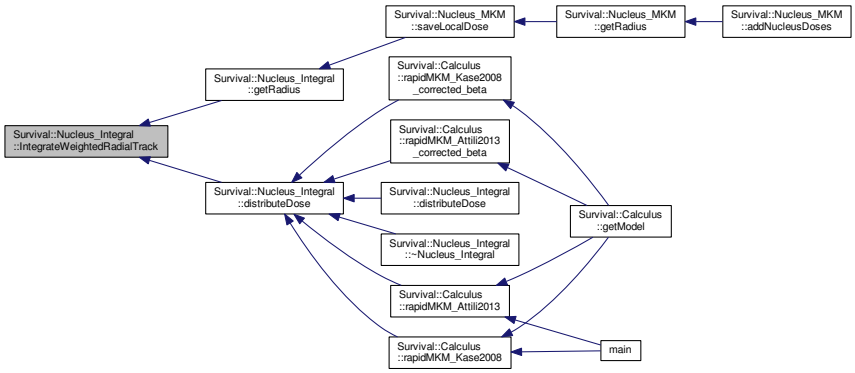
distributeDose()

Definition at line 216 of file Nucleus_Integral.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.4 Member Data Documentation

7.5.4.1 const CellLine& Survival::Nucleus_Integral::cellLine [protected]

A reference to a CellLine object where the characteristics of the cell line to which the nucleus belongs are stored.

Definition at line 190 of file Nucleus_Integral.h.

7.5.4.2 int Survival::Nucleus_Integral::inNucleusCount [protected]

The number of times that the nucleus has been crossed through by a Particle.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 210 of file Nucleus_Integral.h.

7.5.4.3 int Survival::Nucleus_Integral::intersectionCount [protected]

The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 216 of file Nucleus_Integral.h.

7.5.4.4 double Survival::Nucleus_Integral::r_nucleus [protected]

The radius of the nucleus, expressed in um.

It's instantiated in the constructor getting the value from the CellLine object representing the cell line to which the nucleus belongs.

See also

Nucleus_Integral()

Definition at line 198 of file Nucleus_Integral.h.

7.5.4.5 double Survival::Nucleus_Integral::totalNucleusDose [protected]

The total dose absorbed by the nucleus, expressed in Gy.

The value is initially set to zero in the constructor (through the cleanNucleus() method) and then updated by distributeDose() when an interaction with a Particle occurs.

Definition at line 187 of file Nucleus_Integral.h.

7.5.4.6 const double Survival::Nucleus_Integral::x_nucleus [protected]

The position of the center of the nucleus (x coordinate) referred to the beam axis, expressed in mm.

Definition at line 201 of file Nucleus_Integral.h.

7.5.4.7 const double Survival::Nucleus_Integral::y_nucleus [protected]

The position of the center of the nucleus (y coordinate) referred to the beam axis, expressed in mm.

Definition at line 204 of file Nucleus_Integral.h.

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

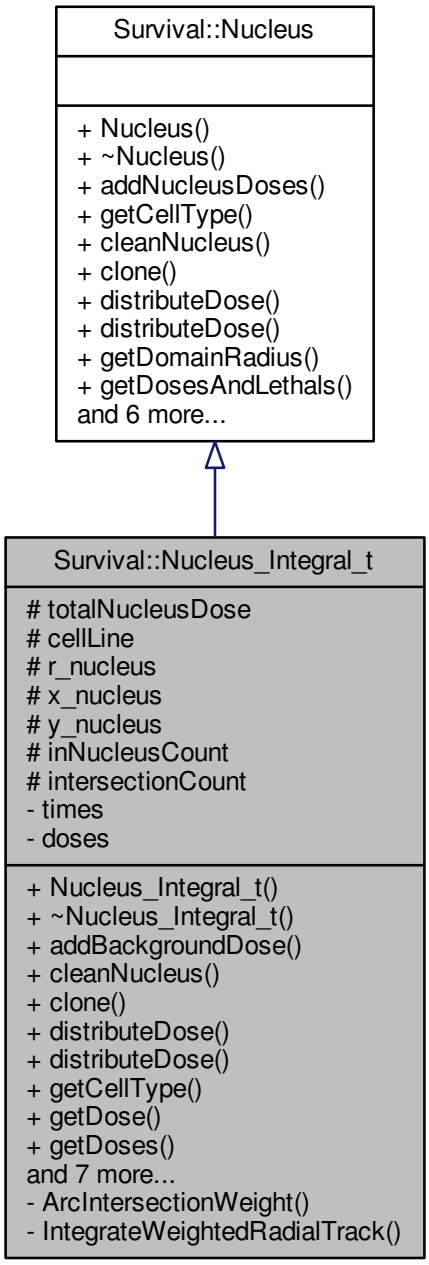
- include/Nucleus_Integral.h
- src/Nucleus_Integral.cpp

7.6 Survival::Nucleus_Integral_t Class Reference

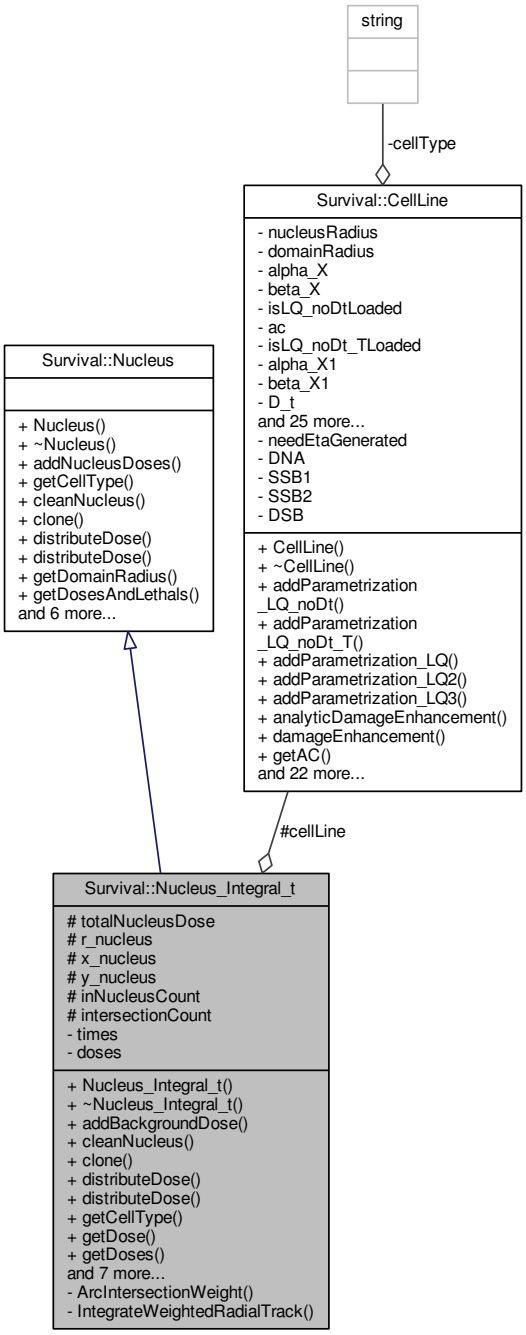
Implements a nucleus as a 2D circular object and provides methods to evaluate the number of lethal events observed taking into account the time structure of the irradiation.

`#include <Nucleus_Integral_t.h>`

Inheritance diagram for Survival::Nucleus_Integral_t:



Collaboration diagram for Survival::Nucleus_Integral_t:



Public Member Functions

- Nucleus_Integral_t (const CellLine &cellLineRef, const double xPosition=0.0, const double yPosition=0.0)

  Constructor. Instantiates and sets the object.
- virtual ~Nucleus_Integral_t ()

  Destructor.
- void addBackgroundDose (const double dose, const double t)

Adds a constant value of dose absorbed by the nucleus in a specific instant.

- virtual void cleanNucleus ()

  Resets to zero inNucleusCount and intersectionCount counters, the total dose absorbed (totalNucleusDose) and the vectors containing the history of times and doses deposited.
- virtual Nucleus_Integral_t * clone (const CellLine &)

  Returns a pointer to a new Nucleus_Integral_t object. It not really a clone but a new clean object.
- virtual void distributeDose (const Track &track)

  Integrates the radial profile of the track in the intersection area with the nucleus to evaluate the dose deposited.
- virtual void distributeDose (const Tracks &tracks)

  Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for every track of the container.
- virtual std::string getCellType () const

  Returns the name of the cell line to which the nucleus belongs.
- void getDose (double &dose) const

  Returns the total dose absorbed by the nucleus, expressed in Gy, overwriting a double variable passed by reference.
- std::vector< double > getDoses () const

  Returns the vector representing each dose deposited in the nucleus by a Particle when an interaction occurs.
- virtual void getDoseAndSurvival (double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty) const

  Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated survival, with respective uncertainties.
- void getDoseAndLethals (double &dose, double &doseUncertainty, double &lethals, double &lethals↩ Uncertainty) const

  Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and associated lethal events, with respective uncertainties.
- virtual int getInNucleusCount () const

  Returns the number of times that the nucleus has been crossed through by a Particle.
- virtual int getIntersectionCount () const

  Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.
- virtual void getPosition (double &returnX, double &returnY) const

  Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.
- virtual double getRadius () const

  Returns the radius of the nucleus expressed in um.
- std::vector< double > getTimes () const

  Returns the vector representing each instant in which an interaction with a Particle occurs. Each value is expressed in hours.

Protected Attributes

- double totalNucleusDose

  The total dose absorbed by the nucleus, expressed in Gy.
- const CellLine & cellLine

  A reference to a CellLine object where the characteristics of the cell line to which the nucleus belongs are stored.
- double r_nucleus

  The radius of the nucleus, expressed in um.
- const double x_nucleus

  The position of the center of the nucleus (`x` coordinate) referred to the beam axis, expressed in mm.
- const double y_nucleus

  The position of the center of the nucleus (`y` coordinate) referred to the beam axis, expressed in mm.
- int inNucleusCount

  The number of times that the nucleus has been crossed through by a Particle.
- int intersectionCount

  The number of times that the nucleus interacted with a Particle.

Private Member Functions

- double ArcIntersectionWeight (double r, double b)

  Evaluate the length of the arc of circumference (expressed in radians) derived from the intersection between the nucleus and a circumference whose radius is equal to `r` *and whose center is far* `b` *from the center of the nucleus.*
- double IntegrateWeightedRadialTrack (const Track &track, double rMin, double rMax, double b, double &area, double step)

  Performs the integral of the radial profile of the track in the intersection area with the nucleus.

Private Attributes

- std::vector< double > times

  Vector containing the sequence of interaction times (expressed in hours), each elements is associated to one interaction.
- std::vector< double > doses

  Vector containing the sequence of doses deposited in the nucleus, expressed in Gy, each elements is associated to one interaction.

7.6.1 Detailed Description

Implements a nucleus as a 2D circular object and provides methods to evaluate the number of lethal events observed taking into account the time structure of the irradiation.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2015

This class inherits from the Nucleus pure virtual class and it has the same structure of the Nucleus_Integral class. It defines the nucleus used in the MCt-MKM model and its peculiarity is to provide some methods to evaluate the temporal effect of the irradiation keeping track of its time structure. Every time an interaction between the nucleus and a particle occurs, the time of the event and the dose deposited are stored in dedicated vectors to be passed at the Nucleus_tMKM class.

Definition at line 17 of file Nucleus_Integral_t.h.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Nucleus_Integral_t::Nucleus_Integral_t (const CellLine & *cellLineRef*, const double *xPosition* = `0.0`, const double *yPosition* = `0.0`)

Constructor. Instantiates and sets the object.

Parameters

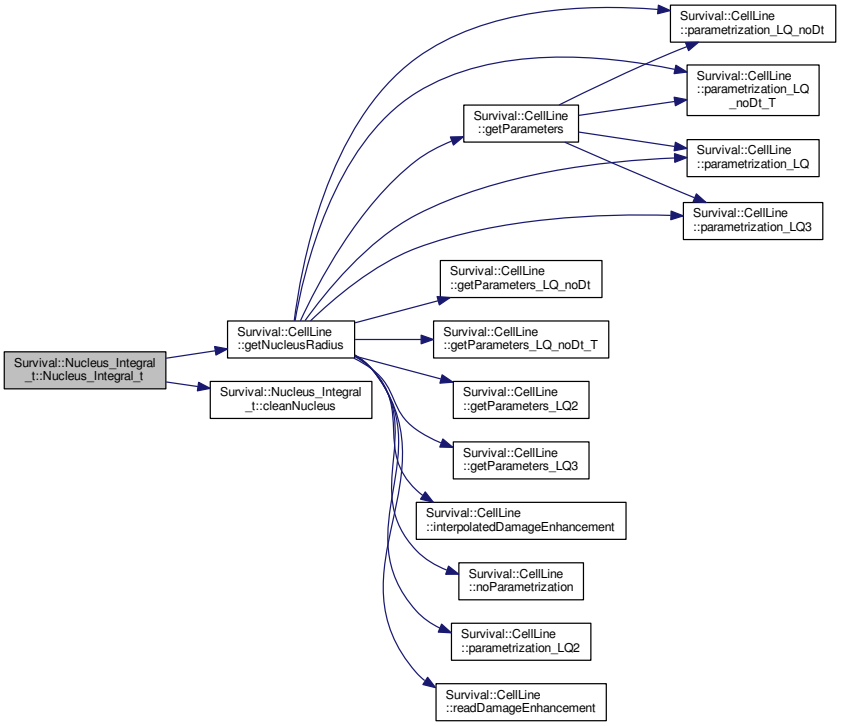
cellLineRef	A reference to the corresponding CellLine.
xPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.
yPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.

See also

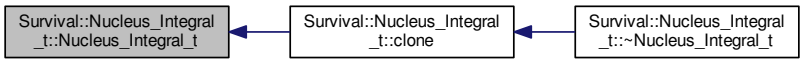
cleanNucleus() and Nucleus_tMKM::createDomains()

Definition at line 18 of file Nucleus_Integral_t.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

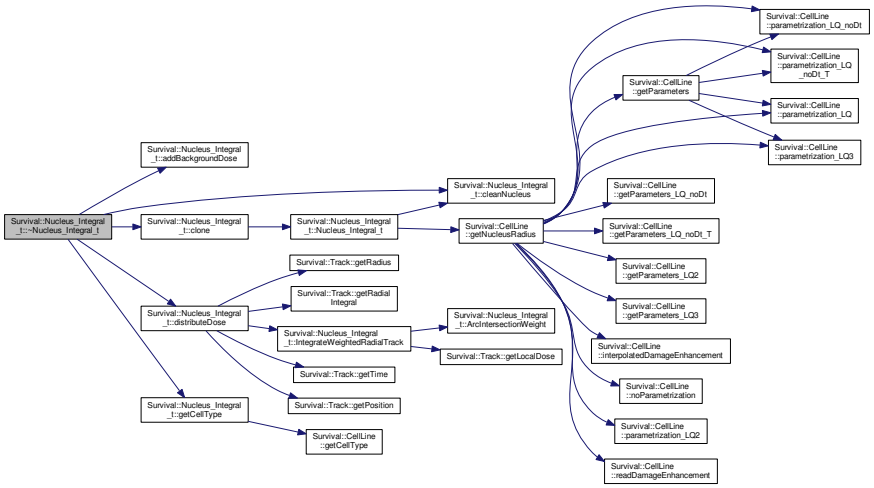


7.6.2.2 virtual Survival::Nucleus_Integral_t::~Nucleus_Integral_t ( ) [inline], [virtual]

Destructor.

Definition at line 37 of file Nucleus_Integral_t.h.

Here is the call graph for this function:



7.6.3 Member Function Documentation

7.6.3.1 void Nucleus_Integral_t::addBackgroundDose (const double *dose*, const double *t*)

Adds a constant value of dose absorbed by the nucleus in a specific instant.

The method added an element to doses and times vectors, representing a constant dose absorbed by the nucleus in a specific instant. The method updates also the totalNucleusDose value, adding a constant value chosen by the user.

Parameters

dose	The value of dose absorbed to be added, expressed in Gy.
t	The time associated to the dose added, expressed in hours.

See also

Nucleus_tMKM::addBackgroundDose()

Definition at line 49 of file Nucleus_Integral_t.cpp.

Here is the caller graph for this function:

7.6.3.2 double Nucleus_Integral_t::ArcIntersectionWeight (double *r*, double *b*) [private]

Evaluate the length of the arc of circumference (expressed in radians) derived from the intersection between the nucleus and a circumference whose radius is equal to r and whose center is far b from the center of the nucleus.

The calculus is performed by considering all possible cases and relative subcases:

1. if b is smaller than the radius of the nucleus (R_N), i.e. center inside the nucleus, and:
   - if $r < R_N - b$ returns 2π (full circle);
   - else returns the length of the arc, that is $2 \arccos\left(\frac{b}{2r} + \frac{r}{2b} - \frac{R_N^2}{2br}\right)$.
2. if the center is outside the nucleus and:
   - if $r < b - R_N$ no intersection occurs, therefore it returns 0;
   - else if $r < b + R_N$ it returns the length of the arc, that is $2 \arccos\left(\frac{b}{2r} + \frac{r}{2b} - \frac{R_N^2}{2br}\right)$. Else no intersection occurs and it returns 0.

Parameters

r	The radius of the circle, expressed in um.
b	The impact parameter of the track, expressed in um.

Returns

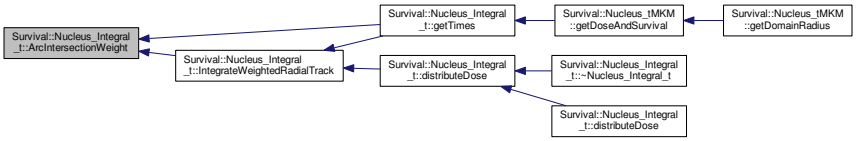
The length of the arc of circumference, expressed in radians.

See also

IntegrateWeightedRadialTrack() and distributeDose()

Definition at line 59 of file Nucleus_Integral_t.cpp.

Here is the caller graph for this function:



7.6.3.3 void Nucleus_Integral_t::cleanNucleus ( ) [virtual]

Resets to zero inNucleusCount and intersectionCount counters, the total dose absorbed (totalNucleusDose) and the vectors containing the history of times and doses deposited.

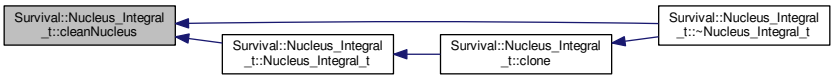
See also

Nucleus_tMKM::cleanNucleus()

Implements Survival::Nucleus.

Definition at line 86 of file Nucleus_Integral_t.cpp.

Here is the caller graph for this function:



7.6.3.4 Nucleus_Integral_t * Nucleus_Integral_t::clone (const CellLine & *cellLine*) [virtual]

Returns a pointer to a new Nucleus_Integral_t object. It not really a clone but a new clean object.

Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

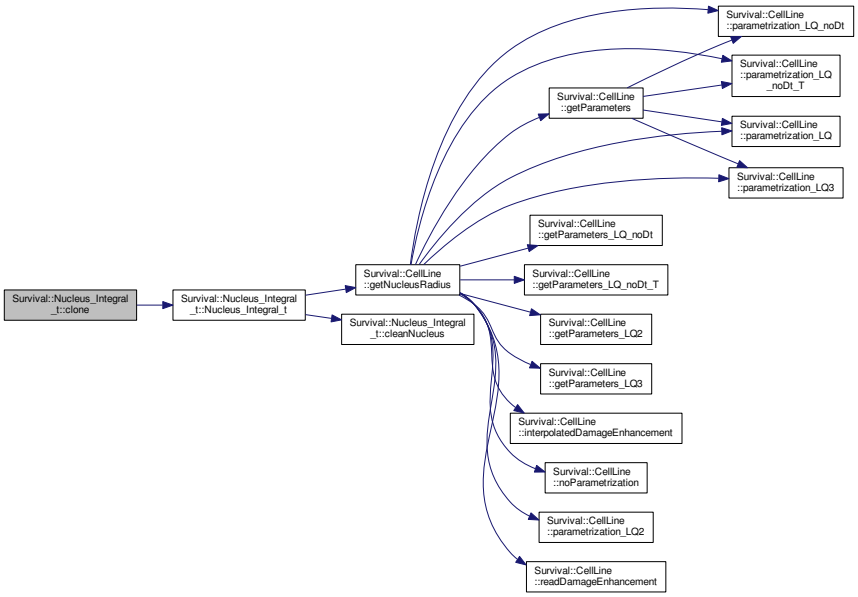
Note

To create a real clone of another nucleus, a better implementation of the copy constructor is needed.

Implements Survival::Nucleus.

Definition at line 97 of file Nucleus_Integral_t.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

7.6.3.5 void Nucleus_Integral_t::distributeDose (const Track & *track*) [virtual]

Integrates the radial profile of the track in the intersection area with the nucleus to evaluate the dose deposited.

It has the same implementation of Nucleus_Integral::distributeDose(), but once the dose deposited is calculated this method adds an element to doses and times vectors (through the `push.back()` method defined in the STL): the dose is the one calculated by the function while the time is got by means of the Track::getTime() method.

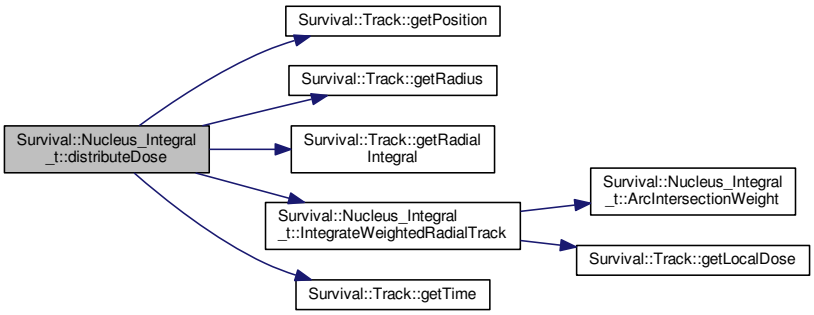
Parameters

track	The Track of the particle interacting with the nucleus.

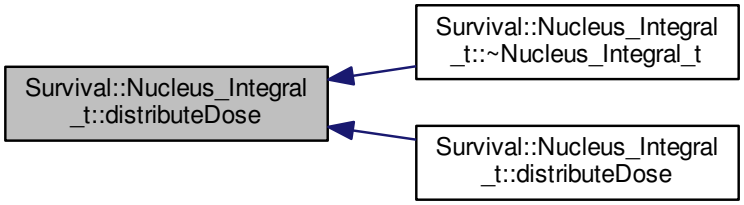
Implements Survival::Nucleus.

Definition at line 105 of file Nucleus_Integral_t.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.6 void Nucleus_Integral_t::distributeDose (const Tracks & *tracks*) [virtual]

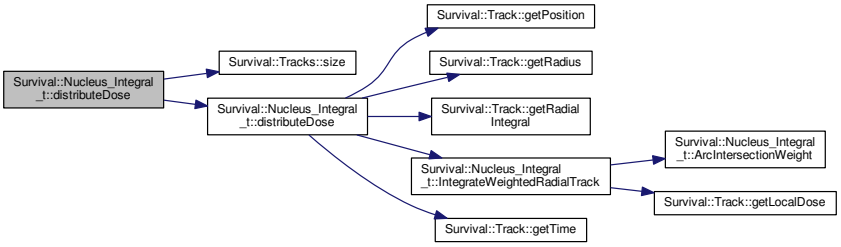
Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for every track of the container.

Since the Tracks class is a container for Track objects, this method calls distributeDose(const Track &track) in a `for` loop over each track contained in the Tracks object.

Implements Survival::Nucleus.

Definition at line 171 of file Nucleus_Integral_t.cpp.

Here is the call graph for this function:



7.6.3.7 string Nucleus_Integral_t::getCellType ( ) const [virtual]

Returns the name of the cell line to which the nucleus belongs.

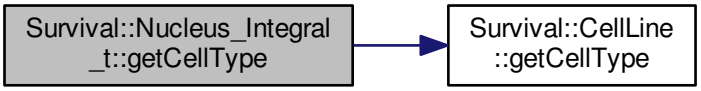
Returns

A `string` corresponding to the name of the cell line to which the nucleus belongs, getting the information by cellLine.

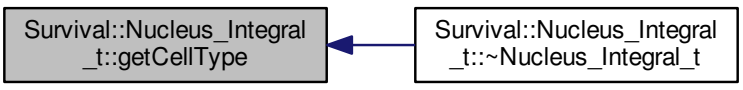
Implements Survival::Nucleus.

Definition at line 179 of file Nucleus_Integral_t.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.8 void Survival::Nucleus_Integral_t::getDose (double & *dose*) const [inline]

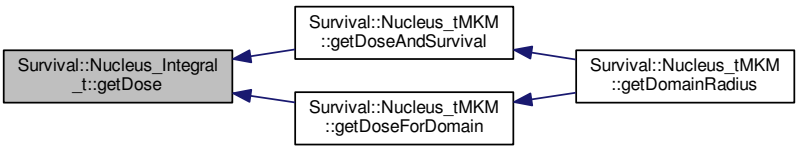
Returns the total dose absorbed by the nucleus, expressed in Gy, overwriting a double variable passed by reference.

See also

totalNucleusDose

Definition at line 90 of file Nucleus_Integral_t.h.

Here is the caller graph for this function:



7.6.3.9 void Nucleus_Integral_t::getDoseAndLethals (double & *dose,* double & *doseUncertainty,* double & *lethals,* double & *lethalsUncertainty*) const

Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and associated lethal events, with respective uncertainties.

The dose absorbed coincides with totalNucleusDose, the number of lethal events is evaluated by means of the CellLine::getLogSurvival_X() method.

Parameters

dose	The total dose absorbed by the nucleus, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
lethals	The lethal events observed in the nucleus, passed by reference to be overwritten.
lethalsUncertainty	The uncertainty associated to the lethal events observed, passed by reference to be overwritten.

Note

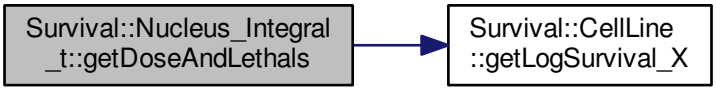
The method was thought to associate also an uncertainty to dose and lethal events, but this possibility hasn't been implemented yet, therefore actually -1 is assigned to those values.

Warning

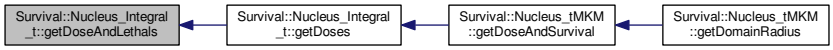
This method doesn't consider the time structure of the irradiation. The reason is that this class was thought to implement the MCt-MKM model, therefore the time structure is considered directly in the Nucleus_tMKM class.

Definition at line 200 of file Nucleus_Integral_t.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.10 void Nucleus_Integral_t::getDoseAndSurvival (double & *dose,* double & *doseUncertainty,* double & *survival,* double & *survivalUncertainty*) const [virtual]

Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated survival, with respective uncertainties.

The dose absorbed coincides with totalNucleusDose, the survival is evaluated by means of the CellLine::getLog↩Survival_X() method.

Parameters

dose	The total dose absorbed by the nucleus, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival associated to the dose absorbed by the nucleus, passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.

Note

The method was thought to associate also an uncertainty to dose and survival, but this possibility hasn't been implemented yet, therefore actually -1 is assigned to those values.

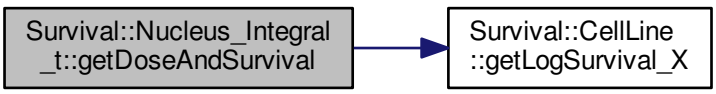
Warning

The implementation of this method is identical to the one defined in Nucleus_Integral::getDoseAndSurvival(); it isn't considered the time structure of the irradiation. The reason is that this class was thought to implement the MCt-MKM model, therefore the time structure is considered directly in the Nucleus_tMKM class.

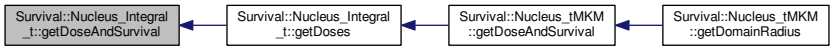
Implements Survival::Nucleus.

Definition at line 186 of file Nucleus_Integral_t.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.11 std::vector<double> Survival::Nucleus_Integral_t::getDoses ( ) const [inline]

Returns the vector representing each dose deposited in the nucleus by a Particle when an interaction occurs.

Returns

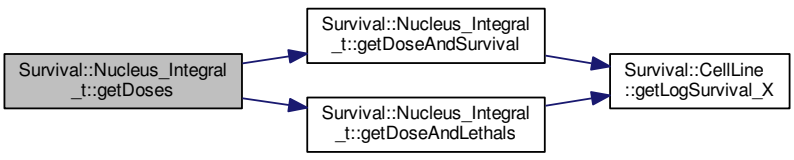
A vector representing each dose deposited in the nucleus by a Particle when an interaction occurs, expressed in Gy.

See also

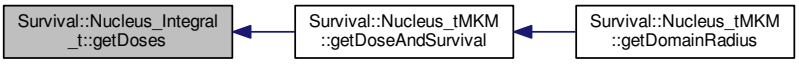
doses

Definition at line 98 of file Nucleus_Integral_t.h.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.12 virtual int Survival::Nucleus_Integral_t::getInNucleusCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus has been crossed through by a Particle.

Returns

inNucleusCount The number of times that the nucleus has been crossed through by a Particle.

See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 142 of file Nucleus_Integral_t.h.

7.6.3.13 virtual int Survival::Nucleus_Integral_t::getIntersectionCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

Returns

intersectionCount The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

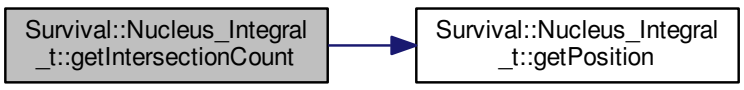
See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 150 of file Nucleus_Integral_t.h.

Here is the call graph for this function:



7.6.3.14 void Nucleus_Integral_t::getPosition (double & *returnX,* double & *returnY*) const [virtual]

Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the nucleus referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `y` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.

See also

Nucleus_tMKM::getPosition()

Implements Survival::Nucleus.

Definition at line 213 of file Nucleus_Integral_t.cpp.

Here is the caller graph for this function:

7.6.3.15 virtual double Survival::Nucleus_Integral_t::getRadius ( ) const [inline], [virtual]

Returns the radius of the nucleus expressed in um.

Returns

r_nucleus The radius of the nucleus expressed in um.

Implements Survival::Nucleus.

Definition at line 168 of file Nucleus_Integral_t.h.

7.6.3.16 std::vector<double> Survival::Nucleus_Integral_t::getTimes ( ) const [inline]

Returns the vector representing each instant in which an interaction with a Particle occurs. Each value is expressed in hours.

Returns

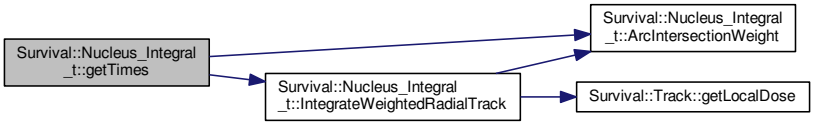
A vector representing each instant in which an interaction with a Particle occurs, expressed in hours.

See also

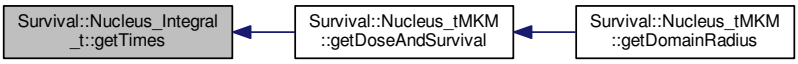
times

Definition at line 176 of file Nucleus_Integral_t.h.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.17 double Nucleus_Integral_t::IntegrateWeightedRadialTrack (const Track & *track*, double *rMin*, double *rMax*, double *b*, double & *area*, double *step*) [private]

Performs the integral of the radial profile of the track in the intersection area with the nucleus.

The problem is to integrate a function over a complex area originated by the random intersection of two circles. The way this method performs the task is to evaluate it numerically, dividing the area (or the radius to be covered) in a number of finite (small) step and evaluating for each step the length of the arc of circumference by means of the ArcIntersectionWeight() method. The sum of all these lengths is equal to the intersection area and it's overwritten in the correspondent parameter. For each step, defined by a specific radius, the method gets the local dose from the track (Track::getLocalDose()) and the integral is evaluated considering a step function constructed in this way. Finally the value of the integral is normalized over the intersection area.

Parameters

track	A reference to the Track of the particle interacting with the nucleus.
rMin	Minimum radius, lower limit of integration, expressed in um.
rMax	Maximum radius, upper limit of integration, expressed in um.
b	The impact parameters of the track, expressed in um.
area	The intersection area between Track and Nucleus, passed by reference to be overwritten with its correct value.
step	The length of the radial step of integration.

Returns

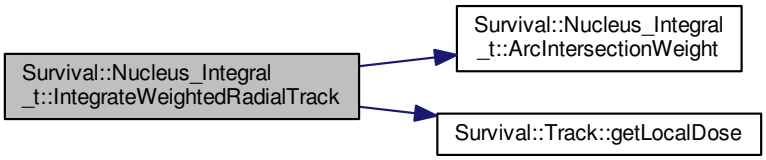
The value of the integral normalized over the intersection area, expressed in Gy.

See also

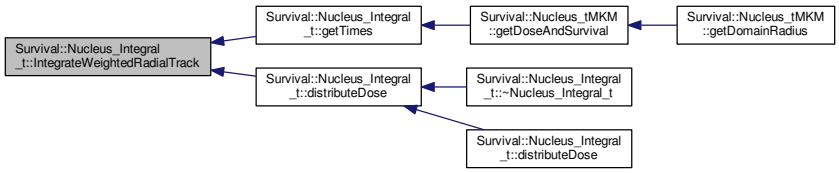
distributeDose()

Definition at line 222 of file Nucleus_Integral_t.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.4 Member Data Documentation

7.6.4.1 const CellLine& Survival::Nucleus_Integral_t::cellLine [protected]

A reference to a CellLine object where the characteristics of the cell line to which the nucleus belongs are stored.

Definition at line 249 of file Nucleus_Integral_t.h.

7.6.4.2 std::vector<double> Survival::Nucleus_Integral_t::doses [private]

Vector containing the sequence of doses deposited in the nucleus, expressed in Gy, each elements is associated to one interaction.

It's updated by distributeDose() every times an interaction occurs. This vector has the same length of times: they are strongly coupled because they represent the same event. The sum of the elements of the vector is equal to totalNucleusDose, representing the total dose absorbed by the nucleus.

See also

getDoses()

Definition at line 238 of file Nucleus_Integral_t.h.

7.6.4.3 int Survival::Nucleus_Integral_t::inNucleusCount [protected]

The number of times that the nucleus has been crossed through by a Particle.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 269 of file Nucleus_Integral_t.h.

7.6.4.4 int Survival::Nucleus_Integral_t::intersectionCount [protected]

The number of times that the nucleus interacted with a Particle.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 275 of file Nucleus_Integral_t.h.

7.6.4.5 double Survival::Nucleus_Integral_t::r_nucleus [protected]

The radius of the nucleus, expressed in um.

It's instantiated in the constructor getting the value from the CellLine object representing the cell line to which the nucleus belongs to.

See also

Nucleus_Integral()

Definition at line 257 of file Nucleus_Integral_t.h.

7.6.4.6 std::vector<double> Survival::Nucleus_Integral_t::times [private]

Vector containing the sequence of interaction times (expressed in hours), each elements is associated to one interaction.

It's updated by distributeDose() every times an interaction occurs. This vector has the same length of doses: they are strongly coupled because they represent the same event.

See also

getTimes()

Definition at line 230 of file Nucleus_Integral_t.h.

7.6.4.7 double Survival::Nucleus_Integral_t::totalNucleusDose [protected]

The total dose absorbed by the nucleus, expressed in Gy.

The value is initially set to zero in the constructor (through the cleanNucleus() method) and then updated by distributeDose() when an interaction with a Particle occurs.

Definition at line 246 of file Nucleus_Integral_t.h.

7.6.4.8 const double Survival::Nucleus_Integral_t::x_nucleus [protected]

The position of the center of the nucleus (x coordinate) referred to the beam axis, expressed in mm.

Definition at line 260 of file Nucleus_Integral_t.h.

7.6.4.9 const double Survival::Nucleus_Integral_t::y_nucleus [protected]

The position of the center of the nucleus (y coordinate) referred to the beam axis, expressed in mm.

Definition at line 263 of file Nucleus_Integral_t.h.

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

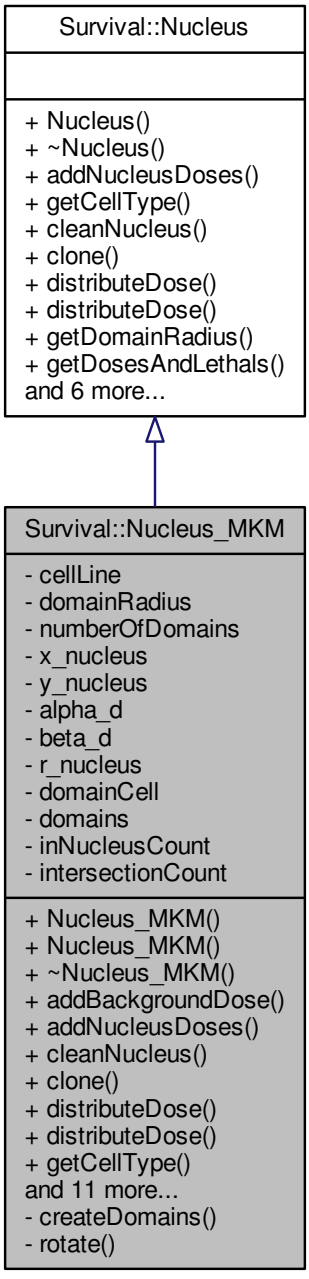
- include/Nucleus_Integral_t.h
- src/Nucleus_Integral_t.cpp

7.7 Survival::Nucleus_MKM Class Reference

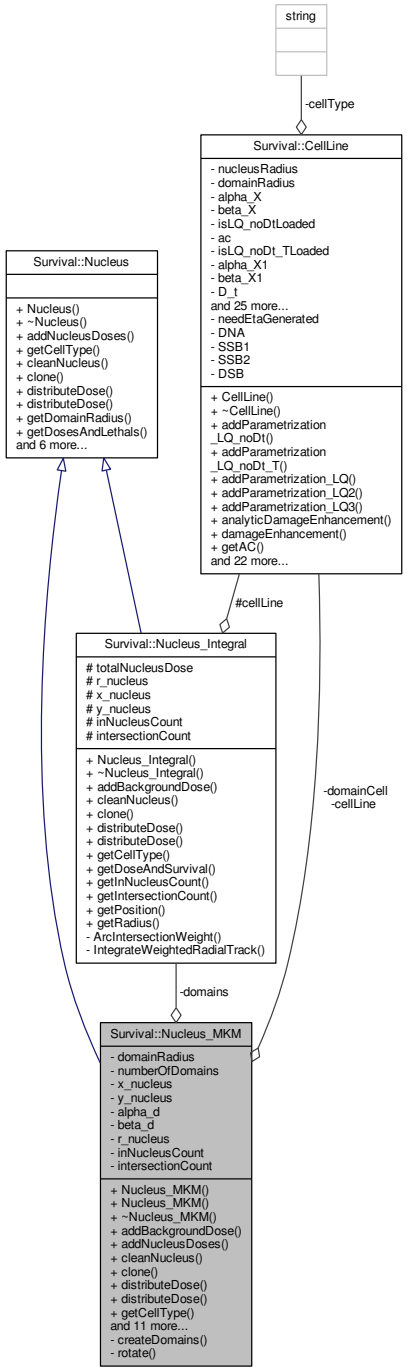
Inherited from the Nucleus pure virtual class, it implements the cellular nucleus as defined and used in the MKM model.

`#include <Nucleus_MKM.h>`

Inheritance diagram for Survival::Nucleus_MKM:



Collaboration diagram for Survival::Nucleus_MKM:



Public Member Functions

- Nucleus_MKM (const CellLine &cellLineRef, const double xPosition=0.0, const double yPosition=0.0)

  Constructor. Instantiates and sets the object.
- Nucleus_MKM (const CellLine &cellLineRef, double domainRadius, int numberOfDomains, const double x↩ Position=0.0, const double yPosition=0.0)

  Instantiates and sets the object. Overload of the constructor.

- virtual ~Nucleus_MKM ()

  Destructor.
- void addBackgroundDose (const double dose)

  Adds a constant value of dose absorbed in each domain of the nucleus.
- virtual void addNucleusDoses (Nucleus_MKM &nucleus)

  Performs the sum of the dose absorbed by two different nucleus domain to domain (adds one to the other).
- virtual void cleanNucleus ()

  Resets to zero all counters (inNucleusCount and intersectionCount) and doses in the nucleus and his domain.
- virtual Nucleus_MKM * clone (const CellLine &)

  Returns a pointer to a new Nucleus_MKM object. It not really a clone but a new clean object.
- virtual void distributeDose (const Track &track)

  When an interaction between a Particle and the MKM nucleus occurs, the method increases inNucleusCount and intersectionCount counters and proceeds to distribute the dose in each domain.
- virtual void distributeDose (const Tracks &tracks)

  Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for each track of the container.
- virtual std::string getCellType () const

  Returns the name of the cell line to which the nucleus belongs.
- virtual double getDomainRadius ()

  Returns the radius of the domain corresponding to the CellLine to which the MKM nucleus belongs.
- void getDoseAndLethalForDomain (int domainIndex, double &dose, double &doseUncertainty, double &lethal, double &lethalUncertainty) const

  Returns (overwriting parameters passed by reference) the dose absorbed and the number of lethal events observed in a specified domain identified by domainIndex, with respective uncertainties.
- virtual void getDosesAndLethals (std::vector< double > &doses, std::vector< double > &dosesUncertainty, std::vector< double > &lethals, std::vector< double > &lethalsUncertainty) const

  Returns (overwriting parameters passed by reference) the vectors containing doses absorbed and number of lethal events observed with respective uncertainties, each element of the vector refers to one of the domains.
- virtual void getDoseAndSurvival (double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty) const

  Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated survival, with respective uncertainties.
- double getDoseForDomain (int indexOfDomain) const

  Return the dose absorbed in the indexOfDomain-th domain, expressed in Gy.
- virtual int getInNucleusCount () const

  Returns the number of times that the nucleus has been crossed through by a Particle.
- virtual int getIntersectionCount () const

  Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.
- virtual int getNumberOfDomains ()

  Returns the number of domains composing the MKM nucleus.
- virtual void getPosition (double &returnX, double &returnY) const

  Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.
- virtual double getRadius () const

  Returns the effective radius of the Nucleus_MKM object.
- void saveLocalDose (const std::string fileName) const

  Save data corresponding to the dose absorbed by each domain and the number of lethal events observed, useful to debug.

Private Member Functions

- void createDomains ()

  Create the domains as pointers to Nucleus_Integral objects, placed to form a hexagonal shape, spiraling from (0,0).
- void rotate (double &xTranslation, double &yTranslation)

  Performs a 60 degrees clockwise rotation.

Private Attributes

- const CellLine & cellLine

  A reference to a CellLine object where the characteristics of the cell line to which the MKM nucleus belongs are stored.
- double domainRadius

  The radius of the domain corresponding to the CellLine to which the MKM nucleus belongs.
- int numberOfDomains

  The number of domains composing the MKM nucleus.
- const double x_nucleus

  The position of the center of the nucleus (x coordinate) referred to the beam axis, expressed in mm.
- const double y_nucleus

  The position of the center of the nucleus (y coordinate) referred to the beam axis, expressed in mm.
- double alpha_d

  The linear-quadratic parameter α associated to each of the domains composing the MKM nucleus.
- double beta_d

  The linear-quadratic parameter β associated to each of the domains composing the MKM nucleus.
- double r_nucleus

  It's the effective radius of the Nucleus_MKM object.
- CellLine $*$ domainCell

  A pointer to a CellLine object, storing the information about the cell line to which the MKM nucleus belongs.
- Nucleus_Integral $**$ domains

  A pointer to pointers, where the objects finally pointed are Nucleus_Integral objects corresponding to the domains composing the MKM nucleus.
- int inNucleusCount

  The number of times that the nucleus has been crossed through by a Particle.
- int intersectionCount

  The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

7.7.1 Detailed Description

Inherited from the Nucleus pure virtual class, it implements the cellular nucleus as defined and used in the MKM model.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2011–2015

The microdosimetric-kinetic model (MKM, 2) is based on a cellular nucleus divided into subcellular structures referred to as *domains* similar to the *sites* defined in the theory of dual radiation action (TDRA) by Kellerer and Rossi (1). This class implements the nucleus structure characteristics of the MKM model in all his formulations and provides some methods to evaluate the dose absorbed in the interaction with particles and get informations about lethal events observed and the associated cellular survival.

1. A. Kellerer and H. Rossi, "A generalized formulation of dual radiation action", *Radiation Research* **75**, 471-488 (1978)

2. The MKM model was formulated by Hawkins in 1994, then modified over subsequent years and recently reformulated (and here implemented) following a Monte Carlo approach. The original published reference for the MKM is:

- R.B. Hawkins, "A Statistical Theory of Cell Killing by Radiation of Varying Linear Energy Transfer", *Radiation Research* **140**, 366-374 (1994). While for the recent Monte Carlo reformulation:
- L. Manganaro, G. Russo, A. Attili, "Advanced modeling of the temporal effect in particle therapy: from radiobiological evaluation to treatment planning", *Medical Physics*, (Submitted)

Definition at line 27 of file Nucleus_MKM.h.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Nucleus_MKM::Nucleus_MKM (const CellLine & *cellLineRef,* const double *xPosition* = 0.0, const double *yPosition* = 0.0)

Constructor. Instantiates and sets the object.

When the constructor is called, it instantiates the object creating a hexagonal structure of circular domains, using the informations stored in the CellLine reference object, such as the radius of the nucleus and the radius of the single domain. This is made by calling the createDomains() function.

Parameters

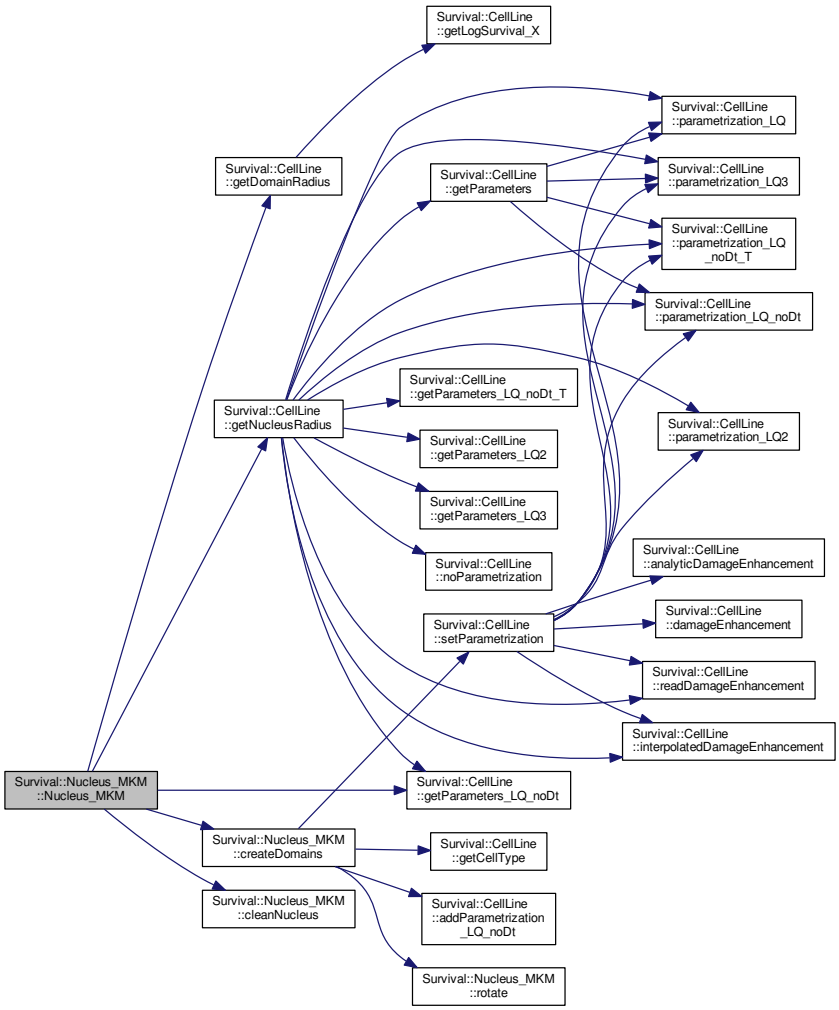
cellLineRef	A reference to the corresponding CellLine.
xPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.
yPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.

See also

createDomains(), cleanNucleus(), Nucleus_MKM(const CellLine&, double, int, const double, const double) and Nucleus_tMKM

Definition at line 38 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

7.7.2.2 Nucleus_MKM::Nucleus_MKM (const CellLine & *cellLineRef,* double *domainRadius,* int *numberOfDomains,* const double *xPosition* = 0.0, const double *yPosition* = 0.0)

Instantiates and sets the object. Overload of the constructor.

Provide the possibility to specify also the total number of domains and the radius of each domain.

Parameters

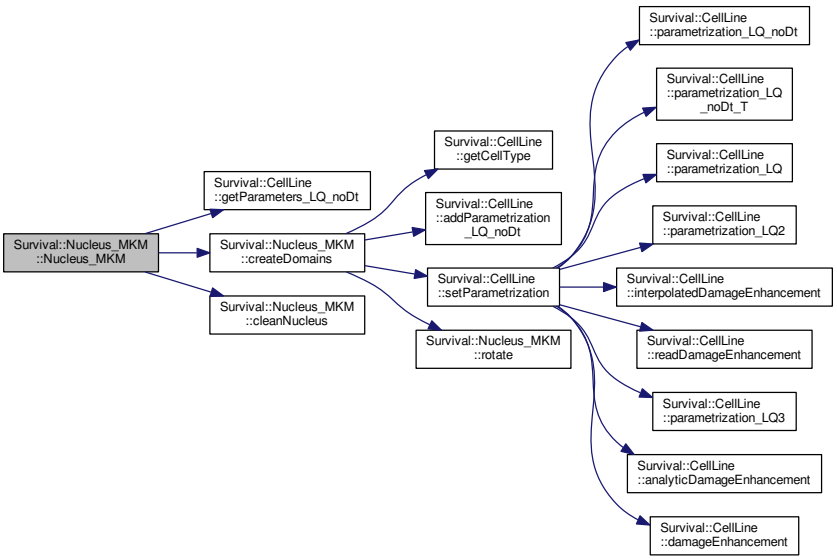
cellLineRef	A reference to the corresponding CellLine.
domainRadius	The radius of the single domain in the MKM nucleus, expressed in um.
numberOfDomains	The number of domains that constitute the MKM nucleus.
xPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.
yPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.

See also

createDomains(), Nucleus_Integral and Nucleus_MKM(const CellLine&, const double, const double)

Definition at line 65 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



7.7.2.3 Nucleus_MKM::~Nucleus_MKM ( ) [virtual]

Destructor.

The destructor deletes the domainCell, domains and each domains[i-th] pointers created when the object is instantiated.

See also

createDomains()

Definition at line 94 of file Nucleus_MKM.cpp.

7.7.3 Member Function Documentation

7.7.3.1 void Nucleus_MKM::addBackgroundDose (const double *dose*)

Adds a constant value of dose absorbed in each domain of the nucleus.

The method calls systematically for each domain the function Nucleus_Integral::addBackgroundDose(), that is the override of this method itself in the Nucleus_Integral class. The result is to add a constant value of dose absorbed in each domain and consequently in the whole nucleus.

Parameters

dose	The dose to be added expressed in Gy.

Definition at line 162 of file Nucleus_MKM.cpp.

7.7.3.2 void Nucleus_MKM::addNucleusDoses (Nucleus_MKM & *nucleus*) [virtual]

Performs the sum of the dose absorbed by two different nucleus domain to domain (adds one to the other).

This method calls systematically for each domain the function Nucleus_Integral::getDoseAndLethalForDomain() to get the respectively domain dose and then adds it to the corresponding domain in the current nucleus via the Nucleus_Integral::addBackgroundDose() function.

Warning

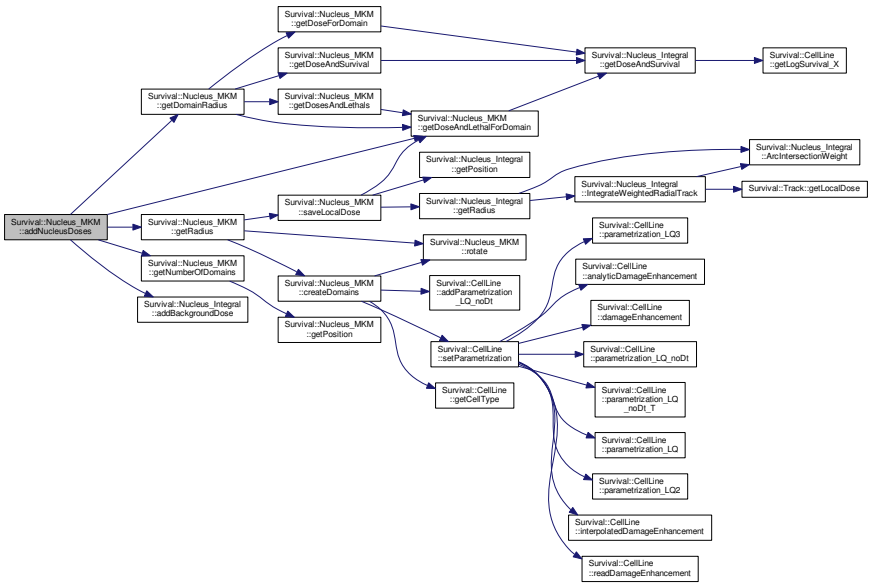
The execution of the program will be terminated if the other nucleus is not geometrically similar to this one (e.g. different r_nucleus, domainRadius or numberOfDomains).

Parameters

nucleus	A reference to another Nucleus_MKM object to evaluate the sum of the doses.

Definition at line 170 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



7.7.3.3 void Nucleus_MKM::cleanNucleus ( ) [virtual]

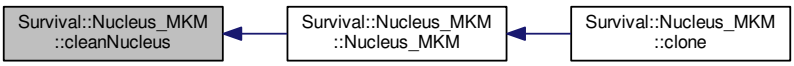
Resets to zero all counters (inNucleusCount and intersectionCount) and doses in the nucleus and his domain.

Resets to zero inNucleusCount and intersectionCount and calls systematically for each domain the function Nucleus_Integral::cleanNucleus(), that this the override of this method itself in the Nucleus_Integral class.

Implements Survival::Nucleus.

Definition at line 192 of file Nucleus_MKM.cpp.

Here is the caller graph for this function:



7.7.3.4 Nucleus_MKM * Nucleus_MKM::clone (const CellLine & *cellLine*) [virtual]

Returns a pointer to a new Nucleus_MKM object. It not really a clone but a new clean object.

Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

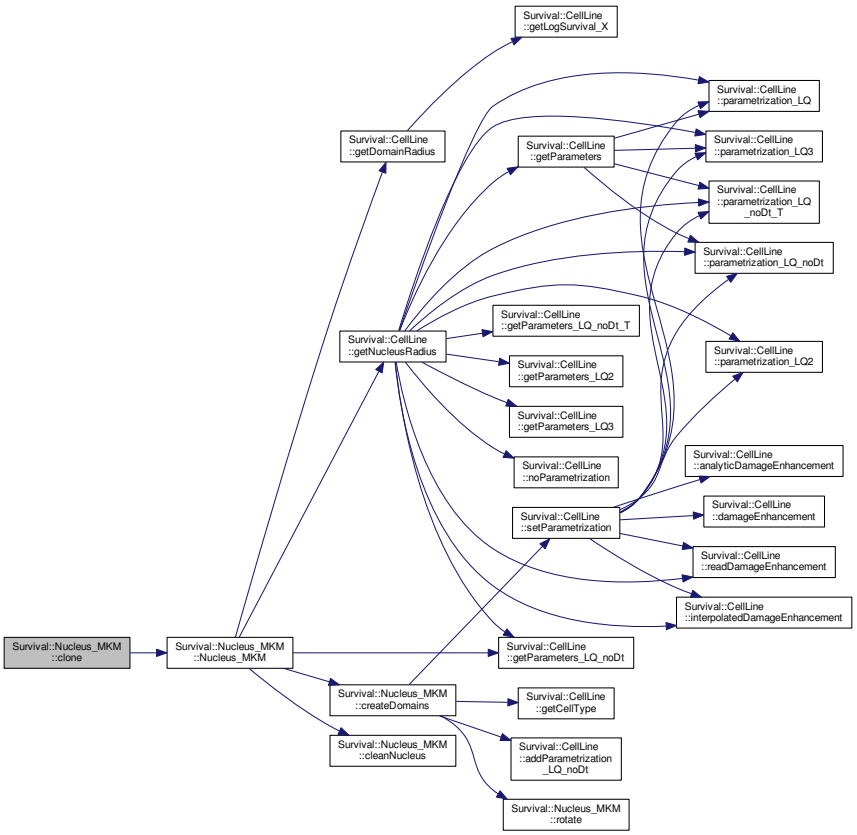
Note

To create a real clone of another nucleus, a better implementation of the copy constructor is needed.

Implements Survival::Nucleus.

Definition at line 203 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



7.7.3.5 void Nucleus_MKM::createDomains ( ) [private]

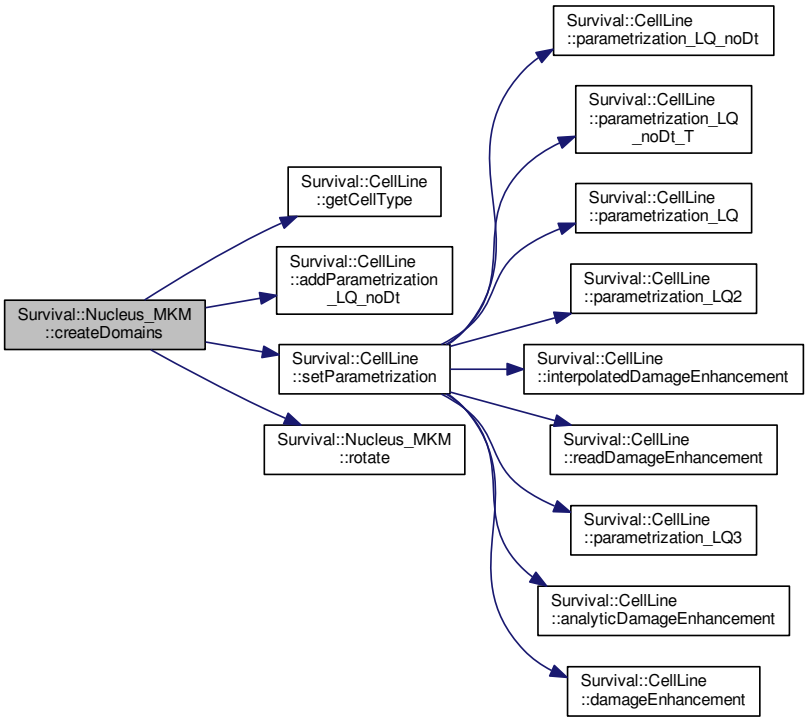
Create the domains as pointers to Nucleus_Integral objects, placed to form a hexagonal shape, spiraling from (0,0).

This function is called by the constructor every times a Nucleus_MKM is created, and it's responsible to instantiate and place the domains in the right position.

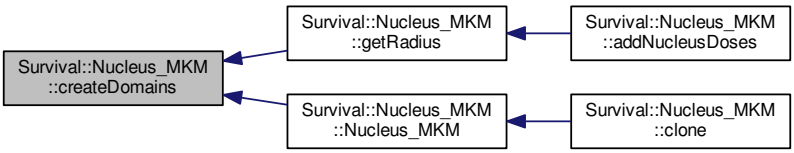
The structure is created in such a way that the center of each domain is placed on the vertex of a regular hexagon and the distance between two nearest neighbors is exactly equal to twice the radius of the domain (domainRadius). Some concentric hexagons are created to places all the domains defined (numberOfDomains). The center of each hexagon coincides with the position of the first domain created, that is also the center of the MKM nucleus. This structure is created by means of the rotate() method.

Definition at line 108 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.6 void Nucleus_MKM::distributeDose (const Track & *track*) [virtual]

When an interaction between a Particle and the MKM nucleus occurs, the method increases inNucleusCount and intersectionCount counters and proceeds to distribute the dose in each domain.

This function is the first step to evaluate the dose deposited by the radiation in the nucleus or, better, in each domain of the MKM nucleus. It checks if any interaction exists between Nucleus and Particle (that is the Track generated by the particle) looking at their positions and radius. If it's true, it increases the respective counters (inNucleusCount and intersectionCount) and calls the method Nucleus_Integral::distributeDose() in a `for` loop over the total number of domains.

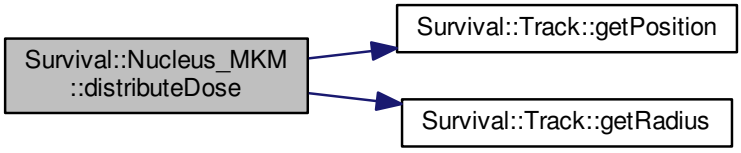
Parameters

track	A reference to the Track generated by the particle in the nucleus.

Implements Survival::Nucleus.

Definition at line 211 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

7.7.3.7 void Nucleus_MKM::distributeDose (const Tracks & *tracks*) [virtual]

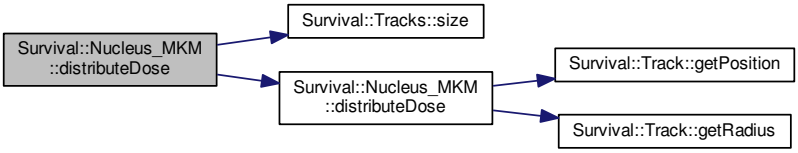
Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for each track of the container.

Since the Tracks class is a container for Track objects, this method calls distributeDose(const Track &track) in a `for` loop over each track contained in the Tracks object.

Implements Survival::Nucleus.

Definition at line 233 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



7.7.3.8 string Nucleus_MKM::getCellType ( ) const [virtual]

Returns the name of the cell line to which the nucleus belongs.

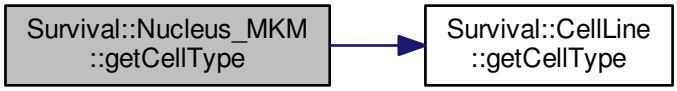
Returns

A `string` corresponding to the name of the cell line to which the nucleus belongs, getting the information by cellLine.

Implements Survival::Nucleus.

Definition at line 241 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



7.7.3.9 virtual double Survival::Nucleus_MKM::getDomainRadius ( ) [inline], [virtual]

Returns the radius of the domain corresponding to the CellLine to which the MKM nucleus belongs.

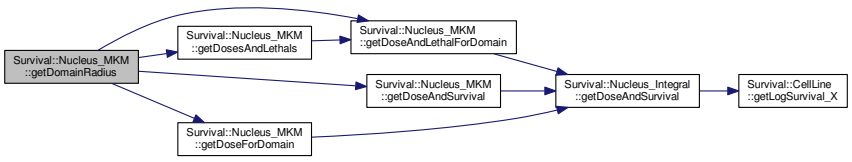
Returns

domainRadius That is the radius of the domain corresponding to the CellLine to which the MKM nucleus belongs.

Reimplemented from Survival::Nucleus.

Definition at line 127 of file Nucleus_MKM.h.

Here is the call graph for this function:



Here is the caller graph for this function:

7.7.3.10 void Nucleus_MKM::getDoseAndLethalForDomain (int *domainIndex,* double & *dose,* double & *doseUncertainty,* double & *lethal,* double & *lethalUncertainty*) const

Returns (overwriting parameters passed by reference) the dose absorbed and the number of lethal events observed in a specified domain identified by domainIndex, with respective uncertainties.

The function calls Nucleus_Integral::getDoseAndSurvival() to get the dose absorbed by the domainIndex-th domain and then evaluates the number of lethal events by means of the linear-quadratic relation:

$$L = \alpha_d \, D + \beta_d \, D^2$$

Parameters

domainIndex	The index referred to the domain, passed by reference to be overwritten.
dose	The dose absorbed by the domainIndex-th domain, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
lethal	The lethal events observed in the domainIndex-th domain, passed by reference to be overwritten.
lethalUncertainty	The uncertainty associated to the lethal events observed, passed by reference to be overwritten.

Note

The method was thought to associate also an uncertainty to dose and lethals, but this possibility hasn't been implemented yet, therefore actually -1 is assigned to those values.

Warning

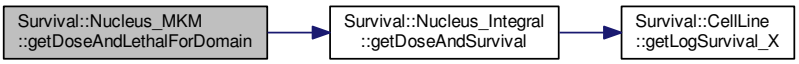
If an incorrect domainIndex is chosen (e.g. greater than the real number of domains) -1 will be assigned also to dose and lethals, no exceptions are thrown.

See also

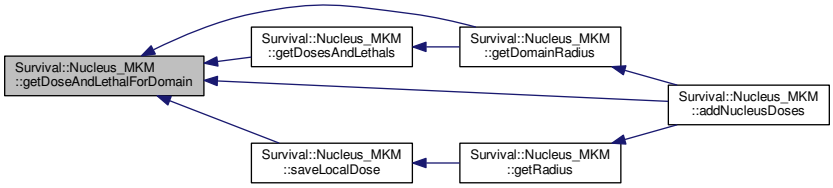
getDosesAndLethals(), getDoseAndSurvival() and getDoseForDomain()

Definition at line 248 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.11 void Nucleus_MKM::getDoseAndSurvival (double & *dose*, double & *doseUncertainty*, double & *survival*, double & *survivalUncertainty*) const [virtual]

Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated survival, with respective uncertainties.

The function calls, in a `for` loop over the total number of domains, the method Nucleus_Integral::getDoseAnd↩Survival() to get the dose absorbed by each domain and for each one evaluates the number of lethal events by means of the linear-quadratic relation:

$$L_i = \alpha_d \, D + \beta_d \, D^2$$

(An equivalent solution could be reached calling directly getDoseAndLethalForDomain()). Then the total dose absorbed by the nucleus is obtained by the average of the doses absorbed by domains while the cellular survival is evaluated as a negative exponential function of the total number of lethal events (according to the poissonian statistics):

$$S = \exp(-L_{TOT})$$

where $L_{TOT} = \sum L_i$

Parameters

dose	The total dose absorbed by the nucleus, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival associated to the dose absorbed by the nucleus, passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.

Note

The method was thought to associate also an uncertainty to dose and survival, but this possibility hasn't been implemented yet, therefore actually -1 is assigned to those values.

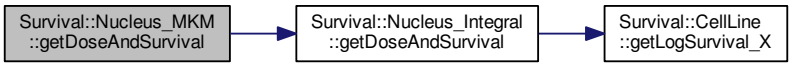
See also

getDosesAndLethals(), getDoseAndLethalForDomain() and getDoseForDomain()

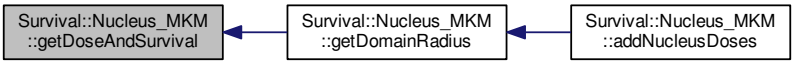
Implements Survival::Nucleus.

Definition at line 280 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.12 double Nucleus_MKM::getDoseForDomain (int *indexOfDomain*) const

Return the dose absorbed in the indexOfDomain-th domain, expressed in Gy.

The function calls the method Nucleus_Integral::getDoseAndSurvival() from the indexOfDomain-th domain.

If an incorrect index is selected (e.g. greater than the total number of domains) the dose absorbed is set to -1.

Parameters

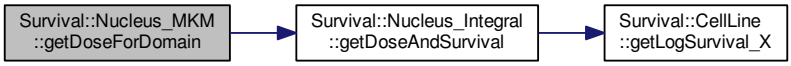
indexOfDomain	The index associated to the domain.

Returns

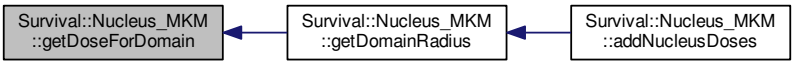
The dose absorbed in the indexOfDomain-th domain, expressed in Gy.

Definition at line 309 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.13 void Nucleus_MKM::getDosesAndLethals (std::vector< double > & *doses*, std::vector< double > & *dosesUncertainty*, std::vector< double > & *lethals*, std::vector< double > & *lethalsUncertainty*) const [virtual]

Returns (overwriting parameters passed by reference) the vectors containing doses absorbed and number of lethal events observed with respective uncertainties, each element of the vector refers to one of the domains.

The function calls, in a `for` loop over the total number of domains, the method getDoseAndLethalForDomain() to get the informations desired.

Parameters

doses	The vector of doses absorbed (in Gy), each element refers to a specific domain, passed by reference to be overwritten.
dosesUncertainty	The vector of uncertainties associated to doses absorbed (in Gy), each element refers to a specific domain, passed by reference to be overwritten.
lethals	The vector of lethal events observed, each element refers to a specific domain, passed by reference to be overwritten.
lethalsUncertainty	The vector of uncertainties associated to the number of lethal events observed, each element refers to a specific domain, passed by reference to be overwritten.

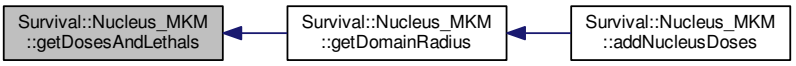
See also

getDoseAndSurvival() and getDoseForDomain()

Definition at line 269 of file Nucleus_MKM.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



7.7.3.14 virtual int Survival::Nucleus_MKM::getInNucleusCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus has been crossed through by a Particle.

Returns

inNucleusCount The number of times that the nucleus has been crossed through by a Particle.

See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 215 of file Nucleus_MKM.h.

7.7.3.15 virtual int Survival::Nucleus_MKM::getIntersectionCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

Returns

intersectionCount The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 223 of file Nucleus_MKM.h.

7.7.3.16 virtual int Survival::Nucleus_MKM::getNumberOfDomains ( ) [inline], [virtual]

Returns the number of domains composing the MKM nucleus.

Returns

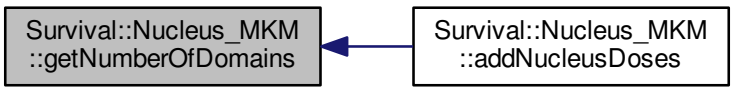
The number of domains composing the MKM nucleus.

Reimplemented from Survival::Nucleus.

Definition at line 229 of file Nucleus_MKM.h.

Here is the call graph for this function:

Here is the caller graph for this function:



7.7.3.17 void Nucleus_MKM::getPosition (double & *returnX*, double & *returnY*) const [virtual]

Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the nucleus referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `y` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.

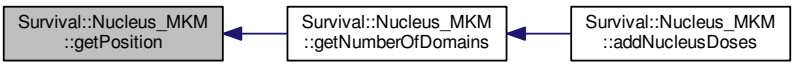
See also

Nucleus_Integral::getPosition()

Implements Survival::Nucleus.

Definition at line 322 of file Nucleus_MKM.cpp.

Here is the caller graph for this function:



7.7.3.18 virtual double Survival::Nucleus_MKM::getRadius ( ) const [inline], [virtual]

Returns the effective radius of the Nucleus_MKM object.

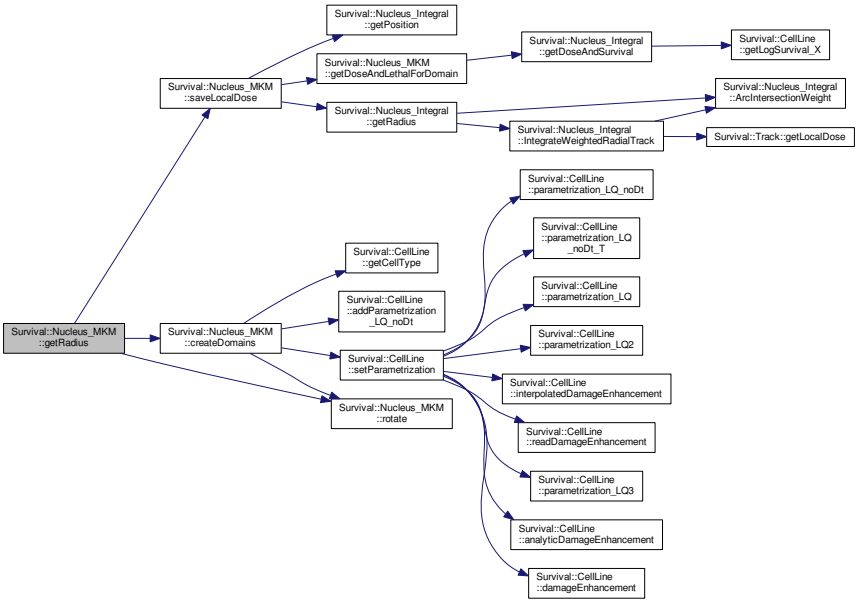
Returns

r_nucleus, the effective radius of the Nucleus_MKM object. Since the structure of the final MKM nucleus is "hexagon-like" this radius is different from the radius stored in the cellLine reference. It's the distance between the center of the nucleus and the farthest point of the nucleus itself, expressed in um.

Implements Survival::Nucleus.

Definition at line 247 of file Nucleus_MKM.h.

Here is the call graph for this function:



Here is the caller graph for this function:

7.7.3.19 void Nucleus_MKM::rotate (double & *xTranslation,* double & *yTranslation*) [private]

Performs a 60 degrees clockwise rotation.

The rotation matrix is defined by:

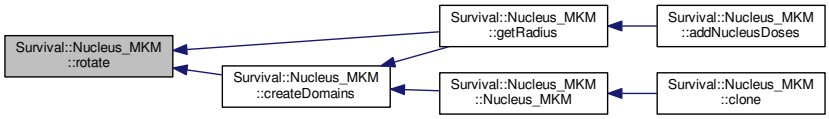
$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

Parameters

xTranslation	The x coordinate of the point where to start the 60 degrees clockwise rotation.
yTranslation	The y coordinate of the point where to start the 60 degrees clockwise rotation.

Definition at line 151 of file Nucleus_MKM.cpp.

Here is the caller graph for this function:



7.7.3.20 void Nucleus_MKM::saveLocalDose (const std::string *fileName*) const

Save data corresponding to the dose absorbed by each domain and the number of lethal events observed, useful to debug.

This method write on a file, for each domain of the nucleus:

- The index of the domain;
- The radius of the domain;
- The x and y coordinates (in um) identifying its position referred to the beam axis;
- The dose absorbed by the domain;
- The number of lethal events observed.

Parameters

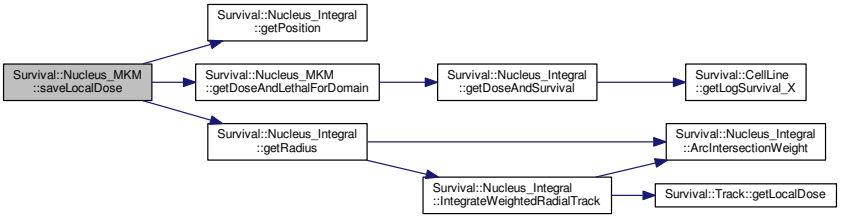
fileName	The name of the file where to save data.

Warning

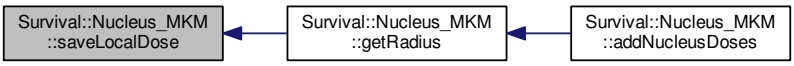
The execution of the program will be terminated if fileName refers to an inexistent file.

Definition at line 331 of file Nucleus_MKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.4 Member Data Documentation

7.7.4.1 double Survival::Nucleus_MKM::alpha_d [private]

The linear-quadratic parameter α associated to each of the domains composing the MKM nucleus.

It's instantiated in the constructor dividing the α parameter stored in the cellLine reference by the total number of domains (numberOfDomains): $\frac{\alpha}{N_d}$. Note that α is a parameter of the model, that ideally represents the value of the linear-quadratic $alpha_X$ parameter identified in the case of irradiation with a photon beam.

See also

CellLine, CellLine::getParameters_LQ_noDt() and Nucleus_MKM()

Definition at line 323 of file Nucleus_MKM.h.

7.7.4.2 double Survival::Nucleus_MKM::beta_d [private]

The linear-quadratic parameter β associated to each of the domains composing the MKM nucleus.

It's instantiated in the constructor dividing the β parameter stored in the cellLine reference by the total number of domains (numberOfDomains): $\frac{\beta}{N_d}$. Note that β is a parameter of the model, that ideally represents the value of the linear-quadratic $beta_X$ parameter identified in the case of irradiation with a photon beam.

See also

CellLine, CellLine::getParameters_LQ_noDt() and Nucleus_MKM()

Definition at line 331 of file Nucleus_MKM.h.

7.7.4.3 const CellLine& Survival::Nucleus_MKM::cellLine [private]

A reference to a CellLine object where the characteristics of the cell line to which the MKM nucleus belongs are stored.

Definition at line 290 of file Nucleus_MKM.h.

7.7.4.4 CellLine* Survival::Nucleus_MKM::domainCell [private]

A pointer to a CellLine object, storing the information about the cell line to which the MKM nucleus belongs.

It's defined in the createDomains() method and deleted in the destructor ~Nucleus_MKM.

Definition at line 347 of file Nucleus_MKM.h.

7.7.4.5 double Survival::Nucleus_MKM::domainRadius [private]

The radius of the domain corresponding to the CellLine to which the MKM nucleus belongs.

This information is stored in the cellLine reference and then copied to this variable in the constructor. It's expressed in um.

See also

Nucleus_MKM()

Definition at line 298 of file Nucleus_MKM.h.

7.7.4.6 Nucleus_Integral** Survival::Nucleus_MKM::domains [private]

A pointer to pointers, where the objects finally pointed are Nucleus_Integral objects corresponding to the domains composing the MKM nucleus.

Definition at line 350 of file Nucleus_MKM.h.

7.7.4.7 int Survival::Nucleus_MKM::inNucleusCount [private]

The number of times that the nucleus has been crossed through by a Particle.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 356 of file Nucleus_MKM.h.

7.7.4.8 int Survival::Nucleus_MKM::intersectionCount [private]

The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 362 of file Nucleus_MKM.h.

7.7.4.9 int Survival::Nucleus_MKM::numberOfDomains [private]

The number of domains composing the MKM nucleus.

It's evaluated in the constructor as the ratio between the areas of the MKM nucleus, whose radius R_N is stored in the cellLine reference, and the single domain, characterized by a radius R_d (domainRadius):

$$N_d = \frac{R_N^2}{R_d^2}$$

See also

Nucleus_MKM()

Definition at line 309 of file Nucleus_MKM.h.

7.7.4.10 double Survival::Nucleus_MKM::r_nucleus [private]

It's the effective radius of the Nucleus_MKM object.

Since the structure of the final MKM nucleus is "hexagon-like" this radius is different from the radius stored in the cellLine reference. It's the distance between the center of the nucleus and the farthest point of the nucleus itself.

It's is defined in the createDomains() function, called in the constructor, and it's expressed in um.

See also

Nucleus_MKM()

Definition at line 341 of file Nucleus_MKM.h.

7.7.4.11 const double Survival::Nucleus_MKM::x_nucleus [private]

The position of the center of the nucleus (x coordinate) referred to the beam axis, expressed in mm.

Definition at line 312 of file Nucleus_MKM.h.

7.7.4.12 const double Survival::Nucleus_MKM::y_nucleus [private]

The position of the center of the nucleus (y coordinate) referred to the beam axis, expressed in mm.

Definition at line 315 of file Nucleus_MKM.h.

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

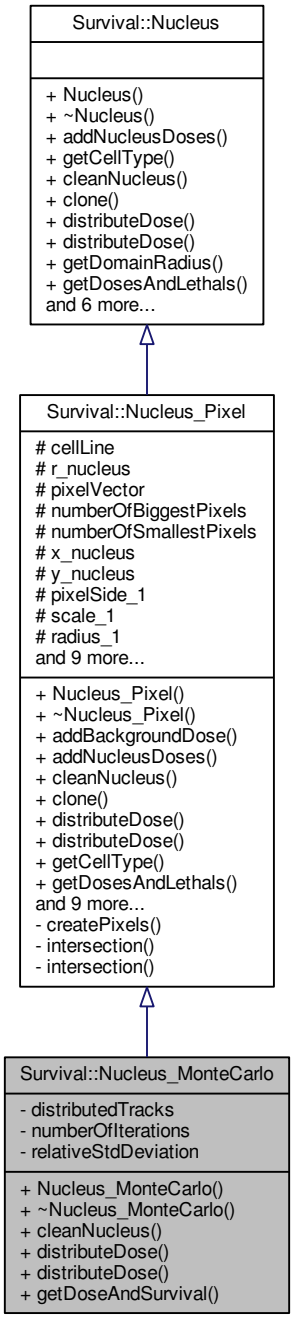
- include/Nucleus_MKM.h
- src/Nucleus_MKM.cpp

7.8 Survival::Nucleus_MonteCarlo Class Reference

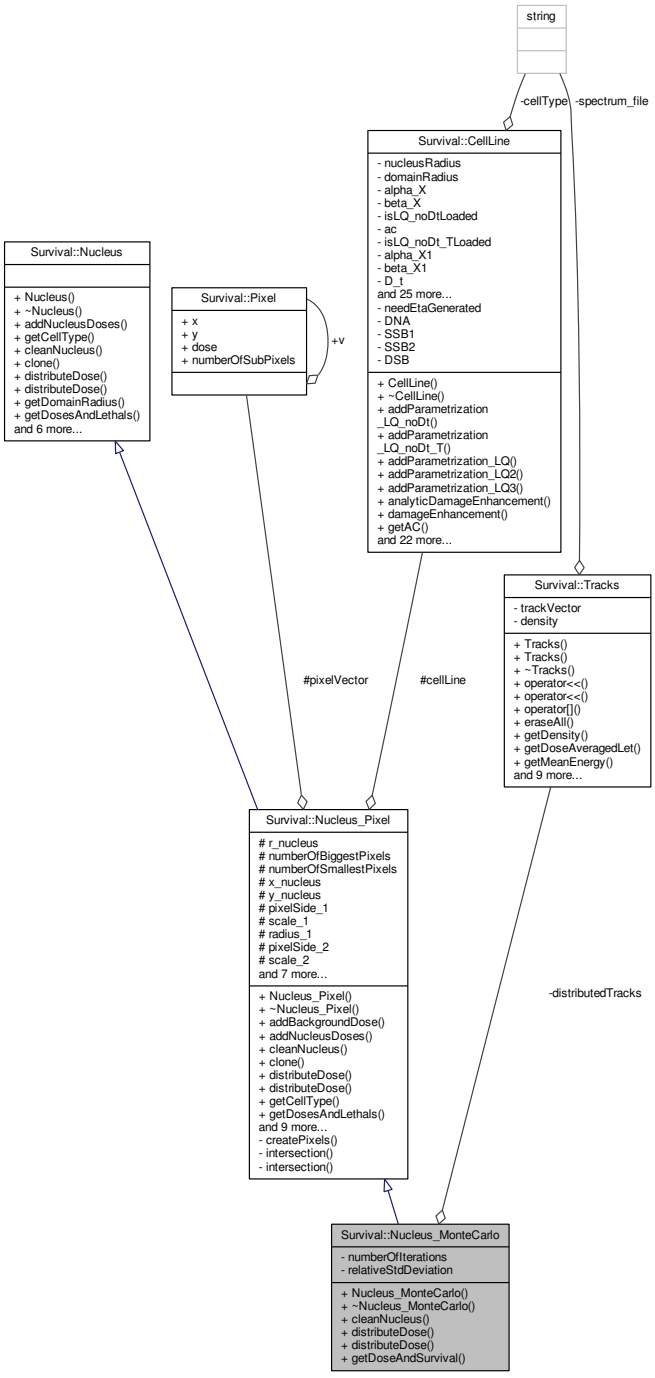
Inherited from the Nucleus_Pixel class, it performs the integration of the dose deposited by the track via the Monte Carlo *importance sampling* method.

```
#include <Nucleus_MonteCarlo.h>
```

Inheritance diagram for Survival::Nucleus_MonteCarlo:



Collaboration diagram for Survival::Nucleus_MonteCarlo:



Public Member Functions

- Nucleus_MonteCarlo (const CellLine &cellLineRef, const double precision=3e-3, const double xPosition=0.0, const double yPosition=0.0, const double pixelSide1=0.005, const int scale1=2, const double radius1=0.1, const int scale2=10, const double radius2=1.0, const int scale3=10, const double radius3=10.0)

  Constructor. Instantiates and sets the object.
- ~Nucleus_MonteCarlo ()

Destructor.

- void cleanNucleus ()

  Clean the object by calling the Tracks::eraseAll() and Nucleus_Pixel::cleanNucleus() methods.

- void distributeDose (const Track &track)

  It simply calls Nucleus_Pixel::distributeDose(const Track &track) to evaluate the dose deposited in the nucleus and appends the Track object to distributedTracks.

- void distributeDose (const Tracks &tracks)

  Overload of distributeDose(const Track &track) to manage a Tracks object.

- void getDoseAndSurvival (double &dose, double &doseUncertainty, double &survival, double &survival↩ Uncertainty)

  Perform a Monte Carlo simulation to integrate over the nucleus the dose deposited by a track distribution in a stochastic way.

Private Attributes

- Tracks distributedTracks

  A Tracks object storing all Track objects interacting with the nucleus.

- long int numberOfIterations

  One of the two ending conditions of the Monte Carlo simulation. Fix the maximum number of iterations executable.

- double relativeStdDeviation

  One of the two ending conditions of the Monte Carlo simulation. Fix a constraint on the precision that is the maximum relative error on the cell survival evaluated.

Additional Inherited Members

7.8.1 Detailed Description

Inherited from the Nucleus_Pixel class, it performs the integration of the dose deposited by the track via the Monte Carlo *importance sampling* method.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2008

The class was thought to be a test class, useful to perform a simulation making use of the Nucleus_Pixel class and verifying the correctness and the precision of the nucleus structure and the way it integrates the track radial profile.

The approximation of the integrand is given by the pixel-wise constant dose profile generated with the Nucleus_↩ Pixel class; to avoid having zero values in some points of the nucleus, an arbitrary local dose background of 1 Gy is added. This approximation of dose profile is then normalized by its integral and is interpreted as a probability distribution; the related cumulative distribution is used to extract a position in the nucleus where to compute the exact local dose value and the consequent local number of lethal events. The average of the obtained lethal events values divided by their probability of extraction gives an estimate of the integral. The precision of the estimate is of course dependent on the number of extractions; the algorithm cycles until the user defined precision is reached (in the probabilistic convergence sense).

Definition at line 22 of file Nucleus_MonteCarlo.h.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 Nucleus_MonteCarlo::Nucleus_MonteCarlo (const CellLine & *cellLineRef*, const double *precision* = 3e-3, const double *xPosition* = 0.0, const double *yPosition* = 0.0, const double *pixelSide1* = 0.005, const int *scale1* = 2, const double *radius1* = 0.1, const int *scale2* = 10, const double *radius2* = 1.0, const int *scale3* = 10, const double *radius3* = 10.0)

Constructor. Instantiates and sets the object.

Calls explicitly Nucleus_Pixel constructor. It differs only in the `precision` parameter, used to set the ending condition of the Monte Carlo simulation. There are two way to set such a condition:

- A fixed number of iterations, hence the `precision` has to be an integer value greater (or at least equal) to 1.
- A constraint on the precision to reach in the simulation in the evaluation of the cell survival (precisely the relative error on the survival), hence the `precision` has to be a `double` in (0, 1).

Warning

The execution of the program will be terminated if the precision is not set correctly.

Parameters

cellLineRef	A reference to the corresponding CellLine.
precision	Fix the ending condition of the Monte Carlo simulation.
xPosition	The nucleus position (`x` coordinate of the center) referred to the beam axis, expressed in mm.
yPosition	The nucleus position (`y` coordinate of the center) referred to the beam axis, expressed in mm.
pixelSide1	The side of the smallest (or the third) sub-grid of pixels (pixelSide_1), expressed in um.
scale1	The scale factor between the second and the third subgrid of pixels (scale_1).
radius1	The radius of the smallest circumference that defines the sampling of the track, expressed in um.
scale2	The scale factor between the first and the second subgrid of pixels (scale_2).
radius2	The radius of the second circumference that defines the sampling of the track, expressed in um.
scale3	The scale factor between the biggest grid of pixel and the first sub-grid (scale_3).
radius3	The radius of the biggest circumference that defines the sampling of the track, expressed in um.

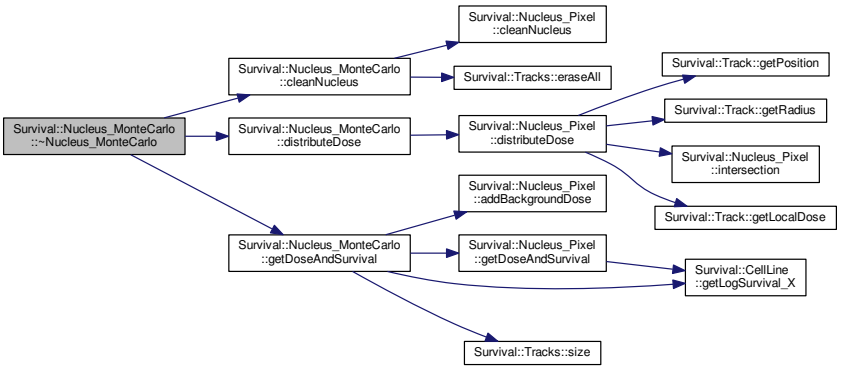
Definition at line 23 of file Nucleus_MonteCarlo.cpp.

7.8.2.2 Survival::Nucleus_MonteCarlo::~Nucleus_MonteCarlo ( ) [inline]

Destructor.

Definition at line 59 of file Nucleus_MonteCarlo.h.

Here is the call graph for this function:



7.8.3 Member Function Documentation

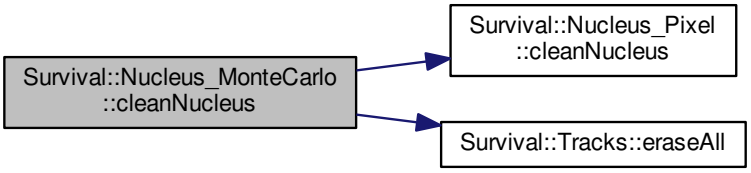
7.8.3.1 void Nucleus_MonteCarlo::cleanNucleus ( ) [virtual]

Clean the object by calling the Tracks::eraseAll() and Nucleus_Pixel::cleanNucleus() methods.

Reimplemented from Survival::Nucleus_Pixel.

Definition at line 60 of file Nucleus_MonteCarlo.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

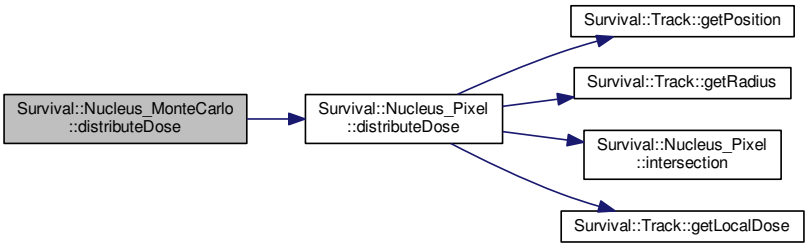
7.8.3.2 void Nucleus_MonteCarlo::distributeDose (const Track & *track*) [virtual]

It simply calls Nucleus_Pixel::distributeDose(const Track &track) to evaluate the dose deposited in the nucleus and appends the Track object to distributedTracks.

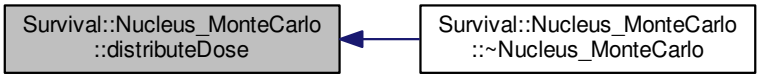
Reimplemented from Survival::Nucleus_Pixel.

Definition at line 69 of file Nucleus_MonteCarlo.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.3.3 void Nucleus_MonteCarlo::distributeDose (const Tracks & *tracks*) [virtual]

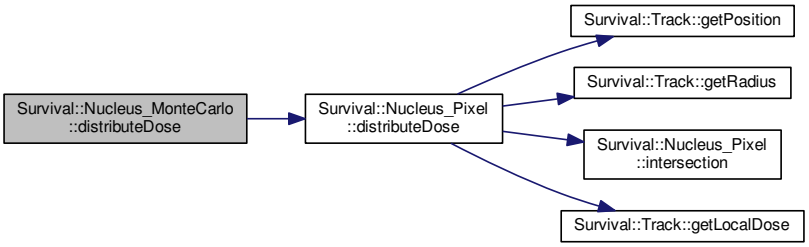
Overload of distributeDose(const Track &track) to manage a Tracks object.

It simply calls the method Nucleus_Pixel::distributeDose(const Tracks &tracks) and appends the Tracks object to distributedTracks.

Reimplemented from Survival::Nucleus_Pixel.

Definition at line 78 of file Nucleus_MonteCarlo.cpp.

Here is the call graph for this function:



7.8.3.4 void Nucleus_MonteCarlo::getDoseAndSurvival (double & *dose,* double & *doseUncertainty,* double & *survival,* double & *survivalUncertainty*)

Perform a Monte Carlo simulation to integrate over the nucleus the dose deposited by a track distribution in a stochastic way.

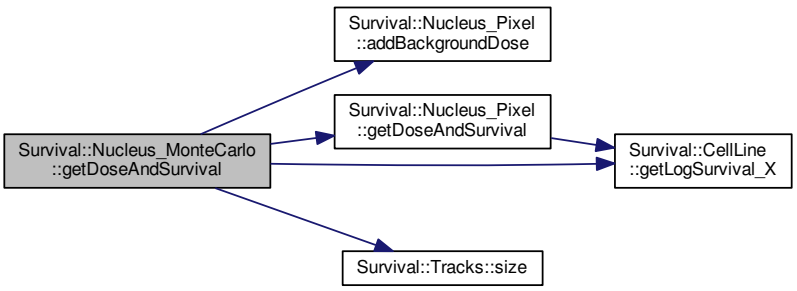
The approximation of the integrand is given by the pixel-wise constant dose profile generated with the Nucleus_↩ Pixel class; to avoid having zero values in some points of the nucleus, an arbitrary local dose background of 1 Gy is added. This approximation of dose profile is then normalized by its integral and is interpreted as a probability distribution; the related cumulative distribution is used to extract a position in the nucleus where to compute the exact local dose value and the consequent local number of lethal events. The average of the obtained lethal events values divided by their probability of extraction gives an estimate of the integral. The precision of the estimate is of course dependent on the number of extractions; the algorithm cycles until the user defined precision is reached (see Nucleus_MonteCarlo()).

Parameters

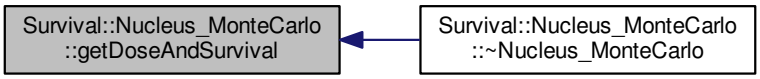
dose	The total dose deposited in the nucleus by the radiation, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival observed related to the dose absorbed.
survivalUncertainty	The uncertainty associated to the cellular survival.

Definition at line 87 of file Nucleus_MonteCarlo.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.4 Member Data Documentation

7.8.4.1 Tracks Survival::Nucleus_MonteCarlo::distributedTracks [private]

A Tracks object storing all Track objects interacting with the nucleus.

Definition at line 90 of file Nucleus_MonteCarlo.h.

7.8.4.2 long int Survival::Nucleus_MonteCarlo::numberOfIterations [private]

One of the two ending conditions of the Monte Carlo simulation. Fix the maximum number of iterations executable.

Definition at line 93 of file Nucleus_MonteCarlo.h.

7.8.4.3 double Survival::Nucleus_MonteCarlo::relativeStdDeviation [private]

One of the two ending conditions of the Monte Carlo simulation. Fix a constraint on the precision that is the maximum relative error on the cell survival evaluated.

Definition at line 96 of file Nucleus_MonteCarlo.h.

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

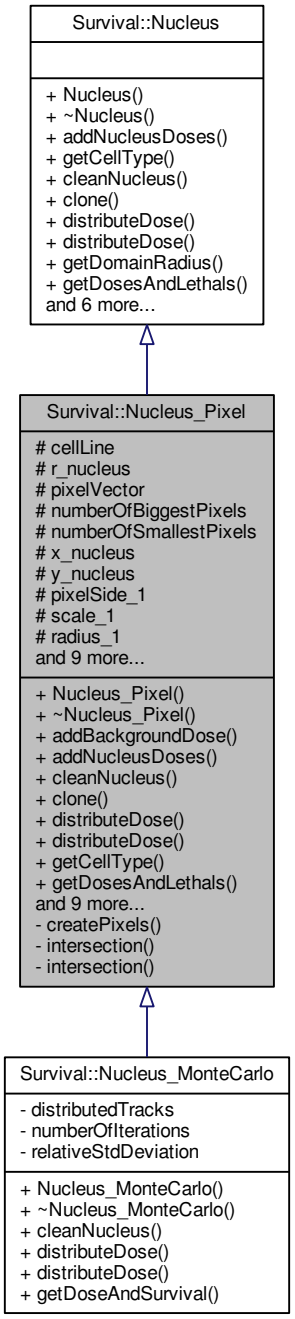
- include/Nucleus_MonteCarlo.h
- src/Nucleus_MonteCarlo.cpp

7.9 Survival::Nucleus_Pixel Class Reference

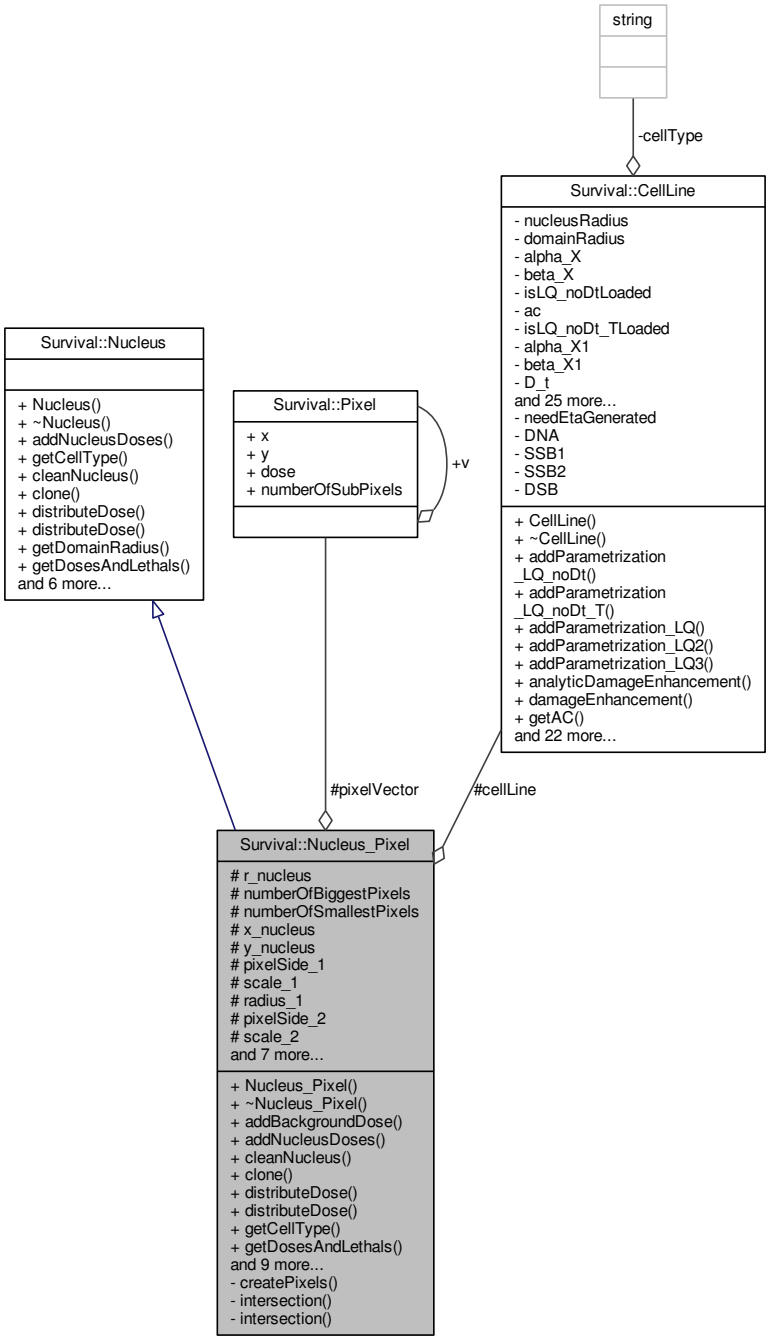
Inherited from the Nucleus pure virtual class, it implements the nucleus structure used in the LEM.

`#include <Nucleus_Pixel.h>`

Inheritance diagram for Survival::Nucleus_Pixel:



Collaboration diagram for Survival::Nucleus_Pixel:



Public Member Functions

- Nucleus_Pixel (const CellLine &cellLineRef, const double xPosition=0.0, const double yPosition=0.0, const double pixelSide1=0.005, const int scale1=2, const double radius1=0.1, const int scale2=10, const double radius2=1.0, const int scale3=10, const double radius3=10.0)

  Constructor. Instantiates and sets the object.
- virtual ~Nucleus_Pixel ()

Destructor.

- void addBackgroundDose (const double dose)

  Adds a constant value of dose absorbed in each pixel of the nucleus.
- virtual void addNucleusDoses (Nucleus_Pixel &nucleus)

  Performs the sum of the dose absorbed by two different nucleus pixel by pixel, starting from the smallest grid.
- virtual void cleanNucleus ()

  Resets to zero all counters (inNucleusCount and intersectionCount) and doses absorbed in the nucleus, pixel by pixel.
- virtual Nucleus_Pixel * clone (const CellLine &)

  Returns a pointer to a new Nucleus_Pixel object. It not really a clone but a new clean object.
- virtual void distributeDose (const Track &track)

  Distributes the dose deposited by a Track in the pixels constituting the nucleus.
- virtual void distributeDose (const Tracks &tracks)

  Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for every track of the container.
- virtual std::string getCellType () const

  Returns the name of the cell line to which the nucleus belongs.
- void getDosesAndLethals (std::vector< double > &doses, std::vector< double > &dosesUncertainty, std::vector< double > &lethals, std::vector< double > &lethalsUncertainty) const

  Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated lethal events, with respective uncertainties.
- virtual void getDoseAndSurvival (double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty) const

  Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated survival, with respective uncertainties.
- virtual int getInNucleusCount () const

  Returns the number of times that the nucleus has been crossed through by a Particle.
- virtual int getIntersectionCount () const

  Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.
- int getNumberOfBiggestPixels ()

  Returns the number of pixels constituting the biggest grid.
- int getNumberOfSmallestPixels ()

  The number of pixels constituting the smallest grid.
- virtual void getPosition (double &returnX, double &returnY) const

  Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.
- virtual double getRadius () const

  Returns the radius of the nucleus, expressed in um.
- void saveLocalDose (const std::string fileName) const

  Saves (writing on a file) the local dose deposited in the smallest pixels and their coordinates.
- void writeDoses (std::vector< double > &doses)

  Sets a value of dose in each of the smallest pixels constituting the nucleus, getting it from an external vector.

Protected Attributes

- const CellLine & cellLine

  A reference to a CellLine object where the characteristics of the cell line to which the nucleus belongs are stored.
- double r_nucleus

  The radius of the nucleus, expressed in um.
- Pixel * pixelVector

  A pointer to the pixels of the biggest sub-grid.
- int numberOfBiggestPixels

The number of pixels constituting the biggest grid.

- int numberOfSmallestPixels

  The number of pixels constituting the smallest grid.

- const double x_nucleus

  The position of the center of the nucleus (x coordinate) referred to the beam axis, expressed in mm.

- const double y_nucleus

  The position of the center of the nucleus (y coordinate) referred to the beam axis, expressed in mm.

- const double pixelSide_1

  The side of the smallest (or the third) sub-grid of pixels, expressed in um. Default: 0.005 um.

- const int scale_1

  The scale factor between the second and the third subgrid of pixels. Default: 2.

- const double radius_1

  The radius of the smallest circumference that defines the sampling of the track, expressed in um.

- const double pixelSide_2

  The side of the second sub-grid of pixels, expressed in um. Default: 0.01 um.

- const int scale_2

  The scale factor between the first and the second subgrid of pixels. Default: 10.

- const double radius_2

  The radius of the second circumference that defines the sampling of the track, expressed in um.

- const double pixelSide_3

  The side of the first sub-grid of pixels, expressed in um. Default: 0.1 um.

- const int scale_3

  The scale factor between the biggest grid of pixel and the first sub-grid. Default: 10.

- const double radius_3

  The radius of the biggest circumference that defines the sampling of the track, expressed in um.

- const double pixelSide_4

  The side of the biggest grid of pixels, expressed in um. Default: 1 um.

- int inNucleusCount

  The number of times that the nucleus has been crossed through by a Particle.

- int intersectionCount

  The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

Private Member Functions

- void createPixels ()

  Called by the constructor to create the pixels structure.

- bool intersection (const double x_pixel, const double y_pixel, const double pixel_side) const

  Determines if there is intersection between a pixel of the grid and the nucleus.

- bool intersection (const double x_pixel, const double y_pixel, const double pixel_side, const double x_track, const double y_track, const double radius, double &distance) const

  Determines if there is intersection between a pixel and a circle with a specified radius centered in the track position.

7.9.1 Detailed Description

Inherited from the Nucleus pure virtual class, it implements the nucleus structure used in the LEM.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2007

It performs the integration on several grids of pixels of varying resolution. Those grids are used to sample with a higher spatial frequency only when needed (i.e. near the position of ion transversals, where the local dose is rapidly varying). Thanks to this approach the single-event survival evaluation is both fast and accurate.

Definition at line 55 of file Nucleus_Pixel.h.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Nucleus_Pixel::Nucleus_Pixel (const CellLine & *cellLineRef*, const double *xPosition* = 0.0, const double *yPosition* = 0.0, const double *pixelSide1* = 0.005, const int *scale1* = 2, const double *radius1* = 0.1, const int *scale2* = 10, const double *radius2* = 1.0, const int *scale3* = 10, const double *radius3* = 10.0)

Constructor. Instantiates and sets the object.

It divided the nucleus creating the pixel-structure by means of the createPixels() method. It creates four grid of pixels of decreasing dimension.

Parameters

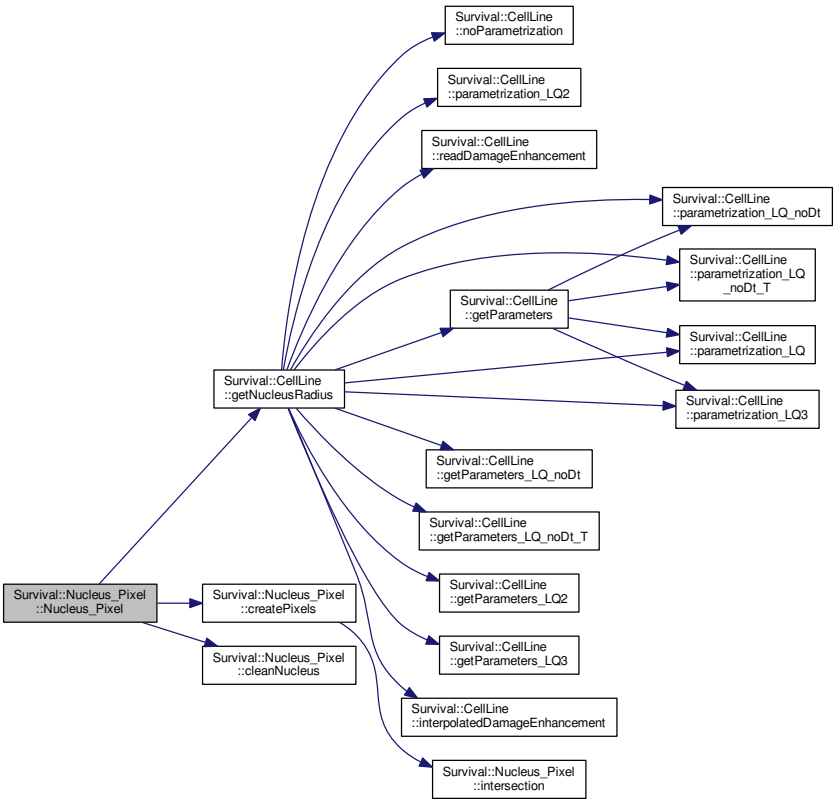
cellLineRef	A reference to the corresponding CellLine.
xPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.
yPosition	The nucleus position (y coordinate of the center) referred to the beam axis, expressed in mm.
pixelSide1	The side of the smallest (or the third) sub-grid of pixels (pixelSide_1), expressed in um.
scale1	The scale factor between the second and the third subgrid of pixels (scale_1).
radius1	The radius of the smallest circumference that defines the sampling of the track, expressed in um.
scale2	The scale factor between the first and the second subgrid of pixels (scale_2).
radius2	The radius of the second circumference that defines the sampling of the track, expressed in um.
scale3	The scale factor between the biggest grid of pixel and the first sub-grid (scale_3).
radius3	The radius of the biggest circumference that defines the sampling of the track, expressed in um.

See also

createPixels() and distributeDose()

Definition at line 35 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

7.9.2.2 Nucleus_Pixel::~Nucleus_Pixel ( ) [virtual]

Destructor.

Cyclically deletes all pixels created, starting from the smallest grid and ending with the biggest one.

See also

Pixel and createPixels()

Definition at line 72 of file Nucleus_Pixel.cpp.

7.9.3 Member Function Documentation

7.9.3.1 void Nucleus_Pixel::addBackgroundDose (const double *dose*)

Adds a constant value of dose absorbed in each pixel of the nucleus.

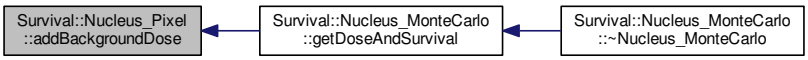
The method adds systematically a constant value of dose in each pixel my means of a `for` loop over pixelVector.

Parameters

dose	The dose to be added expressed in Gy.

Definition at line 90 of file Nucleus_Pixel.cpp.

Here is the caller graph for this function:



7.9.3.2 void Nucleus_Pixel::addNucleusDoses (**Nucleus_Pixel** & *nucleus*) [virtual]

Performs the sum of the dose absorbed by two different nucleus pixel by pixel, starting from the smallest grid.

This method cycles in some nested `for` loops over the pixels of each subgrid, starting from the smallest one, and adds the doses of the correspondent pixel in the other nucleus.

Warning

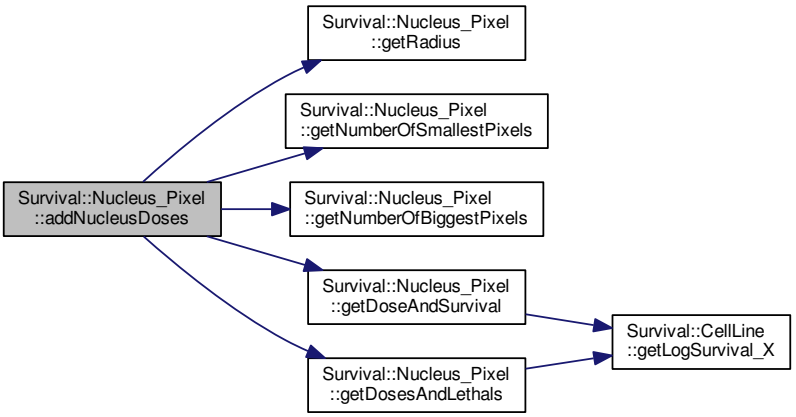
> The function doesn't do anything if the other nucleus is not geometrically similar to this one (e.g. different r_nucleus of number of pixels - numberOfSmallestPixels, numberOfBiggestPixels), but the execution of the program is not terminated.

Parameters

nucleus	A reference to another Nucleus_MKM object to evaluate the sum of the doses.

Definition at line 98 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:



7.9.3.3 void Nucleus_Pixel::cleanNucleus ( ) [virtual]

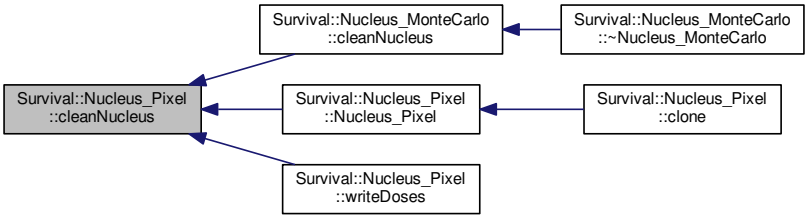
Resets to zero all counters (inNucleusCount and intersectionCount) and doses absorbed in the nucleus, pixel by pixel.

Implements Survival::Nucleus.

Reimplemented in Survival::Nucleus_MonteCarlo.

Definition at line 141 of file Nucleus_Pixel.cpp.

Here is the caller graph for this function:



7.9.3.4 Nucleus_Pixel * Nucleus_Pixel::clone (const CellLine & *cellLine*) [virtual]

Returns a pointer to a new Nucleus_Pixel object. It not really a clone but a new clean object.

Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

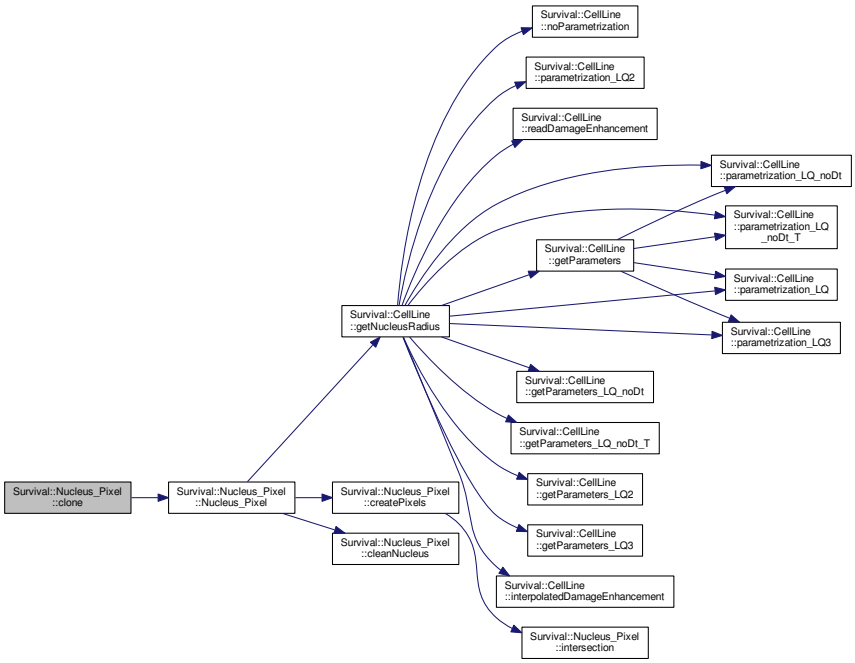
Note

To create a real clone of another nucleus, a better implementation of the copy constructor is needed.

Implements Survival::Nucleus.

Definition at line 166 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:



7.9.3.5 void Nucleus_Pixel::createPixels ( ) [private]

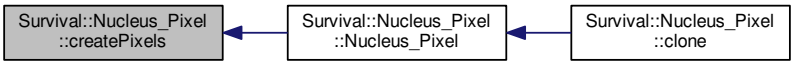
Called by the constructor to create the pixels structure.

The nucleus is divided into four grid of pixels of decreasing dimension. Starting from the biggest grid, the function perform a loop over the pixels and for each pixel it defines the `x` and `y` coordinates with respect to the beam axis (in um). In this way the first grid is created. Then, pixel by pixel, the method verify if there is intersection with the nucleus by means of the intersection(const double, const double, const double) function and, if there is, it proceeds recursively creating a subgrid of pixels inside it in the same way, and so on till the smallest grid.

Definition at line 402 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



7.9.3.6 void Nucleus_Pixel::distributeDose (const Track & *track*) [virtual]

Distributes the dose deposited by a Track in the pixels constituting the nucleus.

The sampling of the track is divided into four zone delimited by three concentric circumferences of decreasing radius. Starting from the biggest grid, for each pixel it determines if there is intersection with the biggest circle defined for the sampling by calling the intersection(const double, const double, const double, const double, const double, const double, double) method; if the answer is:

- no: then it calls Track::getLocalDose() to evaluate the dose deposited in that pixel (obviously only if its distance from the track is smaller than the track radius).
- yes: then it recursively does the same process on the inner grid of pixels. In this way, the track is sampled with a higher spatial frequency only when needed (i.e. near the position of ion transversals, where the local dose is rapidly varying); thanks to this approach the single-event survival evaluation is both fast and accurate.

Parameters

track	A reference to the track interacting with the nucleus.

See also

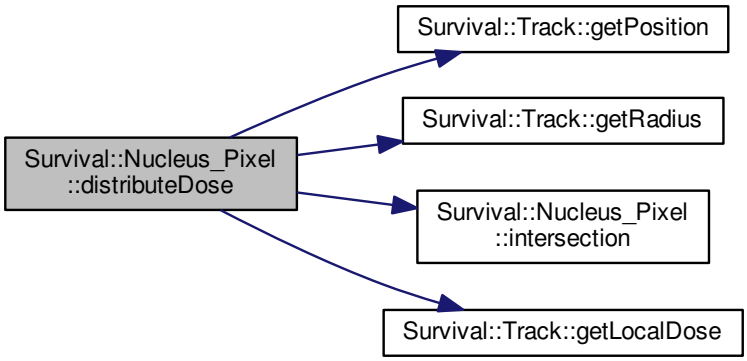
createPixels()

Implements Survival::Nucleus.

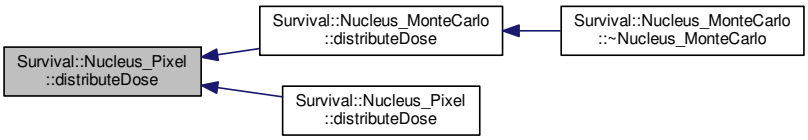
Reimplemented in Survival::Nucleus_MonteCarlo.

Definition at line 174 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.3.7 void Nucleus_Pixel::distributeDose (const Tracks & *tracks*) [virtual]

Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for every track of the container.

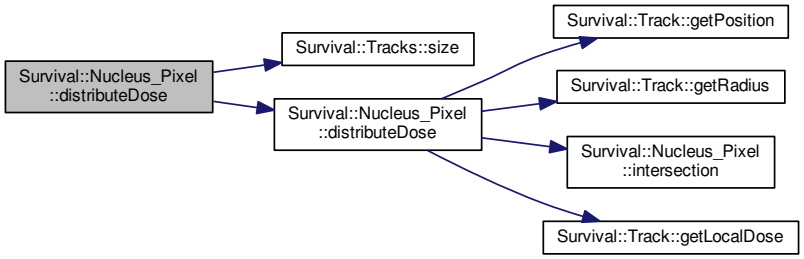
Since the Tracks class is a container for Track objects, this method calls distributeDose(const Track &track) in a `for` loop over each track contained in the Tracks object.

Implements Survival::Nucleus.

Reimplemented in Survival::Nucleus_MonteCarlo.

Definition at line 242 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:



7.9.3.8 string Nucleus_Pixel::getCellType ( ) const [virtual]

Returns the name of the cell line to which the nucleus belongs.

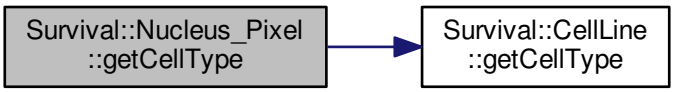
Returns

A `string` corresponding to the name of the cell line to which the nucleus belongs, getting the information by cellLine.

Implements Survival::Nucleus.

Definition at line 250 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:



7.9.3.9 void Nucleus_Pixel::getDoseAndSurvival (double & *dose*, double & *doseUncertainty*, double & *survival*, double & *survivalUncertainty*) const [virtual]

Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated survival, with respective uncertainties.

Through four nested `for` loops over the grid of pixels it evaluates the total dose absorbed by the nucleus, considering the dose absorbed by each pixel opportunely normalized, parallel evaluating the total number of lethal events observed, calculated by means of the selected parametrization (CellLine::getLogSurvival_X()). The cellular survival is then evaluated by means of the following relation:

$$S = \exp(-L_{TOT})$$

where L_{TOT} is the sum of all the lethal events observed.

Parameters

dose	The total dose absorbed by the nucleus, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival associated to the dose absorbed by the nucleus, passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.

Note

The method was thought to associate also an uncertainty to dose and survival, but this possibility hasn't been implemented yet, therefore actually -1 is assigned to those values.

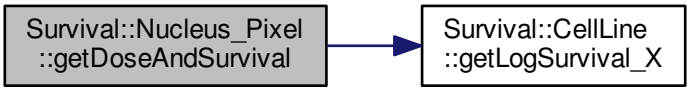
See also

getDosesAndLethals()

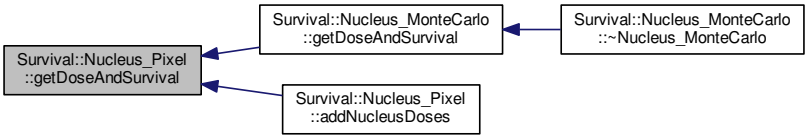
Implements Survival::Nucleus.

Definition at line 298 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.3.10 void Nucleus_Pixel::getDosesAndLethals (std::vector< double > & *doses*, std::vector< double > & *dosesUncertainty*, std::vector< double > & *lethals*, std::vector< double > & *lethalsUncertainty*) const

Returns (overwriting parameters passed by reference) the dose absorbed by the nucleus and the associated lethal events, with respective uncertainties.

Through four nested `for` loops over the grid of pixels it evaluates the total dose absorbed by the nucleus, considering the dose absorbed by each pixel opportunely normalized, parallel evaluating the total number of lethal events observed, calculated by means of the selected parametrization (CellLine::getLogSurvival_X()).

Parameters

doses	The vector of doses absorbed (in Gy), each element refers to a specific pixel, passed by reference to be overwritten.
dosesUncertainty	The vector of uncertainties associated to doses absorbed (in Gy), each element refers to a specific pixel, passed by reference to be overwritten.
lethals	The vector of lethal events observed, each element refers to a specific pixel, passed by reference to be overwritten.
lethalsUncertainty	The vector of uncertainties associated to the number of lethal events observed, each element refers to a specific pixel, passed by reference to be overwritten.

Note

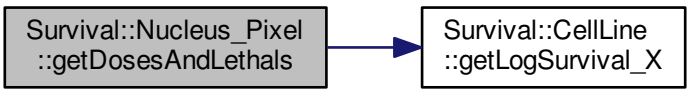
The method was thought to associate also an uncertainty to doses and lethals, but this possibility hasn't been implemented yet, therefore actually -1 is assigned to those values.

See also

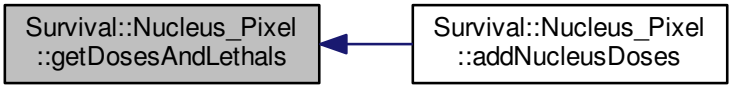
getDoseAndSurvival()

Definition at line 257 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.3.11 virtual int Survival::Nucleus_Pixel::getInNucleusCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus has been crossed through by a Particle.

Returns

inNucleusCount The number of times that the nucleus has been crossed through by a Particle.

See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 197 of file Nucleus_Pixel.h.

7.9.3.12 virtual int Survival::Nucleus_Pixel::getIntersectionCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

Returns

intersectionCount The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 205 of file Nucleus_Pixel.h.

7.9.3.13 int Survival::Nucleus_Pixel::getNumberOfBiggestPixels ( ) [inline]

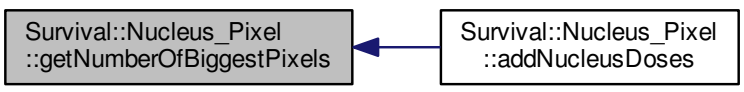
Returns the number of pixels constituting the biggest grid.

Returns

The number of pixels constituting the biggest grid (numberOfBiggestPixels).

Definition at line 211 of file Nucleus_Pixel.h.

Here is the caller graph for this function:



7.9.3.14 int Survival::Nucleus_Pixel::getNumberOfSmallestPixels ( ) [inline]

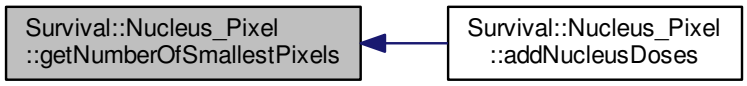
The number of pixels constituting the smallest grid.

Returns

The number of pixels constituting the biggest grid (numberOfSmallestPixels).

Definition at line 217 of file Nucleus_Pixel.h.

Here is the caller graph for this function:



7.9.3.15 void Nucleus_Pixel::getPosition (double & *returnX*, double & *returnY*) const [virtual]

Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the nucleus referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `y` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.

See also

Track::getPosition()

Implements Survival::Nucleus.

Definition at line 337 of file Nucleus_Pixel.cpp.

7.9.3.16 virtual double Survival::Nucleus_Pixel::getRadius ( ) const [inline], [virtual]

Returns the radius of the nucleus, expressed in um.

Returns

The radius of the nucleus expressed in um.

Implements Survival::Nucleus.

Definition at line 235 of file Nucleus_Pixel.h.

Here is the caller graph for this function:

7.9.3.17 bool Nucleus_Pixel::intersection (const double *x_pixel,* const double *y_pixel,* const double *pixel_side*) const [inline],[private]

Determines if there is intersection between a pixel of the grid and the nucleus.

Evaluates if there is at least one point of the pixel intersecting the circular nucleus looking at the pixel coordinates first, then at the innermost vertex and finally at the innermost point of the edge with respect to the center of the nucleus.

Parameters

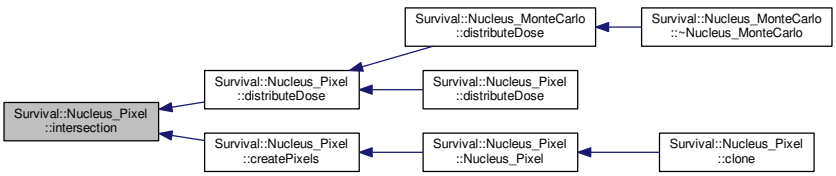
x_pixel	The x coordinate of the pixel referred to the beam axis, expressed in um.
y_pixel	The x coordinate of the pixel referred to the beam axis, expressed in um.
pixel_side	The length of the pixel side, expressed in um.

Returns

A boolean value to indicate if an intersection occurs.

Definition at line 548 of file Nucleus_Pixel.cpp.

Here is the caller graph for this function:



7.9.3.18 bool Nucleus_Pixel::intersection (const double *x_pixel,* const double *y_pixel,* const double *pixel_side,* const double *x_track,* const double *y_track,* const double *radius,* double & *distance*) const [inline], [private]

Determines if there is intersection between a pixel and a circle with a specified radius centered in the track position.

First identifies the distance between pixel and track, overwriting the distance-parameter. The it evaluates if there is at least one point of the pixel intersecting the circle looking at the distance first, then at the innermost vertex of the pixel and finally at the innermost point of the edge with respect to the position of the track.

Parameters

x_pixel	The position of the pixel (x coordinate) referred to the beam axis, expressed in um.
y_pixel	The position of the pixel (y coordinate) referred to the beam axis, expressed in um.
pixel_side	The length of the pixel edge, expressed in um.
x_track	The position of the track (x coordinate) referred to the beam axis, expressed in um.
y_track	The position of the track (y coordinate) referred to the beam axis, expressed in um.
radius	The radius of the circle expressed in um.
distance	The distance between pixel and track, expressed in um.

Definition at line 567 of file Nucleus_Pixel.cpp.

7.9.3.19 void Nucleus_Pixel::saveLocalDose (const std::string *fileName*) const

Saves (writing on a file) the local dose deposited in the smallest pixels and their coordinates.

This method write on a file, for each of the smallest pixel of the nucleus:

- The x and y coordinates (in um) identifying its position referred to the beam axis;
- The dose absorbed (in Gy);

Parameters

fileName	The name of the file where to save data.

Definition at line 346 of file Nucleus_Pixel.cpp.

7.9.3.20 void Nucleus_Pixel::writeDoses (std::vector< double > & *doses*)

Sets a value of dose in each of the smallest pixels constituting the nucleus, getting it from an external vector.

It runs in a nested for loops over the smallest pixels, for each one setting its Pixel::dose (getting it from an external vector)

Parameters

doses	A vector containing the doses (in Gy) to be assigned at each of the smallest pixels constituting the nucleus.

Definition at line 382 of file Nucleus_Pixel.cpp.

Here is the call graph for this function:

7.9.4 Member Data Documentation

7.9.4.1 const CellLine& Survival::Nucleus_Pixel::cellLine [protected]

A reference to a CellLine object where the characteristics of the cell line to which the nucleus belongs are stored.

Definition at line 300 of file Nucleus_Pixel.h.

7.9.4.2 int Survival::Nucleus_Pixel::inNucleusCount [protected]

The number of times that the nucleus has been crossed through by a Particle.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 374 of file Nucleus_Pixel.h.

7.9.4.3 int Survival::Nucleus_Pixel::intersectionCount [protected]

The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 380 of file Nucleus_Pixel.h.

7.9.4.4 int Survival::Nucleus_Pixel::numberOfBiggestPixels [protected]

The number of pixels constituting the biggest grid.

See also

createPixels()

Definition at line 317 of file Nucleus_Pixel.h.

7.9.4.5 int Survival::Nucleus_Pixel::numberOfSmallestPixels [protected]

The number of pixels constituting the smallest grid.

See also

createPixels()

Definition at line 323 of file Nucleus_Pixel.h.

7.9.4.6 const double Survival::Nucleus_Pixel::pixelSide_1 [protected]

The side of the smallest (or the third) sub-grid of pixels, expressed in um. Default: 0.005 um.

Definition at line 332 of file Nucleus_Pixel.h.

7.9.4.7 const double Survival::Nucleus_Pixel::pixelSide_2 [protected]

The side of the second sub-grid of pixels, expressed in um. Default: 0.01 um.

Definition at line 344 of file Nucleus_Pixel.h.

7.9.4.8 const double Survival::Nucleus_Pixel::pixelSide_3 [protected]

The side of the first sub-grid of pixels, expressed in um. Default: 0.1 um.

Definition at line 356 of file Nucleus_Pixel.h.

7.9.4.9 const double Survival::Nucleus_Pixel::pixelSide_4 [protected]

The side of the biggest grid of pixels, expressed in um. Default: 1 um.

Definition at line 368 of file Nucleus_Pixel.h.

7.9.4.10 Pixel* Survival::Nucleus_Pixel::pixelVector [protected]

A pointer to the pixels of the biggest sub-grid.

Definition at line 311 of file Nucleus_Pixel.h.

7.9.4.11 double Survival::Nucleus_Pixel::r_nucleus [protected]

The radius of the nucleus, expressed in um.

It's instantiated in the constructor getting the value from the CellLine object representing the cell line to which the nucleus belongs.

See also

Nucleus_Pixel()

Definition at line 308 of file Nucleus_Pixel.h.

7.9.4.12 const double Survival::Nucleus_Pixel::radius_1 [protected]

The radius of the smallest circumference that defines the sampling of the track, expressed in um.

See also

distributeDose()

Definition at line 341 of file Nucleus_Pixel.h.

7.9.4.13 const double Survival::Nucleus_Pixel::radius_2 [protected]

The radius of the second circumference that defines the sampling of the track, expressed in um.

See also

distributeDose()

Definition at line 353 of file Nucleus_Pixel.h.

7.9.4.14 const double Survival::Nucleus_Pixel::radius_3 [protected]

The radius of the biggest circumference that defines the sampling of the track, expressed in um.

See also

distributeDose()

Definition at line 365 of file Nucleus_Pixel.h.

7.9.4.15 const int Survival::Nucleus_Pixel::scale_1 [protected]

The scale factor between the second and the third subgrid of pixels. Default: 2.

Definition at line 335 of file Nucleus_Pixel.h.

7.9.4.16 const int Survival::Nucleus_Pixel::scale_2 [protected]

The scale factor between the first and the second subgrid of pixels. Default: 10.

Definition at line 347 of file Nucleus_Pixel.h.

7.9.4.17 const int Survival::Nucleus_Pixel::scale_3 [protected]

The scale factor between the biggest grid of pixel and the first sub-grid. Default: 10.

Definition at line 359 of file Nucleus_Pixel.h.

7.9.4.18 const double Survival::Nucleus_Pixel::x_nucleus [protected]

The position of the center of the nucleus (x coordinate) referred to the beam axis, expressed in mm.

Definition at line 326 of file Nucleus_Pixel.h.

7.9.4.19 const double Survival::Nucleus_Pixel::y_nucleus [protected]

The position of the center of the nucleus (y coordinate) referred to the beam axis, expressed in mm.

Definition at line 329 of file Nucleus_Pixel.h.

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

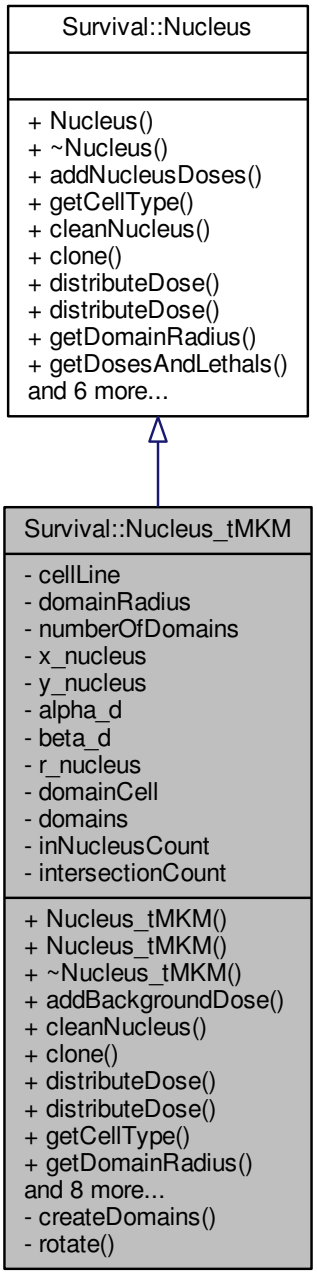
- include/Nucleus_Pixel.h
- src/Nucleus_Pixel.cpp

7.10 Survival::Nucleus_tMKM Class Reference

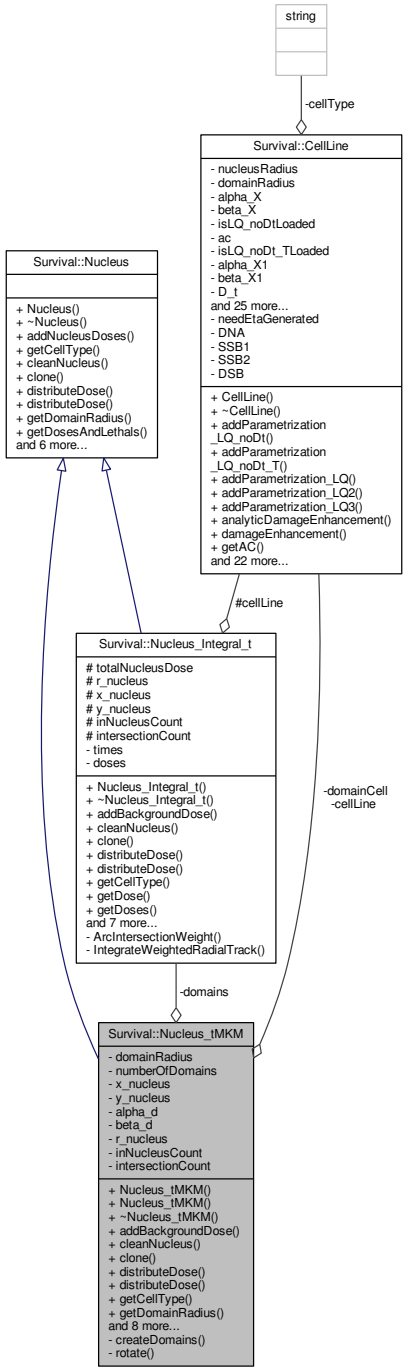
Inherited from the Nucleus pure virtual class, it implements the nucleus defined in the Monte Carlo temporal reformulation of the MKM model (MCt-MKM).

```
#include <Nucleus_tMKM.h>
```

Inheritance diagram for Survival::Nucleus_tMKM:



Collaboration diagram for Survival::Nucleus_tMKM:



Public Member Functions

- Nucleus_tMKM (const CellLine &cellLineRef, const double xPosition=0.0, const double yPosition=0.0)

  Constructor. Instantiates and sets the object.
- Nucleus_tMKM (const CellLine &cellLineRef, double domainRadius, int numberOfDomains, const double x↩ Position=0.0, const double yPosition=0.0)

  Instantiates and sets the object. Overload of the constructor.

- virtual ~Nucleus_tMKM ()

  Destructor.
- void addBackgroundDose (const double dose, const double t)

  Adds a constant value of dose absorbed in each domain of the nucleus in a specific instant.
- virtual void cleanNucleus ()

  Resets to zero all counters (inNucleusCount and intersectionCount) and doses in the nucleus and his domain.
- virtual Nucleus_tMKM * clone (const CellLine &)

  Returns a pointer to a new Nucleus_tMKM object. It not really a clone but a new clean object.
- virtual void distributeDose (const Track &track)

  When an interaction between a Particle and the tMKM nucleus occurs, the method increases inNucleusCount and intersectionCount counters and proceeds to distribute the dose in each domain.
- virtual void distributeDose (const Tracks &tracks)

  Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for each track of the container.
- virtual std::string getCellType () const

  Returns the name of the cell line to which the nucleus belongs.
- virtual double getDomainRadius ()

  Returns the radius of the domain corresponding to the CellLine to which the tMKM nucleus belongs.
- virtual void getDoseAndSurvival (double &dose, double &doseUncertainty, double &survival, double &survivalUncertainty) const

  Returns (overwriting parameters passed by reference) the total dose absorbed by the nucleus and the associated survival, evaluated taking into account the time structure of the irradiation, with respective uncertainties.
- double getDoseForDomain (int indexOfDomain) const

  Return the dose absorbed in the indexOfDomain-th domain, expressed in Gy.
- virtual int getInNucleusCount () const

  Returns the number of times that the nucleus has been crossed through by a Particle.
- virtual int getIntersectionCount () const

  Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.
- virtual int getNumberOfDomains ()

  Returns the number of domains composing the tMKM nucleus.
- virtual void getPosition (double &returnX, double &returnY) const

  Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.
- virtual double getRadius () const

  Returns the effective radius of the Nucleus_tMKM object.
- void saveLocalDose (const std::string fileName) const

  Save data corresponding to the dose absorbed by each domain and the number of lethal events observed, useful to debug.

Private Member Functions

- void createDomains ()

  Create the domains as pointers to Nucleus_Integral_t objects, placed to form a hexagonal shape, spiraling from (0,0).
- void rotate (double &xTranslation, double &yTranslation)

  Performs a 60 degrees clockwise rotation.

Private Attributes

- const CellLine & cellLine

  A reference to a CellLine object where the characteristics of the cell line to which the tMKM nucleus belongs are stored.
- double domainRadius

  The radius of the domain corresponding to the CellLine to which the tMKM nucleus belongs.
- int numberOfDomains

  The number of domains composing the tMKM nucleus.
- const double x_nucleus

  The position of the center of the nucleus (x coordinate) referred to the beam axis, expressed in mm.
- const double y_nucleus

  The position of the center of the nucleus (y coordinate) referred to the beam axis, expressed in mm.
- double alpha_d

  The linear-quadratic parameter α associated to each of the domains composing the tMKM nucleus.
- double beta_d

  The linear-quadratic parameter β associated to each of the domains composing the tMKM nucleus.
- double r_nucleus

  It's the effective radius of the Nucleus_tMKM object.
- CellLine * domainCell

  A pointer to a CellLine object, storing the information about the cell line to which the tMKM nucleus belongs.
- Nucleus_Integral_t ** domains

  A pointer to pointers, where the objects finally pointed are Nucleus_Integral_t objects corresponding to the domains composing the tMKM nucleus.
- int inNucleusCount

  The number of times that the nucleus has been crossed through by a Particle.
- int intersectionCount

  The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

7.10.1 Detailed Description

Inherited from the Nucleus pure virtual class, it implements the nucleus defined in the Monte Carlo temporal reformulation of the MKM model (MCt-MKM).

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2015

Similar in its structure to the Nucleus_MKM class, it provides some method to manage the temporal structure of the irradiation to support the MonteCarlo temporal-Microdosimetric Kinetic Model (MCt-MKM, 1). It keeps track of the history of the irradiation, associating to each dose deposited a precise temporal instant. The total number of lethal events observed and the associated cellular survival are evaluated considering also the repaired kinetics of the cell.

1. L. Manganaro, G. Russo, R. Cirio, F. Dalmasso, S. Giordanengo, V. Monaco, R. Sacchi, A. Vignati, A. Attili, "A novel formulation of the Microdosimetric Kinetic Model to account for dose-delivery time structure effects in ion beam therapy with application in treatment planning simulations", *Medical Physics*, (Submitted).

Definition at line 22 of file Nucleus_tMKM.h.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 Nucleus_tMKM::Nucleus_tMKM (const CellLine & *cellLineRef,* const double *xPosition* = 0.0, const double *yPosition* = 0.0)

Constructor. Instantiates and sets the object.

When the constructor is called, it instantiates the object creating a hexagonal structure of circular domains, using the informations stored in the CellLine reference object, such as the radius of the nucleus and the radius of the single domain. This is made by calling the createDomains() function.

Parameters

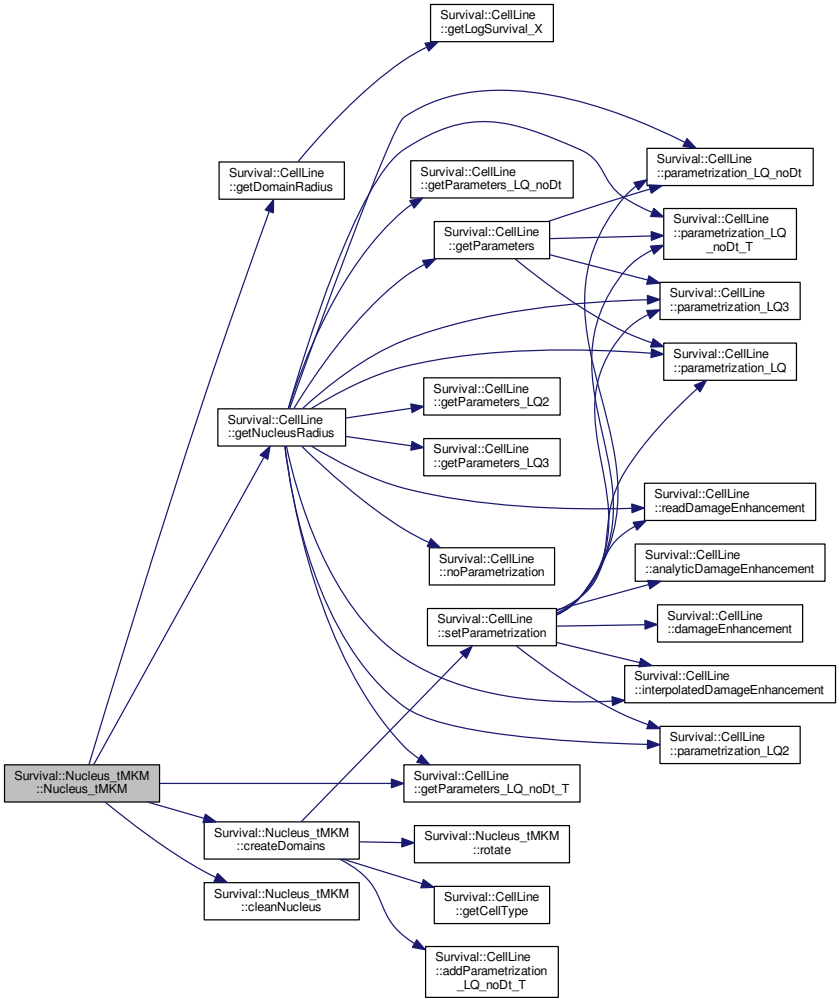
cellLineRef	A reference to the corresponding CellLine.
xPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.
yPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.

See also

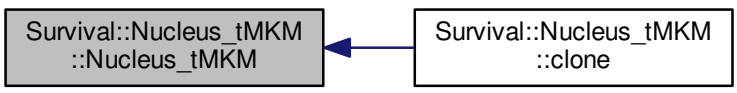
createDomains(), cleanNucleus(), Nucleus_tMKM(const CellLine&, double, int, const double, const double) and Nucleus_MKM

Definition at line 39 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.2.2 Nucleus_tMKM::Nucleus_tMKM (const CellLine & *cellLineRef,* double *domainRadius,* int *numberOfDomains,* const double *xPosition* = 0.0, const double *yPosition* = 0.0)

Instantiates and sets the object. Overload of the constructor.

Provide the possibility to specify also the total number of domains and the radius of each domain.

Parameters

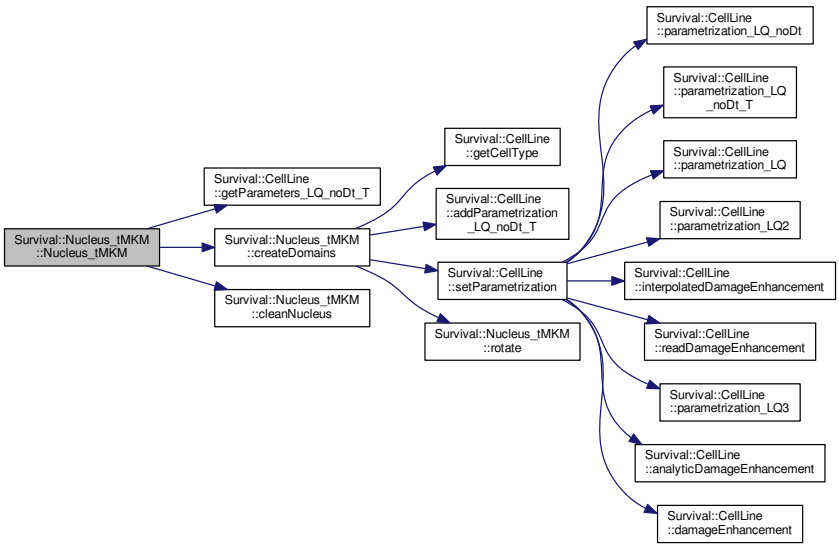
cellLineRef	A reference to the corresponding CellLine.
domainRadius	The radius of the single domain in the tMKM nucleus, expressed in um.
numberOfDomains	The number of domains that constitute the tMKM nucleus.
xPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.
yPosition	The nucleus position (x coordinate of the center) referred to the beam axis, expressed in mm.

See also

createDomains(), Nucleus_Integral_t and Nucleus_tMKM(const CellLine&, const double, const double)

Definition at line 67 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



7.10.2.3 Nucleus_tMKM::~Nucleus_tMKM ( ) [virtual]

Destructor.

The destructor deletes the domainCell, domains and each domains[i-th] pointers created when the object is instantiated.

See also

createDomains()

Definition at line 98 of file Nucleus_tMKM.cpp.

7.10.3 Member Function Documentation

7.10.3.1 void Nucleus_tMKM::addBackgroundDose (const double *dose*, const double *t*)

Adds a constant value of dose absorbed in each domain of the nucleus in a specific instant.

The method calls systematically for each domain the function Nucleus_Integral_t::addBackgroundDose(), that is the override of this method itself in the Nucleus_Integral_t class. The result is to add a constant value of dose absorbed in each domain (at a specific instant) and consequently in the whole nucleus.

Parameters

dose	The dose to be added expressed in Gy.
t	The time associated to the dose added, expressed in hours.

Definition at line 112 of file Nucleus_tMKM.cpp.

7.10.3.2 void Nucleus_tMKM::cleanNucleus ( ) [virtual]

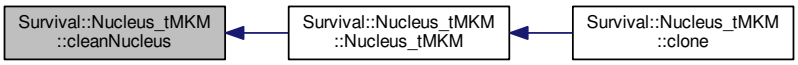
Resets to zero all counters (inNucleusCount and intersectionCount) and doses in the nucleus and his domain.

Resets to zero inNucleusCount and intersectionCount and calls systematically for each domain the function Nucleus_Integral_t::cleanNucleus(), that this the override of this method itself in the Nucleus_Integral_t class.

Implements Survival::Nucleus.

Definition at line 121 of file Nucleus_tMKM.cpp.

Here is the caller graph for this function:



7.10.3.3 Nucleus_tMKM * Nucleus_tMKM::clone (const CellLine & *cellLine*) [virtual]

Returns a pointer to a new Nucleus_tMKM object. It not really a clone but a new clean object.

Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

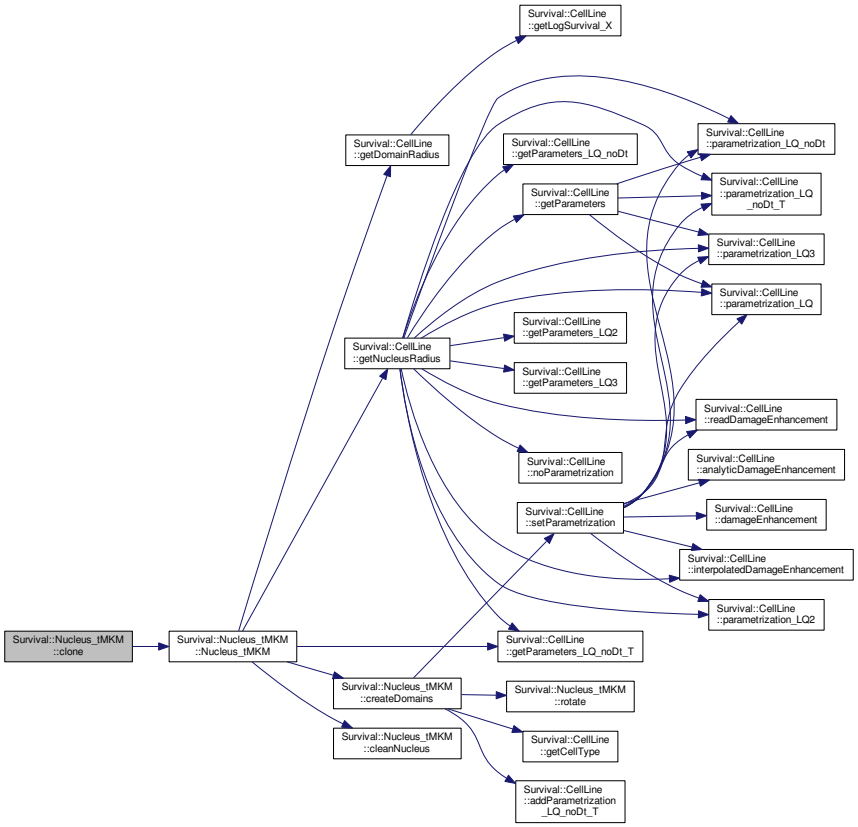
Note

To create a real clone of another nucleus, a better implementation of the copy constructor is needed.

Implements Survival::Nucleus.

Definition at line 132 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



7.10.3.4 void Nucleus_tMKM::createDomains ( ) [private]

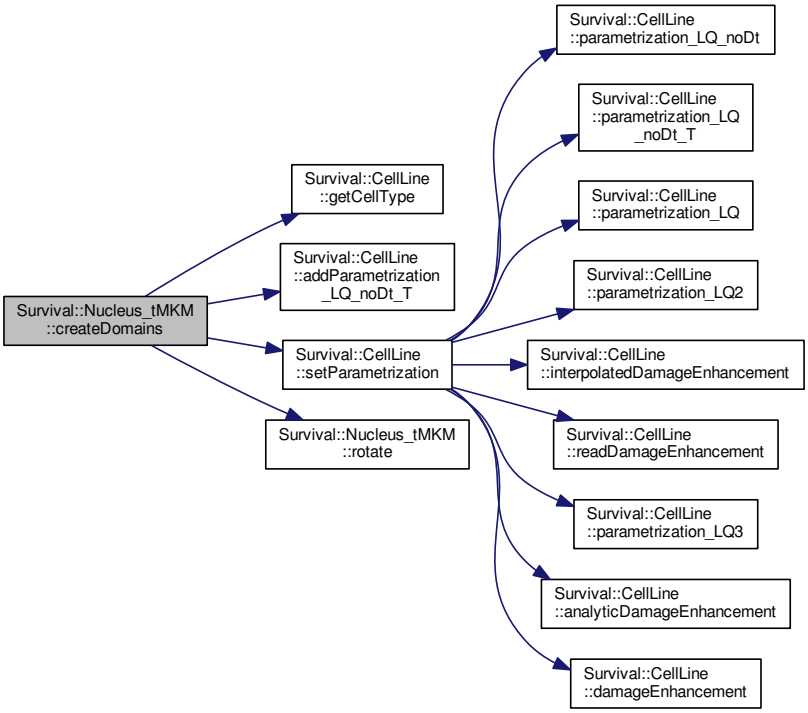
Create the domains as pointers to Nucleus_Integral_t objects, placed to form a hexagonal shape, spiraling from (0,0).

This function is called by the constructor every times a Nucleus_tMKM is created, and it's responsible to instantiate and place the domains in the right position.

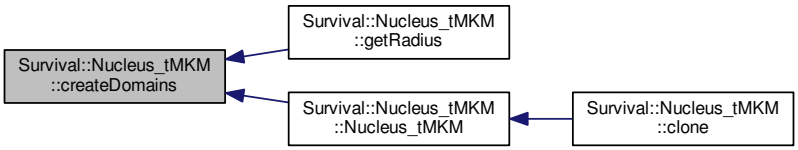
The structure is created in such a way that the center of each domain is placed on the vertex of a regular hexagon and the distance between two nearest neighbors is exactly equal to twice the radius of the domain (domainRadius). Some concentric hexagons are created to places all the domains defined (numberOfDomains). The center of each hexagon coincides with the position of the first domain created, that is also the center of the tMKM nucleus. This structure is created by means of the rotate() method.

Definition at line 141 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.3.5 void Nucleus_tMKM::distributeDose (const Track & *track*) [virtual]

When an interaction between a Particle and the tMKM nucleus occurs, the method increases inNucleusCount and intersectionCount counters and proceeds to distribute the dose in each domain.

This function is the first step to evaluate the dose deposited by the radiation in the nucleus or, better, in each domain of the tMKM nucleus. It checks if any interaction exists between Nucleus and Particle (that is the Track generated by the particle) looking at their positions and radius. If it's true, it increases the respective counters (inNucleus↩ Count and intersectionCount) and calls the method Nucleus_Integral_t::distributeDose() in a `for` loop over the total number of domains.

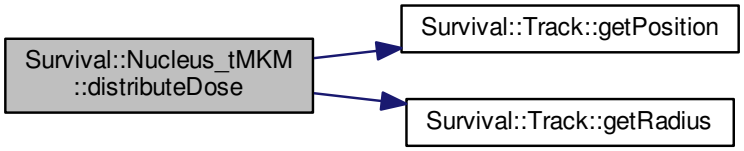
Parameters

track	A reference to the Track generated by the particle in the nucleus.

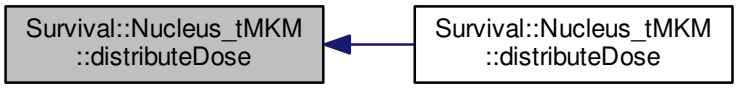
Implements Survival::Nucleus.

Definition at line 181 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.3.6 void Nucleus_tMKM::distributeDose (const Tracks & *tracks*) [virtual]

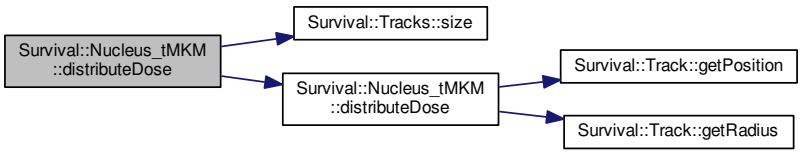
Overload of distributeDose(const Track &track) to manage a Tracks object, it simply calls distributeDose(const Track &track) for each track of the container.

Since the Tracks class is a container for Track objects, this method calls distributeDose(const Track &track) in a `for` loop over each track contained in the Tracks object.

Implements Survival::Nucleus.

Definition at line 203 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



7.10.3.7 string Nucleus_tMKM::getCellType ( ) const [virtual]

Returns the name of the cell line to which the nucleus belongs.

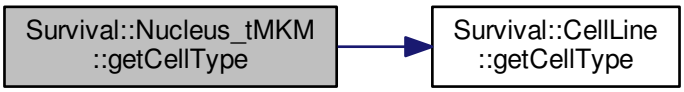
Returns

A `string` corresponding to the name of the cell line to which the nucleus belongs, getting the information by cellLine.

Implements Survival::Nucleus.

Definition at line 221 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



7.10.3.8 virtual double Survival::Nucleus_tMKM::getDomainRadius ( ) [inline], [virtual]

Returns the radius of the domain corresponding to the CellLine to which the tMKM nucleus belongs.

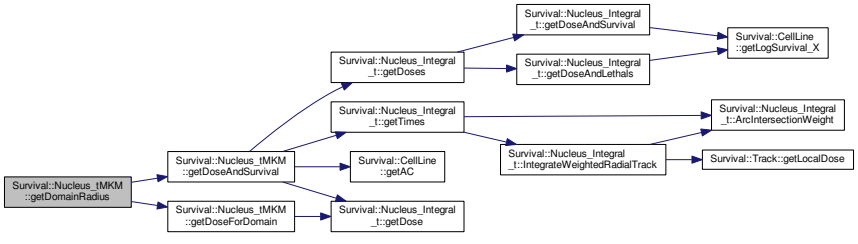
Returns

domainRadius That is the radius of the domain corresponding to the CellLine to which the tMKM nucleus belongs.

Reimplemented from Survival::Nucleus.

Definition at line 117 of file Nucleus_tMKM.h.

Here is the call graph for this function:



7.10.3.9 void Nucleus_tMKM::getDoseAndSurvival (double & *dose,* double & *doseUncertainty,* double & *survival,* double & *survivalUncertainty*) const [virtual]

Returns (overwriting parameters passed by reference) the total dose absorbed by the nucleus and the associated survival, evaluated taking into account the time structure of the irradiation, with respective uncertainties.

The function calls, in a `for` loop over the total number of domains, the methods Nucleus_Integral_t::getDoses() and Nucleus_Integral_t::getTimes() to get the complete history of doses deposited in the nucleus by the radiation, each dose associated to a specific instant. Then it evaluates for each domain the total number of lethal events observed, taking into account the time structure of the irradiation, by means of the following relation:

$$L_d = -\alpha_d \left(\sum_{i=1}^{N} z_i \right) - \beta_d \left(\sum_{i=1}^{N} \right)^2 - 2\beta \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[1 - \exp\left(-(a+c)(t_j - t_i) \right) \right] z_i \, z_j$$

where N represents the total number of interaction events in the domain, the sum (a+c) represents the time constant characteristics of the cellular repair kinetics and z_i and t_i represent the i-th element of the vectors of times and doses absorbed respectively.

The total number of lethal events in the nucleus is evaluated as the sum of the ones observed in each domain and the cellular survival is evaluated as a negative exponential function of the total number of lethal events (according to the poissonian statistics):

$$S = \exp(-L_{TOT})$$

Finally the method overwrite the total dose absorbed by the nucleus and the cellular survival with respective uncertainties.

Parameters

dose	The total dose absorbed by the nucleus, expressed in Gy, passed by reference to be overwritten.
doseUncertainty	The uncertainty associated to the dose absorbed, expressed in Gy, passed by reference to be overwritten.
survival	The cellular survival associated to the dose absorbed by the nucleus (with its specific time structure), passed by reference to be overwritten.
survivalUncertainty	The uncertainty associated to the cellular survival, passed by reference to be overwritten.

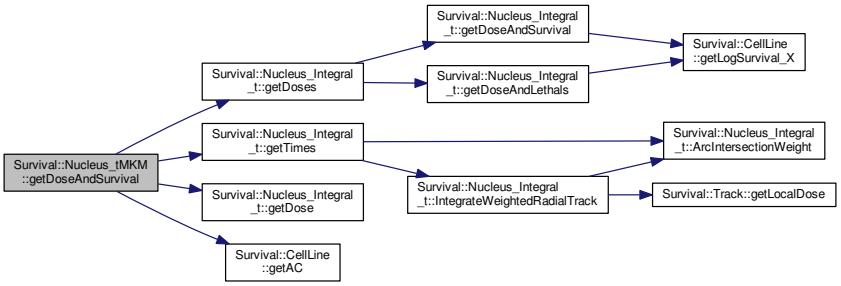
Note

The method was thought to associate also an uncertainty to dose and survival, but this possibility hasn't been implemented yet, therefore actually -1 is assigned to those values.

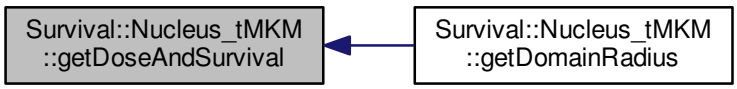
Implements Survival::Nucleus.

Definition at line 228 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.3.10 double Nucleus_tMKM::getDoseForDomain (int *indexOfDomain*) const

Return the dose absorbed in the indexOfDomain-th domain, expressed in Gy.

The function calls the method Nucleus_Integral_t::getDose() from the indexOfDomain-th domain.

If an incorrect index is selected (e.g. greater than the total number of domains) the dose absorbed is set to -1.

Parameters

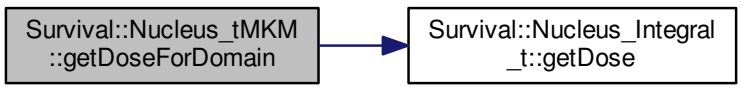
indexOfDomain	The index associated to the domain.

Returns

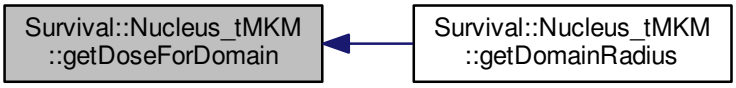
The dose absorbed in the indexOfDomain-th domain, expressed in Gy.

Definition at line 263 of file Nucleus_tMKM.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.3.11 virtual int Survival::Nucleus_tMKM::getInNucleusCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus has been crossed through by a Particle.

Returns

inNucleusCount The number of times that the nucleus has been crossed through by a Particle.

See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 164 of file Nucleus_tMKM.h.

7.10.3.12 virtual int Survival::Nucleus_tMKM::getIntersectionCount ( ) const [inline], [virtual]

Returns the number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

Returns

intersectionCount The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

See also

distributeDose(const Track&)

Implements Survival::Nucleus.

Definition at line 172 of file Nucleus_tMKM.h.

7.10.3.13 virtual int Survival::Nucleus_tMKM::getNumberOfDomains ( ) [inline], [virtual]

Returns the number of domains composing the tMKM nucleus.

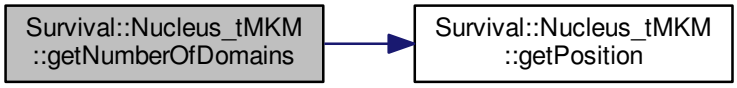
Returns

The number of domains composing the tMKM nucleus.

Reimplemented from Survival::Nucleus.

Definition at line 178 of file Nucleus_tMKM.h.

Here is the call graph for this function:



7.10.3.14 void Nucleus_tMKM::getPosition (double & *returnX*, double & *returnY*) const [virtual]

Returns the nucleus position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the nucleus referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `y` coordinate of the nucleus, expressed in mm, passed by reference to be overwritten.

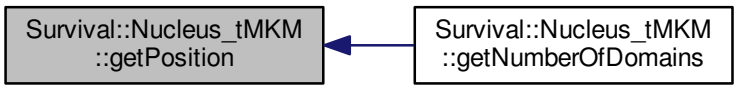
See also

Track_KieferChatterjee::getPosition()

Implements Survival::Nucleus.

Definition at line 276 of file Nucleus_tMKM.cpp.

Here is the caller graph for this function:



7.10.3.15 virtual double Survival::Nucleus_tMKM::getRadius ( ) const [inline], [virtual]

Returns the effective radius of the Nucleus_tMKM object.

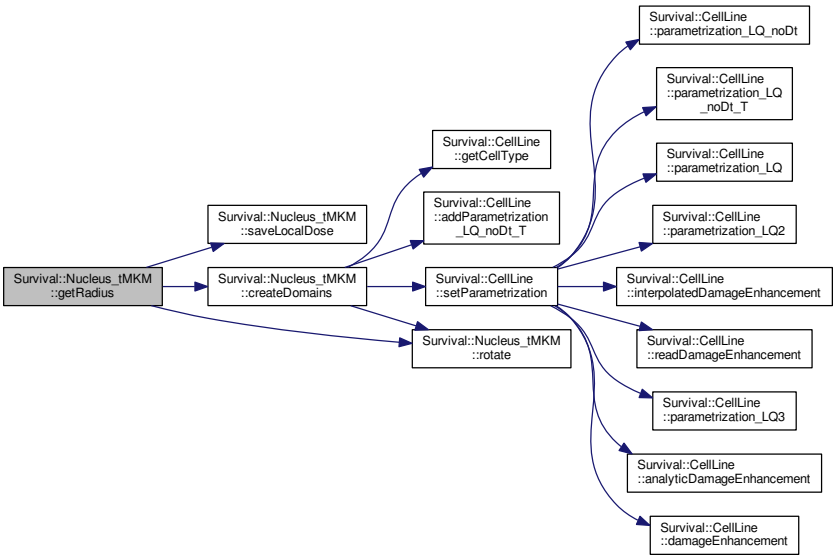
Returns

r_nucleus, the effective radius of the Nucleus_tMKM object. Since the structure of the final tMKM nucleus is "hexagon-like" this radius is different from the radius stored in the cellLine reference. It's the distance between the center of the nucleus and the farthest point of the nucleus itself, expressed in um.

Implements Survival::Nucleus.

Definition at line 196 of file Nucleus_tMKM.h.

Here is the call graph for this function:



7.10.3.16 void Nucleus_tMKM::rotate (double & *xTranslation,* double & *yTranslation*) [private]

Performs a 60 degrees clockwise rotation.

The rotation matrix is defined by:

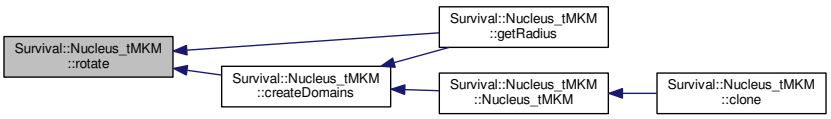
$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

Parameters

xTranslation	The x coordinate of the point where to start the 60 degrees clockwise rotation.
yTranslation	The y coordinate of the point where to start the 60 degrees clockwise rotation.

Definition at line 285 of file Nucleus_tMKM.cpp.

Here is the caller graph for this function:



7.10.3.17 void Nucleus_tMKM::saveLocalDose (const std::string *fileName*) const

Save data corresponding to the dose absorbed by each domain and the number of lethal events observed, useful to debug.

Warning

This method hasn't been implemented yet.

Parameters

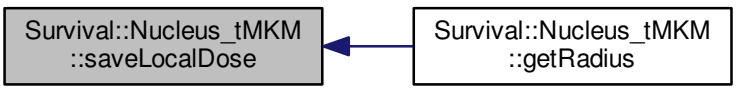
fileName	The name of the file where to save data.

Warning

The execution of the program will be terminated if fileName refers to an inexistent file.

Definition at line 296 of file Nucleus_tMKM.cpp.

Here is the caller graph for this function:



7.10.4 Member Data Documentation

7.10.4.1 double Survival::Nucleus_tMKM::alpha_d [private]

The linear-quadratic parameter α associated to each of the domains composing the tMKM nucleus.

It's instantiated in the constructor dividing the α parameter stored in the cellLine reference by the total number of domains (numberOfDomains): $\frac{\alpha}{N_d}$. Note that α is a parameter of the model, that ideally represents the value of the linear-quadratic $alpha_X$ parameter identified in the case of irradiation with a photon beam.

See also

CellLine, CellLine::getParameters_LQ_noDt() and Nucleus_tMKM()

Definition at line 267 of file Nucleus_tMKM.h.

7.10.4.2 double Survival::Nucleus_tMKM::beta_d [private]

The linear-quadratic parameter β associated to each of the domains composing the tMKM nucleus.

It's instantiated in the constructor dividing the β parameter stored in the cellLine reference by the total number of domains (numberOfDomains): $\frac{\beta}{N_d}$. Note that β is a parameter of the model, that ideally represents the value of the linear-quadratic $beta_X$ parameter identified in the case of irradiation with a photon beam.

See also

CellLine, CellLine::getParameters_LQ_noDt() and Nucleus_tMKM()

Definition at line 275 of file Nucleus_tMKM.h.

7.10.4.3 const CellLine& Survival::Nucleus_tMKM::cellLine [private]

A reference to a CellLine object where the characteristics of the cell line to which the tMKM nucleus belongs are stored.

Definition at line 234 of file Nucleus_tMKM.h.

7.10.4.4 CellLine* Survival::Nucleus_tMKM::domainCell [private]

A pointer to a CellLine object, storing the information about the cell line to which the tMKM nucleus belongs.

It's defined in the createDomains() method and deleted in the destructor ~Nucleus_tMKM.

Definition at line 291 of file Nucleus_tMKM.h.

7.10.4.5 double Survival::Nucleus_tMKM::domainRadius [private]

The radius of the domain corresponding to the CellLine to which the tMKM nucleus belongs.

This information is stored in the cellLine reference and then copied to this variable in the constructor. It's expressed in um.

See also

Nucleus_tMKM()

Definition at line 242 of file Nucleus_tMKM.h.

7.10.4.6 Nucleus_Integral_t** Survival::Nucleus_tMKM::domains [private]

A pointer to pointers, where the objects finally pointed are Nucleus_Integral_t objects corresponding to the domains composing the tMKM nucleus.

Definition at line 294 of file Nucleus_tMKM.h.

7.10.4.7 int Survival::Nucleus_tMKM::inNucleusCount [private]

The number of times that the nucleus has been crossed through by a Particle.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 300 of file Nucleus_tMKM.h.

7.10.4.8 int Survival::Nucleus_tMKM::intersectionCount [private]

The number of times that the nucleus interacted with a Particle that doesn't pass through the nucleus itself.

It's incremented by means of the distributeDose(const Track&) method.

Definition at line 306 of file Nucleus_tMKM.h.

7.10.4.9 int Survival::Nucleus_tMKM::numberOfDomains [private]

The number of domains composing the tMKM nucleus.

It's evaluated in the constructor as the ratio between the areas of the tMKM nucleus, whose radius R_N is stored in the cellLine reference, and the single domain, characterized by a radius R_d (domainRadius):

$$N_d = \frac{R_N^2}{R_d^2}$$

See also

Nucleus_tMKM()

Definition at line 253 of file Nucleus_tMKM.h.

7.10.4.10 double Survival::Nucleus_tMKM::r_nucleus [private]

It's the effective radius of the Nucleus_tMKM object.

Since the structure of the final tMKM nucleus is "hexagon-like" this radius is different from the radius stored in the cellLine reference. It's the distance between the center of the nucleus and the farthest point of the nucleus itself.

It's is defined in the createDomains() function, called in the constructor, and it's expressed in um.

See also

Nucleus_tMKM()

Definition at line 285 of file Nucleus_tMKM.h.

7.10.4.11 const double Survival::Nucleus_tMKM::x_nucleus [private]

The position of the center of the nucleus (x coordinate) referred to the beam axis, expressed in mm.

Definition at line 256 of file Nucleus_tMKM.h.

7.10.4.12 const double Survival::Nucleus_tMKM::y_nucleus [private]

The position of the center of the nucleus (y coordinate) referred to the beam axis, expressed in mm.

Definition at line 259 of file Nucleus_tMKM.h.

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

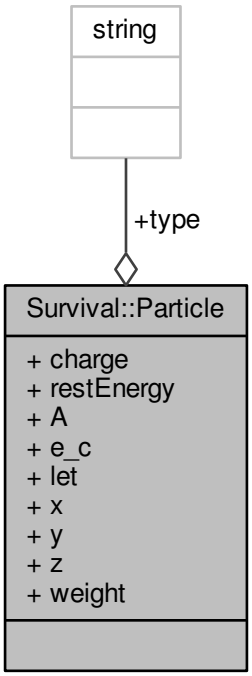
- include/Nucleus_tMKM.h
- src/Nucleus_tMKM.cpp

7.11 Survival::Particle Class Reference

This class defines the object "particle".

```
#include <Particle.h>
```

Collaboration diagram for Survival::Particle:



Public Attributes

- std::string type

  The type of particle (e.g. Chemical symbol for ions: H, He, Li, ...).
- int charge

  The charge of the particle expressed in elementary charge units.
- double restEnergy

  The rest energy of the particle expressed in MeV.
- int A

  The mass number of the particle.
- double e_c

  The kinetic energy of the particle expressed in MeV.
- double let

  The LET in water of the particle expressed in MeV/um (according to the Bethe-Bloch formula).
- double x

  The particle position (x coordinate) referred to the beam axis, expressed in mm.
- double y

  The particle position (y coordinate) referred to the beam axis, expressed in mm.
- double z

  The particle position (z coordinate) referred to the depth of penetration in matter.
- double weight

  The weight of this particular particle type in the beam. Useful in the case of "mixed fields".

7.11.1 Detailed Description

This class defines the object "particle".

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2007

This class defines the object "particle". It is used as a C++ struct to contain, for a certain particle in a given position in space, recorded characteristics like type, charge, mass number, kinetic energy, LET in water and position. It has no member functions; it has only data members that identify the characteristics of the particle. Note that all data members are defined `public`.

Definition at line 17 of file Particle.h.

7.11.2 Member Data Documentation

7.11.2.1 int Survival::Particle::A

The mass number of the particle.

Definition at line 34 of file Particle.h.

7.11.2.2 int Survival::Particle::charge

The charge of the particle expressed in elementary charge units.

Definition at line 28 of file Particle.h.

7.11.2.3 double Survival::Particle::e_c

The kinetic energy of the particle expressed in MeV.

Definition at line 37 of file Particle.h.

7.11.2.4 double Survival::Particle::let

The LET in water of the particle expressed in MeV/um (according to the Bethe-Bloch formula).

Definition at line 40 of file Particle.h.

7.11.2.5 double Survival::Particle::restEnergy

The rest energy of the particle expressed in MeV.

Definition at line 31 of file Particle.h.

7.11.2.6 std::string Survival::Particle::type

The type of particle (e.g. Chemical symbol for ions: H, He, Li, ...).

This data member identify the particle type. Note that, if the particle is an ion, only ions with atomic number $Z <= 10$ are supported.

Definition at line 25 of file Particle.h.

7.11.2.7 double Survival::Particle::weight

The weight of this particular particle type in the beam. Useful in the case of "mixed fields".

Definition at line 52 of file Particle.h.

7.11.2.8 double Survival::Particle::x

The particle position (x coordinate) referred to the beam axis, expressed in mm.

Definition at line 43 of file Particle.h.

7.11.2.9 double Survival::Particle::y

The particle position (y coordinate) referred to the beam axis, expressed in mm.

Definition at line 46 of file Particle.h.

7.11.2.10 double Survival::Particle::z

The particle position (z coordinate) referred to the depth of penetration in matter.

Definition at line 49 of file Particle.h.

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

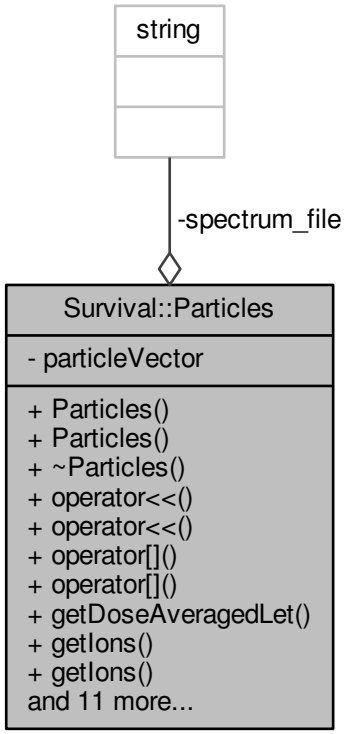
- include/Particle.h

7.12 Survival::Particles Class Reference

This is a container class, used to group together Particle objects. It implements the structure and methods to manage a vector of particles.

`#include <Particles.h>`

Collaboration diagram for Survival::Particles:



Public Member Functions

- Particles (const int numberOfParticles=0)

  Constructor. Instantiates and sets the object.
- Particles (const std::string file_name)

  Overload of the constructor. Instantiates and sets the object by loading a file where a spectrum of particles is defined.
- ~Particles ()

  Destructor.
- void operator<< (const Particle &particle)

  Overload of the << operator to add a new particle at the end of the vector.
- void operator<< (const Particles &particles)

  Overload of the << operator to add a vector of particles at the end of the vector.
- Particle & operator[ ] (const int index)

  Overload of the [] operator to access at the n-th element of the vector.

- const Particle & operator[ ] (const int index) const

  Overload of the [] operator to access at the n-th element of the vector.
- double getDoseAveragedLet () const

  Returns the dose averaged LET of the vector of particles.
- Particles getIons ()

  Selects and returns only the ions identified in the vector of particles.
- Particles * getIons (const int charge)

  Selects and returns only the ions with a particular charge identified in the vector of particles.
- Particles * getIons (const int charge, const int A)

  Selects and returns only the ions with a particular charge and mass number identified in the vector of particles.
- double getMeanLet () const

  Returns the mean LET of the vector of particles, expressed in keV/um.
- std::string getSpectrumFile () const

  Returns a `string` identifying the file containing the spectrum of particles.
- double getTotalLet () const

  Returns the total LET of the vector of particles.
- double getTotalWeight () const

  Returns the total weight of the vector of particles.
- Particles * getWithCoordinatesBetween (const double x_min, const double x_max, const double y_min, const double y_max)

  Selects and returns only the particles with coordinates between $[x_{min}, x_{max}]$ *and* $[y_{min}, y_{max}]$.
- Particles * getWithDistanceBetween (const double distance_min, const double distance_max)

  Selects and returns only the particles with a distance from the origin between $[distance_{min}, distance_{max}]$.
- void loadSpectrum (const std::string file_name)

  Loads a spectrum from a file and stores it in particleVector.
- void reconstructIonLETandEnergy ()

  For each particle of particle of particle vector, if its LET or energy is undefined the method tries to set it using the Bethe-Bloch formula, assuming it's an ion.
- void setSpectrumFile (const std::string file_name)

  Sets the name of the file containing the spectrum of particles.
- int size () const

  Returns the size of the vector of particles.

Private Attributes

- std::vector< Particle > particleVector

  The vector of particles.
- std::string spectrum_file

  A `string` identifying the file containing the spectrum of particles.

7.12.1 Detailed Description

This is a container class, used to group together Particle objects. It implements the structure and methods to manage a vector of particles.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2008

This is a container class, used to group together Particle objects. It implements the structure and methods to manage a vector of particles, which is the only data member of the class. It provides also functionalities to select particles belonging to a specific region of space or corresponding to a certain category (e.g. Ions).

See also

Particle and Tracks

Definition at line 22 of file Particles.h.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Particles::Particles (const int *numberOfParticles* = 0)

Constructor. Instantiates and sets the object.

Parameters

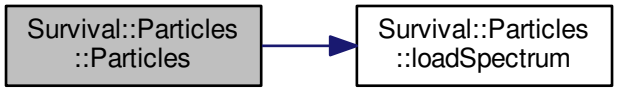
numberOfParticles	The length of the vector or, likewise, the number of particles to be stored in the object.

See also

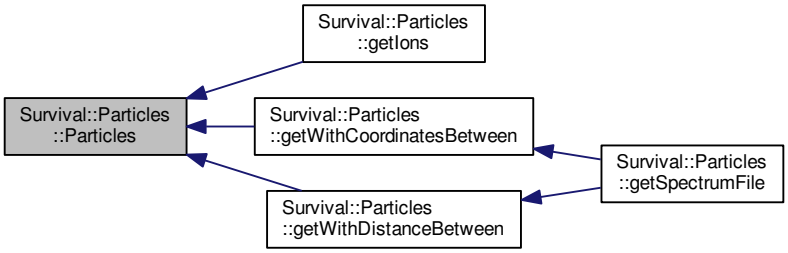
Tracks

Definition at line 28 of file Particles.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.2.2 Survival::Particles::Particles (const std::string *file_name*)

Overload of the constructor. Instantiates and sets the object by loading a file where a spectrum of particles is defined.

The method reads the file "file_name" and, row by row, instantiates, sets and stores the particles in particleVector.

Parameters

file_name	A `string` identifying the file containing the spectrum of particles.

Warning

Almost no check will be done on the file.

See also

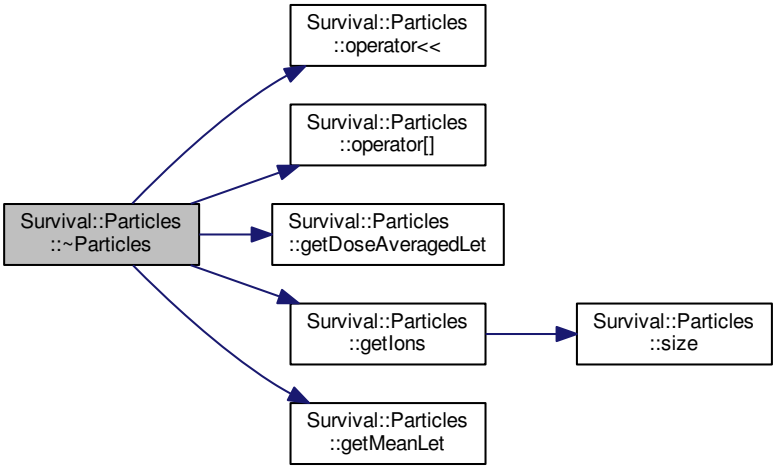
loadSpectrum()

7.12.2.3 Survival::Particles::~Particles ( ) [inline]

Destructor.

Definition at line 47 of file Particles.h.

Here is the call graph for this function:



7.12.3 Member Function Documentation

7.12.3.1 double Particles::getDoseAveragedLet ( ) const

Returns the dose averaged LET of the vector of particles.

Evaluates and returns the dose averaged LET of the group of particles stored in the vector starting from the single LET and weight of each particle by means of the following relation:

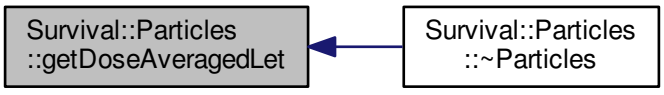
$$LET_d = \frac{\sum w_i \cdot LET_i^2}{\sum w_i \cdot LET_i}$$

Returns

The dose averaged LET.

Definition at line 73 of file Particles.cpp.

Here is the caller graph for this function:



7.12.3.2 Particles Particles::getIons ( )

Selects and returns only the ions identified in the vector of particles.

The method check if both the charge and the mass number are greater than 0.

Returns

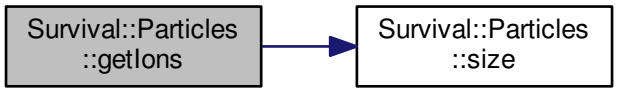
A Particles object which is the subset of all ions stored in the original Particles object.

See also

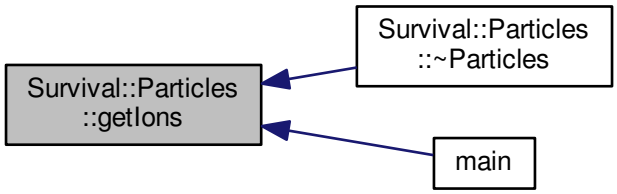
getIons(const int) and getIons(const int, const int)

Definition at line 112 of file Particles.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.3 Particles * Particles::getIons (const int *charge*)

Selects and returns only the ions with a particular charge identified in the vector of particles.

Overload of the `getIons()` function to provide the possibility of selecting a particular charge value.

Parameters

charge	The charge of the ions to be selected.

Returns

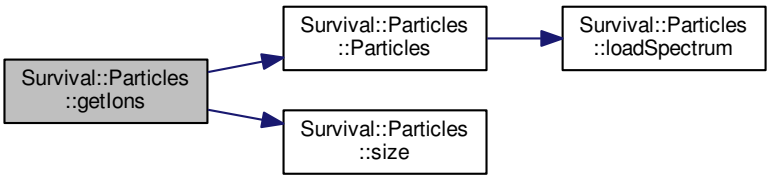
A pointer to an object of the class Particles which is the subset of all ions, with a particular charge value, stored in the original Particles object.

See also

getIons() and getIons(const int, const int)

Definition at line 130 of file Particles.cpp.

Here is the call graph for this function:



7.12.3.4 Particles * Particles::getIons (const int *charge*, const int *A*)

Selects and returns only the ions with a particular charge and mass number identified in the vector of particles.

Overload of the `getIons()` function to provide the possibility of selecting particular value of charge and mass.

Parameters

charge	The charge of the ions to be selected.
A	The mass number of the ions to be selected.

Returns

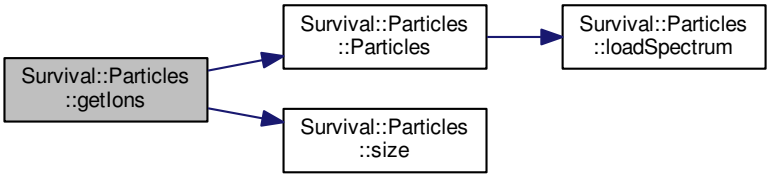
A pointer to an object of the class Particles which is the subset of all ions, with a particular value of charge and mass, stored in the original Particles object.

See also

getIons() and getIons(const int, const int)

Definition at line 148 of file Particles.cpp.

Here is the call graph for this function:



7.12.3.5 double Particles::getMeanLet ( ) const

Returns the mean LET of the vector of particles, expressed in keV/um.

Evaluates and returns the mean LET of the group of particles stored in the vector starting from the single LET and weight of each particle by means of the following relation:

$$\langle LET \rangle = \frac{\sum w_i \cdot LET_i}{\sum w_i}$$

Returns

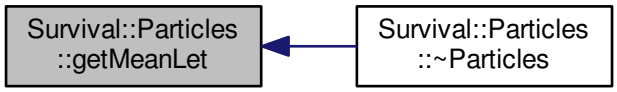
The mean LET, expressed in keV/um.

See also

Particle::let

Definition at line 197 of file Particles.cpp.

Here is the caller graph for this function:



7.12.3.6 std::string Survival::Particles::getSpectrumFile ( ) const [inline]

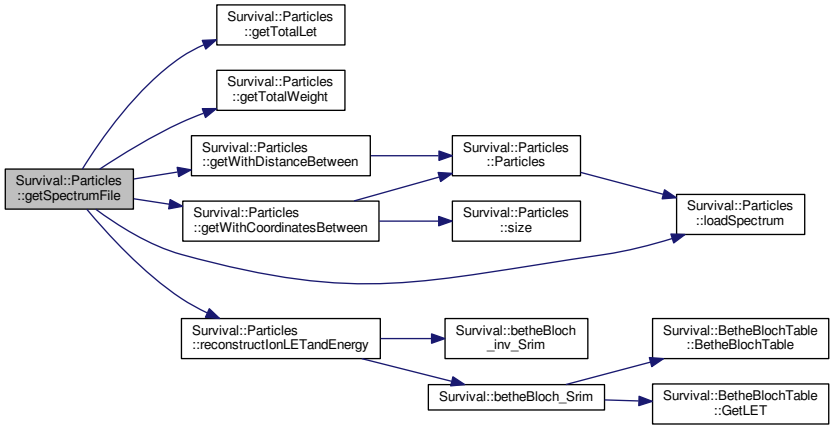
Returns a `string` identifying the file containing the spectrum of particles.

Returns

A `string` identifying the file containing the spectrum of particles (spectrum_file).

Definition at line 144 of file Particles.h.

Here is the call graph for this function:



7.12.3.7 double Particles::getTotalLet ( ) const

Returns the total LET of the vector of particles.

Returns

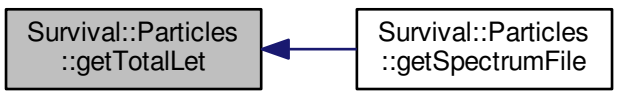
The total LET, evaluated as the sum of the LET of each particle.

See also

Particle::let

Definition at line 245 of file Particles.cpp.

Here is the caller graph for this function:



7.12.3.8 double Particles::getTotalWeight ( ) const

Returns the total weight of the vector of particles.

Returns

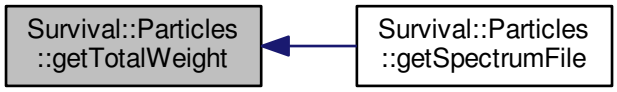
The total weight, evaluated as the sum of the weight of each particle.

See also

Particle::weight

Definition at line 257 of file Particles.cpp.

Here is the caller graph for this function:



7.12.3.9 Particles * Particles::getWithCoordinatesBetween (const double *x_min*, const double *x_max*, const double *y_min*, const double *y_max*)

Selects and returns only the particles with coordinates between $[x_{min},\ x_{max}]$ and $[y_{min},\ y_{max}]$.

Parameters

x_min	The minimum value of the x coordinate, expressed in mm.
x_max	The maximum value of the y coordinate, expressed in mm.
y_min	The minimum value of the x coordinate, expressed in mm.
y_max	The maximum value of the y coordinate, expressed in mm.

Returns

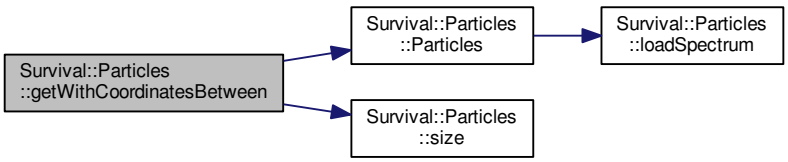
A pointer to an object of the class Particles which is a subset of the original Particles object.

See also

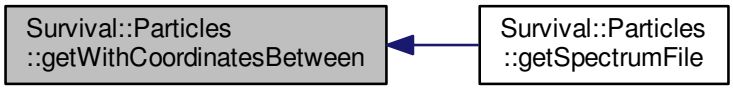
getIons() and getWithDistanceBetween()

Definition at line 268 of file Particles.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.10 Particles * Particles::getWithDistanceBetween (const double *distance_min*, const double *distance_max*)

Selects and returns only the particles with a distance from the origin between $[distance_{min}, distance_{max}]$.

Selects all particles in an annulus between $[distance_{min}, distance_{max}]$.

Parameters

distance_min	The minimum distance from the origin, expressed in mm.
distance_max	The maximum distance from the origin, expressed in mm.

Returns

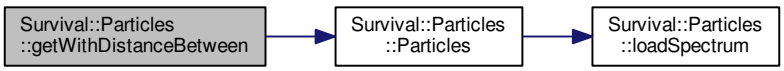
A pointer to an object of the class Particles which is a subset of the original Particles object.

See also

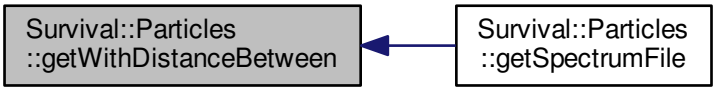
getIons() and getWithCoordinatesBetween()

Definition at line 291 of file Particles.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.11 void Particles::loadSpectrum (const std::string *file_name*)

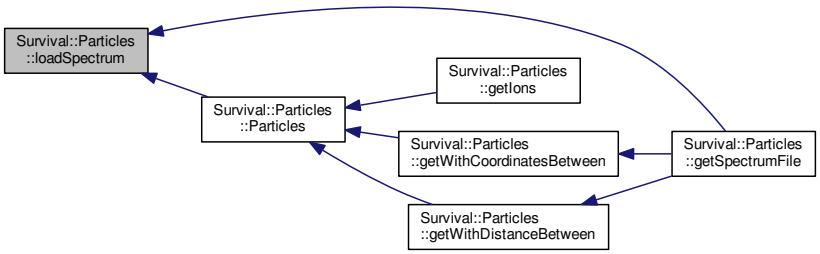
Loads a spectrum from a file and stores it in particleVector.

Parameters

file_name	A `string` identifying the file containing the spectrum of particles.

Definition at line 311 of file Particles.cpp.

Here is the caller graph for this function:



7.12.3.12 void Particles::operator<< (const Particle & *particle*)

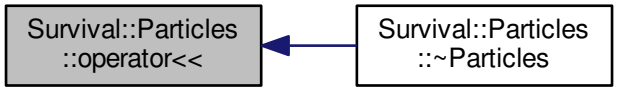
Overload of the << operator to add a new particle at the end of the vector.

Parameters

particle	The particle to be added.

Definition at line 44 of file Particles.cpp.

Here is the caller graph for this function:



7.12.3.13 void Particles::operator<< (const Particles & *particles*)

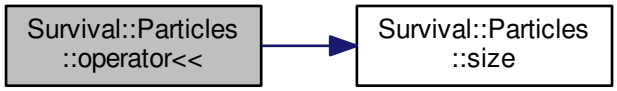
Overload of the << operator to add a vector of particles at the end of the vector.

Parameters

particles	The object containing the vector of particles to be added.

Definition at line 51 of file Particles.cpp.

Here is the call graph for this function:



7.12.3.14 Particle & Particles::operator[ ] (const int *index*)

Overload of the [] operator to access at the n-th element of the vector.

Parameters

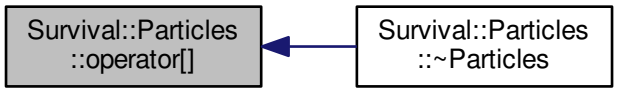
index	The position of the element in the vector.

Returns

A reference to the element at the specified position in the vector.

Definition at line 59 of file Particles.cpp.

Here is the caller graph for this function:



7.12.3.15 const Particle & Particles::operator[ ] (const int *index*) const

Overload of the [] operator to access at the n-th element of the vector.

Parameters

index	The position of the element in the vector.

Returns

A `const` reference to the element at the specified position in the vector.

Definition at line 66 of file Particles.cpp.

7.12.3.16 void Particles::reconstructIonLETandEnergy ( )

For each particle of particle of particle vector, if its LET or energy is undefined the method tries to set it using the Bethe-Bloch formula, assuming it's an ion.

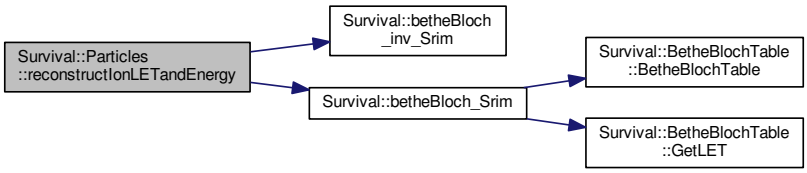
The actual LET of each particle is assumed to be expressed in keV/um, while the kinetic energy in MeV.

Warning

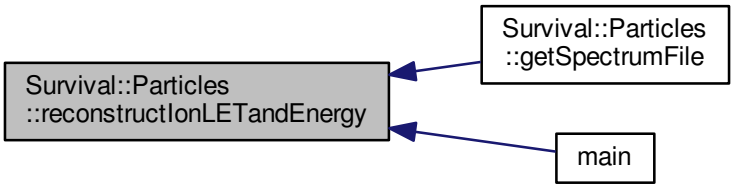
This function is NOT general and thought only for a precise particular purpose. Execution of the program will be terminated if an element of the vector isn't an ion, and there aren't any checks on other particle features.

Definition at line 375 of file Particles.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.17 void Survival::Particles::setSpectrumFile (const std::string *file_name*) [inline]

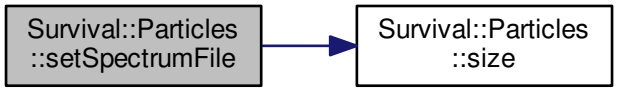
Sets the name of the file containing the spectrum of particles.

Parameters

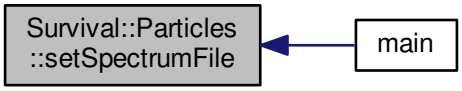
file_name	A `string` identifying the file containing the spectrum of particles.

Definition at line 210 of file Particles.h.

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.18 int Particles::size ( ) const

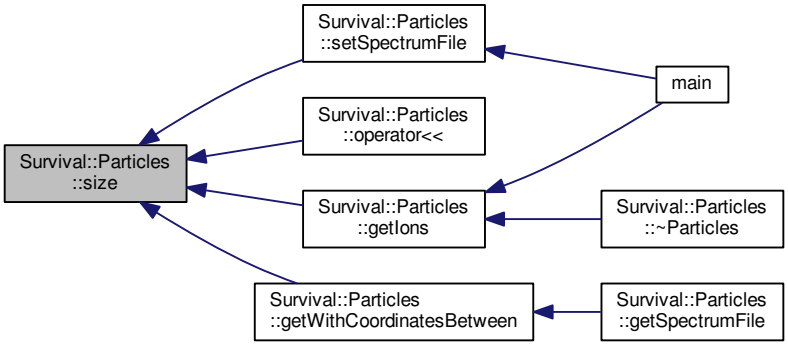
Returns the size of the vector of particles.

Returns

The size of the vector of particles: particleVector

Definition at line 399 of file Particles.cpp.

Here is the caller graph for this function:



7.12.4 Member Data Documentation

7.12.4.1 std::vector< Particle > Survival::Particles::particleVector [private]

The vector of particles.

See also

Particle

Definition at line 224 of file Particles.h.

7.12.4.2 std::string Survival::Particles::spectrum_file [private]

A `string` identifying the file containing the spectrum of particles.

Definition at line 227 of file Particles.h.

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

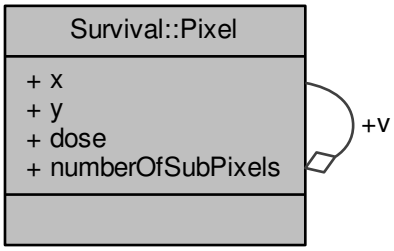
- include/Particles.h
- src/Particles.cpp

7.13 Survival::Pixel Class Reference

Implements the Pixel features to be used in the Nucleus_Pixel class.

`#include <Nucleus_Pixel.h>`

Collaboration diagram for Survival::Pixel:



Public Attributes

- double x

  The position of the pixel (x coordinate) referred to the nucleus center, expressed in um.
- double y

  The position of the pixel (y coordinate) referred to the nucleus center, expressed in um.
- double dose

  The local dose deposited in the pixel, expressed in Gy.
- int numberOfSubPixels

  The number of pixel constituting the first inner grid.
- Pixel * v

  A pointer to the sub-pixels in which the pixel itself is divided (i.e. The pixels of the first subgrid).

7.13.1 Detailed Description

Implements the Pixel features to be used in the Nucleus_Pixel class.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2007

The object has simply a few public data members, like the pixel position or the dose deposited. The most important data members are v and numberOfSubPixels that identify the real structure. The Nucleus_Pixel is divided into four grid of pixels of decreasing dimension (this is useful to sample the track interacting with the nucleus with a higher frequency only when needed, that is near the position of ion transversals, where the local dose is rapidly varying). Hence a "tree structure" is created and the data member v is used to dynamically allocate the memory necessary to contain the grid of pixel just smaller in which each pixel is divided. numberOfSubPixels stores the number of pixels pointed by v.

See also

Nucleus_Pixel::createPixels()

Definition at line 19 of file Nucleus_Pixel.h.

7.13.2 Member Data Documentation

7.13.2.1 double Survival::Pixel::dose

The local dose deposited in the pixel, expressed in Gy.

Definition at line 30 of file Nucleus_Pixel.h.

7.13.2.2 int Survival::Pixel::numberOfSubPixels

The number of pixel constituting the first inner grid.

See also

Nucleus_Pixel::createPixels()

Definition at line 36 of file Nucleus_Pixel.h.

7.13.2.3 Pixel* Survival::Pixel::v

A pointer to the sub-pixels in which the pixel itself is divided (i.e. The pixels of the first subgrid).

See also

Nucleus_Pixel::createPixels()

Definition at line 42 of file Nucleus_Pixel.h.

7.13.2.4 double Survival::Pixel::x

The position of the pixel (x coordinate) referred to the nucleus center, expressed in um.

Definition at line 24 of file Nucleus_Pixel.h.

7.13.2.5 double Survival::Pixel::y

The position of the pixel (y coordinate) referred to the nucleus center, expressed in um.

Definition at line 27 of file Nucleus_Pixel.h.

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

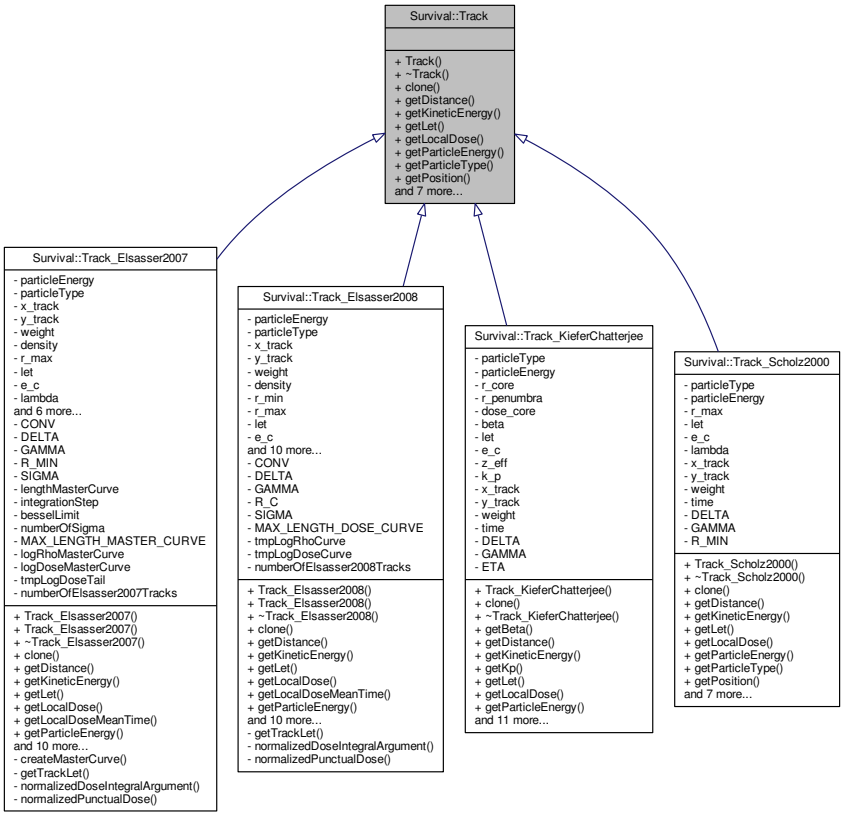
- include/Nucleus_Pixel.h

7.14 Survival::Track Class Reference

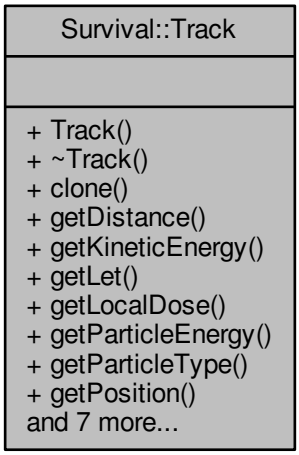
Constructed on the base of an ion Particle object, this class represents the local dose distribution around that ion.

`#include <Track.h>`

Inheritance diagram for Survival::Track:



Collaboration diagram for Survival::Track:



Public Member Functions

- Track ()

  Constructor of a pure virtual class (empty).
- virtual ~Track ()

  Destructor of a pure virtual class (empty).
- virtual Track * clone () const =0

  Declaration of the pure virtual function clone (for a more detailed description see the derived classes).
- virtual double getDistance (const double localDose) const =0

  Declaration of the pure virtual function getDistance (for a more detailed description see the derived classes).
- virtual double getKineticEnergy () const =0

  Declaration of the pure virtual function getKineticEnergy (for a more detailed description see the derived classes).
- virtual double getLet () const =0

  Declaration of the pure virtual function getLet (for a more detailed description see the derived classes).
- virtual double getLocalDose (const double distance) const =0

  Declaration of the pure virtual function getLocalDose (for a more detailed description see the derived classes).
- virtual double getParticleEnergy () const =0

  Declaration of the pure virtual function getParticleEnergy (for a more detailed description see the derived classes).
- virtual std::string getParticleType () const =0

  Declaration of the pure virtual function getParticleType (for a more detailed description see the derived classes).
- virtual void getPosition (double &returnX, double &returnY) const =0

  Declaration of the pure virtual function getPosition (for a more detailed description see the derived classes).
- virtual double getRadialIntegral (const double r_min, const double r_max) const =0

  Declaration of the pure virtual function getRadialIntegral (for a more detailed description see the derived classes).
- virtual double getRadius () const =0

  Declaration of the pure virtual function getRadius (for a more detailed description see the derived classes).
- virtual double getTime () const =0

  Declaration of the pure virtual function getTime (for a more detailed description see the derived classes).
- virtual double getWeight () const =0

  Declaration of the pure virtual function getWeight (for a more detailed description see the derived classes).
- virtual std::string saveTrack () const =0

  Declaration of the pure virtual function saveTrack (for a more detailed description see the derived classes).
- virtual void setPosition (const double x, const double y)=0

  Declaration of the pure virtual function setPosition (for a more detailed description see the derived classes).
- virtual void setTime (double t)=0

  Declaration of the pure virtual function setTime (for a more detailed description see the derived classes).

7.14.1 Detailed Description

Constructed on the base of an ion Particle object, this class represents the local dose distribution around that ion.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2007–2015

Constructed on the base of an ion Particle object, this class represents the local dose distribution around that ion. It is a pure virtual class, defined by the inherited Track_Scholz2000, Track_Elsasser2007, Track_Elsasser2008 and Track_KieferChatterjee classes, which implement the track models of LEM I, II, III and MKM respectively.

Definition at line 17 of file Track.h.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 Survival::Track::Track ( ) [inline]

Constructor of a pure virtual class (empty).

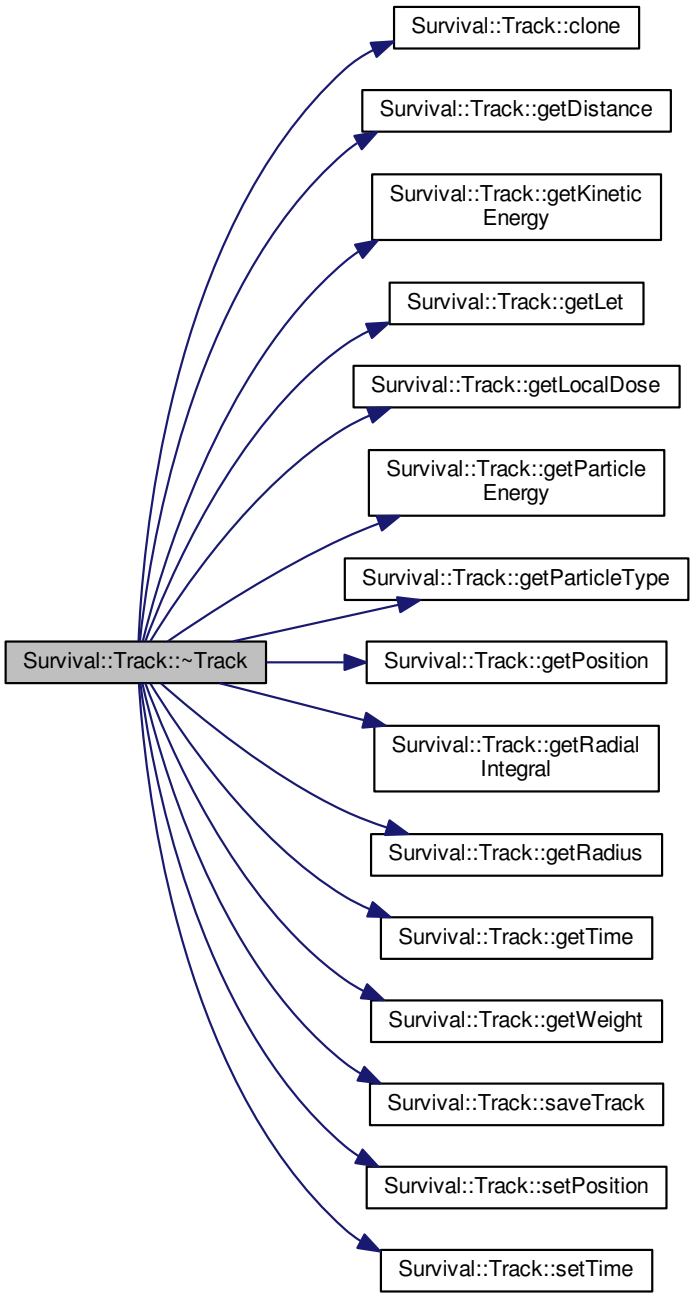
Definition at line 22 of file Track.h.

7.14.2.2 virtual Survival::Track::~Track ( ) [inline], [virtual]

Destructor of a pure virtual class (empty).

Definition at line 25 of file Track.h.

Here is the call graph for this function:



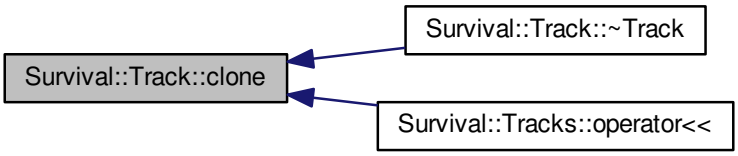
7.14.3 Member Function Documentation

7.14.3.1 virtual Track* Survival::Track::clone ( ) const [pure virtual]

Declaration of the pure virtual function clone (for a more detailed description see the derived classes).

Implemented in Survival::Track_Scholz2000, Survival::Track_Elsasser2008, Survival::Track_Elsasser2007, and Survival::Track_KieferChatterjee.

Here is the caller graph for this function:

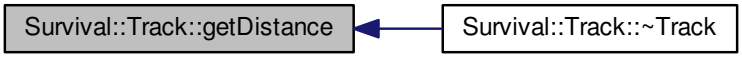


7.14.3.2 virtual double Survival::Track::getDistance (const double *localDose*) const [pure virtual]

Declaration of the pure virtual function getDistance (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Scholz2000, Survival::Track_Elsasser2008, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

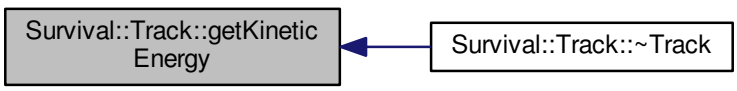


7.14.3.3 virtual double Survival::Track::getKineticEnergy ( ) const [pure virtual]

Declaration of the pure virtual function getKineticEnergy (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Scholz2000, Survival::Track_Elsasser2008, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

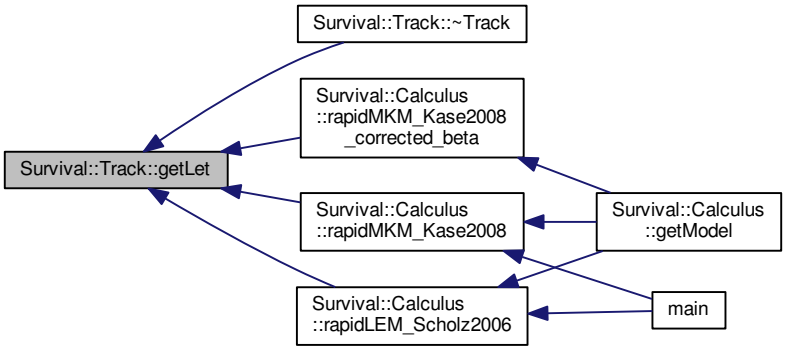


7.14.3.4 virtual double Survival::Track::getLet ( ) const [pure virtual]

Declaration of the pure virtual function getLet (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Scholz2000, Survival::Track_Elsasser2008, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

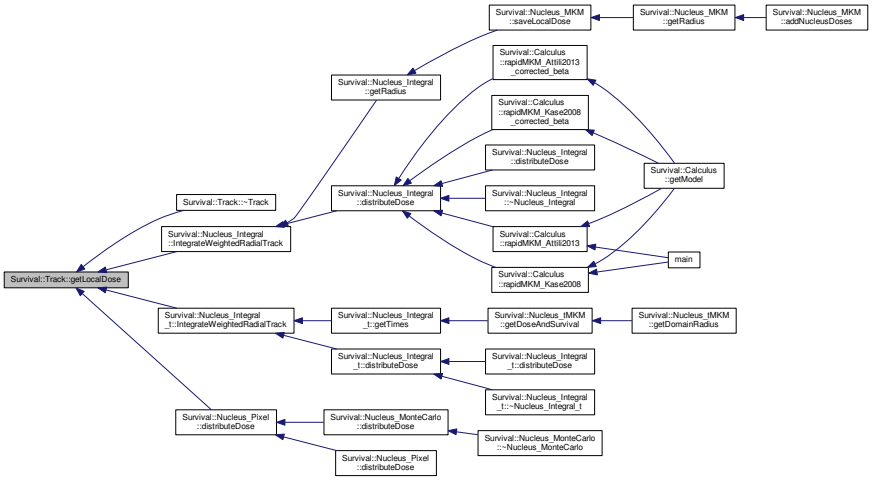


7.14.3.5 virtual double Survival::Track::getLocalDose (const double *distance*) const [pure virtual]

Declaration of the pure virtual function getLocalDose (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Scholz2000, Survival::Track_Elsasser2008, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

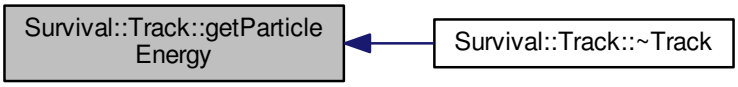


7.14.3.6 virtual double Survival::Track::getParticleEnergy ( ) const [pure virtual]

Declaration of the pure virtual function getParticleEnergy (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Elsasser2008, Survival::Track_Scholz2000, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

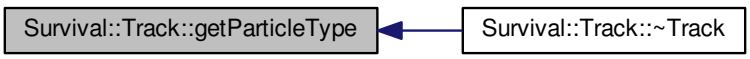


7.14.3.7 virtual std::string Survival::Track::getParticleType ( ) const [pure virtual]

Declaration of the pure virtual function getParticleType (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Elsasser2008, Survival::Track_Scholz2000, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

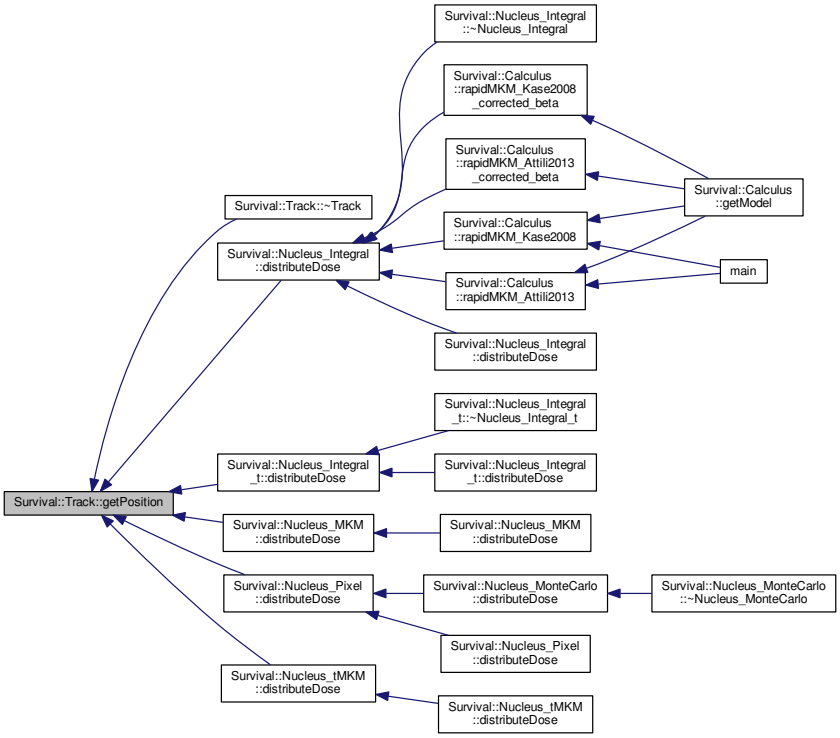


7.14.3.8 virtual void Survival::Track::getPosition (double & *returnX*, double & *returnY*) const [pure virtual]

Declaration of the pure virtual function getPosition (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Elsasser2008, Survival::Track_Scholz2000, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

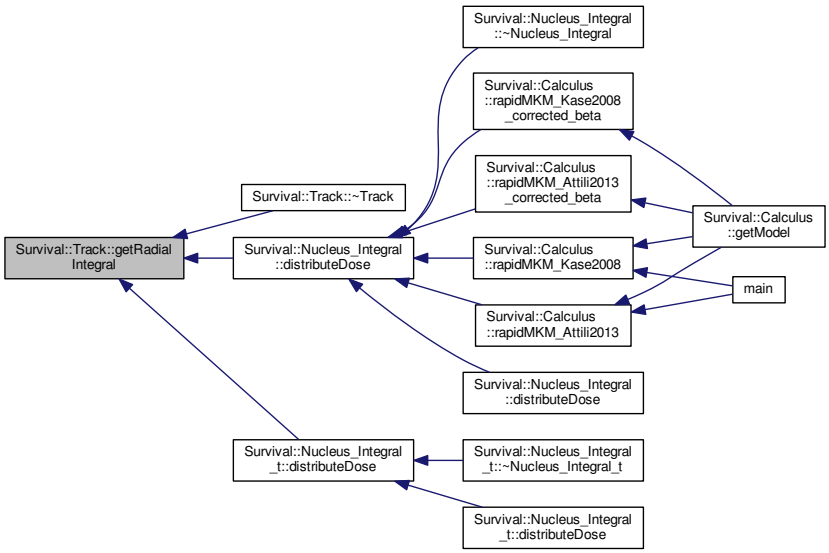


7.14.3.9 virtual double Survival::Track::getRadialIntegral (const double *r_min*, const double *r_max*) const [pure virtual]

Declaration of the pure virtual function getRadialIntegral (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Scholz2000, Survival::Track_Elsasser2008, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

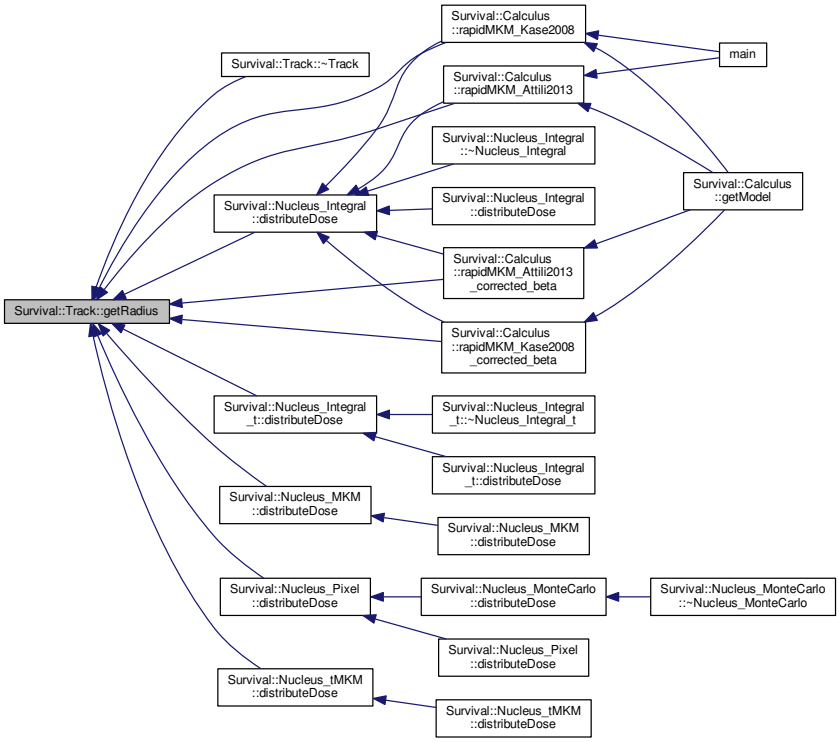


7.14.3.10 virtual double Survival::Track::getRadius ( ) const [pure virtual]

Declaration of the pure virtual function getRadius (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Scholz2000, Survival::Track_Elsasser2008, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

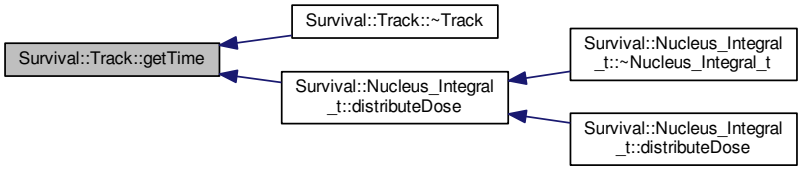


7.14.3.11 virtual double Survival::Track::getTime ( ) const [pure virtual]

Declaration of the pure virtual function getTime (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Elsasser2008, Survival::Track_Scholz2000, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

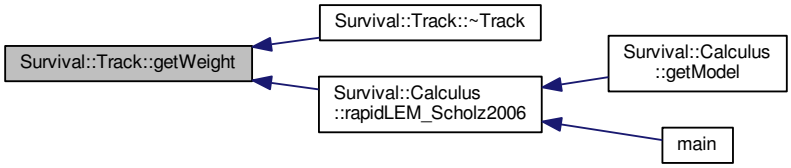


7.14.3.12 virtual double Survival::Track::getWeight ( ) const [pure virtual]

Declaration of the pure virtual function getWeight (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Elsasser2008, Survival::Track_Scholz2000, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

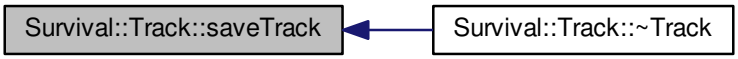


7.14.3.13 virtual std::string Survival::Track::saveTrack ( ) const [pure virtual]

Declaration of the pure virtual function saveTrack (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Elsasser2008, Survival::Track_Scholz2000, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

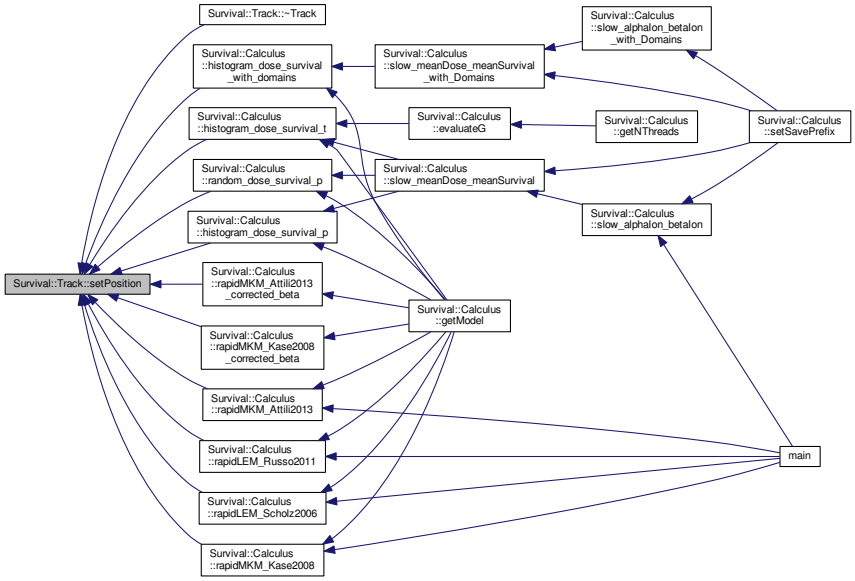


7.14.3.14 virtual void Survival::Track::setPosition (const double *x*, const double *y*) [pure virtual]

Declaration of the pure virtual function setPosition (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Elsasser2008, Survival::Track_Scholz2000, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:

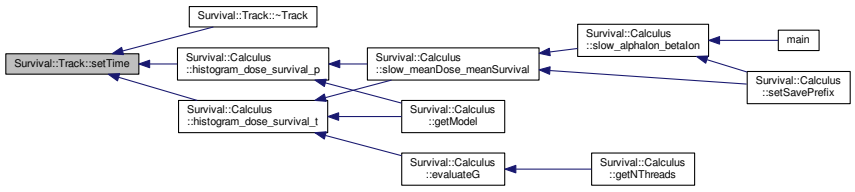


7.14.3.15 virtual void Survival::Track::setTime (double *t*) [pure virtual]

Declaration of the pure virtual function setTime (for a more detailed description see the derived classes).

Implemented in Survival::Track_KieferChatterjee, Survival::Track_Elsasser2008, Survival::Track_Scholz2000, and Survival::Track_Elsasser2007.

Here is the caller graph for this function:



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

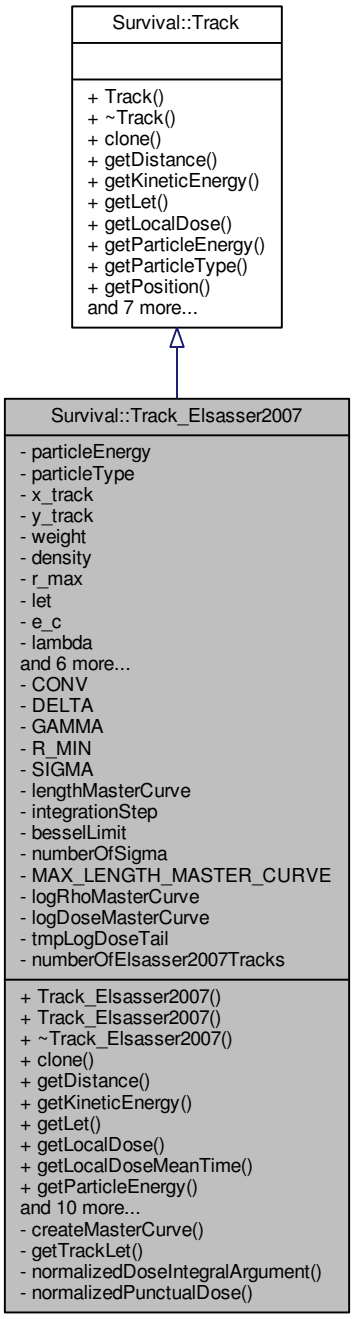
- include/Track.h

7.15 Survival::Track_Elsasser2007 Class Reference

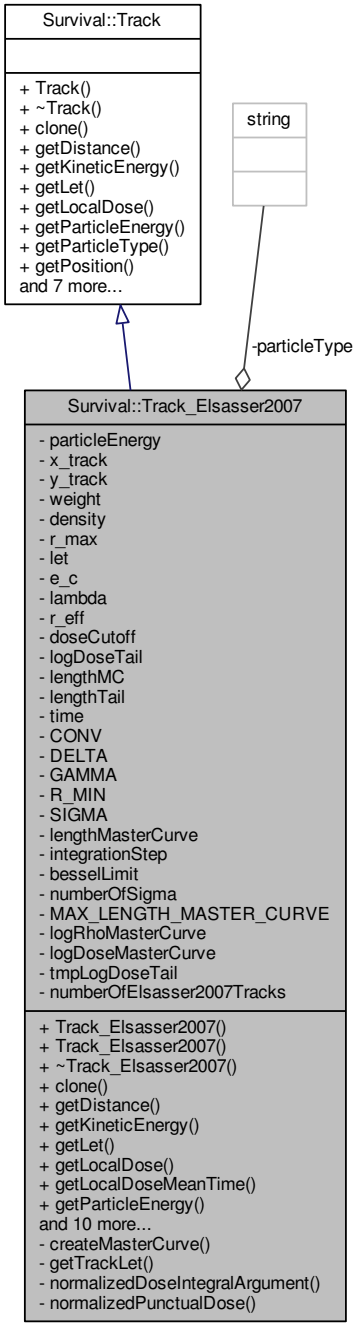
Inherited from the Track class, it implements the LEM II track model.

`#include <Track_Elsasser2007.h>`

Inheritance diagram for Survival::Track_Elsasser2007:



Collaboration diagram for Survival::Track_Elsasser2007:



Public Member Functions

- Track_Elsasser2007 (const Particle &particle, const double density, const double doseCutoff=1e-8, const int lengthMasterCurve=300.0, const double integrationStepFactor=1e-2, const double besselLimit=400.0, const double numberOfSigma=20.0, double t=0.0)

  Constructor. Instantiates and sets the object.
- Track_Elsasser2007 (const Track_Elsasser2007 &track)

Copy constructor. Instantiates a new Track_Elsasser2007 object copying an existent one, including the precalculated master curve.

- virtual ~Track_Elsasser2007 ()

  Destructor.
- virtual Track_Elsasser2007 * clone () const

  Returns a pointer to a new Track_Elsasser2007 object created as a copy of an existent one by means of the copy constructor.
- virtual double getDistance (const double localDose) const

  Returns the distance from the center of the track, knowing the local dose deposited.
- virtual double getKineticEnergy () const

  Returns the kinetic energy of the particle generating the track expressed in MeV.
- virtual double getLet () const

  Returns the LET in water of the particle generating the track expressed in MeV/um.
- virtual double getLocalDose (const double distance) const

  Returns the local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).
- double getLocalDoseMeanTime ()

  Function created to calculate the mean time required to the evaluation the getLocalDose() method.
- virtual double getParticleEnergy () const

  Returns the specific energy of the particle generating the track, expressed in MeV/u.
- virtual std::string getParticleType () const

  Returns the type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).
- virtual void getPosition (double &returnX, double &returnY) const

  Returns the track position (x and y coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.
- virtual double getRadialIntegral (const double r_min, const double r_max) const

  Evaluates the radial integral of the track profile in $[r_{min}, r_{max}]$.
- virtual double getRadius () const

  Returns the effective radius of the track, expressed in um.
- double getRelativePrecision () const

  Evaluates the relative precision of the calculated LET with respect of the original particle LET.
- virtual double getTime () const

  Returns the time associated to a particular event expressed in hours.
- virtual double getWeight () const

  Returns the weight in the beam of the particle generating the track. Useful in the case of "mixed fields".
- virtual std::string saveTrack () const

  Saves the local dose deposited by the track at different distances from its center to reconstruct the track profile.
- virtual void setPosition (const double x, const double y)

  Sets the track position (x and y coordinates) referred to the beam axis.
- virtual void setTime (double t)

  Sets the time associated to a particular event.

Private Member Functions

- void createMasterCurve (const int lengthMasterCurve, const double integrationStepFactor, const double besselLimit, const double numberOfSigma)

  Creates the master curve corresponding to the common track profile.
- double getTrackLet () const

  Evaluates and returns the LET of the track.
- double normalizedDoseIntegralArgument (const double r, const double r1) const

  Evaluates the argument of the normalized integral in the creation of the master curve.
- double normalizedPunctualDose (const double distance) const

  Evaluates the local dose along the radial profile of the master curve.

Private Attributes

- double particleEnergy

  The specific energy of the particle generating the track, expressed in MeV/u.
- std::string particleType

  The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...)
- double x_track

  The track position (x coordinate) referred to the beam axis, expressed in mm.
- double y_track

  The track position (y coordinate) referred to the beam axis, expressed in mm.
- double weight

  The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".
- double density

  The density of the medium expressed in $\frac{g}{cm^3}$.
- double r_max

  The radius of the track expressed in um.
- double let

  The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).
- double e_c

  The kinetic energy of the particle generating the track expressed in MeV.
- double lambda

  A constant value required to evaluate r_max.
- double r_eff

  The effective radius of the track (expressed in um) corresponding to the point where the local dose results equal to doseCutoff.
- double doseCutoff

  Minimum possible dose deposited evaluable, expressed in Gy.
- double * logDoseTail

  A pointer to the values of the calculated (logarithmic) local dose in the tail of the track.
- int lengthMC

  Length of the used master curve.
- int lengthTail

  Length of the created tail.
- double time

  The time associated to a particular event, expressed in hours.

Static Private Attributes

- static const double CONV = 160.2177

  Constants static variables and precalculated feature indices.
- static const double DELTA = 1.7

  Useful constant value.
- static const double GAMMA = 0.062

  Useful constant value.
- static const double R_MIN = 3e-4

  The core radius expressed in um.
- static const double SIGMA = 4e-3

  The radical diffusion length, expressed in um.
- static int lengthMasterCurve

  The length of the master curve expressed in um.

- static double integrationStep

  The integration step for the generation of the master curve expressed in um. It's proportional to R_MIN.
- static double besselLimit

  Limit between the calculation of Bessel function by means of series development and the calculation with the asymptotic exponential approximation.
- static double numberOfSigma

  Half width of non-zero window in units of sigma.
- static const int MAX_LENGTH_MASTER_CURVE = 1000

  Maximum length of the master curve (i.e. maximum number of steps, equal to the length of the arrays logRho↩ MasterCurve and logDoseMasterCurve).
- static double logRhoMasterCurve [MAX_LENGTH_MASTER_CURVE]

  Array to store the progressive (logarithmic) radii corresponding to the profile of the master curve.
- static double logDoseMasterCurve [MAX_LENGTH_MASTER_CURVE]

  Array to store the calculated values of (logarithmic) local dose deposited, constituting the radial profile of the master curve.
- static double tmpLogDoseTail [MAX_LENGTH_MASTER_CURVE]

  Temporary storage of the local dose in the track tail.
- static int numberOfElsasser2007Tracks = 0

  The number of existing Track_Elsasser2007 objects.

7.15.1 Detailed Description

Inherited from the Track class, it implements the LEM II track model.

Author

Andrea Attili
Giuseppe Falvo D'Urso Labate
Lorenzo Manganaro
Germano Russo

Date

2008

With respect to the LEM I, the LEM II (1) track model is extended to explicitly include the effect of radical diffusion: the previous parametric representation is still used (see Track_Scholz2000) but changing the core radius R_↩ MIN value from 10 nm to 0.3 nm (value more in agreement with experimental data) in order to represent the instantaneous average ionization pattern which occurs some nanoseconds after the passage of the ion; this pattern is then convoluted with a gaussian kernel of 4 nm sigma that models the spreading of the induced radical species taking place at longer time scales (a few microseconds). Since the computation of the convolution with the gaussian radical diffusion profile is quite time-consuming, one should consider that, except for the outer part of the track, all convoluted track profiles are equal, apart from a normalizing factor depending on the particle LET; hence it is possible to preconvolute once for all this common track profile, called *master curve*, shared by all Track_↩ Elsasser2007 instances as a static data member. This class gives also the capability of truncating the track profile providing a dose cut-off.

1. T. Elsässer and M. Scholz, "Cluster effects within the local effect model", *Radiation Research* **167**, 319-329 (2007).

Definition at line 22 of file Track_Elsasser2007.h.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 Track_Elsasser2007::Track_Elsasser2007 (const Particle & *particle*, const double *density*, const double *doseCutoff* = 1e-8, const int *lengthMasterCurve* = 300.0, const double *integrationStepFactor* = 1e-2, const double *besselLimit* = 400.0, const double *numberOfSigma* = 20.0, double *t* = 0.0)

Constructor. Instantiates and sets the object.

Converts a particle (object of class Particle), passed by reference, in a track according to the LEM II amorphous track model. Some of the data members are instantiated on the basis of the informations stored in the Particle object. (For a more detailed description of the instantiation of each member respectively look at its specific documentation).

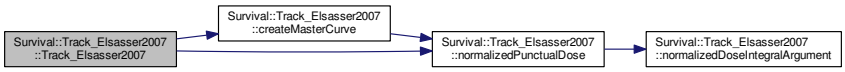
It calculates also, once for all, the master curve by calling the createMasterCurve() method and completes it generating the tail, if necessary.

Parameters

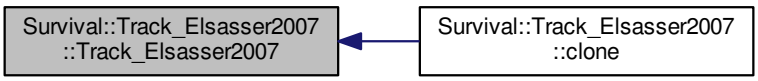
particle	The particle generating the track in the medium, passed by reference.
density	The density of the medium expressed in $\frac{g}{cm^3}$. The default value is the density of water.
doseCutoff	The minimum possible dose deposited evaluable, expressed in Gy (see doseCutoff).
lengthMasterCurve	The length of the master curve expressed in um.
integrationStepFactor	Dimensionless integration factor (multiplied by R_MIN gives the integrationStep)
besselLimit	The limit between the calculation of Bessel function by means of series development and the calculation with the asymptotic exponential approximation (see besselLimit).
numberOfSigma	Half width of non-zero window in units of sigma (see numberOfSigma).
t	The time corresponding to the generation of the track in the target. The default value is 0. (See also the documentation of the data member time).

Definition at line 55 of file Track_Elsasser2007.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.2.2 Track_Elsasser2007::Track_Elsasser2007 (const Track_Elsasser2007 & *track*)

Copy constructor. Instantiates a new Track_Elsasser2007 object copying an existent one, including the precalculated master curve.

Definition at line 154 of file Track_Elsasser2007.cpp.

7.15.2.3 Track_Elsasser2007::~Track_Elsasser2007 ( ) [virtual]

Destructor.

Delete the object, deallocating also the memory occupied by logDoseTail, and decrements the counter of Track_Elsasser2007 objects (numberOfElsasser2007Tracks).

Definition at line 166 of file Track_Elsasser2007.cpp.

7.15.3 Member Function Documentation

7.15.3.1 Track_Elsasser2007 * Track_Elsasser2007::clone ( ) const [virtual]

Returns a pointer to a new Track_Elsasser2007 object created as a copy of an existent one by means of the copy constructor.

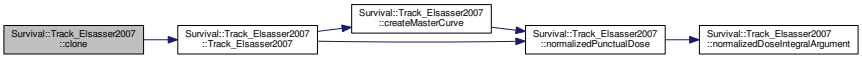
Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

Implements Survival::Track.

Definition at line 175 of file Track_Elsasser2007.cpp.

Here is the call graph for this function:



7.15.3.2 void Track_Elsasser2007::createMasterCurve (const int *lengthMasterCurve,* const double *integrationStepFactor,* const double *besselLimit,* const double *numberOfSigma*) [private]

Creates the master curve corresponding to the common track profile.

Create the master curve calling the normalizedPunctualDose() function in a for loop over the radial length of the master curve itself. It stores the calculated doses and radii in logRhoMasterCurve and logDoseMasterCurve data members.

Warning

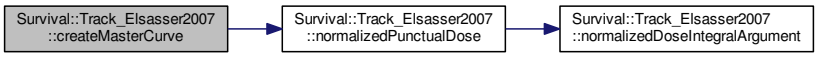
The execution of the program will be terminated if the function is called when the master curve already exist (or if the length required is greater than the maximum length imposed - MAX_LENGTH_MASTER_CURVE).

See also

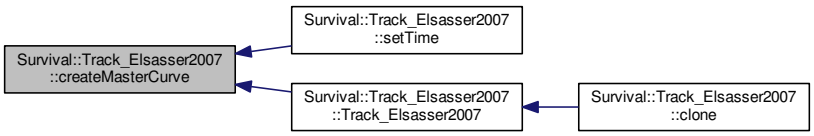
Track_Elsasser2007

Definition at line 359 of file Track_Elsasser2007.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.3.3 double Track_Elsasser2007::getDistance (const double *localDose*) const [virtual]

Returns the distance from the center of the track, knowing the local dose deposited.

It's the inverse function of getLocalDose(). The function runs in a loop over the precalculated values, starting from the master curve and continuing with the tail until it finds a value smaller than the required dose deposited, then it interpolates the nearest neighbors to get the correct value.

Note

If the dose deposited is smaller than the doseCutoff it returns r_eff.

Parameters

localDose	The local dose deposited, expressed in Gy.

Returns

The distance from the center of the track, expressed in um.

See also

getLocalDose()

Implements Survival::Track.

Definition at line 182 of file Track_Elsasser2007.cpp.

7.15.3.4 virtual double Survival::Track_Elsasser2007::getKineticEnergy ( ) const [inline], [virtual]

Returns the kinetic energy of the particle generating the track expressed in MeV.

Returns

The kinetic energy of the particle generating the track expressed in MeV.

See also

e_c

Implements Survival::Track.

Definition at line 86 of file Track_Elsasser2007.h.

7.15.3.5 virtual double Survival::Track_Elsasser2007::getLet ( ) const [inline], [virtual]

Returns the LET in water of the particle generating the track expressed in MeV/um.

Returns

The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

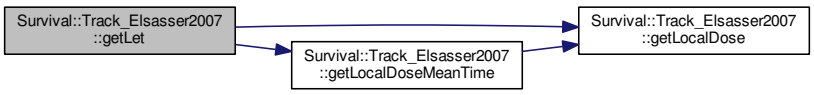
See also

let

Implements Survival::Track.

Definition at line 94 of file Track_Elsasser2007.h.

Here is the call graph for this function:



7.15.3.6 double Track_Elsasser2007::getLocalDose (const double *distance*) const [virtual]

Returns the local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

The function evaluates some possible cases:

- If the distance is smaller than the minimum radius stored in the master curve it returns the dose at the minimum radius evaluated
- If the distance is greater than the effective radius of the track (r_eff) it returns 0
- Else it returns the precalculated local dose at the required distance obtained by an interpolation of the nearest neighbors, discriminating if that distance corresponds to the tail or to the master curve.

Parameters

distance	The distance from the track center expressed in um.

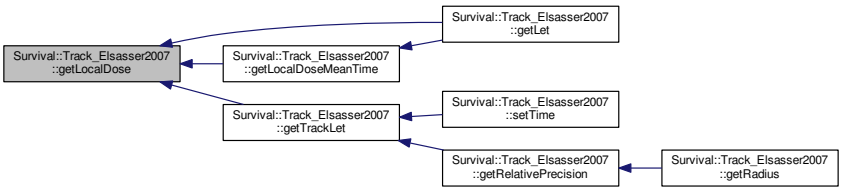
Returns

The local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

Implements Survival::Track.

Definition at line 225 of file Track_Elsasser2007.cpp.

Here is the caller graph for this function:



7.15.3.7 double Track_Elsasser2007::getLocalDoseMeanTime ( )

Function created to calculate the mean time required to the evaluation the getLocalDose() method.

It cyclically calls getLocalDose() 1000000 times, timing the total elapsed time and dividing it by 1000000.

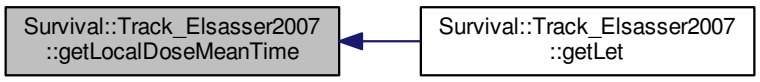
Returns

The mean time needed to a complete evaluation of the getLocalDose() method, expressed in s.

Definition at line 265 of file Track_Elsasser2007.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



7.15.3.8 virtual double Survival::Track_Elsasser2007::getParticleEnergy ( ) const [inline], [virtual]

Returns the specific energy of the particle generating the track, expressed in MeV/u.

Returns

The specific energy of the particle generating the track, expressed in MeV/u.

See also

particleEnergy

Implements Survival::Track.

Definition at line 123 of file Track_Elsasser2007.h.

7.15.3.9 virtual std::string Survival::Track_Elsasser2007::getParticleType ( ) const [inline], [virtual]

Returns the type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

Returns

The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

See also

Particle::type

Implements Survival::Track.

Definition at line 131 of file Track_Elsasser2007.h.

7.15.3.10 virtual void Survival::Track_Elsasser2007::getPosition (double & *returnX*, double & *returnY*) const [inline], [virtual]

Returns the track position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the track referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the track, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `x` coordinate of the track, expressed in mm, passed by reference to be overwritten.

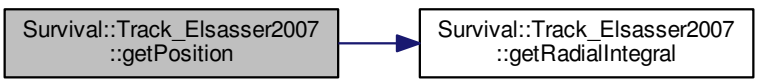
See also

setPosition()

Implements Survival::Track.

Definition at line 142 of file Track_Elsasser2007.h.

Here is the call graph for this function:



7.15.3.11 double Track_Elsasser2007::getRadialIntegral (const double *r_min*, const double *r_max*) const [virtual]

Evaluates the radial integral of the track profile in $[r_{min}, r_{max}]$.

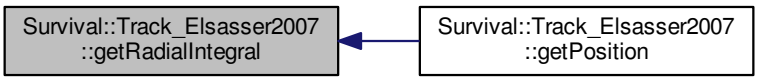
Warning

Not yet implemented.

Implements Survival::Track.

Definition at line 282 of file Track_Elsasser2007.cpp.

Here is the caller graph for this function:



7.15.3.12 virtual double Survival::Track_Elsasser2007::getRadius ( ) const [inline], [virtual]

Returns the effective radius of the track, expressed in um.

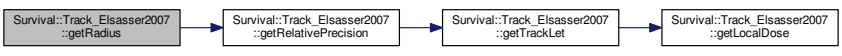
Returns

The effective radius of the track (r_eff) expressed in um.

Implements Survival::Track.

Definition at line 160 of file Track_Elsasser2007.h.

Here is the call graph for this function:



7.15.3.13 double Track_Elsasser2007::getRelativePrecision ( ) const

Evaluates the relative precision of the calculated LET with respect of the original particle LET.

The relative precision is evaluated by the difference between the calculated and the "original" LETs divided by "original" LET.

Returns

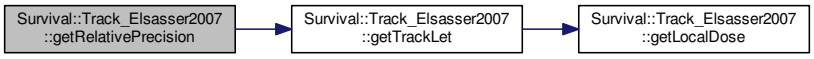
The relative precision of the calculated LET with respect of the original particle LET.

See also

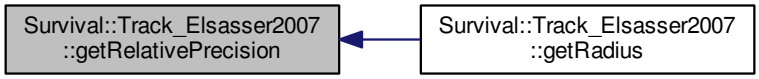
getTrackLet()

Definition at line 321 of file Track_Elsasser2007.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.3.14 virtual double Survival::Track_Elsasser2007::getTime ( ) const [inline], [virtual]

Returns the time associated to a particular event expressed in hours.

Returns

The time associated to a particular event expressed in hours.

See also

time and setTime()

Implements Survival::Track.

Definition at line 178 of file Track_Elsasser2007.h.

7.15.3.15 double Track_Elsasser2007::getTrackLet ( ) const [private]

Evaluates and returns the LET of the track.

Since it was observed a minimal discrepancy from the imposed particle LET, this function calculates the real observed LET of the particle integrating the radial profile.

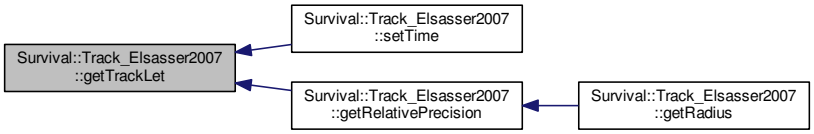
Returns

The calculated LET of the track, expressed in $\frac{MeV}{um}$

Definition at line 402 of file Track_Elsasser2007.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



7.15.3.16 virtual double Survival::Track_Elsasser2007::getWeight ( ) const [inline], [virtual]

Returns the weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

Returns

The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

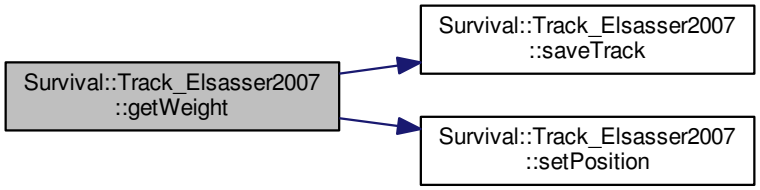
See also

Particle::weight

Implements Survival::Track.

Definition at line 186 of file Track_Elsasser2007.h.

Here is the call graph for this function:



7.15.3.17 double Track_Elsasser2007::normalizedDoseIntegralArgument (const double *r*, const double *r1*) const [private]

Evaluates the argument of the normalized integral in the creation of the master curve.

The calculation is divided in two cases:

- If the argument of the Bessel's function $\rho = \frac{r\,r'}{\sigma^2}$ is smaller than the fixed besselLimit then the evaluation is based on a series development of the Bessel function
- If the argument of the Bessel's function $\rho = \frac{r\,r'}{\sigma^2}$ is grater than the fixed besselLimit then it's used an asymptotic exponential approximation.

Parameters

r	The radial coordinate of the track profile, expressed in um.
r1	The radial coordinate of the gaussian function, expressed in um.

Returns

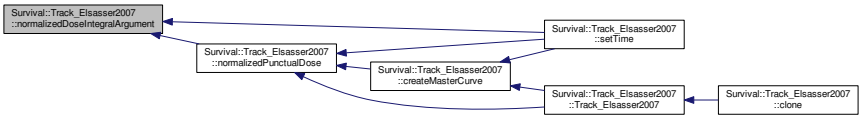
The argument of the integral defining the convolution between the standard radial profile and the gaussian function.

See also

normalizedPunctualDose() and createMasterCurve()

Definition at line 424 of file Track_Elsasser2007.cpp.

Here is the caller graph for this function:



7.15.3.18 double Track_Elsasser2007::normalizedPunctualDose (const double *distance*) const [private]

Evaluates the local dose along the radial profile of the master curve.

The integration process is based on the trapezoidal rule by Newton-Cotes and, step by step, the argument of the integral is evaluated by means of the normalizedDoseIntegralArgument() method.

Parameters

distance	The distance from the track center, expressed in um.

Returns

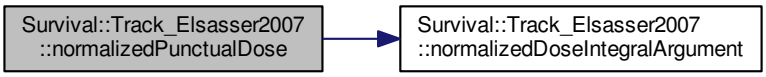
The local dose at a fixed distance from the track center.

See also

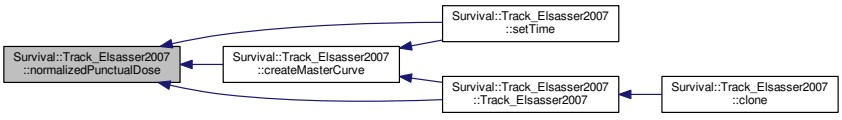
createMasterCurve()

Definition at line 461 of file Track_Elsasser2007.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.15.3.19 string Track_Elsasser2007::saveTrack ( ) const [virtual]

Saves the local dose deposited by the track at different distances from its center to reconstruct the track profile.

It execute a `for` loop saving the values stored in the logRhoMasterCurve, logDoseMasterCurve and logDoseTail data members; that is the local dose deposited and corresponding radii along the whole radial profile.

Returns

The name of the file created.

See also

getLocalDose()

Implements Survival::Track.

Definition at line 328 of file Track_Elsasser2007.cpp.

Here is the caller graph for this function:

7.15.3.20 void Track_Elsasser2007::setPosition (const double *x*, const double *y*) [virtual]

Sets the track position (`x` and `y` coordinates) referred to the beam axis.

Parameters

x	The `x` coordinate of the track to be set, referred to the beam axis and expressed in mm.
y	The `x` coordinate of the track to be set, referred to the beam axis and expressed in mm.

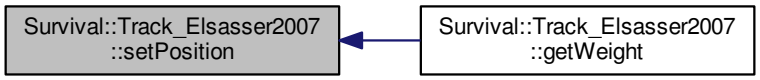
See also

x_track, y_track and getPosition()

Implements Survival::Track.

Definition at line 350 of file Track_Elsasser2007.cpp.

Here is the caller graph for this function:



7.15.3.21 virtual void Survival::Track_Elsasser2007::setTime (double *t*) [inline],[virtual]

Sets the time associated to a particular event.

Parameters

t	The time to be set expressed in hours.

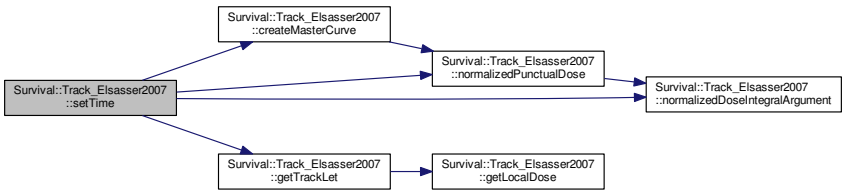
See also

time

Implements Survival::Track.

Definition at line 214 of file Track_Elsasser2007.h.

Here is the call graph for this function:



7.15.4 Member Data Documentation

7.15.4.1 double Track_Elsasser2007::besselLimit [static],[private]

Limit between the calculation of Bessel function by means of series development and the calculation with the asymptotic exponential approximation.

Definition at line 392 of file Track_Elsasser2007.h.

7.15.4.2 const double Track_Elsasser2007::CONV = 160.2177 [static], [private]

Constants static variables and precalculated feature indices.

It's the constant of conversion from $\frac{MeV\,dm^3}{Kg\mu m^3}$ to Gy. It's equal to $160.2177\ \frac{J\mu m^3}{MeV dm^3}$

Definition at line 359 of file Track_Elsasser2007.h.

7.15.4.3 const double Track_Elsasser2007::DELTA = 1.7 [static], [private]

Useful constant value.

It's necessary to evaluate r_max; it's equal to 1.7, according to the LEM II parametrization.

Definition at line 365 of file Track_Elsasser2007.h.

7.15.4.4 double Survival::Track_Elsasser2007::density [private]

The density of the medium expressed in $\frac{g}{cm^3}$.

Definition at line 296 of file Track_Elsasser2007.h.

7.15.4.5 double Survival::Track_Elsasser2007::doseCutoff [private]

Minimum possible dose deposited evaluable, expressed in Gy.

Definition at line 336 of file Track_Elsasser2007.h.

7.15.4.6 double Survival::Track_Elsasser2007::e_c [private]

The kinetic energy of the particle generating the track expressed in MeV.

See also

Particle::e_c

Definition at line 318 of file Track_Elsasser2007.h.

7.15.4.7 const double Track_Elsasser2007::GAMMA = 0.062 [static], [private]

Useful constant value.

It's necessary to evaluate r_max; it's equal to 0.062 $\frac{\mu m}{MeV^\delta}$, according to the LEM II parametrization.

Definition at line 371 of file Track_Elsasser2007.h.

7.15.4.8 double Track_Elsasser2007::integrationStep [static], [private]

The integration step for the generation of the master curve expressed in um. It's proportional to R_MIN.

Definition at line 389 of file Track_Elsasser2007.h.

7.15.4.9 double Survival::Track_Elsasser2007::lambda [private]

A constant value required to evaluate r_max.

It's defined as:

$$\lambda = \frac{1}{\rho\pi(1+\ln(\rho_{max}^2/\rho_{min}^2))}$$

where ρ represents the density of the medium while ρ_{max} and ρ_{min} represent r_max and R_MIN respectively.

It's expressed in $\frac{Gy\,\mu m^3}{MeV}$.

Definition at line 330 of file Track_Elsasser2007.h.

7.15.4.10 int Track_Elsasser2007::lengthMasterCurve [static], [private]

The length of the master curve expressed in um.

Definition at line 386 of file Track_Elsasser2007.h.

7.15.4.11 int Survival::Track_Elsasser2007::lengthMC [private]

Length of the used master curve.

Definition at line 342 of file Track_Elsasser2007.h.

7.15.4.12 int Survival::Track_Elsasser2007::lengthTail [private]

Length of the created tail.

Definition at line 345 of file Track_Elsasser2007.h.

7.15.4.13 double Survival::Track_Elsasser2007::let [private]

The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

See also

Particle::let

Definition at line 312 of file Track_Elsasser2007.h.

7.15.4.14 double Track_Elsasser2007::logDoseMasterCurve [static], [private]

Array to store the calculated values of (logarithmic) local dose deposited, constituting the radial profile of the master curve.

Definition at line 404 of file Track_Elsasser2007.h.

7.15.4.15 double* Survival::Track_Elsasser2007::logDoseTail [private]

A pointer to the values of the calculated (logarithmic) local dose in the tail of the track.

Definition at line 339 of file Track_Elsasser2007.h.

7.15.4.16 double Track_Elsasser2007::logRhoMasterCurve [static], [private]

Array to store the progressive (logarithmic) radii corresponding to the profile of the master curve.

Definition at line 401 of file Track_Elsasser2007.h.

7.15.4.17 const int Track_Elsasser2007::MAX_LENGTH_MASTER_CURVE = 1000 [static], [private]

Maximum length of the master curve (i.e. maximum number of steps, equal to the length of the arrays logRho↩MasterCurve and logDoseMasterCurve).

Definition at line 398 of file Track_Elsasser2007.h.

7.15.4.18 int Track_Elsasser2007::numberOfElsasser2007Tracks = 0 [static], [private]

The number of existing Track_Elsasser2007 objects.

It's incremented in the constructor and decremented in the destructor.

See also

Track_Elsasser2007() and ~Track_Elsasser2007()

Definition at line 415 of file Track_Elsasser2007.h.

7.15.4.19 double Track_Elsasser2007::numberOfSigma [static], [private]

Half width of non-zero window in units of sigma.

Definition at line 395 of file Track_Elsasser2007.h.

7.15.4.20 double Survival::Track_Elsasser2007::particleEnergy [private]

The specific energy of the particle generating the track, expressed in MeV/u.

Definition at line 269 of file Track_Elsasser2007.h.

7.15.4.21 std::string Survival::Track_Elsasser2007::particleType [private]

The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...)

See also

Particle::type

Definition at line 275 of file Track_Elsasser2007.h.

7.15.4.22 double Survival::Track_Elsasser2007::r_eff [private]

The effective radius of the track (expressed in um) corresponding to the point where the local dose results equal to doseCutoff.

Definition at line 333 of file Track_Elsasser2007.h.

7.15.4.23 double Survival::Track_Elsasser2007::r_max [private]

The radius of the track expressed in um.

According to the LEM II parametrization it's evaluated (and instantiated in the constructor) as:

$$r\_max = \gamma\, E^{\delta}$$

where γ is GAMMA, δ is DELTA and *E* represents the specific energy of the ion.

Definition at line 306 of file Track_Elsasser2007.h.

7.15.4.24 const double Track_Elsasser2007::R_MIN = 3e-4 [static], [private]

The core radius expressed in um.

It's taken equal to 0.3 nm according to the LEM II parametrization.

Definition at line 377 of file Track_Elsasser2007.h.

7.15.4.25 const double Track_Elsasser2007::SIGMA = 4e-3 [static], [private]

The radical diffusion length, expressed in um.

It's a constant value equal to 4 nm representing the spreading of the induced radical species taking place at longer time scales (a few microseconds).

Definition at line 383 of file Track_Elsasser2007.h.

7.15.4.26 double Survival::Track_Elsasser2007::time [private]

The time associated to a particular event, expressed in hours.

The *time* isn't really a variable associated to the track, time flows during the irradiation process. For this reason, the initial value of this member represent the instant in which the track is generated, but then it will change during the execution assuming a value representing the interaction time between the track and a specific Nucleus.

Note

Since this track structure is used in the LEM model, that doesn't take into account (yet) the time structure of the irradiation, this data member is actually unuseful.

Definition at line 353 of file Track_Elsasser2007.h.

7.15.4.27 double Track_Elsasser2007::tmpLogDoseTail [static], [private]

Temporary storage of the local dose in the track tail.

Definition at line 407 of file Track_Elsasser2007.h.

7.15.4.28 double Survival::Track_Elsasser2007::weight [private]

The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

See also

Particle::weight

Definition at line 293 of file Track_Elsasser2007.h.

7.15.4.29 double Survival::Track_Elsasser2007::x_track [private]

The track position (x coordinate) referred to the beam axis, expressed in mm.

See also

Particle::x

Definition at line 281 of file Track_Elsasser2007.h.

7.15.4.30 double Survival::Track_Elsasser2007::y_track [private]

The track position (y coordinate) referred to the beam axis, expressed in mm.

See also

Particle::y

Definition at line 287 of file Track_Elsasser2007.h.

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

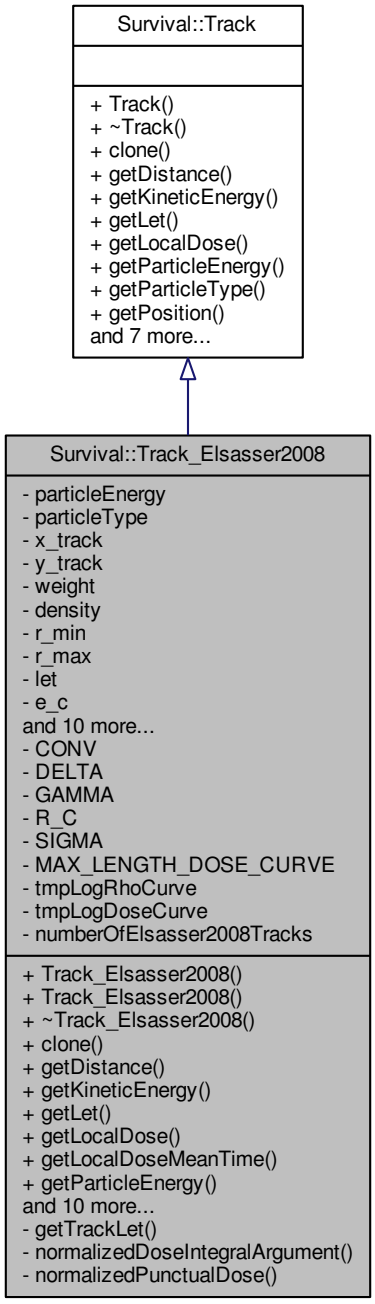
- include/Track_Elsasser2007.h
- src/Track_Elsasser2007.cpp

7.16 Survival::Track_Elsasser2008 Class Reference

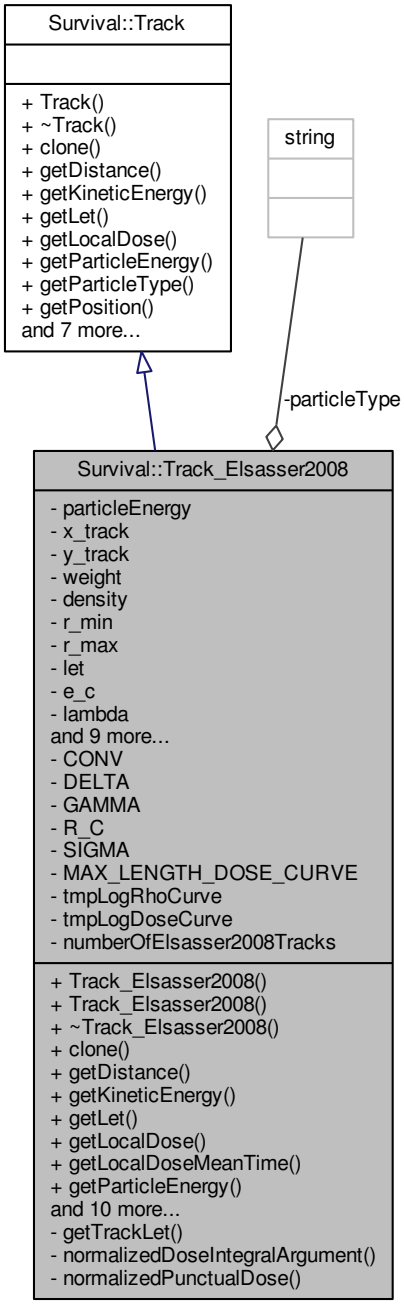
Inherited from the Track class, it implements the LEM III track model.

`#include <Track_Elsasser2008.h>`

Inheritance diagram for Survival::Track_Elsasser2008:



Collaboration diagram for Survival::Track_Elsasser2008:



Public Member Functions

- Track_Elsasser2008 (const Particle &particle, const double density, const double doseCutoff=1e-8, const int lengthDoseCurve=300.0, const double integrationStepFactor=1e-2, const double besselLimit=400.0, const double numberOfSigma=20.0, double t=0.0)

  Constructor. Instantiates and sets the object.
- Track_Elsasser2008 (const Track_Elsasser2008 &track)

Copy constructor. Instantiates a new Track_Elsasser2008 object copying an existent one, including the logRhoCurve and logDoseCurve values.

- virtual ~Track_Elsasser2008 ()

  Destructor.

- virtual Track_Elsasser2008 * clone () const

  Returns a pointer to a new Track_Elsasser2008 object created as a copy of an existent one by means of the copy constructor.

- virtual double getDistance (const double localDose) const

  Returns the distance from the center of the track, knowing the local dose deposited.

- virtual double getKineticEnergy () const

  Returns the kinetic energy of the particle generating the track expressed in MeV.

- virtual double getLet () const

  Returns the LET in water of the particle generating the track expressed in MeV/um.

- virtual double getLocalDose (const double distance) const

  Returns the local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

- double getLocalDoseMeanTime ()

  Function created to calculate the mean time required to the evaluation the getLocalDose() method.

- virtual double getParticleEnergy () const

  Returns the specific energy of the particle generating the track, expressed in MeV/u.

- virtual std::string getParticleType () const

  Returns the type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

- virtual void getPosition (double &returnX, double &returnY) const

  Returns the track position (x and y coordinates) referred to the beam axis and expressed in mm overwriting two `double` *variables passed by reference.*

- virtual double getRadialIntegral (const double r_min, const double r_max) const

  Evaluates the radial integral of the track profile in $[r_{min}, r_{max}]$.

- virtual double getRadius () const

  Returns the effective radius of the track, expressed in um.

- double getRelativePrecision () const

  Evaluates the relative precision of the calculated LET with respect of the original particle LET.

- virtual double getTime () const

  Returns the time associated to a particular event expressed in hours.

- virtual double getWeight () const

  Returns the weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

- virtual std::string saveTrack () const

  Saves the local dose deposited by the track at different distances from its center to reconstruct the track profile.

- virtual void setPosition (const double x, const double y)

  Sets the track position (x and y coordinates) referred to the beam axis.

- virtual void setTime (double t)

  Sets the time associated to a particular event.

Private Member Functions

- double getTrackLet () const

  Evaluates and returns the LET of the track.

- double normalizedDoseIntegralArgument (const double r, const double r1) const

  Evaluates the argument of the normalized integral in the construction of the track profile.

- double normalizedPunctualDose (const double distance) const

  Evaluates the local dose along the radial profile of the track.

Private Attributes

- double particleEnergy

  The specific energy of the particle generating the track, expressed in MeV/u.
- std::string particleType

  The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...)
- double x_track

  The track position (x coordinate) referred to the beam axis, expressed in mm.
- double y_track

  The track position (y coordinate) referred to the beam axis, expressed in mm.
- double weight

  The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".
- double density

  The density of the medium expressed in $\frac{g}{cm^3}$.
- double r_min

  The core radius expressed in um.
- double r_max

  The radius of the track expressed in um.
- double let

  The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).
- double e_c

  The kinetic energy of the particle generating the track expressed in MeV.
- double lambda

  A constant value required to evaluate r_max.
- double r_eff

  The effective radius of the track (expressed in um) corresponding to the point where the local dose results equal to doseCutoff.
- double doseCutoff

  Minimum possible dose deposited evaluable, expressed in Gy.
- double integrationStep

  The integration step for the construction of the track profile expressed in um. It's created proportional to r_min.
- double besselLimit

  Limit between the calculation of Bessel function by means of series development and the calculation with the asymptotic exponential approximation.
- double numberOfSigma

  Half width of non-zero window in units of sigma.
- int lengthDoseCurve

  The length of the dose profile expressed in um.
- double $*$ logRhoCurve

  A pointer to the (logarithmic) radii corresponding to the calculated local dose of the track profile.
- double $*$ logDoseCurve

  A pointer to the values of the calculated (logarithmic) local dose of the track profile.
- double time

  The time associated to a particular event, expressed in hours.

Static Private Attributes

- static const double CONV = 160.2177

  Constants static variables and precalculated feature indices.
- static const double DELTA = 1.7

  Useful constant value.
- static const double GAMMA = 0.062

  Useful constant value.
- static const double R_C = 4e-2

  The maximum core radius, corresponding to ions traveling at the speed of light, expressed in um.
- static const double SIGMA = 4e-3

  The radical diffusion length, expressed in um.
- static const int MAX_LENGTH_DOSE_CURVE = 1000

  Maximum length of the dose profile (i.e. maximum number of steps, equal to the length of the arrays tmpLogRhoCurve and tmpLogDoseCurve).
- static double tmpLogRhoCurve [MAX_LENGTH_DOSE_CURVE]

  Array to temporary store the progressive radii corresponding to the profile of the curve.
- static double tmpLogDoseCurve [MAX_LENGTH_DOSE_CURVE]

  Array to temporary store the calculated values of local dose deposited, constituting the radial profile of the curve.
- static int numberOfElsasser2008Tracks = 0

  The number of existing Track_Elsasser2008 objects.

7.16.1 Detailed Description

Inherited from the Track class, it implements the LEM III track model.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2009

To further increase the level of agreement with carbon ion data with respect to the LEM II formulation (see Track_↩ Elsasser2007) a LEM III version (1) was implemented where the track core radius r_min has been made proportional to the particle velocity, according to the following relation:

$$\rho_{min} = \rho_c \cdot \beta$$

where $\rho_c = 40\,nm$ (R_C) represents the maximum core radius, corresponding to ions traveling at the speed of light, and $beta$ is the relativistic ion velocity.

In this case the *master curve* approach (see Track_Elsasser2007) is not applicable, because the shape of track core deposition varies with the kinetic energy of the particle (due to the previous relation). Therefore the track profile is evaluated at the time of the object construction (object by object) and stored it as a member variable. This could bring issues with memory consumption if a large number of Track_Elsasser2008 objects needs to be instantiate at the same time.

1. T. Elsässer, M. Krämer and M. Scholz, "Accuracy of the local effect model for the prediction of biologic effects of carbon ion beams \em in \em vitro and \em in \em vivo", *International Journal of Radiation Oncology-Biology-Physics **71**, 866-872* (2008).

Definition at line 27 of file Track_Elsasser2008.h.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 Track_Elsasser2008::Track_Elsasser2008 (const Particle & *particle*, const double *density*, const double *doseCutoff* = 1e−8, const int *lengthDoseCurve* = 300.0, const double *integrationStepFactor* = 1e−2, const double *besselLimit* = 400.0, const double *numberOfSigma* = 20.0, double *t* = 0.0)

Constructor. Instantiates and sets the object.

Converts a particle (object of class Particle), passed by reference, in a track according to the LEM III amorphous track model. Some of the data members are instantiated on the basis of the informations stored in the Particle object. (For a more detailed description of the instantiation of each member respectively look at its specific documentation).

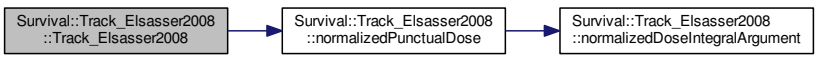
It constructs also the track radial profile via a process similar to the one defined in the Track_Elssasser2007 implementation but without the general purpose of the *master curve*.

Parameters

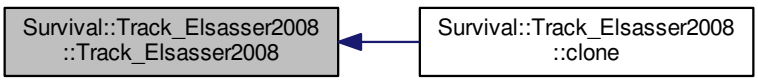
particle	The particle generating the track in the medium, passed by reference.
density	The density of the medium expressed in $\frac{g}{cm^3}$. The default value is the density of water.
doseCutoff	The minimum possible dose deposited evaluable, expressed in Gy (see doseCutoff).
lengthDoseCurve	The length of the curve representing the track radial profile, expressed in um.
integrationStepFactor	Dimensionless integration factor (multiplied by r_min gives the integrationStep)
besselLimit	The limit between the calculation of Bessel function by means of series development and the calculation with the asymptotic exponential approximation (see besselLimit).
numberOfSigma	Half width of non-zero window in units of sigma (see numberOfSigma).
t	The time corresponding to the generation of the track in the target. The default value is 0. (See time).

Definition at line 48 of file Track_Elsasser2008.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.2.2 Track_Elsasser2008::Track_Elsasser2008 (const Track_Elsasser2008 & *track*)

Copy constructor. Instantiates a new Track_Elsasser2008 object copying an existent one, including the logRho↩Curve and logDoseCurve values.

Definition at line 126 of file Track_Elsasser2008.cpp.

7.16.2.3 Track_Elsasser2008::~Track_Elsasser2008 ( ) [virtual]

Destructor.

Delete the object, deallocating also the memory occupied by logRhoCurve and logDoseCurve, and decrements the counter of Track_Elsasser2008 objects (numberOfElsasser2008Tracks).

Definition at line 143 of file Track_Elsasser2008.cpp.

7.16.3 Member Function Documentation

7.16.3.1 Track_Elsasser2008 * Track_Elsasser2008::clone ( ) const [virtual]

Returns a pointer to a new Track_Elsasser2008 object created as a copy of an existent one by means of the copy constructor.

Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

Implements Survival::Track.

Definition at line 153 of file Track_Elsasser2008.cpp.

Here is the call graph for this function:

7.16.3.2 double Track_Elsasser2008::getDistance (const double *localDose*) const [virtual]

Returns the distance from the center of the track, knowing the local dose deposited.

It's the inverse function of getLocalDose(). The function runs in a loop over the precalculated values of the track profile stored in logRhoCurve and logDoseCurve data members until it finds a value smaller than the required dose deposited, then it interpolates the nearest neighbors to get the correct value.

Note

If the dose deposited is smaller than the doseCutoff it returns r_eff.

Parameters

localDose	The local dose deposited, expressed in Gy.

Returns

The distance from the center of the track, expressed in um.

See also

getLocalDose()

Implements Survival::Track.

Definition at line 162 of file Track_Elsasser2008.cpp.

7.16.3.3 virtual double Survival::Track_Elsasser2008::getKineticEnergy ( ) const [inline], [virtual]

Returns the kinetic energy of the particle generating the track expressed in MeV.

Returns

The kinetic energy of the particle generating the track expressed in MeV.

See also

e_c

Implements Survival::Track.

Definition at line 91 of file Track_Elsasser2008.h.

7.16.3.4 virtual double Survival::Track_Elsasser2008::getLet ( ) const [inline], [virtual]

Returns the LET in water of the particle generating the track expressed in MeV/um.

Returns

The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

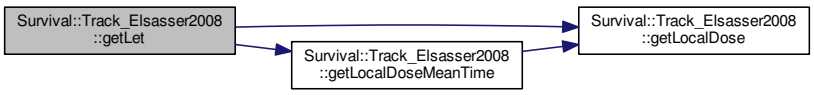
See also

let

Implements Survival::Track.

Definition at line 99 of file Track_Elsasser2008.h.

Here is the call graph for this function:



7.16.3.5 double Track_Elsasser2008::getLocalDose (const double *distance*) const [virtual]

Returns the local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

The function evaluates some possible cases:

- If the distance is smaller than the minimum radius stored in the logRhoCurve it returns the dose at the minimum radius evaluated
- If the distance is greater than the effective radius of the track (r_eff) it returns 0
- Else it returns the precalculated local dose at the required distance obtained by an interpolation of the nearest neighbors.

Parameters

distance	The distance from the track center expressed in um.

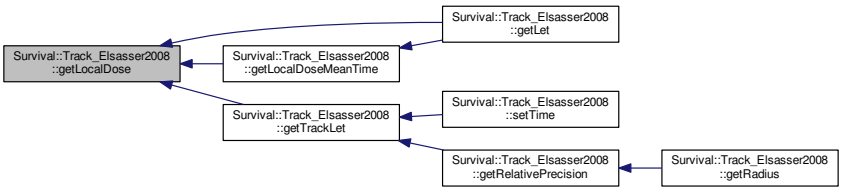
Returns

The local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

Implements Survival::Track.

Definition at line 187 of file Track_Elsasser2008.cpp.

Here is the caller graph for this function:



7.16.3.6 double Track_Elsasser2008::getLocalDoseMeanTime ( )

Function created to calculate the mean time required to the evaluation the getLocalDose() method.

It cyclically calls getLocalDose() 1000000 times, timing the total elapsed time and dividing it by 1000000.

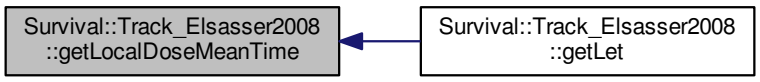
Returns

The mean time needed to a complete evaluation of the getLocalDose() method, expressed in s.

Definition at line 218 of file Track_Elsasser2008.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



7.16.3.7 virtual double Survival::Track_Elsasser2008::getParticleEnergy ( ) const [inline], [virtual]

Returns the specific energy of the particle generating the track, expressed in MeV/u.

Returns

The specific energy of the particle generating the track, expressed in MeV/u.

See also

particleEnergy

Implements Survival::Track.

Definition at line 128 of file Track_Elsasser2008.h.

7.16.3.8 virtual std::string Survival::Track_Elsasser2008::getParticleType ( ) const [inline], [virtual]

Returns the type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

Returns

The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

See also

Particle::type

Implements Survival::Track.

Definition at line 136 of file Track_Elsasser2008.h.

7.16.3.9 virtual void Survival::Track_Elsasser2008::getPosition (double & *returnX*, double & *returnY*) const [inline], [virtual]

Returns the track position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the track referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the track, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `x` coordinate of the track, expressed in mm, passed by reference to be overwritten.

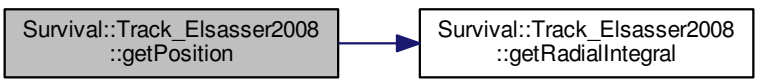
See also

setPosition()

Implements Survival::Track.

Definition at line 147 of file Track_Elsasser2008.h.

Here is the call graph for this function:



7.16.3.10 double Track_Elsasser2008::getRadialIntegral (const double *r_min*, const double *r_max*) const [virtual]

Evaluates the radial integral of the track profile in $[r_{min}, r_{max}]$.

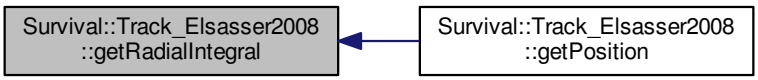
Warning

Not yet implemented.

Implements Survival::Track.

Definition at line 235 of file Track_Elsasser2008.cpp.

Here is the caller graph for this function:



7.16.3.11 virtual double Survival::Track_Elsasser2008::getRadius ( ) const [inline], [virtual]

Returns the effective radius of the track, expressed in um.

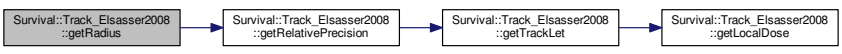
Returns

The effective radius of the track (r_eff) expressed in um.

Implements Survival::Track.

Definition at line 165 of file Track_Elsasser2008.h.

Here is the call graph for this function:



7.16.3.12 double Track_Elsasser2008::getRelativePrecision ( ) const

Evaluates the relative precision of the calculated LET with respect of the original particle LET.

The relative precision is evaluated by the difference between the calculated and the "original" LETs divided by "original" LET.

Returns

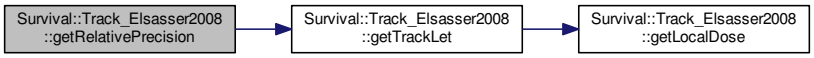
The relative precision of the calculated LET with respect of the original particle LET.

See also

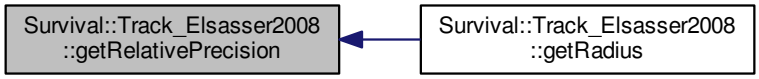
getTrackLet()

Definition at line 274 of file Track_Elsasser2008.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.3.13 virtual double Survival::Track_Elsasser2008::getTime ( ) const [inline], [virtual]

Returns the time associated to a particular event expressed in hours.

Returns

The time associated to a particular event expressed in hours.

See also

time and setTime()

Implements Survival::Track.

Definition at line 183 of file Track_Elsasser2008.h.

7.16.3.14 double Track_Elsasser2008::getTrackLet ( ) const [private]

Evaluates and returns the LET of the track.

Since it was observed a minimal discrepancy from the imposed particle LET, this function calculates the real observed LET of the particle integrating the radial profile.

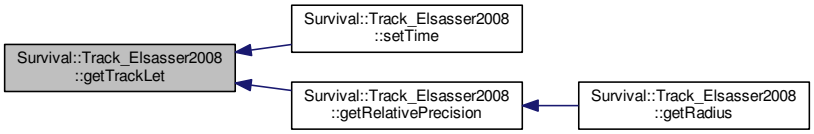
Returns

The calculated LET of the track, expressed in $\frac{MeV}{um}$

Definition at line 308 of file Track_Elsasser2008.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



7.16.3.15 virtual double Survival::Track_Elsasser2008::getWeight ( ) const [inline], [virtual]

Returns the weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

Returns

The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

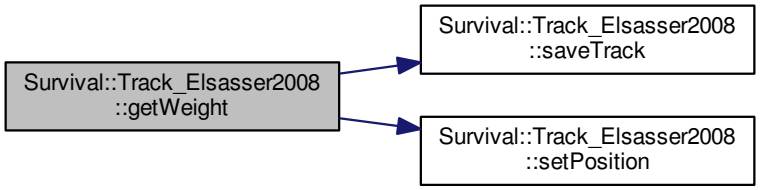
See also

Particle::weight

Implements Survival::Track.

Definition at line 191 of file Track_Elsasser2008.h.

Here is the call graph for this function:



7.16.3.16 double Track_Elsasser2008::normalizedDoseIntegralArgument (const double *r*, const double *r1*) const [private]

Evaluates the argument of the normalized integral in the construction of the track profile.

The calculation is divided in two cases:

- If the argument of the Bessel's function $\rho = \frac{r\,r'}{\sigma^2}$ is smaller than the fixed besselLimit then the evaluation is based on a series development of the Bessel function
- If the argument of the Bessel's function $\rho = \frac{r\,r'}{\sigma^2}$ is grater than the fixed besselLimit then it's used an asymptotic exponential approximation.

Parameters

r	The radial coordinate of the track profile, expressed in um.
r1	The radial coordinate of the gaussian function, expressed in um.

Returns

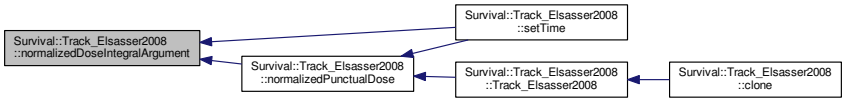
The argument of the integral defining the convolution between the standard radial profile and the gaussian function.

See also

normalizedPunctualDose()

Definition at line 330 of file Track_Elsasser2008.cpp.

Here is the caller graph for this function:



7.16.3.17 double Track_Elsasser2008::normalizedPunctualDose (const double *distance*) const [private]

Evaluates the local dose along the radial profile of the track.

The integration process is based on the trapezoidal rule by Newton-Cotes and, step by step, the argument of the integral is evaluated by means of the normalizedDoseIntegralArgument() method.

Parameters

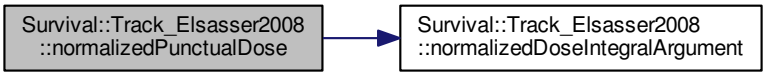
distance	The distance from the track center, expressed in um.

Returns

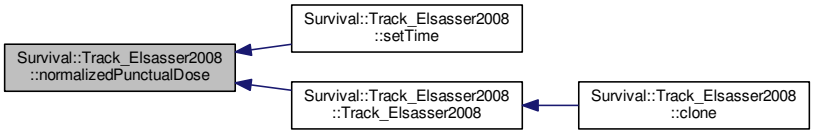
The local dose at a fixed distance from the track center.

Definition at line 368 of file Track_Elsasser2008.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.16.3.18 string Track_Elsasser2008::saveTrack ( ) const [virtual]

Saves the local dose deposited by the track at different distances from its center to reconstruct the track profile.

It execute a `for` loop saving the values stored in the logRhoCurve and logDoseCurve data members; that is the local dose deposited and corresponding radii along the whole radial profile.

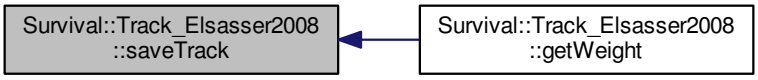
Returns

The name of the file created.

Implements Survival::Track.

Definition at line 281 of file Track_Elsasser2008.cpp.

Here is the caller graph for this function:



7.16.3.19 void Track_Elsasser2008::setPosition (const double *x*, const double *y*) [virtual]

Sets the track position (`x` and `y` coordinates) referred to the beam axis.

Parameters

x	The `x` coordinate of the track to be set, referred to the beam axis and expressed in mm.
y	The `x` coordinate of the track to be set, referred to the beam axis and expressed in mm.

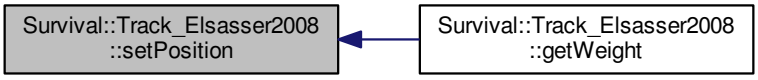
See also

x_track, y_track and getPosition()

Implements Survival::Track.

Definition at line 299 of file Track_Elsasser2008.cpp.

Here is the caller graph for this function:



7.16.3.20 virtual void Survival::Track_Elsasser2008::setTime (double *t*) [inline], [virtual]

Sets the time associated to a particular event.

Parameters

t	The time to be set expressed in hours.

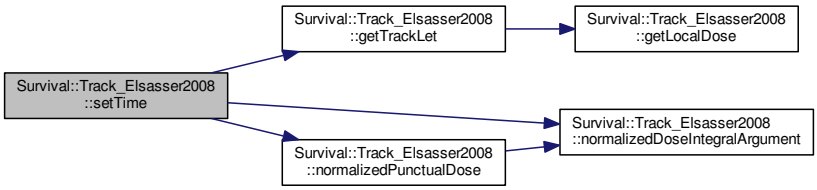
See also

time

Implements Survival::Track.

Definition at line 217 of file Track_Elsasser2008.h.

Here is the call graph for this function:



7.16.4 Member Data Documentation

7.16.4.1 double Survival::Track_Elsasser2008::besselLimit [private]

Limit between the calculation of Bessel function by means of series development and the calculation with the asymptotic exponential approximation.

Definition at line 341 of file Track_Elsasser2008.h.

7.16.4.2 const double Track_Elsasser2008::CONV = 160.2177 [static], [private]

Constants static variables and precalculated feature indices.

It's the constant of conversion from $\frac{MeV\,dm^3}{Kg\mu m^3}$ to Gy. It's equal to $160.2177\ \frac{J\mu m^3}{MeV\,dm^3}$

Definition at line 367 of file Track_Elsasser2008.h.

7.16.4.3 const double Track_Elsasser2008::DELTA = 1.7 [static], [private]

Useful constant value.

It's necessary to evaluate r_max; it's equal to 1.7, according to the LEM III parametrization.

Definition at line 373 of file Track_Elsasser2008.h.

7.16.4.4 double Survival::Track_Elsasser2008::density [private]

The density of the medium expressed in $\frac{g}{cm^3}$.

Definition at line 284 of file Track_Elsasser2008.h.

7.16.4.5 double Survival::Track_Elsasser2008::doseCutoff [private]

Minimum possible dose deposited evaluable, expressed in Gy.

Definition at line 335 of file Track_Elsasser2008.h.

7.16.4.6 double Survival::Track_Elsasser2008::e_c [private]

The kinetic energy of the particle generating the track expressed in MeV.

See also

Particle::e_c

Definition at line 317 of file Track_Elsasser2008.h.

7.16.4.7 const double Track_Elsasser2008::GAMMA = 0.062 [static], [private]

Useful constant value.

It's necessary to evaluate r_max; it's equal to 0.062 $\frac{\mu m}{MeV^{\delta}}$, according to the LEM III parametrization.

Definition at line 379 of file Track_Elsasser2008.h.

7.16.4.8 double Survival::Track_Elsasser2008::integrationStep [private]

The integration step for the construction of the track profile expressed in um. It's created proportional to r_min.

Definition at line 338 of file Track_Elsasser2008.h.

7.16.4.9 double Survival::Track_Elsasser2008::lambda [private]

A constant value required to evaluate r_max.

It's defined as:

$$\lambda = \frac{1}{\rho\pi(1 + \ln(\rho_{max}^2/\rho_{min}^2))}$$

where ρ represents the density of the medium while ρ_{max} and ρ_{min} represent r_max and r_min respectively.

It's expressed in $\frac{Gy\,\mu m^3}{MeV}$.

Definition at line 329 of file Track_Elsasser2008.h.

7.16.4.10 int Survival::Track_Elsasser2008::lengthDoseCurve [private]

The length of the dose profile expressed in um.

Definition at line 347 of file Track_Elsasser2008.h.

7.16.4.11 double Survival::Track_Elsasser2008::let [private]

The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

See also

Particle::let

Definition at line 311 of file Track_Elsasser2008.h.

7.16.4.12 double* Survival::Track_Elsasser2008::logDoseCurve [private]

A pointer to the values of the calculated (logarithmic) local dose of the track profile.

Definition at line 353 of file Track_Elsasser2008.h.

7.16.4.13 double* Survival::Track_Elsasser2008::logRhoCurve [private]

A pointer to the (logarithmic) radii corresponding to the calculated local dose of the track profile.

Definition at line 350 of file Track_Elsasser2008.h.

7.16.4.14 const int Track_Elsasser2008::MAX_LENGTH_DOSE_CURVE = 1000 [static], [private]

Maximum length of the dose profile (i.e. maximum number of steps, equal to the length of the arrays tmpLogRho↩Curve and tmpLogDoseCurve).

Definition at line 391 of file Track_Elsasser2008.h.

7.16.4.15 int Track_Elsasser2008::numberOfElsasser2008Tracks = 0 [static], [private]

The number of existing Track_Elsasser2008 objects.

It's incremented in the constructor and decremented in the destructor.

See also

Track_Elsasser2008() and ~Track_Elsasser2008()

Definition at line 405 of file Track_Elsasser2008.h.

7.16.4.16 double Survival::Track_Elsasser2008::numberOfSigma [private]

Half width of non-zero window in units of sigma.

Definition at line 344 of file Track_Elsasser2008.h.

7.16.4.17 double Survival::Track_Elsasser2008::particleEnergy [private]

The specific energy of the particle generating the track, expressed in MeV/u.

Definition at line 257 of file Track_Elsasser2008.h.

7.16.4.18 std::string Survival::Track_Elsasser2008::particleType [private]

The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...)

See also

Particle::type

Definition at line 263 of file Track_Elsasser2008.h.

7.16.4.19 const double Track_Elsasser2008::R_C = 4e-2 [static], [private]

The maximum core radius, corresponding to ions traveling at the speed of light, expressed in um.

Definition at line 382 of file Track_Elsasser2008.h.

7.16.4.20 double Survival::Track_Elsasser2008::r_eff [private]

The effective radius of the track (expressed in um) corresponding to the point where the local dose results equal to doseCutoff.

Definition at line 332 of file Track_Elsasser2008.h.

7.16.4.21 double Survival::Track_Elsasser2008::r_max [private]

The radius of the track expressed in um.

According to the LEM III parametrization it's evaluated (and instantiated in the constructor) as:

$$r\_max = \gamma \, E^{\delta}$$

where γ is GAMMA, δ is DELTA and *E* represents the specific energy of the ion.

Definition at line 305 of file Track_Elsasser2008.h.

7.16.4.22 double Survival::Track_Elsasser2008::r_min [private]

The core radius expressed in um.

According to the LEM III formulation, it's proportional to the relativistic ion velocity:

$$\rho_{min} = \rho_c \cdot \beta$$

See also

R_C

Definition at line 295 of file Track_Elsasser2008.h.

7.16.4.23 const double Track_Elsasser2008::SIGMA = 4e-3 [static], [private]

The radical diffusion length, expressed in um.

It's a constant value equal to 4 nm representing the spreading of the induced radical species taking place at longer time scales (a few microseconds).

Definition at line 388 of file Track_Elsasser2008.h.

7.16.4.24 double Survival::Track_Elsasser2008::time [private]

The time associated to a particular event, expressed in hours.

The *time* isn't really a variable associated to the track, time flows during the irradiation process. For this reason, the initial value of this member represent the instant in which the track is generated, but then it will change during the execution assuming a value representing the interaction time between the track and a specific Nucleus.

Note

Since this track structure is used in the LEM model, that doesn't take into account (yet) the time structure of the irradiation, this data member is actually useless.

Definition at line 361 of file Track_Elsasser2008.h.

7.16.4.25 double Track_Elsasser2008::tmpLogDoseCurve [static], [private]

Array to temporary store the calculated values of local dose deposited, constituting the radial profile of the curve.

Definition at line 397 of file Track_Elsasser2008.h.

7.16.4.26 double Track_Elsasser2008::tmpLogRhoCurve [static], [private]

Array to temporary store the progressive radii corresponding to the profile of the curve.

Definition at line 394 of file Track_Elsasser2008.h.

7.16.4.27 double Survival::Track_Elsasser2008::weight [private]

The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

See also

Particle::weight

Definition at line 281 of file Track_Elsasser2008.h.

7.16.4.28 double Survival::Track_Elsasser2008::x_track [private]

The track position (x coordinate) referred to the beam axis, expressed in mm.

See also

Particle::x

Definition at line 269 of file Track_Elsasser2008.h.

7.16.4.29 double Survival::Track_Elsasser2008::y_track [private]

The track position (y coordinate) referred to the beam axis, expressed in mm.

See also

Particle::y

Definition at line 275 of file Track_Elsasser2008.h.

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

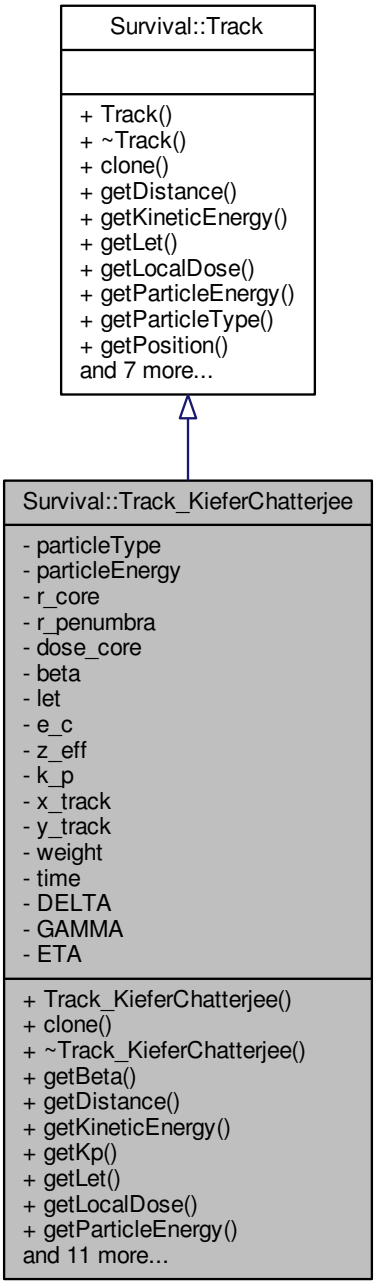
- include/Track_Elsasser2008.h
- src/Track_Elsasser2008.cpp

7.17 Survival::Track_KieferChatterjee Class Reference

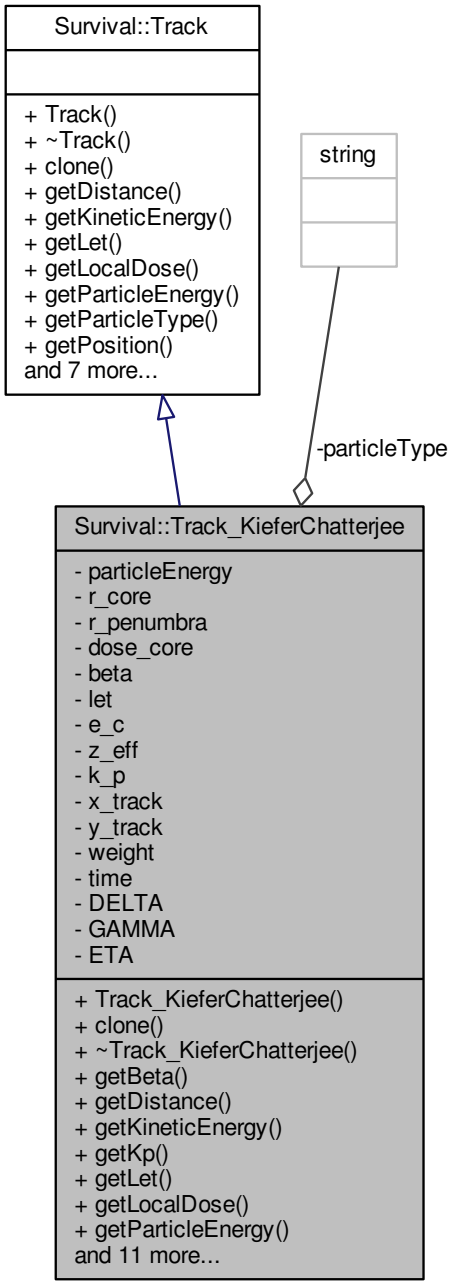
Inherited from the Track pure virtual class, it implements the Kiefer-Chatterje amorphous track structure, used in the MKM model.

`#include <Track_KieferChatterjee.h>`

Inheritance diagram for Survival::Track_KieferChatterjee:



Collaboration diagram for Survival::Track_KieferChatterjee:



Public Member Functions

- Track_KieferChatterjee (const Particle &particle, const double density, double t=0.0)

  Constructor. Instantiates and sets the object.
- virtual Track_KieferChatterjee ∗ clone () const

  Returns a pointer to a new Track_KieferChatterjee object created as a copy of an existent one by means of the copy constructor.

- virtual ~Track_KieferChatterjee ()

  Destructor.
- double getBeta () const

  Returns the ratio between the speed of the ion generating the track and the speed of light.
- virtual double getDistance (const double localDose) const

  Returns the distance from the center of the track, knowing the local dose deposited.
- virtual double getKineticEnergy () const

  Returns the kinetic energy of the particle generating the track expressed in MeV.
- double getKp () const

  Returns the k_p value used to evaluate the dose to the core and to the penumbra.
- virtual double getLet () const

  Returns the LET in water of the particle generating the track expressed in MeV/um.
- virtual double getLocalDose (const double distance) const

  Returns the local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).
- virtual double getParticleEnergy () const

  Returns the specific energy of the particle generating the track, expressed in MeV/u.
- virtual std::string getParticleType () const

  Returns the type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).
- virtual void getPosition (double &returnX, double &returnY) const

  Returns the track position (x and y coordinates) referred to the beam axis and expressed in mm overwriting two `double` *variables passed by reference.*
- virtual double getRadialIntegral (const double r_min, const double r_max) const

  Returns the dose (in Gy) deposited by the track evaluated as the radial integral of the track profile between r_{min} and r_{max} (expressed in um).
- virtual double getRadius () const

  Returns the penumbra radius of the track expressed in um.
- double getRCore () const

  Returns the core radius of the track expressed in um.
- virtual double getTime () const

  Returns the time associated to a particular event expressed in hours.
- virtual double getWeight () const

  Returns the weight in the beam of the particle generating the track. Useful in the case of "mixed fields".
- double getZBarkas () const

  Returns the Barkas effective charge.
- virtual std::string saveTrack () const

  Saves the local dose deposited by the track at different distances from its center to reconstruct the track profile.
- virtual void setPosition (const double x, const double y)

  Sets the track position (x and y coordinates) referred to the beam axis.
- virtual void setTime (double t)

  Sets the time associated to a particular event.

Private Attributes

- std::string particleType

  The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).
- double particleEnergy

  The specific energy of the particle generating the track, expressed in MeV/u.
- double r_core

  The core radius of the track expressed in μm.
- double r_penumbra

The penumbra radius of the track expressed in μm.

- double dose_core

  Constant dose in the core of the track expressed in Gy.

- double beta

  It's the ratio between the speed of the ion and the speed of light.

- double let

  The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

- double e_c

  The kinetic energy of the particle generating the track expressed in MeV.

- double z_eff

  Barkas effective charge.

- double k_p

  Value used to evaluate the dose to the core and to the penumbra. It's a function of the Barkas effective charge and of the β_{Ion} (beta).

- double x_track

  The track position (x coordinate) referred to the beam axis, expressed in mm.

- double y_track

  The track position (y coordinate) referred to the beam axis, expressed in mm.

- double weight

  The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

- double time

  The time associated to a particular event, expressed in hours.

Static Private Attributes

- static const double DELTA = 1.7

  Constant value defined in the Kiefer-Chatterjee amorphous model.

- static const double GAMMA = 0.0616

  Constant value defined in the Kiefer-Chatterjee amorphous model.

- static const double ETA = 0.0116

  Constant value defined in the Kiefer-Chatterjee amorphous model.

7.17.1 Detailed Description

Inherited from the Track pure virtual class, it implements the Kiefer-Chatterje amorphous track structure, used in the MKM model.

Author

Andrea Attili
Lorenzo Manganaro
Lorenzo Marengo
Germano Russo

Date

2011-2015

The structure of the track is defined in the publication by Kase *et al.* (1) in which they suggest to use the Kiefer-↩ Chatterjee amorphous model. The track profile is characterized by an inner region, the *core*, and an outer region, the *penumbra*. The radius of the core and the penumbra respectively are evaluated by means of the following relations

$$R_{core} = \eta \cdot \beta_{Ion}$$

$$R_{penumbra} = \gamma \left(\frac{E}{A}\right)^{\delta}$$

Where γ, δ and η are constants defined in the publication reference (see below), β_{Ion} represents the ratio between the speed of the ion and the speed of light, E is the energy of the ion and A his mass number. The values implemented for γ, δ and η are:

- $\gamma = 0.0616 \frac{\mu m}{MeV^{\delta}}$
- $\delta = 1.7$
- $\eta = 0.0116\,\mu m$

This class provides some methods to evaluate and get the local dose deposited by the ion along the track and the radial integral of the track.

1. Y. Kase, T. Kanai, N. Matsufuji, Y. Furusawa, T. Elsasser, and M. Scholz, "Biophysical calculation of cell survival probabilities using amorphous track structure models for heavy-ion irradiation", *Physics in Medicine and Biology* **53**, 37-59 (2008).

See also

Track, Nucleus_MKM and Nucleus_tMKM

Definition at line 39 of file Track_KieferChatterjee.h.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Track_KieferChatterjee::Track_KieferChatterjee (const Particle & *particle*, const double *density*, double *t* = 0.0)

Constructor. Instantiates and sets the object.

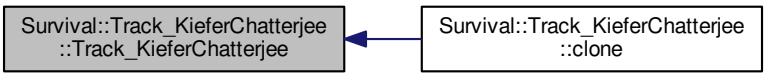
Converts a particle (object of class Particle), passed by reference, in a track according to the Kiefer-Chatterjee amorphous track model. All data members are instantiated on the basis of the informations stored in the Particle object. (For a more detailed description of the instantiation of each member respectively look at its specific documentation).

Parameters

particle	The particle generating the track in the medium, passed by reference.
density	The density of the medium expressed in $\frac{g}{cm^3}$. The default value is the density of water.
t	The time corresponding to the generation of the track in the target. The default value is 0. (See also the documentation of the data member time).

Definition at line 37 of file Track_KieferChatterjee.cpp.

Here is the caller graph for this function:



7.17.2.2 virtual Survival::Track_KieferChatterjee::~Track_KieferChatterjee ( ) [inline], [virtual]

Destructor.

Definition at line 63 of file Track_KieferChatterjee.h.

7.17.3 Member Function Documentation

7.17.3.1 Track_KieferChatterjee * Track_KieferChatterjee::clone ( ) const [virtual]

Returns a pointer to a new Track_KieferChatterjee object created as a copy of an existent one by means of the copy constructor.

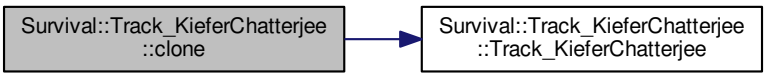
Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

Implements Survival::Track.

Definition at line 77 of file Track_KieferChatterjee.cpp.

Here is the call graph for this function:



7.17.3.2 double Survival::Track_KieferChatterjee::getBeta ( ) const [inline]

Returns the ratio between the speed of the ion generating the track and the speed of light.

Returns

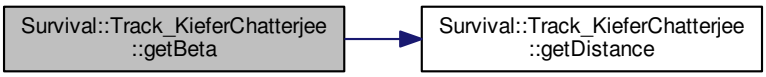
The ratio between the speed of the ion generating the track and the speed of light.

See also

beta

Definition at line 71 of file Track_KieferChatterjee.h.

Here is the call graph for this function:



7.17.3.3 double Track_KieferChatterjee::getDistance (const double *localDose*) const [virtual]

Returns the distance from the center of the track, knowing the local dose deposited.

It's the inverse function of getLocalDose(). Since the maximum possible local dose deposited is: $d_{MAX} = \frac{k_p}{R_c^2}$, if d is greater than d_{MAX} it returns -1 (nonsense value). Else it returns:

$$\sqrt{\frac{k_p}{d}}$$

where d is the local dose, R_c is r_core and k_p is k_p.

Note

If $d < d_{MIN} = \frac{k_p}{R_p^2}$ it returns r_penumbra (R_p).

Parameters

localDose	The local dose deposited expressed in Gy.

Returns

The distance from the center of the track, expressed in um.

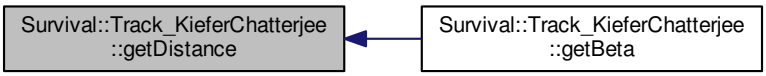
See also

getLocalDose()

Implements Survival::Track.

Definition at line 84 of file Track_KieferChatterjee.cpp.

Here is the caller graph for this function:



7.17.3.4 virtual double Survival::Track_KieferChatterjee::getKineticEnergy ( ) const [inline], [virtual]

Returns the kinetic energy of the particle generating the track expressed in MeV.

Returns

The kinetic energy of the particle generating the track expressed in MeV.

See also

e_c

Implements Survival::Track.

Definition at line 97 of file Track_KieferChatterjee.h.

7.17.3.5 double Survival::Track_KieferChatterjee::getKp ( ) const [inline]

Returns the k_p value used to evaluate the dose to the core and to the penumbra.

Returns

The k_p value used to evaluate the dose to the core and to the penumbra.

Definition at line 103 of file Track_KieferChatterjee.h.

7.17.3.6 virtual double Survival::Track_KieferChatterjee::getLet ( ) const [inline], [virtual]

Returns the LET in water of the particle generating the track expressed in MeV/um.

Returns

The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

See also

let

Implements Survival::Track.

Definition at line 111 of file Track_KieferChatterjee.h.

Here is the call graph for this function:

7.17.3.7 double Track_KieferChatterjee::getLocalDose (const double *distance*) const [virtual]

Returns the local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

The function evaluates some possible cases:

- If the distance is smaller than the core radius it returns dose_core
- If the distance is greater than the penumbra radius it returns 0
- Else it returns the dose to the penumbra calculated as: $\frac{k_p}{d^2}$, where *d* is the "distance" parameter

Parameters

distance	The distance from the track center expressed in um.

Returns

The local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

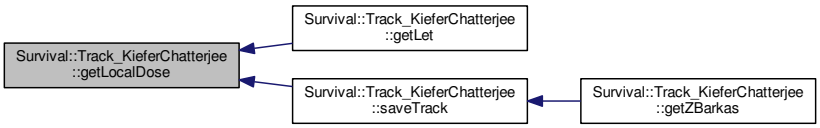
See also

r_core and r_penumbra

Implements Survival::Track.

Definition at line 96 of file Track_KieferChatterjee.cpp.

Here is the caller graph for this function:



7.17.3.8 virtual double Survival::Track_KieferChatterjee::getParticleEnergy ( ) const [inline], [virtual]

Returns the specific energy of the particle generating the track, expressed in MeV/u.

Returns

The specific energy of the particle generating the track, expressed in MeV/u.

See also

particleEnergy

Implements Survival::Track.

Definition at line 134 of file Track_KieferChatterjee.h.

7.17.3.9 virtual std::string Survival::Track_KieferChatterjee::getParticleType ( ) const [inline], [virtual]

Returns the type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

Returns

The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

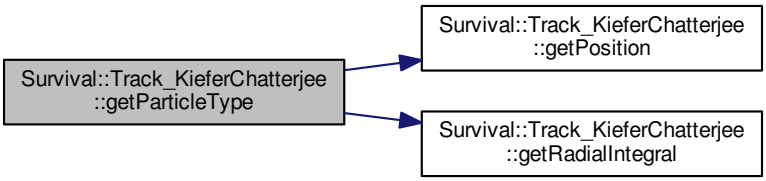
See also

Particle::type

Implements Survival::Track.

Definition at line 142 of file Track_KieferChatterjee.h.

Here is the call graph for this function:



7.17.3.10 void Track_KieferChatterjee::getPosition (double & *returnX*, double & *returnY*) const [virtual]

Returns the track position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the track referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the track, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `x` coordinate of the track, expressed in mm, passed by reference to be overwritten.

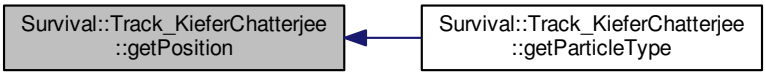
See also

setPosition()

Implements Survival::Track.

Definition at line 111 of file Track_KieferChatterjee.cpp.

Here is the caller graph for this function:



7.17.3.11 double Track_KieferChatterjee::getRadialIntegral (const double *r_min*, const double *r_max*) const [virtual]

Returns the dose (in Gy) deposited by the track evaluated as the radial integral of the track profile between r_{min} and r_{max} (expressed in um).

Parameters

r_min	Lower limit of integration, expressed in um.
r_max	Upper limit of integration, expressed in um.

Returns

The dose (in Gy) deposited by the track evaluated as the radial integral of the track profile between r_{min} and r_{max}.

Warning

The execution of the program will be terminated if incorrect limits of integration are chosen, that is:

- If $r_{min} < 0$
- If $r_{max} < r_{min}$

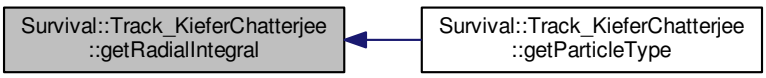
See also

dose_core and getLocalDose()

Implements Survival::Track.

Definition at line 120 of file Track_KieferChatterjee.cpp.

Here is the caller graph for this function:



7.17.3.12 virtual double Survival::Track_KieferChatterjee::getRadius ( ) const [inline], [virtual]

Returns the penumbra radius of the track expressed in um.

Returns

The penumbra radius of the track expressed in um.

See also

r_penumbra and r_core

Implements Survival::Track.

Definition at line 178 of file Track_KieferChatterjee.h.

7.17.3.13 double Survival::Track_KieferChatterjee::getRCore ( ) const [inline]

Returns the core radius of the track expressed in um.

Returns

The core radius of the track expressed in um.

See also

r_core and r_penumbra

Definition at line 186 of file Track_KieferChatterjee.h.

7.17.3.14 virtual double Survival::Track_KieferChatterjee::getTime ( ) const [inline], [virtual]

Returns the time associated to a particular event expressed in hours.

Returns

The time associated to a particular event expressed in hours.

See also

time and setTime()

Implements Survival::Track.

Definition at line 194 of file Track_KieferChatterjee.h.

7.17.3.15 virtual double Survival::Track_KieferChatterjee::getWeight ( ) const [inline], [virtual]

Returns the weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

Returns

The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

See also

Particle::weight

Implements Survival::Track.

Definition at line 202 of file Track_KieferChatterjee.h.

7.17.3.16 double Survival::Track_KieferChatterjee::getZBarkas ( ) const [inline]

Returns the Barkas effective charge.

Returns

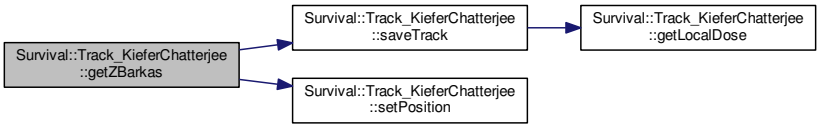
The Barkas effective charge.

See also

z_eff

Definition at line 210 of file Track_KieferChatterjee.h.

Here is the call graph for this function:



7.17.3.17 string Track_KieferChatterjee::saveTrack ( ) const [virtual]

Saves the local dose deposited by the track at different distances from its center to reconstruct the track profile.

The function divides the track penumbra radius into 300 logarithmically spaced distances from the track center, for each of these distances it evaluates the local dose deposited calling the function getLocalDose(); during the process it saves each distance and the corresponding dose in a new file. This allow to reconstruct the track profile.

Returns

The name of the file created.

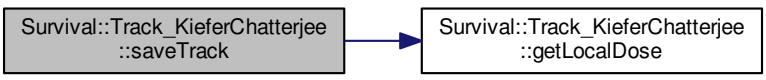
See also

getLocalDose()

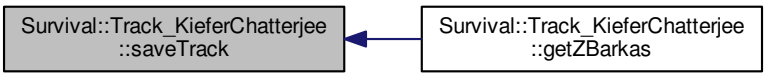
Implements Survival::Track.

Definition at line 152 of file Track_KieferChatterjee.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.18 void Track_KieferChatterjee::setPosition (const double *x*, const double *y*) [virtual]

Sets the track position (x and y coordinates) referred to the beam axis.

Parameters

x	The x coordinate of the track to be set, referred to the beam axis and expressed in mm.
y	The x coordinate of the track to be set, referred to the beam axis and expressed in mm.

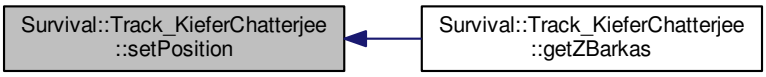
See also

x_track, y_track and getPosition()

Implements Survival::Track.

Definition at line 173 of file Track_KieferChatterjee.cpp.

Here is the caller graph for this function:



7.17.3.19 virtual void Survival::Track_KieferChatterjee::setTime (double *t*) [inline], [virtual]

Sets the time associated to a particular event.

Parameters

t	The time to be set expressed in hours.

See also

time

Implements Survival::Track.

Definition at line 238 of file Track_KieferChatterjee.h.

7.17.4 Member Data Documentation

7.17.4.1 double Survival::Track_KieferChatterjee::beta [private]

It's the ratio between the speed of the ion and the speed of light.

It is evaluated by the information stored in the Particle object using the following relation:

$$\beta = \sqrt{1 - \left(\frac{E_k}{E_0} + 1\right)^{-2}}$$

Where E_k represents the kinetic energy of the particle (e_c; Particle::e_c) and E_0 his rest energy (Particle::rest↩ Energy).

It is instantiated in the Constructor.

See also

Track_KieferChatterjee(const Particle&, const double, double)

Definition at line 306 of file Track_KieferChatterjee.h.

7.17.4.2 const double Track_KieferChatterjee::DELTA = 1.7 [static], [private]

Constant value defined in the Kiefer-Chatterjee amorphous model.

It's equal to:

$$\delta = 1.7$$

Definition at line 377 of file Track_KieferChatterjee.h.

7.17.4.3 double Survival::Track_KieferChatterjee::dose_core [private]

Constant dose in the core of the track expressed in Gy.

Following the Kiefer-Chatterjee amorphous model, the dose in the core is a constant value evaluated by means of the relation:

$$D_c = \frac{1}{\pi R_c^2}\left(\frac{LET}{\rho} - 2\pi k_p \ln\left(\frac{R_p}{R_c}\right)\right)$$

Where R_c and R_p represent respectively the radius of the core and the penumbra; ρ represents the density of the medium (Tracks::density), LET is the unrestricted linear energy transfer for the incident ion, k_p (k_p) is a function of β_{Ion} (beta) and Z_{eff} (z_eff), that is the ratio between the speed of the ion and the speed of light and the Barkas effective charge respectively.

It is instantiated in the Constructor.

See also

Track_KieferChatterjee(const Particle&, const double, double)

Definition at line 292 of file Track_KieferChatterjee.h.

7.17.4.4 double Survival::Track_KieferChatterjee::e_c [private]

The kinetic energy of the particle generating the track expressed in MeV.

See also

Particle::e_c

Definition at line 318 of file Track_KieferChatterjee.h.

7.17.4.5 const double Track_KieferChatterjee::ETA = 0.0116 [static], [private]

Constant value defined in the Kiefer-Chatterjee amorphous model.

It's equal to:

$$\eta = 0.0116\,\mu m$$

Definition at line 395 of file Track_KieferChatterjee.h.

7.17.4.6 const double Track_KieferChatterjee::GAMMA = 0.0616 [static], [private]

Constant value defined in the Kiefer-Chatterjee amorphous model.

It's equal to:

$$\gamma = 0.0616\,\frac{\mu m}{MeV^{\delta}}$$

Definition at line 386 of file Track_KieferChatterjee.h.

7.17.4.7 double Survival::Track_KieferChatterjee::k_p [private]

Value used to evaluate the dose to the core and to the penumbra. It's a function of the Barkas effective charge and of the β_{Ion} (beta).

The value is evaluated by means of the following relation:

$$k_p = 1.25 \cdot 10^{-4} \cdot \left(\frac{Z_{eff}}{\beta_{Ion}}\right)^2$$

It is instantiated in the Constructor.

See also

Track_KieferChatterjee(const Particle&, const double, double)

Definition at line 344 of file Track_KieferChatterjee.h.

7.17.4.8 double Survival::Track_KieferChatterjee::let [private]

The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

See also

Particle::let

Definition at line 312 of file Track_KieferChatterjee.h.

7.17.4.9 double Survival::Track_KieferChatterjee::particleEnergy [private]

The specific energy of the particle generating the track, expressed in MeV/u.

Definition at line 250 of file Track_KieferChatterjee.h.

7.17.4.10 std::string Survival::Track_KieferChatterjee::particleType [private]

The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

See also

Particle::type

Definition at line 238 of file Track_KieferChatterjee.h.

7.17.4.11 double Survival::Track_KieferChatterjee::r_core [private]

The core radius of the track expressed in μm.

It is initialized by means of the following relation:

$$R_{core} = \eta \cdot \beta_{Ion}$$

Where $\eta = 0.0116\,\mu m$ is a constant value and β_{Ion} represents the ratio between the speed of the ion and the speed of light.

It is instantiated in the Constructor.

See also

Track_KieferChatterjee(const Particle&, const double, double)

Definition at line 264 of file Track_KieferChatterjee.h.

7.17.4.12 double Survival::Track_KieferChatterjee::r_penumbra [private]

The penumbra radius of the track expressed in μm.

It is initialized by means of the following relation:

$$R_{penumbra} = \gamma \left(\frac{E}{A}\right)^{\delta}$$

Where $\gamma = 0.0616 \frac{\mu m}{MeV^{\delta}}$ and $\delta = 1.7$ are constant values, E represents the energy of the ion and A his mass number (Particle::A).

It is instantiated in the Constructor.

See also

Track_KieferChatterjee(const Particle&, const double, double)

Definition at line 278 of file Track_KieferChatterjee.h.

7.17.4.13 double Survival::Track_KieferChatterjee::time [private]

The time associated to a particular event, expressed in hours.

The *time* isn't really a variable associated to the track, time flows during the irradiation process. For this reason, the initial value of this member represent the instant in which the track is generated, but then it will change during the execution assuming a value representing the interaction time between the track and a specific Nucleus.

Definition at line 368 of file Track_KieferChatterjee.h.

7.17.4.14 double Survival::Track_KieferChatterjee::weight [private]

The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

See also

Particle::weight

Definition at line 362 of file Track_KieferChatterjee.h.

7.17.4.15 double Survival::Track_KieferChatterjee::x_track [private]

The track position (x coordinate) referred to the beam axis, expressed in mm.

See also

Particle::x

Definition at line 350 of file Track_KieferChatterjee.h.

7.17.4.16 double Survival::Track_KieferChatterjee::y_track [private]

The track position (y coordinate) referred to the beam axis, expressed in mm.

See also

Particle::y

Definition at line 356 of file Track_KieferChatterjee.h.

7.17.4.17 double Survival::Track_KieferChatterjee::z_eff [private]

Barkas effective charge.

The Barkas effective charge evaluated by means of the following relation:

$$Z_{eff} = Z \cdot \left(1 - \exp\left(-\frac{125\beta}{Z^{2/3}}\right)\right)$$

Where Z represents the particle charge (Particle::charge) and beta the ratio between the speed of the ion and the speed of light.

It is instantiated in the Constructor.

See also

Track_KieferChatterjee(const Particle&, const double, double)

Definition at line 332 of file Track_KieferChatterjee.h.

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

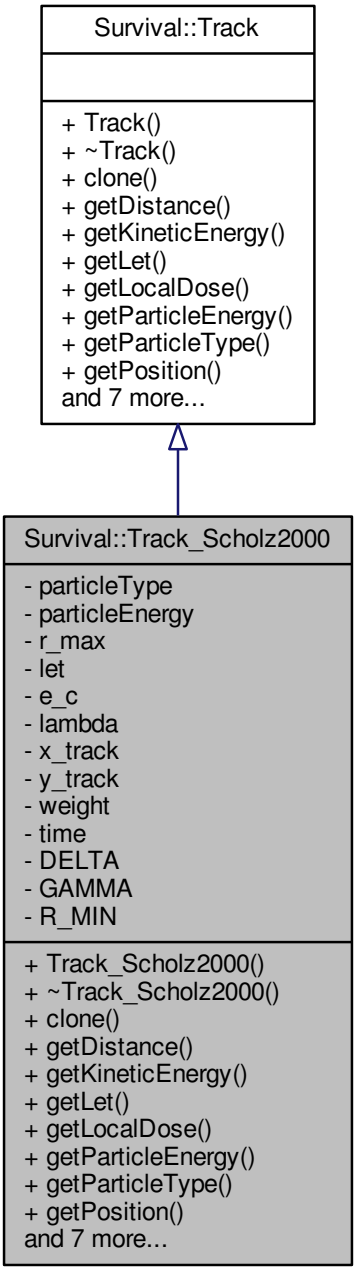
- include/Track_KieferChatterjee.h
- src/Track_KieferChatterjee.cpp

7.18 Survival::Track_Scholz2000 Class Reference

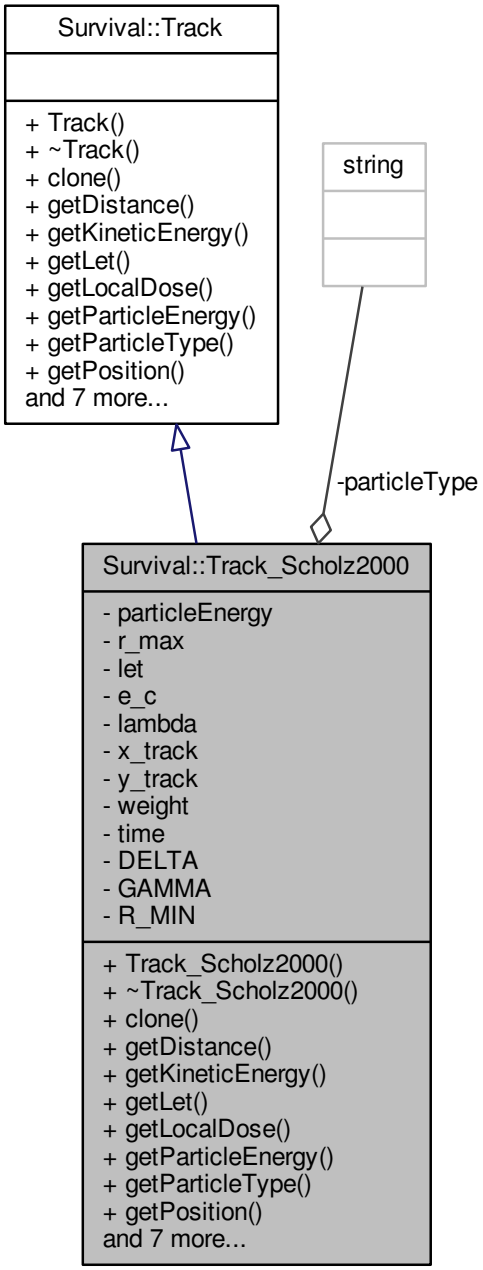
Inherited from the Track class, it implements the LEM I track model.

```
#include <Track_Scholz2000.h>
```

Inheritance diagram for Survival::Track_Scholz2000:



Collaboration diagram for Survival::Track_Scholz2000:



Public Member Functions

- Track_Scholz2000 (const Particle &particle, const double density, double t=0.0)

  Constructor. Instantiates and sets the object.
- virtual ~Track_Scholz2000 ()

  Destructor.
- virtual Track_Scholz2000 * clone () const

Returns a pointer to a new Track_Scholz2000 object created as a copy of an existent one by means of the copy constructor.

- virtual double getDistance (const double localDose) const

  Returns the distance from the center of the track, knowing the local dose deposited.
- virtual double getKineticEnergy () const

  Returns the kinetic energy of the particle generating the track expressed in MeV.
- virtual double getLet () const

  Returns the LET in water of the particle generating the track expressed in MeV/um.
- virtual double getLocalDose (const double distance) const

  Returns the local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).
- virtual double getParticleEnergy () const

  Returns the specific energy of the particle generating the track, expressed in MeV/u.
- virtual std::string getParticleType () const

  Returns the type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).
- virtual void getPosition (double &returnX, double &returnY) const

  Returns the track position (x and y coordinates) referred to the beam axis and expressed in mm overwriting two `double` *variables passed by reference.*
- virtual double getRadialIntegral (const double r_begin, const double r_end) const

  Returns the dose (in Gy) deposited by the track evaluated as the radial integral of the track profile between r_{begin} and r_{end} (expressed in um).
- virtual double getRadius () const

  Returns the radius of the track expressed in um.
- virtual double getTime () const

  Returns the time associated to a particular event expressed in hours.
- virtual double getWeight () const

  Returns the weight in the beam of the particle generating the track. Useful in the case of "mixed fields".
- virtual std::string saveTrack () const

  Saves the local dose deposited by the track at different distances from its center to reconstruct the track profile.
- virtual void setPosition (const double x, const double y)

  Sets the track position (x and y coordinates) referred to the beam axis.
- virtual void setTime (double t)

  Sets the time associated to a particular event.

Private Attributes

- std::string particleType

  The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...)
- double particleEnergy

  The specific energy of the particle generating the track, expressed in MeV/u.
- double r_max

  The radius of the track expressed in um.
- double let

  The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).
- double e_c

  The kinetic energy of the particle generating the track expressed in MeV.
- double lambda

  A constant value required to evaluate r_max.
- double x_track

  The track position (x coordinate) referred to the beam axis, expressed in mm.
- double y_track

  The track position (y coordinate) referred to the beam axis, expressed in mm.

- double weight

  The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".
- double time

  The time associated to a particular event, expressed in hours.

Static Private Attributes

- static const double DELTA = 1.7

  Useful constant value.
- static const double GAMMA = 0.062

  Useful constant value.
- static const double R_MIN = 0.01

  The core radius expressed in um.

7.18.1 Detailed Description

Inherited from the Track class, it implements the LEM I track model.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2007

The structure of the track is defined in a publication by Scholz and Kraft (1) but then (minimally) modified.

According to this parametrization, the average local dose deposition pattern (amorphous track) $d(\rho)$ is divided in a *core*, with a radius $\rho_{min} = 0.01\ um$, where the local dose is constantly equal to

$$d(\rho) = \lambda \frac{LET}{\rho_{min}^2} \qquad \rho < \rho_{min}$$

and an outer region which radius is indicated with ρ_{max} and coincides with the radius of the track where the local dose is defined by

$$d(\rho) = \lambda \frac{LET}{\rho^2} \qquad \rho_{min} < \rho < \rho_{max}$$

while the local dose is equal to zero if the distance is greater than ρ_{max}. λ is a parameter defined as:

$$\lambda = \frac{1}{\rho \pi (1 + \ln(\rho_{max}^2 / \rho_{min}^2))}$$

where ρ in this formula represents the density of the medium.

The radius ρ_{max} of the track is evaluated as a function of the specific energy of the ion, as:

$$\rho_{max} = \gamma\, E^{\delta}$$

where $\gamma = 0.062\ \frac{\mu m}{MeV^{\delta}}$ and $\delta = 1.7$ are constant values.

This class provides some methods to evaluate and get the local dose deposited by the ion along the track and the radial integral of the track.

1. M. Scholz and G. Kraft, "Track structure and the calculation of biological effects of heavy charged particles", *Advances in Space Research* **18**, 5-14 (1996)

See also

Track_Elsasser2007 and Track_Elsasser2008

Definition at line 45 of file Track_Scholz2000.h.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 Track_Scholz2000::Track_Scholz2000 (const Particle & *particle*, const double *density*, double *t* = 0.0)

Constructor. Instantiates and sets the object.

Converts a particle (object of class Particle), passed by reference, in a track according to the amorphous track model defined by Scholz *et al.*

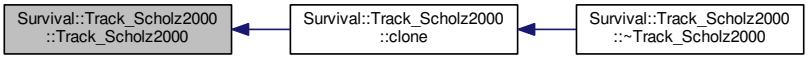
Parameters

particle	The particle generating the track in the medium, passed by reference.
density	The density of the medium expressed in $\frac{g}{cm^3}$. The default value is the density of water.
t	The time corresponding to the generation of the track in the target. The default value is 0. (See also the documentation of the data member time)

Also r_max and lambda are instantiated getting information from particle.

Definition at line 36 of file Track_Scholz2000.cpp.

Here is the caller graph for this function:

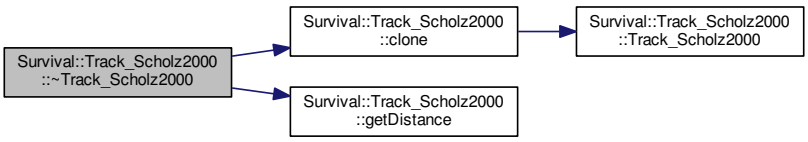


7.18.2.2 virtual Survival::Track_Scholz2000::~Track_Scholz2000 ( ) [inline], [virtual]

Destructor.

Definition at line 64 of file Track_Scholz2000.h.

Here is the call graph for this function:



7.18.3 Member Function Documentation

7.18.3.1 Track_Scholz2000 * Track_Scholz2000::clone ( ) const [virtual]

Returns a pointer to a new Track_Scholz2000 object created as a copy of an existent one by means of the copy constructor.

Warning

It dynamically allocates memory to be deleted (somewhere) by the user.

Implements Survival::Track.

Definition at line 64 of file Track_Scholz2000.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.18.3.2 double Track_Scholz2000::getDistance (const double *localDose*) const [virtual]

Returns the distance from the center of the track, knowing the local dose deposited.

It's the inverse function of getLocalDose(). Since the maximum possible local dose deposited is: $d_{MAX} = \lambda \frac{LET}{\rho^2_{min}}$, if d is greater than d_{MAX} it returns -1 (nonsense value). Else it returns:

$$\sqrt{\frac{\lambda\, LET}{d}}$$

where d is the local dose.

Note

If $d < d_{MIN} = \lambda \frac{LET}{\rho^2_{max}}$ it returns r_max.

Parameters

localDose	The local dose deposited expressed in Gy.

Returns

The distance from the center of the track, expressed in um.

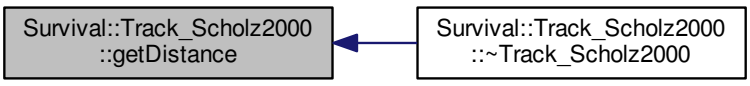
See also

getLocalDose()

Implements Survival::Track.

Definition at line 71 of file Track_Scholz2000.cpp.

Here is the caller graph for this function:



7.18.3.3 virtual double Survival::Track_Scholz2000::getKineticEnergy ( ) const [inline],[virtual]

Returns the kinetic energy of the particle generating the track expressed in MeV.

Returns

The kinetic energy of the particle generating the track expressed in MeV.

See also

e_c

Implements Survival::Track.

Definition at line 96 of file Track_Scholz2000.h.

7.18.3.4 virtual double Survival::Track_Scholz2000::getLet ( ) const [inline], [virtual]

Returns the LET in water of the particle generating the track expressed in MeV/um.

Returns

The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

See also

let

Implements Survival::Track.

Definition at line 104 of file Track_Scholz2000.h.

Here is the call graph for this function:

7.18.3.5 double Track_Scholz2000::getLocalDose (const double *distance*) const [virtual]

Returns the local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

According to the parametrization by Scholz *et al.*, the function evaluates the local dose evaluating some possible cases:

- If the distance ρ is smaller than the core radius R_MIN it returns $d(\rho) = \lambda \frac{LET}{R\_MIN^2}$
- If the distance ρ is greater than the radius of the track r_max it returns 0
- Else it returns $d(\rho) = \lambda \frac{LET}{\rho^2}$

Parameters

distance	The distance from the track center expressed in um.

Returns

The local dose (in Gy) deposited by the track at a certain distance from the track center (expressed in um).

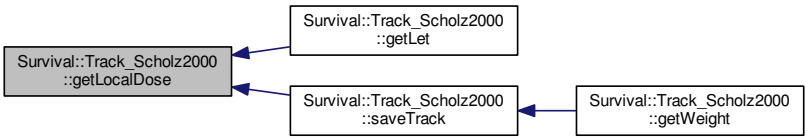
See also

DELTA, GAMMA and lambda

Implements Survival::Track.

Definition at line 83 of file Track_Scholz2000.cpp.

Here is the caller graph for this function:



7.18.3.6 virtual double Survival::Track_Scholz2000::getParticleEnergy ( ) const [inline], [virtual]

Returns the specific energy of the particle generating the track, expressed in MeV/u.

Returns

The specific energy of the particle generating the track, expressed in MeV/u.

See also

particleEnergy

Implements Survival::Track.

Definition at line 127 of file Track_Scholz2000.h.

7.18.3.7 virtual std::string Survival::Track_Scholz2000::getParticleType ( ) const [inline], [virtual]

Returns the type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

Returns

The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...).

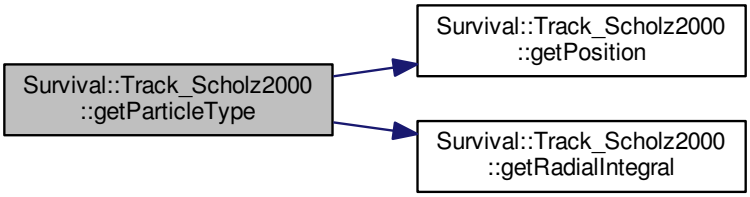
See also

Particle::type

Implements Survival::Track.

Definition at line 135 of file Track_Scholz2000.h.

Here is the call graph for this function:



7.18.3.8 void Track_Scholz2000::getPosition (double & *returnX*, double & *returnY*) const [virtual]

Returns the track position (`x` and `y` coordinates) referred to the beam axis and expressed in mm overwriting two `double` variables passed by reference.

This is an unusual getter which needs two `double` variables, passed by reference, that will be overwritten with the `x` and `y` coordinates of the track referred to the beam axis, expressed in mm.

Parameters

returnX	The variable to be overwritten with the `x` coordinate of the track, expressed in mm, passed by reference to be overwritten.
returnY	The variable to be overwritten with the `x` coordinate of the track, expressed in mm, passed by reference to be overwritten.

See also

setPosition()

Implements Survival::Track.

Definition at line 98 of file Track_Scholz2000.cpp.

Here is the caller graph for this function:

7.18.3.9 double Track_Scholz2000::getRadialIntegral (const double *r_begin*, const double *r_end*) const [virtual]

Returns the dose (in Gy) deposited by the track evaluated as the radial integral of the track profile between r_{begin} and r_{end} (expressed in um).

The local dose varies along the track profile according to the structure defined by Scholz *et al.* (see Track_↩ Scholz2000). This function evaluates the integral of the radial profile in $[r_{begin};\ r_{end}]$.

Parameters

r_begin	Lower limit of integration, expressed in um.
r_end	Upper limit of integration, expressed in um.

Returns

The dose (in Gy) deposited by the track evaluated as the radial integral of the track profile between r_{begin} and r_{end}.

Warning

The execution of the program will be terminated if incorrect limits of integration are chosen, that is:

- If $r_{begin} < 0$
- If $r_{end} < r_{begin}$

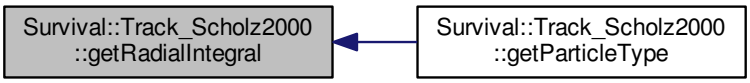
See also

getLocalDose()

Implements Survival::Track.

Definition at line 107 of file Track_Scholz2000.cpp.

Here is the caller graph for this function:



7.18.3.10 virtual double Survival::Track_Scholz2000::getRadius ( ) const [inline], [virtual]

Returns the radius of the track expressed in um.

Returns

The radius of the track expressed in um.

See also

R_MIN

Implements Survival::Track.

Definition at line 173 of file Track_Scholz2000.h.

7.18.3.11 virtual double Survival::Track_Scholz2000::getTime ( ) const [inline], [virtual]

Returns the time associated to a particular event expressed in hours.

Returns

The time associated to a particular event expressed in hours.

See also

time and setTime()

Implements Survival::Track.

Definition at line 181 of file Track_Scholz2000.h.

7.18.3.12 virtual double Survival::Track_Scholz2000::getWeight ( ) const [inline], [virtual]

Returns the weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

Returns

The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

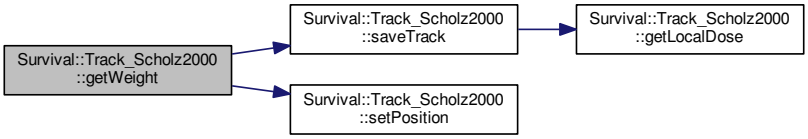
See also

Particle::weight

Implements Survival::Track.

Definition at line 189 of file Track_Scholz2000.h.

Here is the call graph for this function:



7.18.3.13 string Track_Scholz2000::saveTrack ( ) const [virtual]

Saves the local dose deposited by the track at different distances from its center to reconstruct the track profile.

The function divides the track radius (r_max) into 300 logarithmically spaced distances from the track center, for each of these distances it evaluates the local dose deposited calling the function getLocalDose(); during the process it saves each distance and the corresponding dose in a new file. This allow to reconstruct the track profile.

Returns

The name of the file created.

See also

getLocalDose()

Implements Survival::Track.

Definition at line 139 of file Track_Scholz2000.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.18.3.14 void Track_Scholz2000::setPosition (const double *x*, const double *y*) [virtual]

Sets the track position (x and y coordinates) referred to the beam axis.

Parameters

x	The x coordinate of the track to be set, referred to the beam axis and expressed in mm.
y	The x coordinate of the track to be set, referred to the beam axis and expressed in mm.

See also

x_track, y_track and getPosition()

Implements Survival::Track.

Definition at line 161 of file Track_Scholz2000.cpp.

Here is the caller graph for this function:

7.18.3.15 virtual void Survival::Track_Scholz2000::setTime (double *t*) [inline], [virtual]

Sets the time associated to a particular event.

Parameters

t	The time to be set expressed in hours.

See also

time

Implements Survival::Track.

Definition at line 217 of file Track_Scholz2000.h.

7.18.4 Member Data Documentation

7.18.4.1 const double Track_Scholz2000::DELTA = 1.7 [static], [private]

Useful constant value.

It's necessary to evaluate r_max; it's equal to 1.7, according to the LEM I parametrization.

Definition at line 294 of file Track_Scholz2000.h.

7.18.4.2 double Survival::Track_Scholz2000::e_c [private]

The kinetic energy of the particle generating the track expressed in MeV.

See also

Particle::e_c

Definition at line 250 of file Track_Scholz2000.h.

7.18.4.3 const double Track_Scholz2000::GAMMA = 0.062 [static], [private]

Useful constant value.

It's necessary to evaluate r_max; it's equal to 0.062 $\frac{\mu m}{MeV^{\delta}}$, according to the LEM I parametrization.

Definition at line 300 of file Track_Scholz2000.h.

7.18.4.4 double Survival::Track_Scholz2000::lambda [private]

A constant value required to evaluate r_max.

It's defined as:

$$\lambda = \frac{1}{\rho\pi(1 + \ln(\rho_{max}^2/\rho_{min}^2))}$$

where ρ represents the density of the medium while ρ_{max} and ρ_{min} represent r_max and R_MIN respectively.

It's expressed in $\frac{Gy\,\mu m^3}{MeV}$.

Definition at line 262 of file Track_Scholz2000.h.

7.18.4.5 double Survival::Track_Scholz2000::let [private]

The LET in water of the particle generating the track expressed in MeV/um (according to the Bethe-Bloch formula).

See also

Particle::let

Definition at line 244 of file Track_Scholz2000.h.

7.18.4.6 double Survival::Track_Scholz2000::particleEnergy [private]

The specific energy of the particle generating the track, expressed in MeV/u.

Definition at line 228 of file Track_Scholz2000.h.

7.18.4.7 std::string Survival::Track_Scholz2000::particleType [private]

The type of particle generating the track (e.g. Chemical symbol for ions: H, He, Li, ...)

See also

Particle::type

Definition at line 217 of file Track_Scholz2000.h.

7.18.4.8 double Survival::Track_Scholz2000::r_max [private]

The radius of the track expressed in um.

According to the parametrization by Scholz and Kraft it's evaluated (and instantiated in the constructor) as:

$$r\_max = \gamma \, E^{\delta}$$

where γ is GAMMA, δ is DELTA and *E* represents the specific energy of the ion.

Definition at line 238 of file Track_Scholz2000.h.

7.18.4.9 const double Track_Scholz2000::R_MIN = 0.01 [static], [private]

The core radius expressed in um.

It's taken equal to 0.01 um, according to the LEM I parametrization.

Definition at line 306 of file Track_Scholz2000.h.

7.18.4.10 double Survival::Track_Scholz2000::time [private]

The time associated to a particular event, expressed in hours.

The *time* isn't really a variable associated to the track, time flows during the irradiation process. For this reason, the initial value of this member represent the instant in which the track is generated, but then it will change during the execution assuming a value representing the interaction time between the track and a specific Nucleus.

Note

Since this track structure is used in the LEM model, that doesn't take into account (yet) the time structure of the irradiation, this data member is actually useless.

Definition at line 288 of file Track_Scholz2000.h.

7.18.4.11 double Survival::Track_Scholz2000::weight [private]

The weight in the beam of the particle generating the track. Useful in the case of "mixed fields".

See also

Particle::weight

Definition at line 280 of file Track_Scholz2000.h.

7.18.4.12 double Survival::Track_Scholz2000::x_track [private]

The track position (x coordinate) referred to the beam axis, expressed in mm.

See also

Particle::x

Definition at line 268 of file Track_Scholz2000.h.

7.18.4.13 double Survival::Track_Scholz2000::y_track [private]

The track position (y coordinate) referred to the beam axis, expressed in mm.

See also

Particle::y

Definition at line 274 of file Track_Scholz2000.h.

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

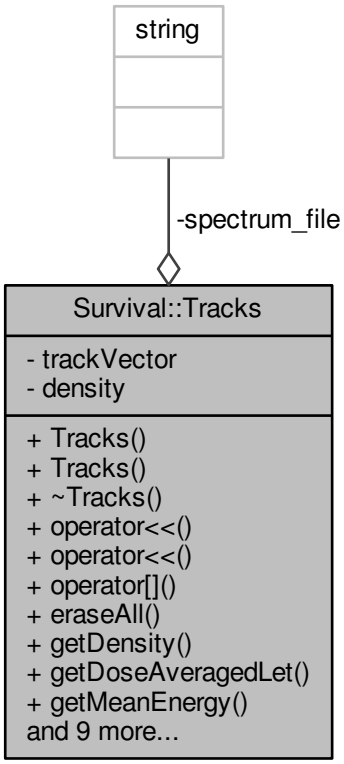
- include/Track_Scholz2000.h
- src/Track_Scholz2000.cpp

7.19 Survival::Tracks Class Reference

This is a container class for Track objects; it implements the structure and methods to manage a vector of tracks.

`#include <Tracks.h>`

Collaboration diagram for Survival::Tracks:



Public Member Functions

- Tracks (const Particles &particles, const std::string trackType, const double massDensity=1.0)

  Constructor. Instantiates and sets the object.
- Tracks (const int numberOfTracks=0, const double massDensity=1.0)

  Instantiates and sets the object. Overload of the constructor.
- ~Tracks ()

  Destructor.
- void operator<< (const Track &track)

  Overload of the << operator to add a new track at the end of the vector.
- void operator<< (const Tracks &tracks)

  Overload of the << operator to add a vector of tracks at the end of the vector.
- const Track & operator[ ] (const int index) const

  Overload of the [] operator to access at the n-th element of the vector.

- void eraseAll ()

  Deletes each element of the vector and erases the vector itself.
- double getDensity () const

  Returns the density of the medium in $\frac{g}{cm^3}$.
- double getDoseAveragedLet () const

  Evaluates and returns the dose averaged LET of the vector of tracks, expressed in keV/um.
- double getMeanEnergy () const

  Evaluates and returns the mean energy of the vector of tracks, expressed in MeV.
- double getMeanLet () const

  Evaluates and returns the mean LET of the vector of tracks, expressed in keV/um.
- double getSigmaDoseAveragedLet () const

  Evaluates and returns the standard deviation of the dose averaged LET of the vector of tracks, expressed in keV/um.
- double getSigmaMeanEnergy () const

  Evaluate and returns standard deviation of the mean energy of the vector of tracks, expressed in MeV.
- double getSigmaMeanLet () const

  Evaluate and returns standard deviation of the mean LET of the vector of tracks, expressed in keV/um.
- std::string getSpectrumFile () const

  Returns a `string` *identifying the file containing the spectrum of particles to be converted in tracks.*
- double getTotalWeight () const

  Returns the total weight of the group of particles generating the vector of tracks.
- bool isMonoenergetic () const

  Check if the vector is monoenergetic.
- void setDensity (const double d)

  Sets the density of the medium.
- int size () const

  Returns the size of the vector of tracks.

Private Attributes

- std::vector< Track * > trackVector

  A dynamic vector of Track pointers.
- double density

  The density of the medium expressed in $\frac{g}{cm^3}$.
- std::string spectrum_file

  A `string` *identifying the file containing the spectrum of particle to be converted in tracks.*

7.19.1 Detailed Description

This is a container class for Track objects; it implements the structure and methods to manage a vector of tracks.

Author

Andrea Attili
Lorenzo Manganaro
Germano Russo

Date

2008

This is a container class for Track objects; it implements the structure and methods to manage a vector of tracks. It can be created directly from a Particles object, specifying a unique Track type, or it can be loaded with single Track objects, making use of polymorphism.

See also

Track and Particles

Definition at line 23 of file Tracks.h.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 Survival::Tracks::Tracks (const Particles & *particles*, const std::string *trackType*, const double *massDensity* = 1.0)

Constructor. Instantiates and sets the object.

Converts a vector of particles (object of class Particles), passed by reference, in a vector of tracks, depending on the parametrization chosen for the Track model. The possible choices for the parametrization are:

- Scholz2000
- Elsasser2007
- Elsasser2008
- KieferChatterjee

Warning

The execution of the program will be terminated if an inexistent parametrization is chosen.

Parameters

particles	The reference to a Particles object, that is the vector of particles generating the tracks.
trackType	The parametrization chosen for the Track model.
massDensity	The density of the medium expressed in $\frac{g}{cm^3}$. The default value is the density of water.

See also

Tracks(const int, const double), Track_Scholz2000, Track_Elsasser2007, Track_Elsasser2008 and Track_↩ KieferChatterjee

7.19.2.2 Tracks::Tracks (const int *numberOfTracks* = 0, const double *massDensity* = 1.0)

Instantiates and sets the object. Overload of the constructor.

Parameters

numberOfTracks	The number of tracks to be stored in the vector
massDensity	The density of the medium expressed in $\frac{g}{cm^3}$. The default value is the density of water.

See also

Tracks(const int, const double)

Definition at line 84 of file Tracks.cpp.

7.19.2.3 Tracks::~Tracks ( )

Destructor.

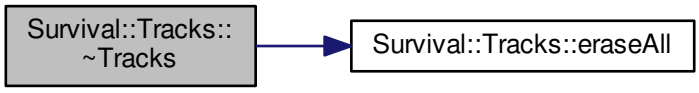
The destructor calls the function eraseAll() that deletes each element of the vector and the vector itself.

See also

eraseAll()

Definition at line 94 of file Tracks.cpp.

Here is the call graph for this function:



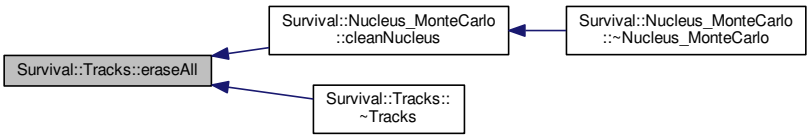
7.19.3 Member Function Documentation

7.19.3.1 void Tracks::eraseAll ( )

Deletes each element of the vector and erases the vector itself.

Definition at line 123 of file Tracks.cpp.

Here is the caller graph for this function:



7.19.3.2 double Survival::Tracks::getDensity ( ) const [inline]

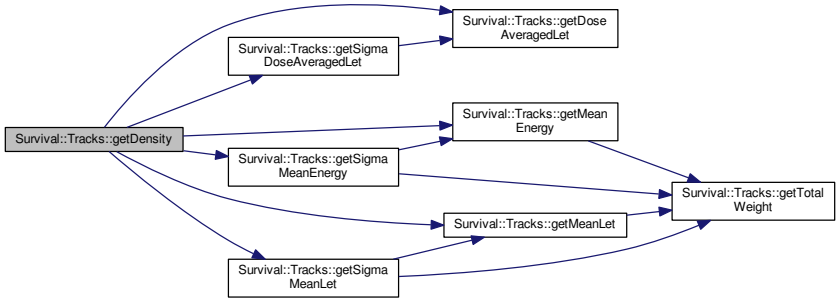
Returns the density of the medium in $\frac{g}{cm^3}$.

Returns

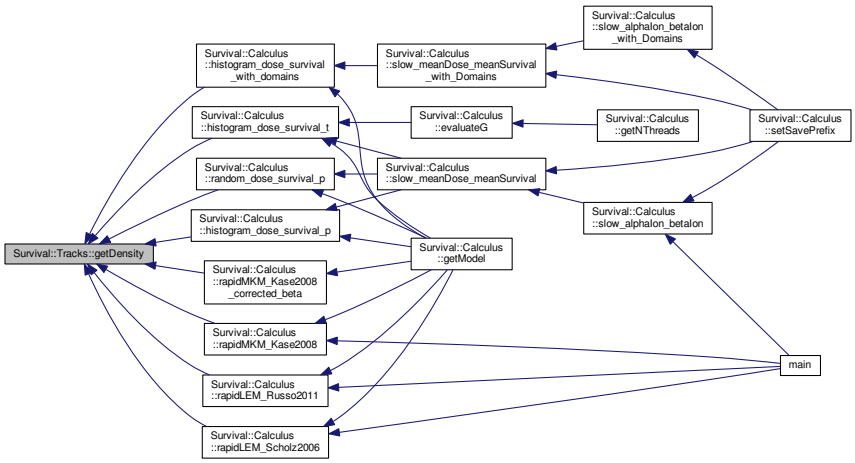
The density of the medium in $\frac{g}{cm^3}$.

Definition at line 93 of file Tracks.h.

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.3.3 double Tracks::getDoseAveragedLet ( ) const

Evaluates and returns the dose averaged LET of the vector of tracks, expressed in keV/um.

It's evaluated as:

$$LET_d = \frac{\sum_i LET_i^2 \, w_i}{\sum_i LET_i \, w_i}$$

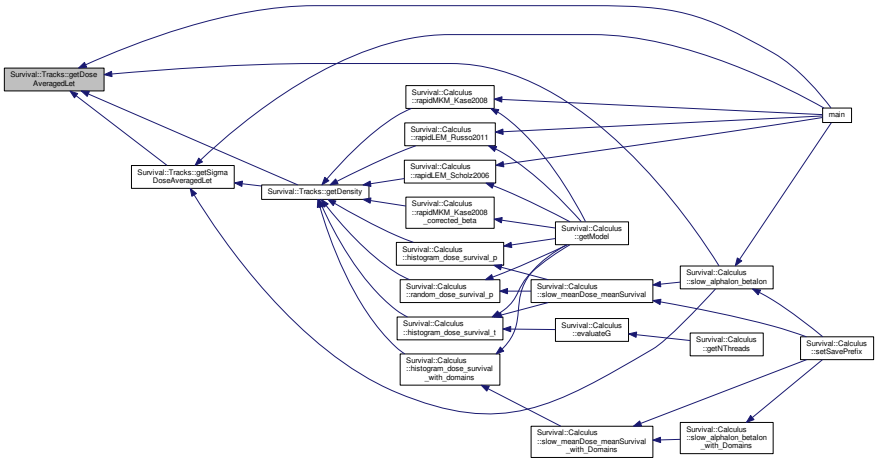
where w_i indicates the weight of the i-th track in the vector.

Returns

The dose averaged LET of the vector of tracks (trackVector), expressed in keV/um.

Definition at line 133 of file Tracks.cpp.

Here is the caller graph for this function:



7.19.3.4 double Tracks::getMeanEnergy ( ) const

Evaluates and returns the mean energy of the vector of tracks, expressed in MeV.

It's evaluated as:

$$\langle E \rangle = \frac{\sum_i E_i \, w_i}{\sum_i w_i}$$

where w_i indicates the weight of the i-th track in the vector.

Returns

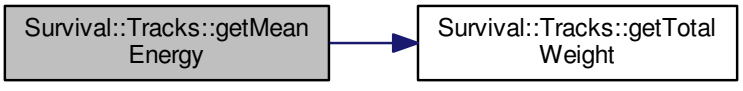
The mean energy of the vector of tracks (trackVector), expressed in MeV.

See also

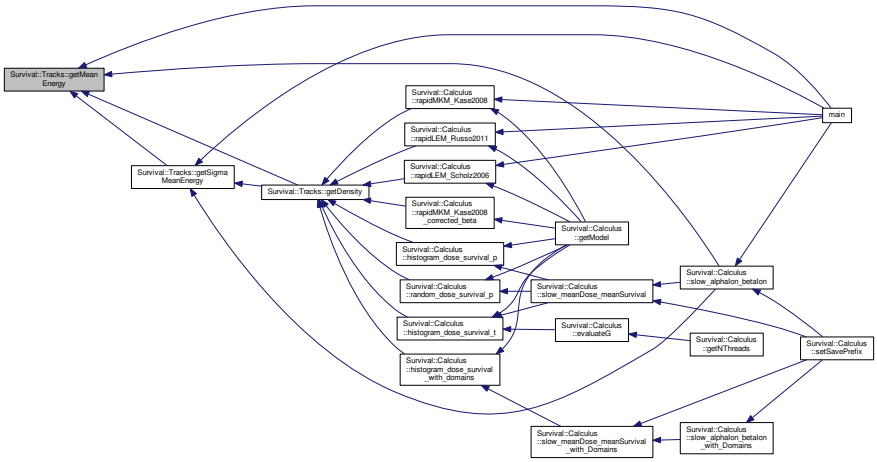
Track::getKineticEnergy()

Definition at line 149 of file Tracks.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.3.5 double Tracks::getMeanLet ( ) const

Evaluates and returns the mean LET of the vector of tracks, expressed in keV/um.

It's evaluated as:

$$\langle LET \rangle = \frac{\sum_i LET_i \, w_i}{\sum_i w_i}$$

where w_i indicates the weight of the i-th track in the vector.

Returns

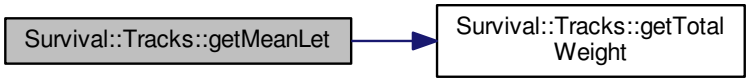
The mean LET of the vector of tracks (trackVector), expressed in keV/um.

See also

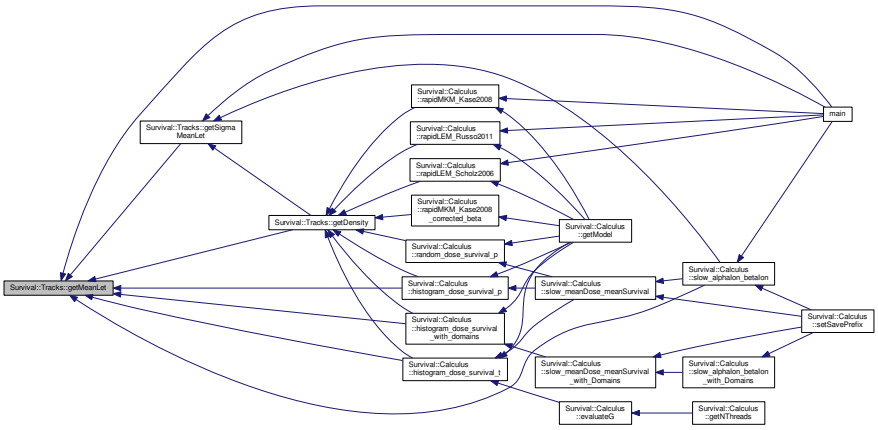
Particles::getMeanLet().

Definition at line 159 of file Tracks.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.3.6 double Tracks::getSigmaDoseAveragedLet ( ) const

Evaluates and returns the standard deviation of the dose averaged LET of the vector of tracks, expressed in keV/um.

It's evaluated as:

$$\sigma_{LET_d} = \left(\frac{\sum_i (LET_i - LET_d)^2 \, LET_i \, w_i}{\sum_i LET_i \, w_i} \right)^{1/2}$$

where w_i indicates the weight of the i-th track in the vector and LET_d the dose averaged LET.

Returns

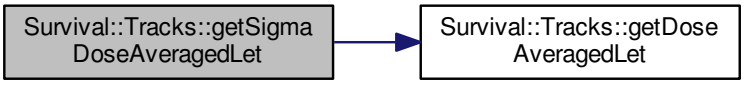
The standard deviation of the dose averaged LET of the vector of tracks (trackVector), expressed in keV/um.

See also

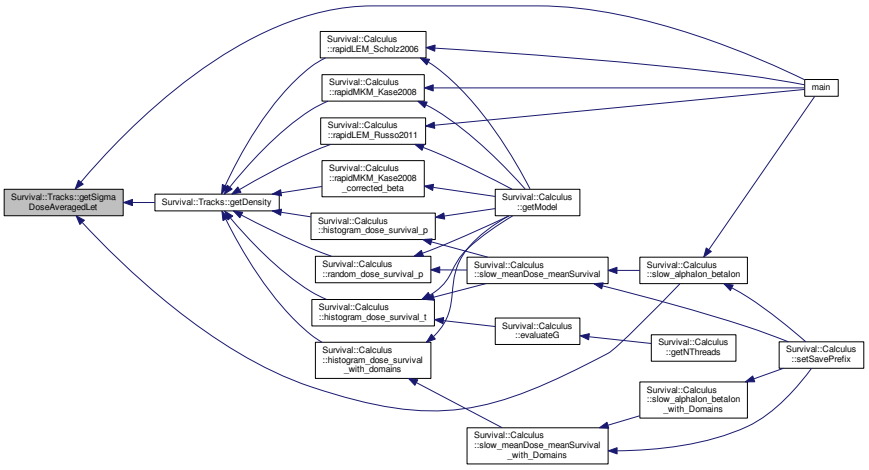
getDoseAveragedLet()

Definition at line 171 of file Tracks.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.3.7 double Tracks::getSigmaMeanEnergy ( ) const

Evaluate and returns standard deviation of the mean energy of the vector of tracks, expressed in MeV.

It's evaluated as:

$$\sigma_{\langle E\rangle} = \left(\frac{\sum_i (E_i - \langle E\rangle)^2 \, w_i}{\sum_i \, w_i} \right)^2$$

where w_i indicates the weight of the i-th track in the vector and $\langle E\rangle$ the mean energy.

Returns

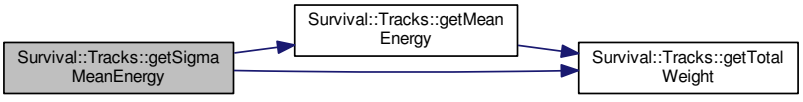
The standard deviation of the mean energy of the vector of tracks (trackVector), expressed in MeV.

See also

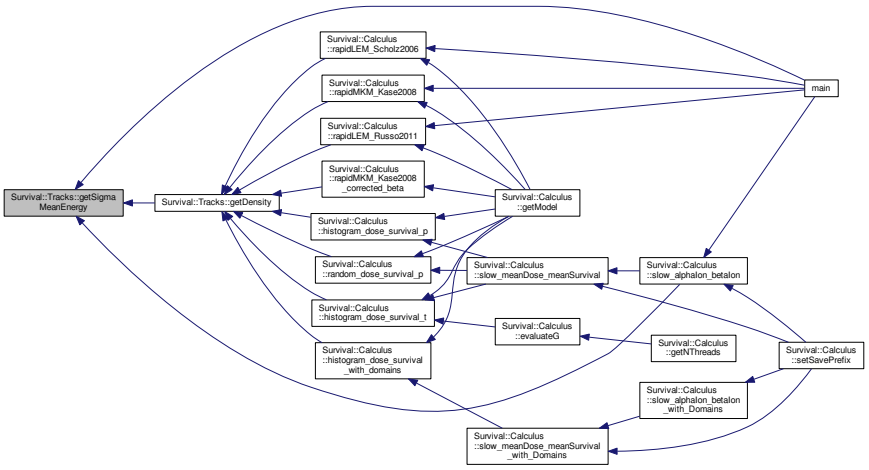
getMeanEnergy()

Definition at line 188 of file Tracks.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.3.8 double Tracks::getSigmaMeanLet ( ) const

Evaluate and returns standard deviation of the mean LET of the vector of tracks, expressed in keV/um.

It's evaluated as:

$$\sigma_{\langle LET \rangle} = \left(\frac{\sum_i (LET_i - \langle LET \rangle)^2 \, w_i}{\sum_i \, w_i} \right)^2$$

where w_i indicates the weight of the i-th track in the vector and $\langle LET \rangle$ the mean LET.

Returns

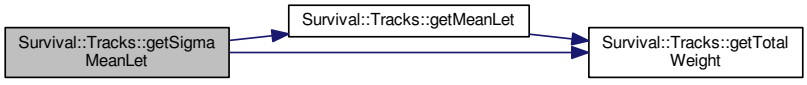
The mean LET of the vector of tracks (trackVector), expressed in keV/um.

See also

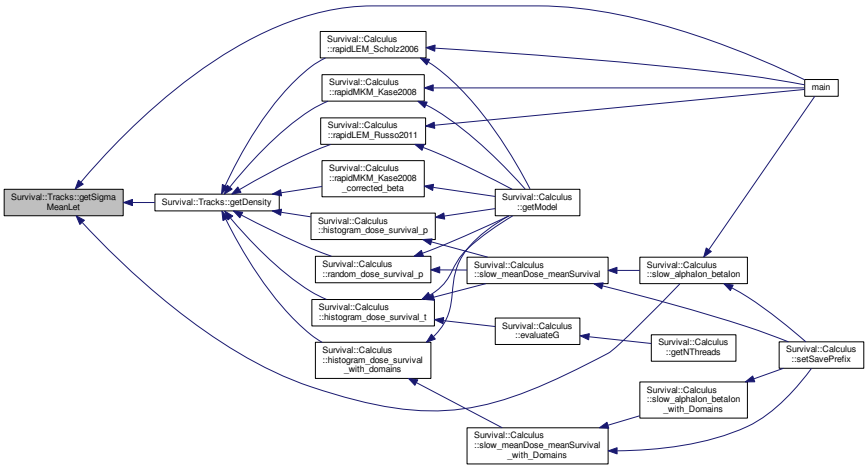
Particles::getMeanLet

Definition at line 203 of file Tracks.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.3.9 std::string Survival::Tracks::getSpectrumFile ( ) const [inline]

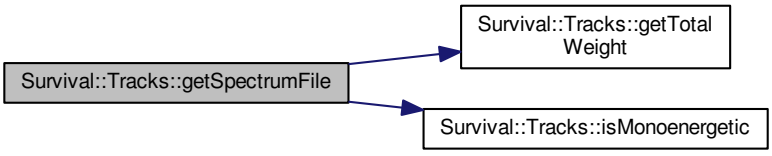
Returns a `string` identifying the file containing the spectrum of particles to be converted in tracks.

Returns

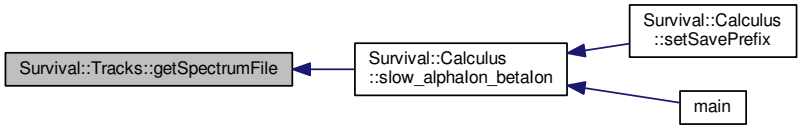
A `string` identifying the file containing the spectrum of particles to be converted in tracks.

Definition at line 181 of file Tracks.h.

Here is the call graph for this function:



Here is the caller graph for this function:



7.19.3.10 double Tracks::getTotalWeight ( ) const

Returns the total weight of the group of particles generating the vector of tracks.

Returns

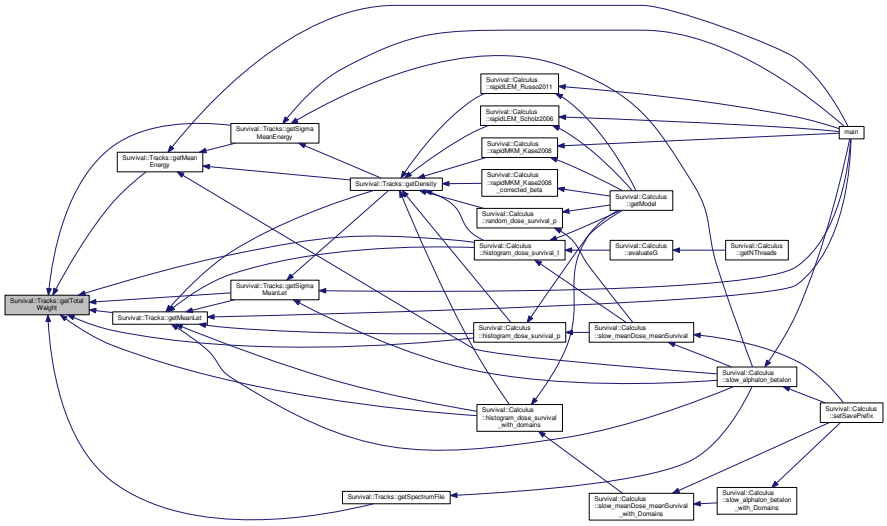
The total weight of the group of particles generating the vector of tracks, evaluated as the sum of the weight of each particle.

See also

Particles::getTotalWeight

Definition at line 218 of file Tracks.cpp.

Here is the caller graph for this function:



7.19.3.11 bool Tracks::isMonoenergetic ( ) const

Check if the vector is monoenergetic.

Returns

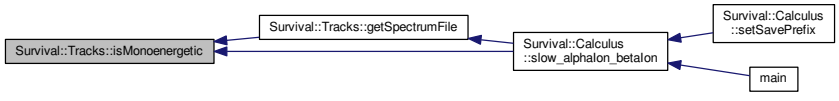
A boolean value identifying if trackVector is monoenergetic, that is $size = 1$.

See also

size()

Definition at line 230 of file Tracks.cpp.

Here is the caller graph for this function:



7.19.3.12 void Tracks::operator<< (const Track & *track*)

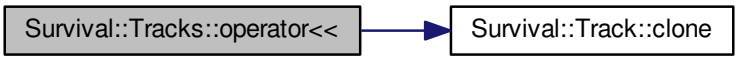
Overload of the << operator to add a new track at the end of the vector.

Parameters

track	The track to be added.

Definition at line 101 of file Tracks.cpp.

Here is the call graph for this function:



7.19.3.13 void Tracks::operator<< (const Tracks & *tracks*)

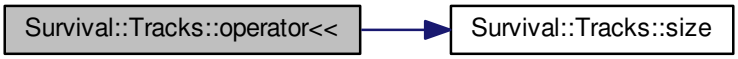
Overload of the << operator to add a vector of tracks at the end of the vector.

Parameters

tracks	The object containing the vector of tracks to be added.

Definition at line 108 of file Tracks.cpp.

Here is the call graph for this function:



7.19.3.14 const Track & Tracks::operator[ ] (const int *index*) const

Overload of the [] operator to access at the n-th element of the vector.

Parameters

index	The position of the element in the vector.

Returns

A `const` reference to the element at the specified position in the vector.

Definition at line 116 of file Tracks.cpp.

7.19.3.15 void Survival::Tracks::setDensity (const double *d*) [inline]

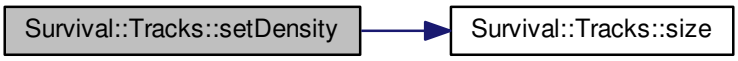
Sets the density of the medium.

Parameters

d	The chosen value of density of the medium to be set, expressed in $\frac{g}{cm^3}$

Definition at line 203 of file Tracks.h.

Here is the call graph for this function:



7.19.3.16 int Tracks::size ( ) const

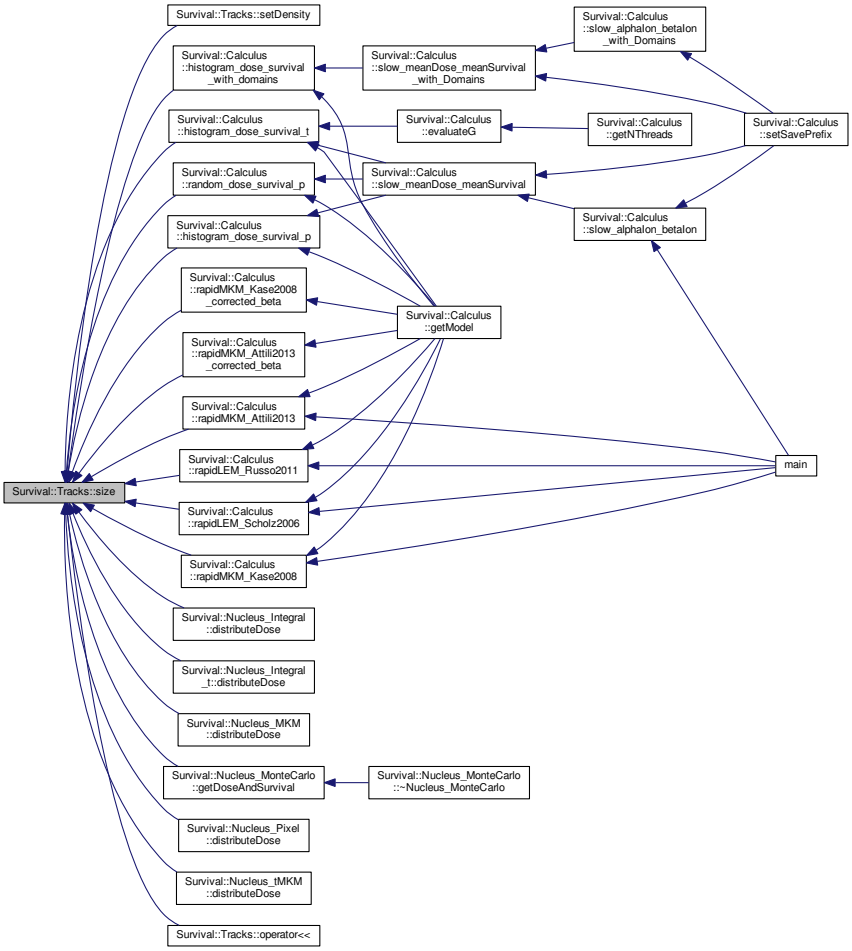
Returns the size of the vector of tracks.

Returns

The size of the vector of tracks: trackVector.

Definition at line 240 of file Tracks.cpp.

Here is the caller graph for this function:



7.19.4 Member Data Documentation

7.19.4.1 double Survival::Tracks::density [private]

The density of the medium expressed in $\frac{g}{cm^3}$.

Definition at line 217 of file Tracks.h.

7.19.4.2 std::string Survival::Tracks::spectrum_file [private]

A `string` identifying the file containing the spectrum of particle to be converted in tracks.

Definition at line 220 of file Tracks.h.

7.19.4.3 std::vector< Track* > Survival::Tracks::trackVector [private]

A dynamic vector of Track pointers.

Definition at line 214 of file Tracks.h.

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

- include/Tracks.h
- src/Tracks.cpp

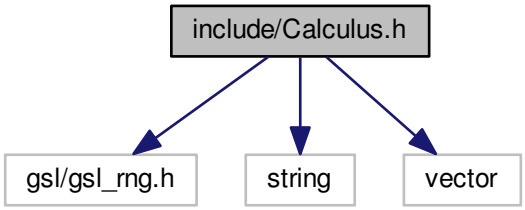
Chapter 8

File Documentation

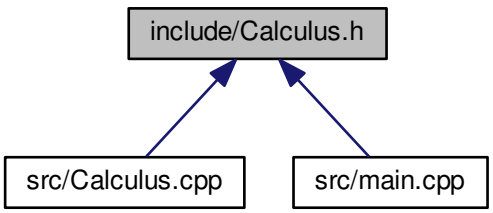
8.1 include/Calculus.h File Reference

```
#include <gsl/gsl_rng.h>
#include <string>
#include <vector>
```

Include dependency graph for Calculus.h:



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



Classes

- class Survival::Calculus

  It implements some methods to simulate the irradiation process evaluating the cellular survival on a population of cells and extrapolating the radiobiological LQ parameters.

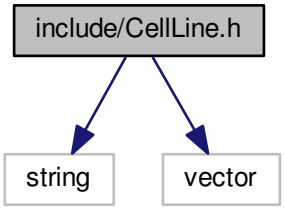
Namespaces

- Survival

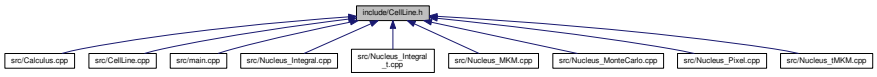
8.2 include/CellLine.h File Reference

```
#include <string>
#include <vector>
```

Include dependency graph for CellLine.h:



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



Classes

- class Survival::CellLine

  Represents the cell line to which the nucleus belongs and hosts the radiobiological and structural characteristics of the cell and the different parametrizations of the models.

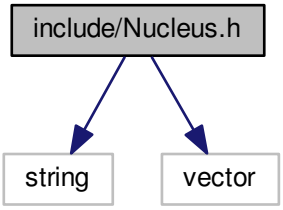
Namespaces

- Survival

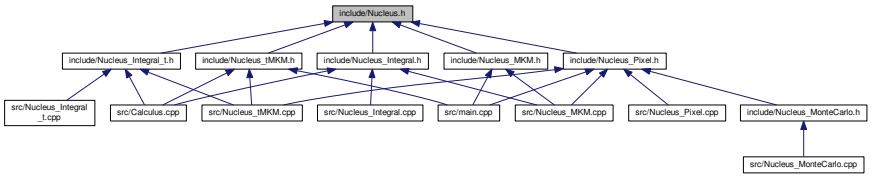
8.3 include/Nucleus.h File Reference

```
#include <string>
#include <vector>
```

Include dependency graph for Nucleus.h:



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



Classes

- class Survival::Nucleus

  Linked to a specific CellLine object, this class defines the structure of the cellular nucleus to be irradiated and evaluates the dose absorbed during the interaction with a track.

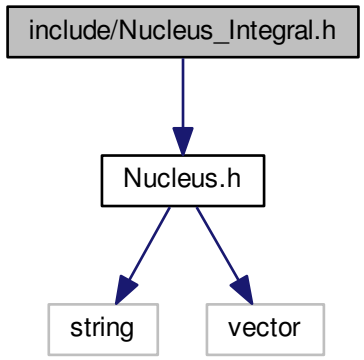
Namespaces

- Survival

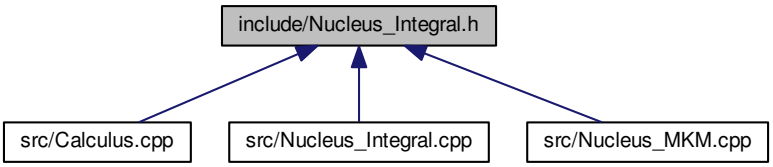
8.4 include/Nucleus_Integral.h File Reference

```
#include "Nucleus.h"
```

Include dependency graph for Nucleus_Integral.h:



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



Classes

- class Survival::Nucleus_Integral

  Implements a nucleus as a 2D circular object and provides methods to evaluate the number of lethal events observed.

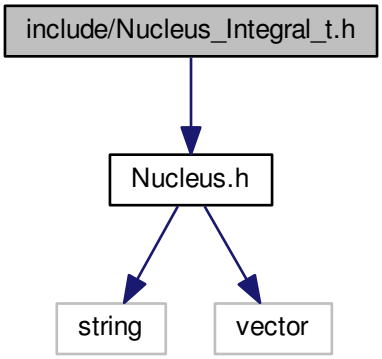
Namespaces

- Survival

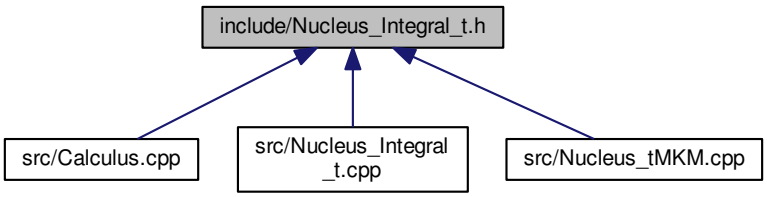
8.5 include/Nucleus_Integral_t.h File Reference

```
#include "Nucleus.h"
```

Include dependency graph for Nucleus_Integral_t.h:



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



Classes

- class Survival::Nucleus_Integral_t

  Implements a nucleus as a 2D circular object and provides methods to evaluate the number of lethal events observed taking into account the time structure of the irradiation.

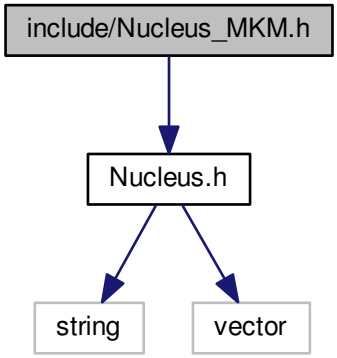
Namespaces

- Survival

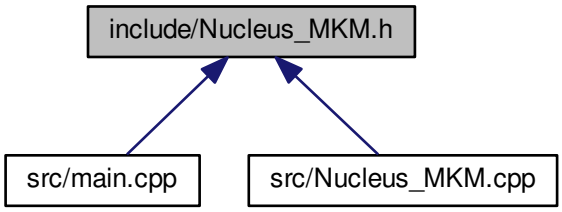
8.6 include/Nucleus_MKM.h File Reference

```
#include "Nucleus.h"
```

Include dependency graph for Nucleus_MKM.h:



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



Classes

- class Survival::Nucleus_MKM

  Inherited from the Nucleus pure virtual class, it implements the cellular nucleus as defined and used in the MKM model.

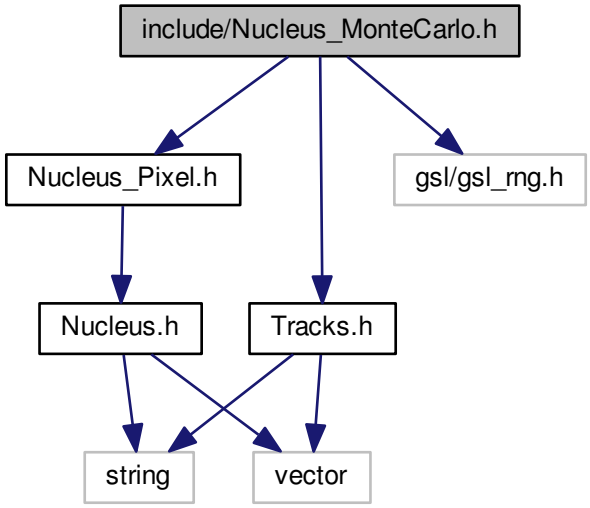
Namespaces

- Survival

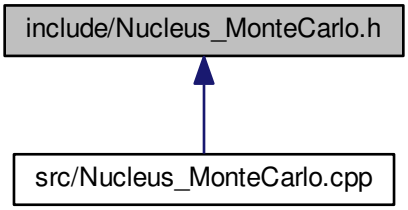
8.7 include/Nucleus_MonteCarlo.h File Reference

```
#include "Nucleus_Pixel.h"
#include "Tracks.h"
#include <gsl/gsl_rng.h>
```

Include dependency graph for Nucleus_MonteCarlo.h:



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



Classes

- class Survival::Nucleus_MonteCarlo

  Inherited from the Nucleus_Pixel class, it performs the integration of the dose deposited by the track via the Monte Carlo importance sampling method.

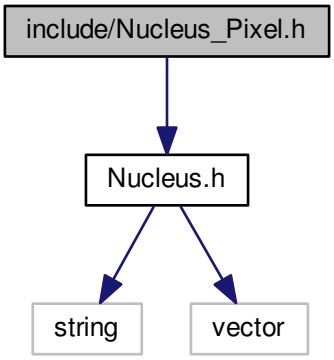
Namespaces

- Survival

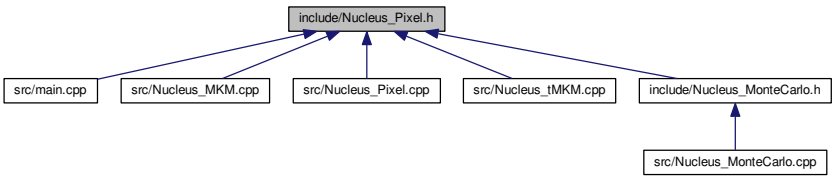
8.8 include/Nucleus_Pixel.h File Reference

```
#include "Nucleus.h"
```

Include dependency graph for Nucleus_Pixel.h:



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



Classes

- class Survival::Pixel

  Implements the Pixel features to be used in the Nucleus_Pixel class.
- class Survival::Nucleus_Pixel

  Inherited from the Nucleus pure virtual class, it implements the nucleus structure used in the LEM.

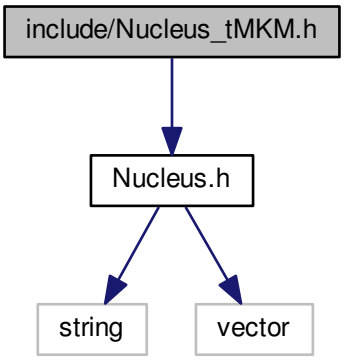
Namespaces

- Survival

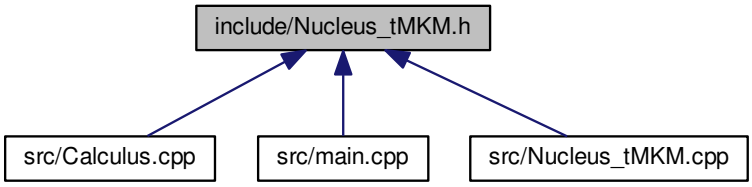
8.9 include/Nucleus_tMKM.h File Reference

```
#include "Nucleus.h"
```

Include dependency graph for Nucleus_tMKM.h:



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



Classes

- class Survival::Nucleus_tMKM

  Inherited from the Nucleus pure virtual class, it implements the nucleus defined in the Monte Carlo temporal reformulation of the MKM model (MCt-MKM).

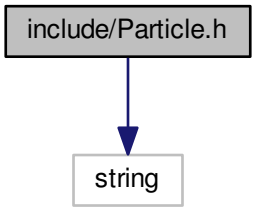
Namespaces

- Survival

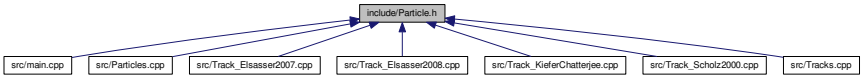
8.10 include/Particle.h File Reference

```
#include <string>
```

Include dependency graph for Particle.h:



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



Classes

- class Survival::Particle

  This class defines the object "particle".

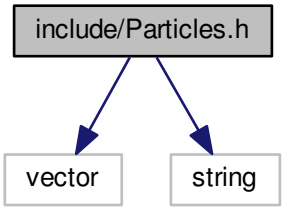
Namespaces

- Survival

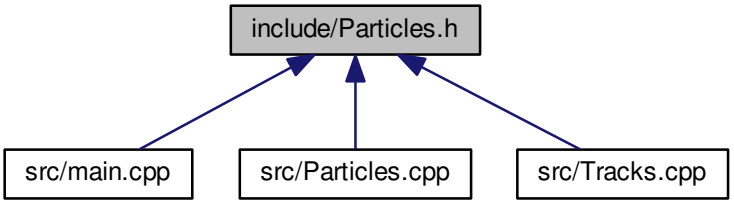
8.11 include/Particles.h File Reference

```
#include <vector>
#include <string>
```

Include dependency graph for Particles.h:



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



Classes

- class Survival::Particles

  This is a container class, used to group together Particle objects. It implements the structure and methods to manage a vector of particles.

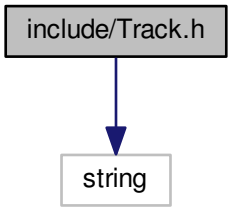
Namespaces

- Survival

8.12 include/Track.h File Reference

```
#include <string>
```

Include dependency graph for Track.h:



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

Classes

- class Survival::Track

  Constructed on the base of an ion Particle object, this class represents the local dose distribution around that ion.

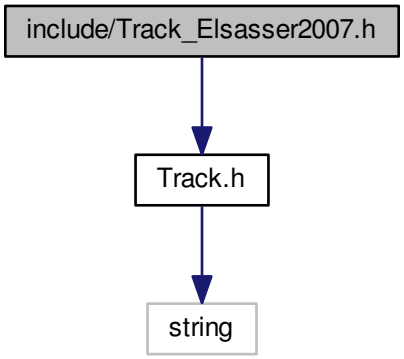
Namespaces

- Survival

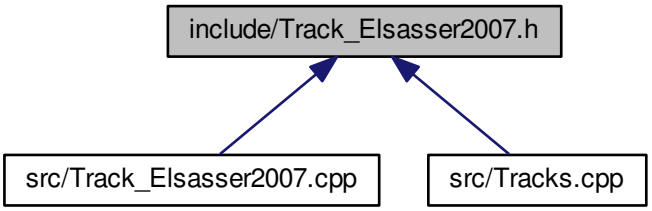
8.13 include/Track_Elsasser2007.h File Reference

```
#include "Track.h"
```

Include dependency graph for Track_Elsasser2007.h:



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



Classes

- class Survival::Track_Elsasser2007

  Inherited from the Track class, it implements the LEM II track model.

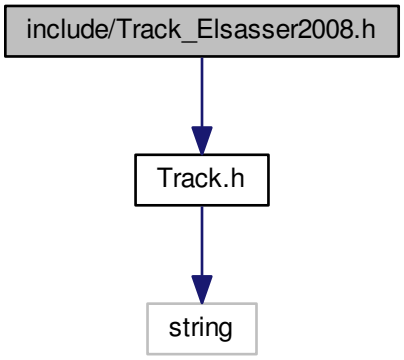
Namespaces

- Survival

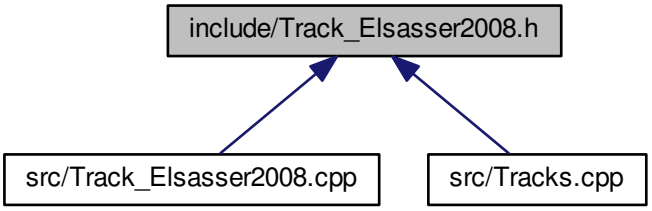
8.14 include/Track_Elsasser2008.h File Reference

```
#include "Track.h"
```

Include dependency graph for Track_Elsasser2008.h:



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



Classes

- class Survival::Track_Elsasser2008

  Inherited from the Track class, it implements the LEM III track model.

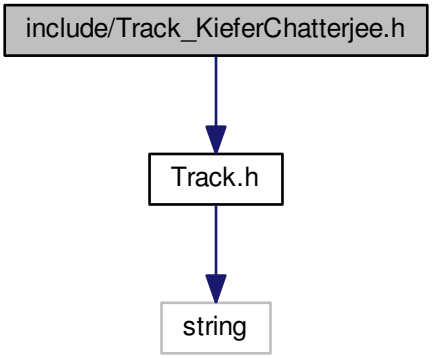
Namespaces

- Survival

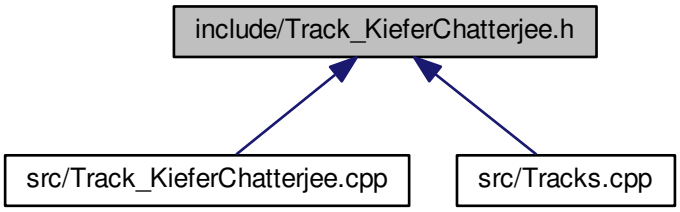
8.15 include/Track_KieferChatterjee.h File Reference

```
#include "Track.h"
```

Include dependency graph for Track_KieferChatterjee.h:



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



Classes

- class Survival::Track_KieferChatterjee

  Inherited from the Track pure virtual class, it implements the Kiefer-Chatterje amorphous track structure, used in the MKM model.

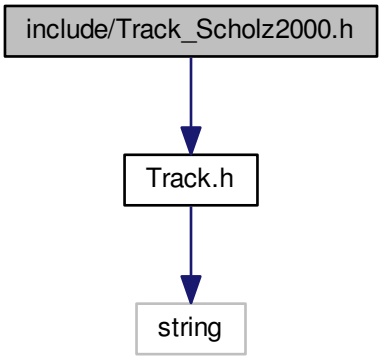
Namespaces

- Survival

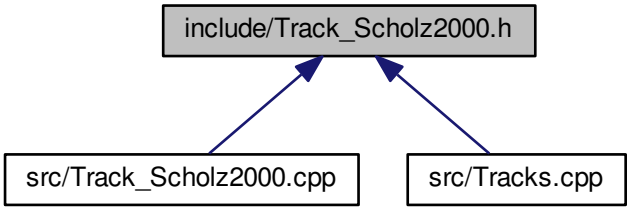
8.16 include/Track_Scholz2000.h File Reference

```
#include "Track.h"
```

Include dependency graph for Track_Scholz2000.h:



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



Classes

- class Survival::Track_Scholz2000

  Inherited from the Track class, it implements the LEM I track model.

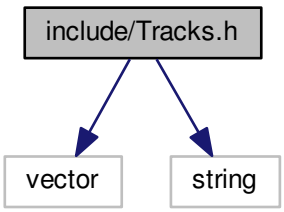
Namespaces

- Survival

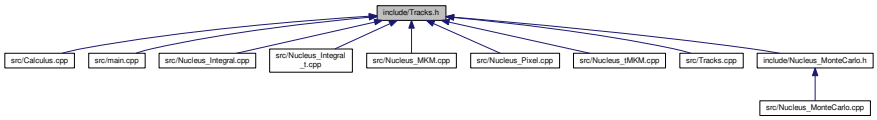
8.17 include/Tracks.h File Reference

```
#include <vector>
#include <string>
```

Include dependency graph for Tracks.h:



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



Classes

- class Survival::Tracks

  This is a container class for Track objects; it implements the structure and methods to manage a vector of tracks.

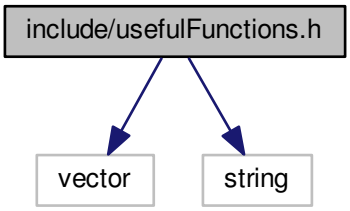
Namespaces

- Survival

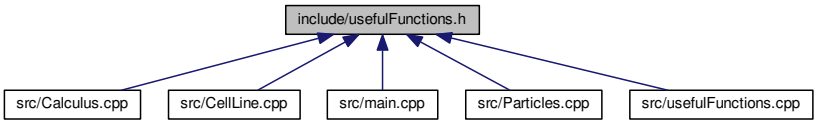
8.18 include/usefulFunctions.h File Reference

```
#include <vector>
#include <string>
```

Include dependency graph for usefulFunctions.h:



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



Namespaces

- Survival

Functions

- int Survival::_mkdir (const char *path)

  Portable wrapper for mkdir. Internally used by mkdir().
- double Survival::betheBloch_inv_Srim (std::string ionType, double let_imposed)

  Returns the kinetic energy associated to the value of LET imposed for a specified ion.
- double Survival::betheBloch_Srim (std::string ionType, double e_c_imposed)
- void Survival::fit_LQ (std::vector< double > dose, std::vector< double > survival, std::vector< double > survivalUncertainty, double &alpha, double &alphaUncertainty, double &beta, double &betaUncertainty, double &chiSquared, double &incompleteGammaQ)

  Perform a fit on a data set of doses and associated survival according to the linear quadratic formula.
- bool Survival::folder_exists (std::string foldername)

  Checks if a folder exists.
- int Survival::mkdir (const char *path)

  Recursive, portable wrapper for mkdir.
- void Survival::parse (int argc, char *argv[], std::string &cellType, std::string &model, std::string &trackType, std::string &parametrizationType, std::string &calculusType, double &precision, int &parallelismType, std::vector< double > &doses, std::vector< std::string > &parameter_name, double &MKM_alpha0, double &MKM_beta0, double &MKM_rNucleus, double &MKM_rDomain, double &tMKM_ac, double &LEM_alpha0, double &LEM_beta0, double &LEM_rNucleus, double &LEM_Dt, std::string &ionType, int &particleA, int &particleZ, std::string &trackMode, std::string &energyType, std::vector< double > &energies, int &nFraction, double &timeSpacing, double &fracDeliveryTime, bool &saveAlphaBeta, bool &saveMeans, bool &saveCell, std::string &projectName, bool &mono, std::string &spectrum_file)

Parses the input arguments in the `main` to set the calculation parameters.

- void Survival::Usage ()

  Display an hint to the user to correctly use the executable.

8.19 src/Calculus.cpp File Reference

```
#include "Calculus.h"
#include "CellLine.h"
#include "Tracks.h"
#include "Track.h"
#include "Nucleus_Integral.h"
#include "Nucleus_Integral_t.h"
#include "Nucleus_tMKM.h"
#include "usefulFunctions.h"
#include <gsl/gsl_randist.h>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_sf_gamma.h>
#include <sstream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <iomanip>
#include <iostream>
#include <algorithm>
#include <sys/types.h>
#include <unistd.h>
#include <omp.h>
```

Include dependency graph for Calculus.cpp:

Macros

- #define _USE_MATH_DEFINES

8.19.1 Macro Definition Documentation

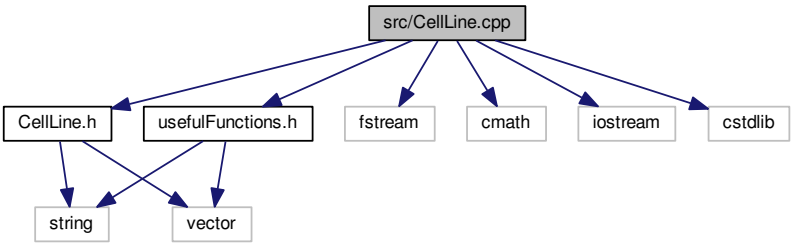
8.19.1.1 #define _USE_MATH_DEFINES

Definition at line 26 of file Calculus.cpp.

8.20 src/CellLine.cpp File Reference

```
#include "CellLine.h"
#include "usefulFunctions.h"
#include <fstream>
#include <cmath>
#include <iostream>
#include <cstdlib>
```

Include dependency graph for CellLine.cpp:



Macros

- #define _USE_MATH_DEFINES

8.20.1 Macro Definition Documentation

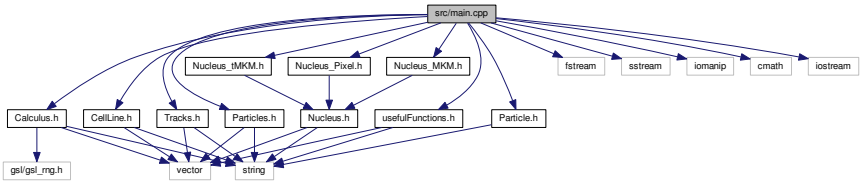
8.20.1.1 #define _USE_MATH_DEFINES

Definition at line 8 of file CellLine.cpp.

8.21 src/main.cpp File Reference

```
#include "Calculus.h"
#include "Nucleus_MKM.h"
#include "Nucleus_tMKM.h"
#include "Nucleus_Pixel.h"
#include "CellLine.h"
#include "Tracks.h"
#include "Particles.h"
#include "Particle.h"
#include "usefulFunctions.h"
#include <fstream>
#include <sstream>
#include <iomanip>
#include <cmath>
#include <iostream>
```

Include dependency graph for main.cpp:



Functions

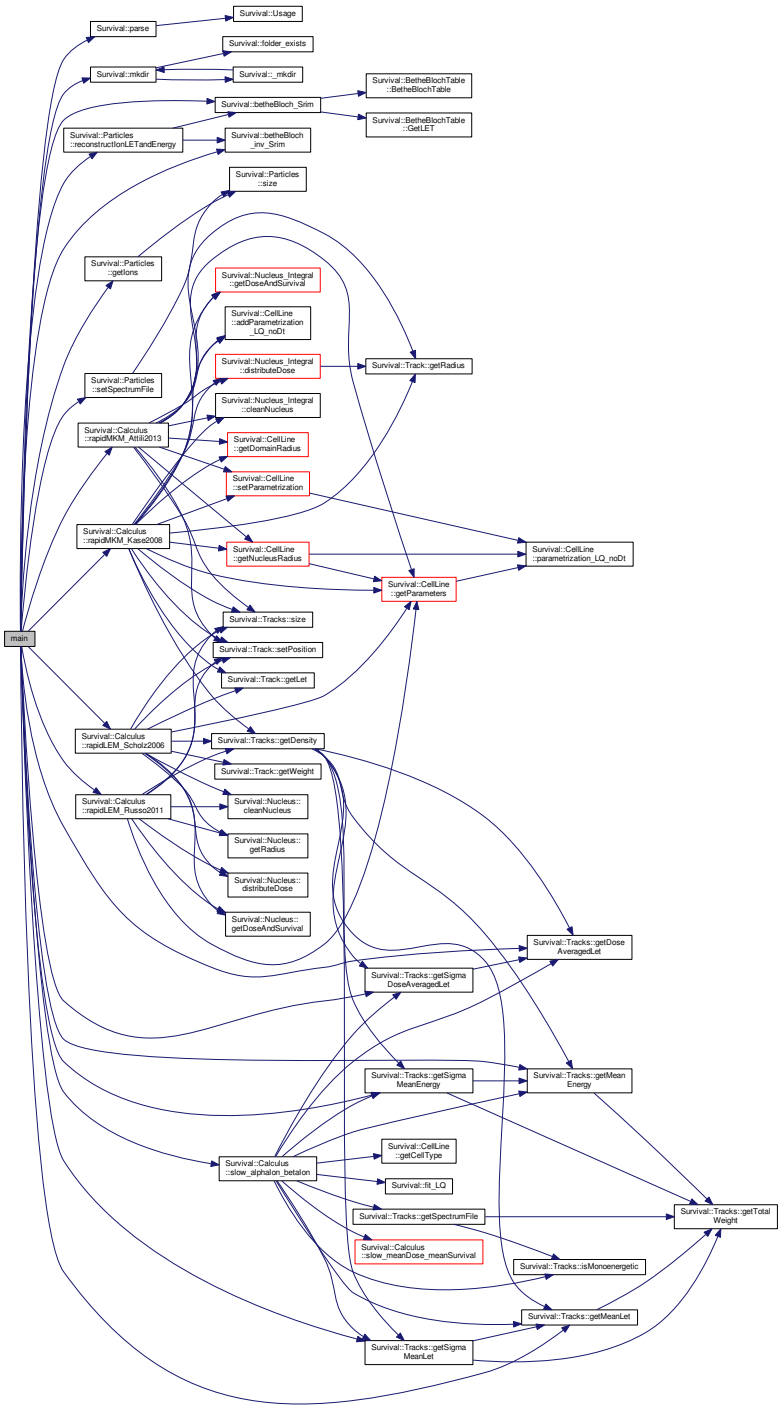
- int main (int argc, char ∗argv[ ])

8.21.1 Function Documentation

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

Definition at line 137 of file main.cpp.

Here is the call graph for this function:

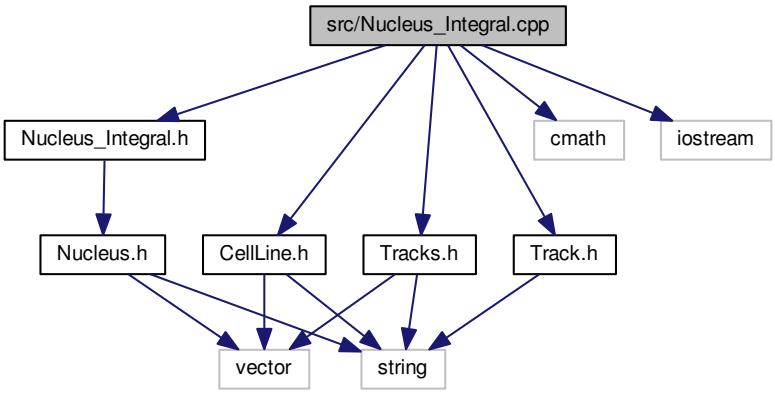


8.22 src/Nucleus_Integral.cpp File Reference

```
#include "Nucleus_Integral.h"
```

```
#include "CellLine.h"
#include "Tracks.h"
#include "Track.h"
#include <cmath>
#include <iostream>
```

Include dependency graph for Nucleus_Integral.cpp:



Macros

- #define _USE_MATH_DEFINES

8.22.1 Macro Definition Documentation

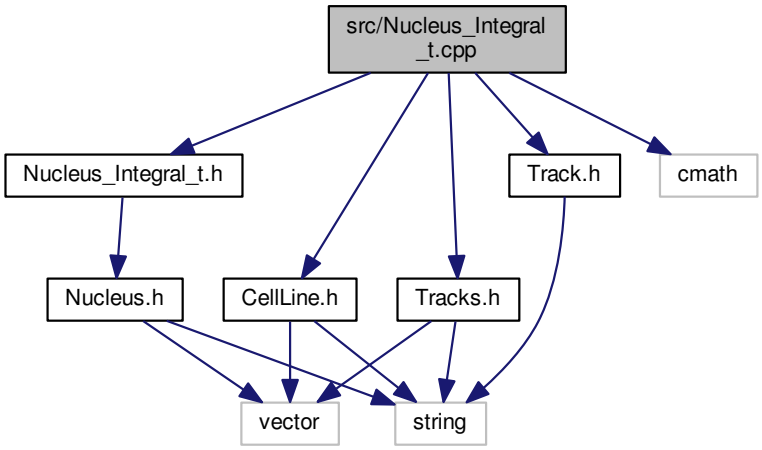
8.22.1.1 #define _USE_MATH_DEFINES

Definition at line 6 of file Nucleus_Integral.cpp.

8.23 src/Nucleus_Integral_t.cpp File Reference

```
#include "Nucleus_Integral_t.h"
#include "CellLine.h"
#include "Tracks.h"
#include "Track.h"
#include <cmath>
```

Include dependency graph for Nucleus_Integral_t.cpp:



Macros

- #define _USE_MATH_DEFINES

8.23.1 Macro Definition Documentation

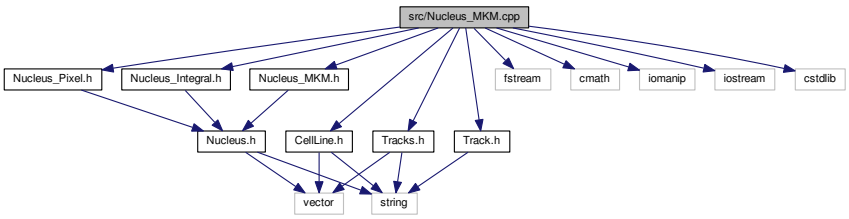
8.23.1.1 #define _USE_MATH_DEFINES

Definition at line 6 of file Nucleus_Integral_t.cpp.

8.24 src/Nucleus_MKM.cpp File Reference

```
#include "Nucleus_MKM.h"
#include "Nucleus_Pixel.h"
#include "Nucleus_Integral.h"
#include "CellLine.h"
#include "Tracks.h"
#include "Track.h"
#include <fstream>
#include <cmath>
#include <iomanip>
#include <iostream>
#include <cstdlib>
```

Include dependency graph for Nucleus_MKM.cpp:



Macros

- #define _USE_MATH_DEFINES

8.24.1 Macro Definition Documentation

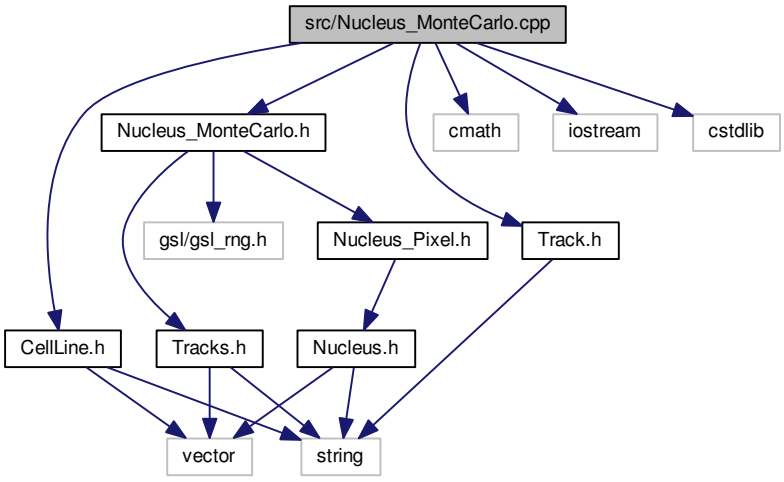
8.24.1.1 #define _USE_MATH_DEFINES

Definition at line 11 of file Nucleus_MKM.cpp.

8.25 src/Nucleus_MonteCarlo.cpp File Reference

```
#include "Nucleus_MonteCarlo.h"
#include "CellLine.h"
#include "Track.h"
#include <cmath>
#include <iostream>
#include <cstdlib>
```

Include dependency graph for Nucleus_MonteCarlo.cpp:



Macros

- #define _USE_MATH_DEFINES

8.25.1 Macro Definition Documentation

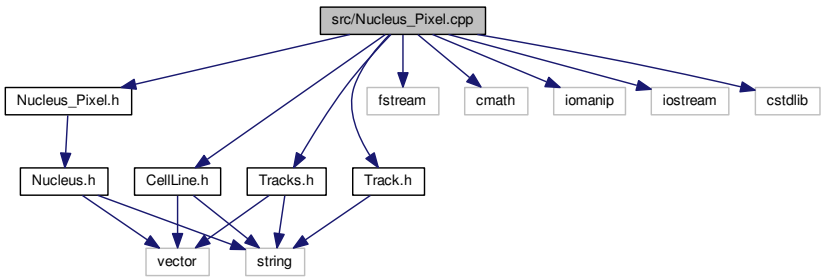
8.25.1.1 #define _USE_MATH_DEFINES

Definition at line 5 of file Nucleus_MonteCarlo.cpp.

8.26 src/Nucleus_Pixel.cpp File Reference

```
#include "Nucleus_Pixel.h"
#include "CellLine.h"
#include "Tracks.h"
#include "Track.h"
#include <fstream>
#include <cmath>
#include <iomanip>
#include <iostream>
#include <cstdlib>
```

Include dependency graph for Nucleus_Pixel.cpp:



Macros

- #define _USE_MATH_DEFINES

8.26.1 Macro Definition Documentation

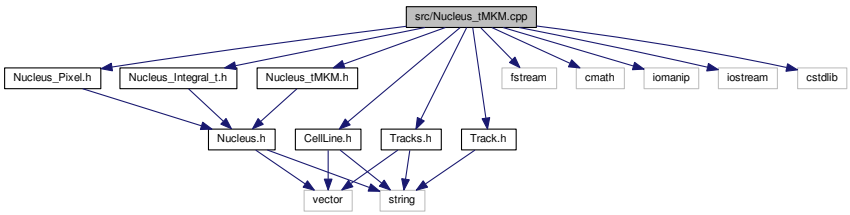
8.26.1.1 #define _USE_MATH_DEFINES

Definition at line 9 of file Nucleus_Pixel.cpp.

8.27 src/Nucleus_tMKM.cpp File Reference

```
#include "Nucleus_tMKM.h"
#include "Nucleus_Pixel.h"
#include "Nucleus_Integral_t.h"
#include "CellLine.h"
#include "Tracks.h"
#include "Track.h"
#include <fstream>
#include <cmath>
#include <iomanip>
#include <iostream>
#include <cstdlib>
```

Include dependency graph for Nucleus_tMKM.cpp:



Macros

- #define _USE_MATH_DEFINES

8.27.1 Macro Definition Documentation

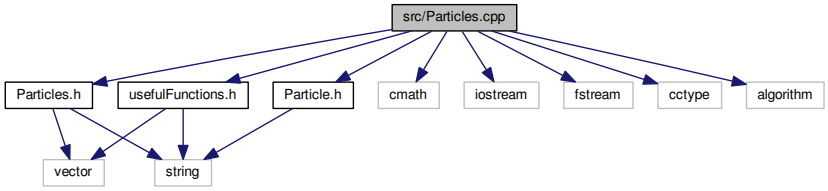
8.27.1.1 #define _USE_MATH_DEFINES

Definition at line 11 of file Nucleus_tMKM.cpp.

8.28 src/Particles.cpp File Reference

```
#include "Particles.h"
#include "Particle.h"
#include "usefulFunctions.h"
#include <cmath>
#include <iostream>
#include <fstream>
#include <cctype>
#include <algorithm>
```

Include dependency graph for Particles.cpp:



Macros

- #define _USE_MATH_DEFINES

8.28.1 Macro Definition Documentation

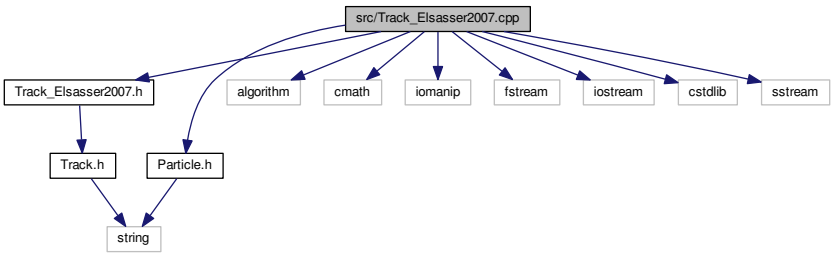
8.28.1.1 #define _USE_MATH_DEFINES

Definition at line 5 of file Particles.cpp.

8.29 src/Track_Elsasser2007.cpp File Reference

```
#include "Track_Elsasser2007.h"
#include "Particle.h"
#include <algorithm>
#include <cmath>
#include <iomanip>
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <sstream>
```

Include dependency graph for Track_Elsasser2007.cpp:



Macros

- #define _USE_MATH_DEFINES

8.29.1 Macro Definition Documentation

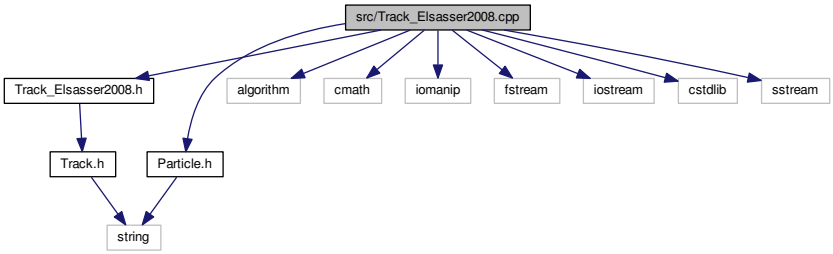
8.29.1.1 #define _USE_MATH_DEFINES

Definition at line 8 of file Track_Elsasser2007.cpp.

8.30 src/Track_Elsasser2008.cpp File Reference

```
#include "Track_Elsasser2008.h"
#include "Particle.h"
#include <algorithm>
#include <cmath>
#include <iomanip>
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <sstream>
```

Include dependency graph for Track_Elsasser2008.cpp:



Macros

- #define _USE_MATH_DEFINES

8.30.1 Macro Definition Documentation

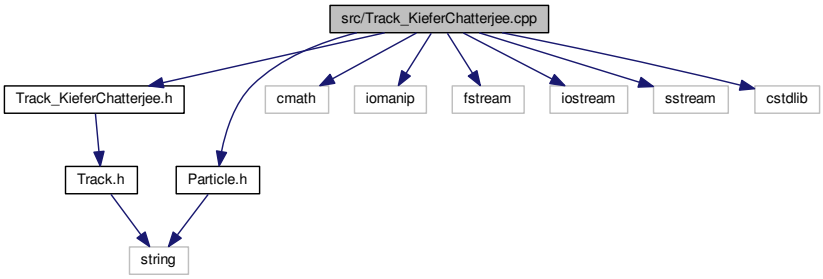
8.30.1.1 #define _USE_MATH_DEFINES

Definition at line 8 of file Track_Elsasser2008.cpp.

8.31 src/Track_KieferChatterjee.cpp File Reference

```
#include "Track_KieferChatterjee.h"
#include "Particle.h"
#include <cmath>
#include <iomanip>
#include <fstream>
#include <iostream>
#include <sstream>
#include <cstdlib>
```

Include dependency graph for Track_KieferChatterjee.cpp:



Macros

- #define _USE_MATH_DEFINES

8.31.1 Macro Definition Documentation

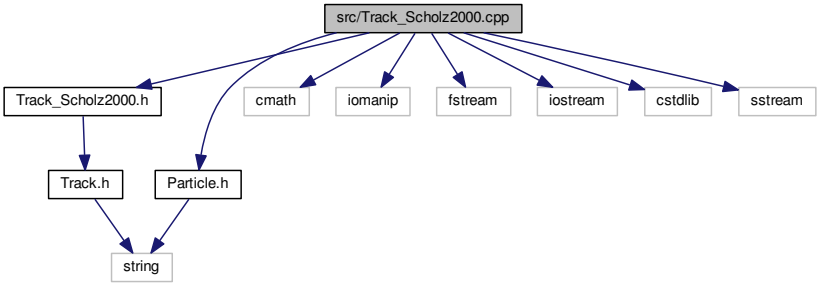
8.31.1.1 #define _USE_MATH_DEFINES

Definition at line 4 of file Track_KieferChatterjee.cpp.

8.32 src/Track_Scholz2000.cpp File Reference

```
#include "Track_Scholz2000.h"
#include "Particle.h"
#include <cmath>
#include <iomanip>
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <sstream>
```

Include dependency graph for Track_Scholz2000.cpp:



Macros

- #define _USE_MATH_DEFINES

8.32.1 Macro Definition Documentation

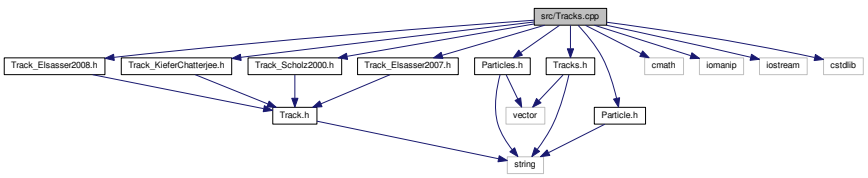
8.32.1.1 #define _USE_MATH_DEFINES

Definition at line 4 of file Track_Scholz2000.cpp.

8.33 src/Tracks.cpp File Reference

```
#include "Tracks.h"
#include "Track_Scholz2000.h"
#include "Track_Elsasser2007.h"
#include "Track_Elsasser2008.h"
#include "Track_KieferChatterjee.h"
#include "Particles.h"
#include "Particle.h"
#include <cmath>
#include <iomanip>
#include <iostream>
#include <cstdlib>
```

Include dependency graph for Tracks.cpp:



Macros

- #define _USE_MATH_DEFINES

8.33.1 Macro Definition Documentation

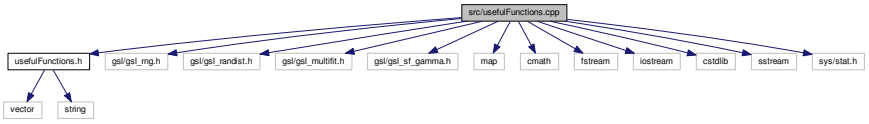
8.33.1.1 #define _USE_MATH_DEFINES

Definition at line 9 of file Tracks.cpp.

8.34 src/usefulFunctions.cpp File Reference

```
#include "usefulFunctions.h"
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_sf_gamma.h>
#include <map>
#include <cmath>
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <sstream>
#include <sys/stat.h>
```

Include dependency graph for usefulFunctions.cpp:



Classes

- class Survival::BetheBlochTable

  Class used to load precomputed values of LET for different ions of varying kinetic energy, evaluated by means of the Bethe-Bloch formula.

Namespaces

- Survival

Macros

- #define _USE_MATH_DEFINES

Functions

- int Survival::_mkdir (const char *path)

  Portable wrapper for mkdir. Internally used by mkdir().
- double Survival::betheBloch_inv_Srim (string ionType, double let_imposed)
- double Survival::betheBloch_Srim (string ionType, double e_c_imposed)
- void Survival::fit_LQ (vector< double > dose, vector< double > survival, vector< double > survivalUncertainty, double &alpha, double &alphaUncertainty, double &beta, double &betaUncertainty, double &chiSquared, double &incompleteGammaQ)
- bool Survival::folder_exists (string foldername)
- int Survival::mkdir (const char *path)

  Recursive, portable wrapper for mkdir.
- void Survival::parse (int argc, char *argv[], string &cellType, string &model, string &trackType, string &parametrizationType, string &calculusType, double &precision, int &parallelismType, vector< double > &doses, vector< string > &parameter_name, double &MKM_alpha0, double &MKM_beta0, double &MKM_rNucleus, double &MKM_rDomain, double &tMKM_ac, double &LEM_alpha0, double &LEM_beta0, double &LEM_rNucleus, double &LEM_Dt, string &ionType, int &particleA, int &particleZ, string &trackMode, string &energyType, vector< double > &energies, int &nFraction, double &timeSpacing, double &fracDeliveryTime, bool &saveAlphaBeta, bool &saveMeans, bool &saveCell, string &projectName, bool &mono, string &spectrum_file)
- void Survival::Usage ()

  Display an hint to the user to correctly use the executable.

8.34.1 Macro Definition Documentation

8.34.1.1 #define _USE_MATH_DEFINES

Definition at line 12 of file usefulFunctions.cpp.

Index