

# PenRed: User Manual

V. Giménez-Alventosa<sup>1</sup>, V. Giménez Gómez<sup>2</sup>, and S. Oliver<sup>3</sup>

<sup>1</sup>Departament de Física Atòmica Molecular i Nuclear, Universitat de València, Dr. Moliner, 50, 46100, Burjassot, València, Spain, gimenez.alventosa.vicent@gmail.com

<sup>2</sup>Departament de Física Teòrica and IFIC, Universitat de València-CSIC, Dr. Moliner, 50, 46100, Burjassot, València, Spain,

<sup>3</sup>Instituto de Seguridad Industrial, Radiofísica y Medioambiental (ISIRYM), Universitat Politècnica de València, Camí de Vera s/n, 46022, València, Spain, sanolgi@upvnet.upv.es

October 5, 2024

## Abstract

The present document is a manual of the PenRed code system which content is focused on the description of the framework usage. PenRed is a general-purpose, stand-alone and extensible framework code based on PENELOPE for parallel Monte Carlo simulations of radiation transport through matter. It has been implemented in C++ programming language and takes advantage of modern object-oriented technologies. In addition, PenRed offers, among other features, the capability to read and process DICOM images to perform simulations. These feature facilitate its usage in medical applications. Our framework has been successfully tested against the original PENELOPE Fortran code.


## 1 Introduction


The present document contains all the necessary information to use the PenRed [1] framework. That is, all the available modules already included in the package and the provided main program. This document is not intended to describe how to implement new components or describe in detail the code structure. That information can be found in the PenRed implementation manual and component manual (in development) respectively.

### 1.1 What is PenRed?

PenRed is a fully parallel, modular and customizable framework for Monte Carlo simulations of the passage of radiation through matter. It is based on the PENELOPE [2] code system, from which inherits its unique physics models and tracking algorithms for charged particles. For further details on the interactions models, the reader is referred to the excellent PENELOPE manual [3]. PenRed has been coded in C++ following an object-oriented

---

V. Giménez-Alventosa  <https://orcid.org/0000-0003-1646-6094>

V. Giménez Gómez  <https://orcid.org/0000-0003-3855-2567>

S. Oliver  <https://orcid.org/0000-0001-8258-3972>

programming paradigm restricted to the C++14 standard. Our engine implements parallelism via a double approach: on the one hand, by using standard C++ threads for shared memory, improving the access and usage of the memory, and, on the other hand, via the MPI standard for distributed memory infrastructures. Notice that both kinds of parallelism can be combined together in the same simulation. In addition, PenRed provides a modular structure with methods designed to easily extend its functionality. Thus, users can create their own independent modules to adapt our engine to their needs without changing the existing ones. Furthermore, user extensions will take advantage of the built-in parallelism without any extra effort or knowledge of parallel programming.

## 1.2 Document structure

The manual is organized as follows. The section 2 shows a useful guide of installation and execution of the framework to run simulations. Section 3 provides some information of the internal data structure format used to configure the simulations. In section 4 a briefly description of important definitions, constants, parameters and units is shown. Following, section 5, shows how to execute the main program provided within the PenRed package, and describes all the implemented tallies, sources, geometries, variance reduction techniques and other parameters to configure the simulation. Then, in the section 7, the examples located in the **examples** folder, which are ready to be simulated, are described. Finally, the offered utilities in the PenRed package and how to use them are explained in the section 8.

Notice that the section 4 summarise some useful definitions necessary to understand the code and its operation.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	What is PenRed? . . . . .	1
1.2	Document structure . . . . .	2
<b>2</b>	<b>Installation and execution</b>	<b>6</b>
2.1	Manual installation with CMake . . . . .	7
2.1.1	User interface . . . . .	7
2.1.2	Command line . . . . .	9
2.2	MSVS installation . . . . .	10
2.3	Basic usage . . . . .	14
2.4	Optional features . . . . .	14
<b>3</b>	<b>Internal data format</b>	<b>15</b>
3.1	Implementation . . . . .	15
<b>4</b>	<b>Constants, parameters and definitions</b>	<b>16</b>
4.1	Important definitions . . . . .	16
4.2	Units . . . . .	18
4.3	Particle indexes . . . . .	18
4.4	Particle interaction indexes . . . . .	19
4.5	Atomic electron shells . . . . .	19
<b>5</b>	<b>Framework usage</b>	<b>20</b>
5.1	Materials . . . . .	20
5.1.1	Configuration . . . . .	21
5.2	Particle sources . . . . .	24
5.2.1	Spatial source samplers . . . . .	27
5.2.2	Direction samplers . . . . .	30
5.2.3	Energy samplers . . . . .	31
5.2.4	Time samplers . . . . .	32
5.2.5	Specific Sources . . . . .	33
5.3	Geometries . . . . .	37
5.3.1	Geometry visualization . . . . .	37
5.3.2	Quadric . . . . .	38
5.3.3	Triangular Mesh Surfaces . . . . .	38
5.3.4	Voxel . . . . .	43
5.3.5	DICOM . . . . .	45
5.3.6	Combo . . . . .	49
5.4	Tallies . . . . .	51
5.4.1	Radial and Cylindrical Dose Distribution . . . . .	52
5.4.2	Emerging Particles Distribution . . . . .	53
5.4.3	Energy Deposition Body . . . . .	53
5.4.4	Energy Deposition Material . . . . .	54
5.4.5	Impact Detector . . . . .	54
5.4.6	Spatial Dose Distribution . . . . .	56
5.4.7	Angular Detector . . . . .	57
5.4.8	Particle Generation . . . . .	58
5.4.9	Spherical Dose Distribution . . . . .	58

5.4.10	Phase Space File (PSF)	59
5.4.11	Kerma track length estimator	60
5.4.12	CT tally	62
5.4.13	DICOM kerma tally (Dose Volume Histogram)	63
5.4.14	Tracks	65
5.4.15	Primary and Scatter Separated (PSS)	66
5.4.16	Detection Spatial Distribution	67
5.4.17	Emerging Spherical Distribution	68
5.5	Variance Reduction	70
5.5.1	Context specific	70
5.5.2	Specific VR techniques	71
5.5.3	Generic VR techniques	72
5.6	Simulation parameters	73
5.6.1	Dump	73
5.6.2	Threading	75
5.6.3	Seeds	75
5.6.4	Simulation time	76
5.7	Logs	76
5.8	Image output	76
5.9	Load balance parameters	77
<b>6</b>	<b>Python wrapper</b>	<b>77</b>
6.1	Configuration format	78
6.2	Usage	79
6.3	API	82
<b>7</b>	<b>Examples</b>	<b>82</b>
7.1	1-disc	83
7.1.1	1-disc-no-vr	83
7.1.2	1-disc-vr	83
7.2	2-plane	83
7.3	3-detector	84
7.4	4-x-ray-tube	84
7.5	5-accelerator	84
7.5.1	5-accelerator-1	84
7.5.2	5-accelerator-2	84
7.5.3	5-accelerator-3	84
7.6	6-polarisation	85
7.7	7-aba	85
7.8	8-fake-chamber	85
7.8.1	8-fake-chamber-novr	85
7.8.2	8-fake-chamber-vr	86
<b>8</b>	<b>Utilities</b>	<b>86</b>
8.1	Blender Geometry Plugin	86
8.1.1	Quadric Geometries	88
8.1.2	Triangular Mesh Surfaces	90
8.2	Mutren	90
8.3	iaeaPSF	90
8.4	geo2voxel	91
8.5	geo2mesh	92

8.6	range . . . . .	94
8.7	PSF spectrum . . . . .	94
8.8	PSF to ASCII . . . . .	94
8.9	Registers . . . . .	94
8.10	CompositionsDB . . . . .	95
8.11	internalData2YAML . . . . .	96
<b>A</b>	<b>Predefined materials</b>	<b>97</b>

## 2 Installation and execution

Although PenRed can be compiled with no external dependencies, some features depend on external libraries that should be installed to activate the corresponding functionalities (see below). It is the user's responsibility to ensure these external dependencies are installed before enabling a feature. Users should refer to the installation guides of the respective optional packages.

PenRed can be compiled with any compiler supporting the C++14 standard and CMake<sup>1</sup> version 3.4 or greater. Once a suitable C++ compiler is available, to compile PenRed, first download the source code from our GitHub repository<sup>2</sup>. This can be done manually from the repository page or using *git* with the following command line instruction:

```
git clone https://github.com/PenRed/PenRed.git
```

With the code downloaded, navigate to the `src` folder to compile it. This folder includes a CMake file and two scripts to compile it automatically, depending on the operating system:

- **Linux/Unix:** Run the bash script *compile.sh* with  
`bash compile.sh`
- **Windows:** Run the batch script *compile.bat*

The compilation scripts will create a *build* folder where all the temporary files required by the compilation will be stored. Finally, a *compilation* folder will be created in the *src* directory with all generated executables. Among all the executables, the PenRed main program, the one used to run the simulation examples, is located at:

*src/compiled/mains/pen\_main*

Editing the corresponding script, allows the user to enable/disable the following main optional features:

- **DICOMs:** If enabled, PenRed's capabilities to read and simulate DICOM images will be active. This option requires the DICOM toolkit (dcmtool)[4].
- **Multi-threading:** This option enables multi-threading capabilities. PenRed implements multi-threading via the standard thread library specified in the C++ standard. Thus, no extra library is required to enable this option.
- **MPI:** This option enables MPI simulations. It requires a library with an implementation of the MPI standard, such as OpenMPI[5] or MPICH[6].
- **Load Balance:** Enables load balancing between threads and MPI processes. This option requires at least multi-threading capabilities, as it uses threads to handle MPI communications.

These options, except for multi-threading capabilities which are included in the C++ standard, require some external libraries. These libraries can be found in most Linux and Unix package repositories or must be downloaded for Windows. For example, to compile

---

<sup>1</sup><https://cmake.org/download/>

<sup>2</sup><https://github.com/PenRed/PenRed>

PenRed with DICOM support on Fedora, the *dcmtdk* library can be installed with the `dnf` command:

```
sudo dnf install dcmtdk dcmtdk-devel
```

and also its dependencies,

```
sudo dnf install libjpeg libjpeg-devel libxml2 libxml2-devel openssl openssl-devel
```

## 2.1 Manual installation with CMake

To access all available compilation options, users can either modify the compilation script by adding definitions manually or use *CMake* to select the desired options.

In this case, we will use the *CMake* application directly. First, download *CMake* from repositories or its webpage<sup>3</sup> and follow the installation instructions. Note that a C++ compiler is still required.

Once installed, the *CMake* utility can be used with its user interface or directly via the command line.

### 2.1.1 User interface

To configure PenRed using the *CMake* user interface, open it and set the source folder to the PenRed *src* directory, and the build folder to the desired path where the intermediate files will be stored (Figure 1). It is recommended to create a new directory for the build path, such as a folder named *build* in the *src* directory.

---

<sup>3</sup><https://cmake.org/download/>

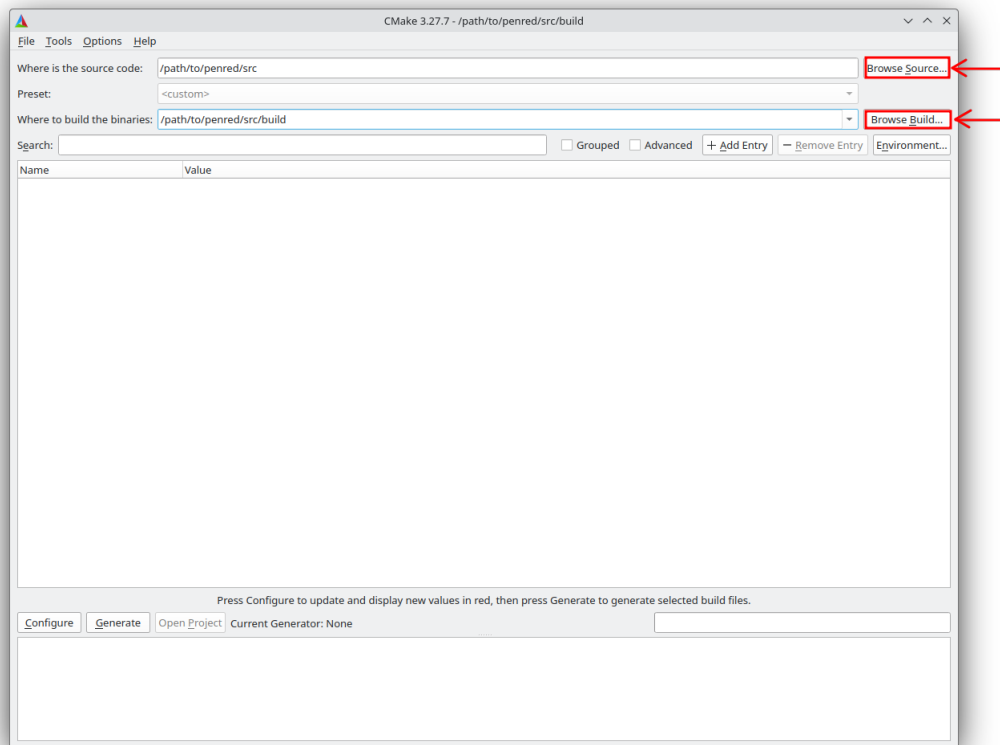


Figure 1: Set source and build paths with cmake user interface.

Then, push the *configure* button at the bottom left of the window. This step will generate the required files in the **build** folder to show a list of all compilation options (Figure 2).



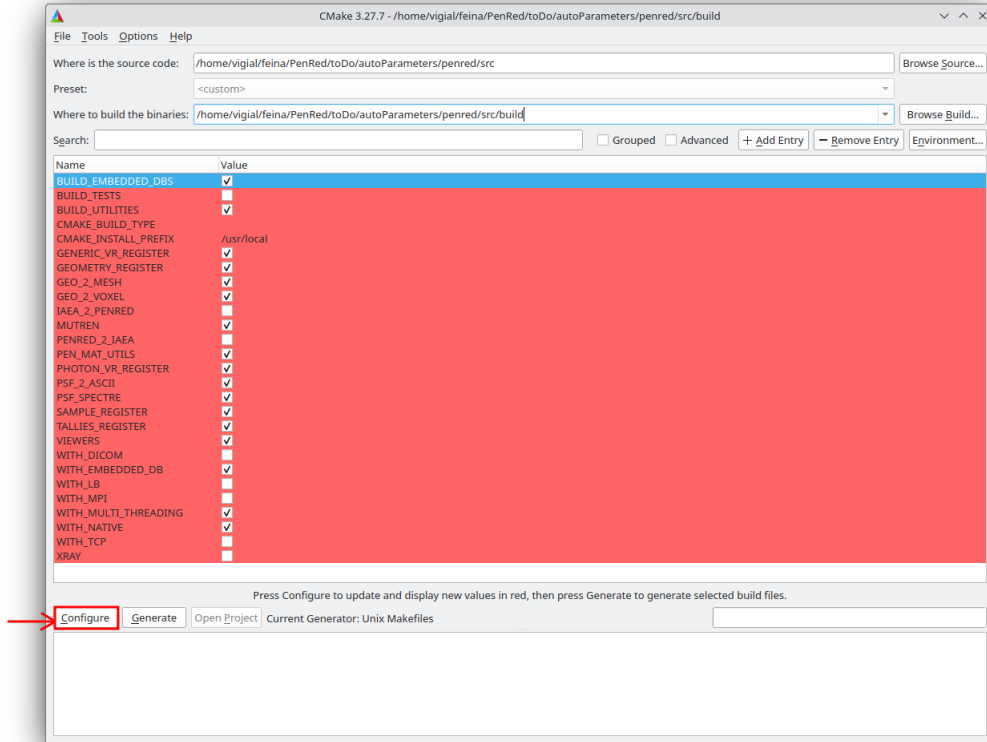


Figure 2: Configure project with cmake user interface, generating a list of compilation options.

Once the options have been selected, press the *generate* button to create all the files in the *build* folder. Finally, if using a compiler with an IDE, like MSVC, to compile the code, the *Open Project* button will open the project with the corresponding IDE, where the user can build and install the package.

If not, and the *Open Project* button remains disabled, the build must be done by command line in two steps. First, navigate to the build folder with the instruction:

```
cd path/to/folder/build
```

then, run the *CMake* build and install instruction:

```
cmake --build . --config Release --target install
```

Once completed, the corresponding executables will be stored in the *compiled* folder inside the penRed *src* directory. The main program is stored in:

```
src/compiled/mains/pen_main
```

## 2.1.2 Command line

To configure the build options via command line, first, navigate to the *src* directory, create a **build** folder and go inside:

```
cd /path/to/PenRed/repository/src
```

```
mkdir build
```

```
cd build
```

Then, run the *ccmake* command

```
ccmake ../
```

This will open a terminal application to configure the PenRed build. First, configure the project by pressing *c* on the keyboard. If no error occurs, a list of compilation options will be displayed. The user can navigate and edit the options with the keyboard. Once the configuration is set, generate the build system by pressing *g*. If the *g* option does not appear, press the configure button again. The build system is now ready to be compiled. Exit the *ccmake* application with *q* and, in the same folder, run the following instruction:

```
cmake --build . --config Release --target install
```

Once the code has been compiled, the penRed main program executable, ready to simulate, can be found at,

```
src/compiled/mains/pen_main
```

## 2.2 MSVS installation

To compile the PenRed code using MSVS, first, select “Clone a repository” from the Visual Studio start window (Figure 3).

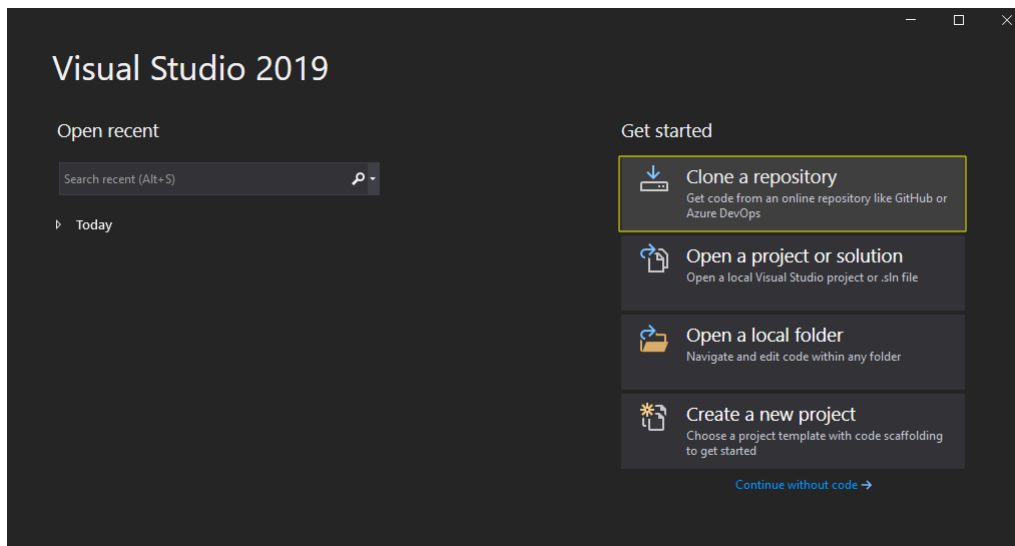


Figure 3: Clone a repository in MSVS

Then, set the PenRed repository url and push on the clone button (Figure 4).

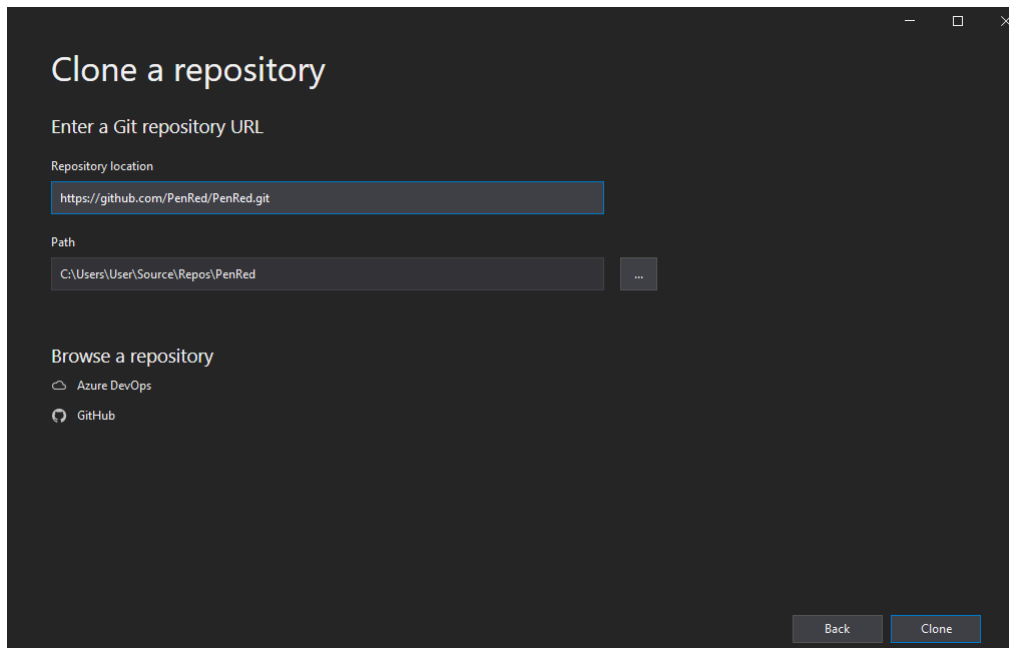


Figure 4: Clone PenRed repository with MSVS

The download will start automatically. Once the Cmake configuration ends, to avoid compiling with a debug profile, add a new configuration with the configuration manager (Figure 5).

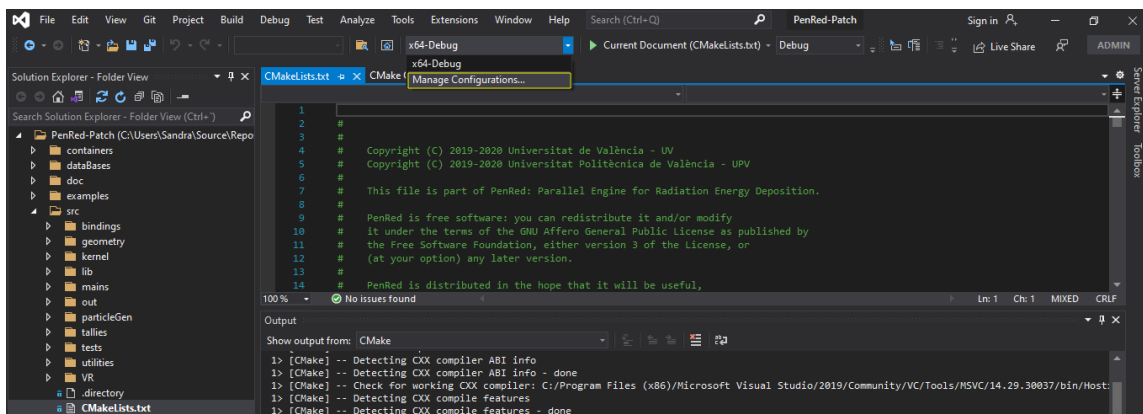


Figure 5: Manage MSVS configurations

In the configuration panel to the left, click on the button with the green *plus* sign to add a new configuration and select the release depending on your system. In the following image, we selected a release for 64-bit system (Figure 6).

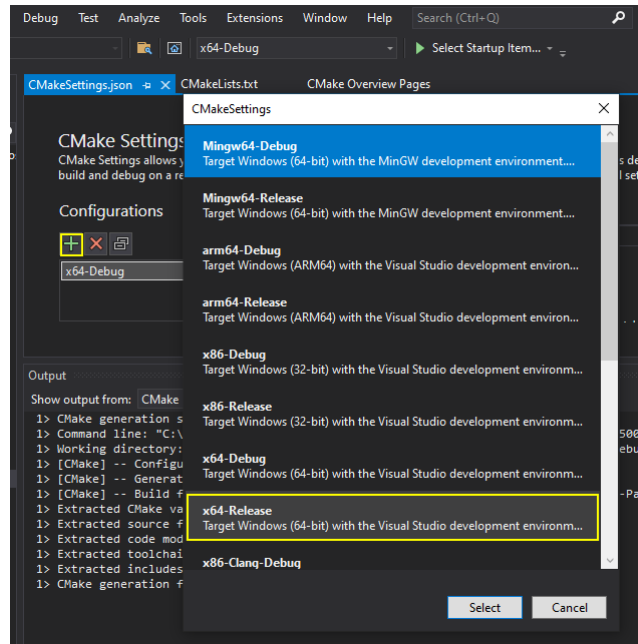


Figure 6: Add configuration in MSVS

Once you have selected the new configuration, and push on “Save and generate CMake cache to load variables” to be able to change the CMake variables for this configuration and compile it (Figure 7).

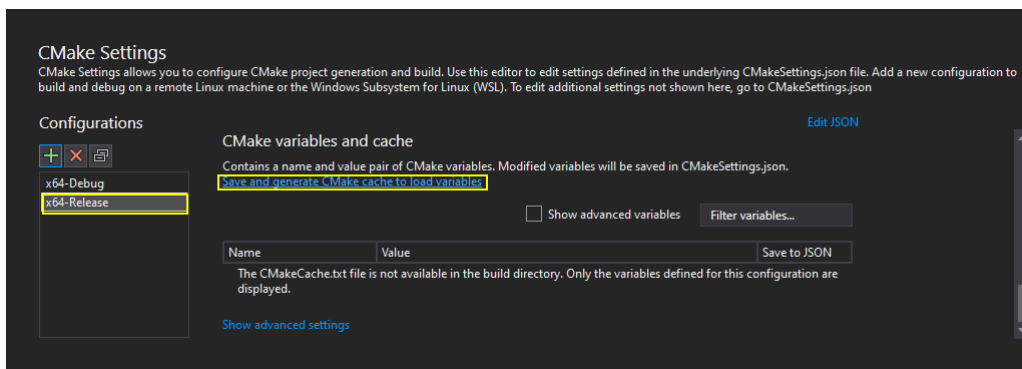


Figure 7: Select configuration in MSVS

Now, we can change all the Cmake configuration variables to enable or disable MPI support and other features (Figure 8).

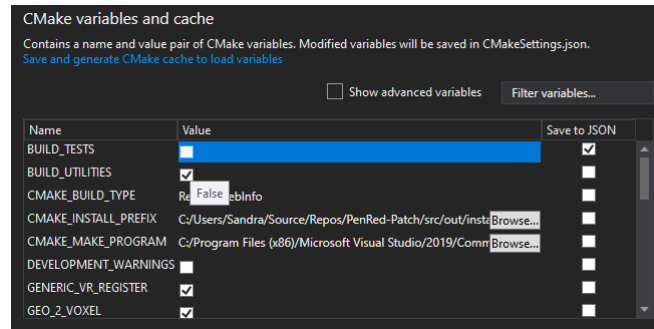


Figure 8: Set compilation flags in MSVS

Finally, build and install PenRed (Figure 9).

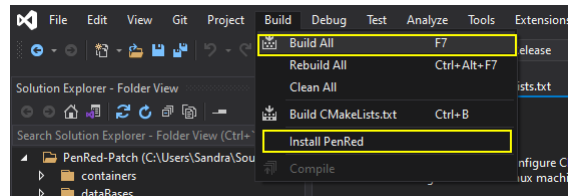


Figure 9: Build and install PenRed in MSVS

If the compilation finishes successfully, a new folder named *compiled* will be created inside the *src* folder containing the PenRed's main program and the executable for all other enabled utilities (Figure 10).

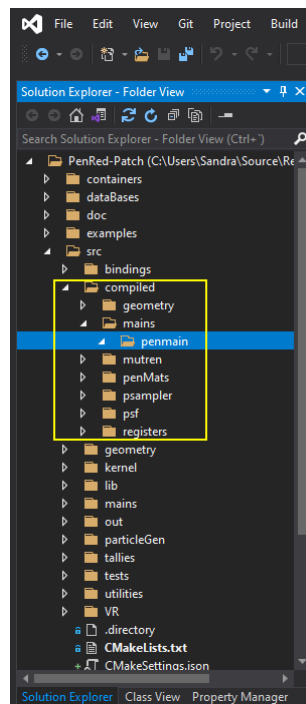


Figure 10: PenRed compiled programs

## 2.3 Basic usage

To execute the main program, the user needs a configuration file and, likely, the required database files, such as material and geometry files. Their paths should be specified in the configuration file, leaving the configuration file path as the only program argument. All details regarding the simulation configuration can be found in section 5. Additionally, the *examples* folder contains several configuration file examples with the corresponding material and geometry files ready to be executed (section 7). To execute the program, the user must use the command line instruction:

```
./pen_main path/to/configuration/file
```

Therefore, to run the provided examples, copy the main program executable in the corresponding example folder and run the instruction:

```
./pen_main config.in
```

If MPI capabilities have been enabled during compilation, the code should be executed as any MPI program. For example:

```
mpirun -np Nprocesses ./pen_main path/to/configuration/file
```

where *Nprocesses* specify the number of MPI processes to use. Of course, the user can use any other options of the *mpirun* command, such as specify the hosts where execute the code via the *hostfile* option.

## 2.4 Optional features

The following list summarise the optional compilation flags available during the PenRed compilation procedure.

- **BUILD\_UTILITIES:** Enables or disable the compilation of utilities, such as *geo2vox*, sampler, geometry and tally registers or load balance server. If enabled, the compilation of each utility can be enabled or disabled individually.
- **WITH\_MULTI\_THREADING:** Handles multithreading capabilities.
- **WITH\_MPI:** Handles MPI capabilities. Require an MPI library installed.
- **WITH\_NATIVE:** Switch on or off the compilation with native optimizations. Actually, this flag only works on GCC and Intel C++ compilers.
- **WITH\_DICOM:** Handles DICOM capabilities. Requires the *dcmtk* library.
- **WITH\_LB:** Enables or disables load balance capabilities. If enabled, some optional features related with the balance system can be configured:
  - **WITH\_SSL:** Enables or disables SSL secure connections for TCP or HTTPS communications. Requires OpenSSL libraries and its dependences.
  - **WITH\_HTTP:** Enables or disables connections to a HTTP/HTTPS rest api as balance server (Experimental feature). Requires the libCurl library.
- **BUILD\_TESTS:** Enables or disable the compilation of test codes.
- **DEVELOPMENT\_WARNINGS:** Enables or disables development warning flags.

### 3 Internal data format

#### src/kernel/parsers

To provide a unified format for input and configuration, PenRed implements a set of data structures. This implementation can be found in the following files,

*src/kernel/parsers/includes/pen\_data.hh*

and

*src/kernel/parsers/source/pen\_data.cpp*

The format used in PenRed is basically a *key/value* pair where the key format is based on unix folder system, i.e. a generic key is a string with the following structure:

*/folder1/folder2/.../element*

On the other hand, the value can be a number, bool (True or False), character, string or an array of numbers, bools or characters. Notice that these types can be combined in a single array. Also, strings must be made with double quotes ("text") to be parsed correctly. For example, code 1 shows a configuration for a cylindrical dose distribution tally.

```
1 tallies/cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"
2 tallies/cylDoseDistrib/print-xyz true
3 tallies/cylDoseDistrib/rmin 0.0
4 tallies/cylDoseDistrib/rmax 30.0
5 tallies/cylDoseDistrib/nbinsr 60
6 tallies/cylDoseDistrib/zmin 0
7 tallies/cylDoseDistrib/zmax 30.0
8 tallies/cylDoseDistrib/nbinsz 60
```

Code 1: Internal data example

#### 3.1 Implementation

This section provides a brief explanation of internal data implementation. PenRed internal data structure consists of four classes where each one uses the previous class to create a more complex structure. First, *pen\_parserData* is basically a union with four allowed types: *char*, *int*, *double* and *bool*. Notice that *pen\_parserData* can't be a string or pointer. Next, *pen\_parserArray* is a standard C++ vector of *pen\_parserData* variables. The third class is *pen\_parserElement*, which can store a C++ string or a *pen\_parserArray* variable. Finally, *pen\_parserSection* stores a C++ map where each key-value pair uses the types,

*< std::string|pen\_parserArray >*

where the left element is the pair key and the right element is the pair value. As we mentioned at previous section, key format follows the structure of Unix paths, i.e. ,

*/folder1/folder2/element.*

In our schema, each "folder" will be considered as a "section". On the other hand, a key with no sub-folders is considered an "element". Notice that keys whose include an element

path as a section are not allowed and the program will return an error or overwrite this element according to the called function. Thus, the following configuration can't exist in a input format map structure,

```
/folder1/folder2/element
/folder1/folder2/element/element2
```

since *element* is not a section and therefore cannot contain any other section nor element.

For further information and usage examples, consult the provided examples in *src/tests/internalData* folder.

## 4 Constants, parameters and definitions

This section summarises some important constants and values of the PenRed engine.

### 4.1 Important definitions

- **Void region/volume:** Geometry region with material index 0.
- **Primary particle:** Particle produced directly by a source and not as a consequence of any interaction of some other particle.
- **Secondary particle:** Particle produced by some interaction of other particle.
- **History:** A *history* consists of a primary particle and all its derived secondary particles.
- **Particle simulation:** Encompasses since the particle enters into a non void region of the geometry system, until it is absorbed or escaped from the system.
- **History simulation:** Includes all the particle simulations of the particles conforming the history.
- **Source simulation:** Includes all the history simulations due primary particles produced by the source.
- **Global simulation:** Consists of all source simulations.
- **Particle state:** All variables stored at the corresponding state structure, which base type should be the provided *pen\_particleState*.
- **Object state:** We refer as a object state to the values of its internal variables, regardless if they are private, public or protected. Also, inherited variables are considered as part of the object state. For example, lets see the objects defined at the code 2. The state of an object of the first type *A*, consists of the variables *a1*, *a2* and *a3*. On the other hand, the state of an object *B* consists of *a1*, *a2*, *a3*, *b1* and *b2*, due the inheritance from the class *A*.

```
1
2 class A{
3     protected:
4         int a1, a2;
5     public:
6         double a3;
7
```



```

8      A() : a1(0),a2(2),a3(5.3){}
9
10     void do1();
11     void do2();
12 };
13
14 class B : public A{
15     private:
16         int b1;
17     public:
18         double b2;
19
20     B() : b1(0),{}
21
22     void doB1();
23     void doB2();
24 };
25

```

Code 2: Source configuration parameters

- **Verbose levels:** Verbose levels consists of a set of non negative indexes that indicates how verbose will be our execution. The meaning of that levels are summarised following,

- 0: Any message should be printed.
- 1: Only error messages should be printed.
- 2: Errors, warnings and important information should be printed.
- > 2: Prints all relevant information.

Optionally, greater verbose levels could be used to filter very verbose information.

- **ILB values:** ILB labels follows the definition of the PENELOPE package, which are summarised at the table extracted from the PENELOPE manual [3]. Notice that our index begins from 0 because C++ syntax but at the FORTRAN code the first index is 1.

<b>ILB</b>	<b>description</b>
ILB[0]	Generation of the particle; 1 for primary particles, 2 for their direct descendants and so on. Primary particles are assumed to be labelled with $ILB[0] = 1$ .
ILB[1]	Parent particle <i>kpar</i> index (see table 3), only if $ILB[0] > 1$ .
ILB[2]	Interaction mechanism <i>ICOL</i> (see tables at section 4.4) that originated the particle, only when $ILB[0] > 1$ .
ILB[3]	<p>A non-zero value identifies particles emitted from atomic relaxation events and describes the atomic transition where the particle was released. The numerical value is,</p> $ILB[3] = Z \times 10^6 + IS1 \times 10^4 + IS2 \times 100 + IS3 \quad (1)$ <p>where <math>Z</math> is the atomic number of the emitting atom and <math>IS1</math>, <math>IS2</math> and <math>IS3</math> are the labels of the active atomic electron shells (see table 7).  For instance, <math>ILB[3] = 29010300</math> designates a <math>K - L2</math> x ray from copper (<math>Z = 29</math>), and <math>ILB[3] = 29010304</math> indicates a <math>K - L2 - L3</math> Auger electron from the same element.  When <math>ILB[3] \neq 0</math>, the value of <math>ILB[2]</math> indicates the interaction mechanism that caused the initial vacancy in the decaying atom.</p>
ILB[4]	This label can be defined by the user and must be transferred to all particle descendants.

Table 1: Description of the *ILB* components

## 4.2 Units

PenRed supposes the use of specific units internally. These units are summarised at the table 2.

<b>Magnitude</b>	<b>Unit</b>
Length	cm
Energy	eV
Time	s
Material mass	g
Density	g/cm <sup>3</sup>

Table 2: PenRed internal units.

## 4.3 Particle indexes

In penRed, each particle has a text identifier, known as particle name, and a numeric identifier set by an internal enumeration. Both values for each particle are summarised in the Table 3.

<b>particle name</b>	<b>Enumeration identifier</b>	<b>Numerical index</b>
electron	PEN_ELECTRON	0
gamma	PEN_PHOTON	1
positron	PEN_POSITRON	2

Table 3: Particle indexes and names.

#### 4.4 Particle interaction indexes

PenRed uses enumerations to indexing the interactions of each particle. This section show the corresponding index for each particle interaction.

<b>Interaction</b>	<b>Enumeration identifier</b>	<b>Numerical index</b>
Elastic collision	BETAe_HARD_ELASTIC	0
Inelastic collision	BETAe_HARD_INELASTIC	1
Bremsstrahlung	BETAe_HARD_BREMSSTRAHLUNG	2
Inner shell interaction	BETAe_HARD_INNER_SHELL	3
Delta interaction	BETAe_DELTA	4
Soft interaction	BETAe_SOFT_INTERACTION	5

Table 4: Electron interaction indexes.

<b>Interaction</b>	<b>Enumeration identifier</b>	<b>Numerical index</b>
Rayleigh	GAMMA_RAYLEIGH	0
Compton	GAMMA_COMPTON	1
Photoelectric	GAMMA_PHOTOELECTRIC	2
Pair production	GAMMA_PAIR_PRODUCTION	3
Delta interaction	GAMMA_DELTA	4

Table 5: Photon interaction indexes.

<b>Interaction</b>	<b>Enumeration identifier</b>	<b>Numerical index</b>
Elastic collision	BETAp_HARD_ELASTIC	0
Inelastic collision	BETAp_HARD_INELASTIC	1
Bremsstrahlung	BETAp_HARD_BREMSSTRAHLUNG	2
Inner shell interaction	BETAp_HARD_INNER_SHELL	3
Annihilation	BETAp_ANNIHILATION	4
Delta interaction	BETAp_DELTA	5
Soft interaction	BETAp_SOFT_INTERACTION	6

Table 6: Positron interaction indexes.

#### 4.5 Atomic electron shells

The atomic electron shells labels are designed following the indexes used at PENELOPE code. These indexes are summarised at the table 7, which has been extracted from the PENELOPE manual [3].

Label	Shell	Label	Shell	Label	Shell
1	K ( $1s_{1/2}$ )	11	N2 ( $4p_{1/2}$ )	21	O5 ( $5d_{5/2}$ )
2	L1 ( $2s_{1/2}$ )	12	N3 ( $4p_{3/2}$ )	22	O6 ( $5f_{5/2}$ )
3	L2 ( $2p_{1/2}$ )	13	N4 ( $4d_{3/2}$ )	23	O7 ( $5f_{7/2}$ )
4	L3 ( $2p_{3/2}$ )	14	N5 ( $4d_{5/2}$ )	24	P1 ( $6s_{1/2}$ )
5	M1 ( $3s_{1/2}$ )	15	N6 ( $4f_{5/2}$ )	25	P2 ( $6p_{1/2}$ )
6	M2 ( $3p_{1/2}$ )	16	N7 ( $4f_{7/2}$ )	26	P3 ( $6p_{3/2}$ )
7	M3 ( $3p_{3/2}$ )	17	O1 ( $5s_{1/2}$ )	27	P4 ( $6d_{3/2}$ )
8	M4 ( $3d_{3/2}$ )	18	O2 ( $5p_{1/2}$ )	28	P5 ( $6d_{5/2}$ )
9	M5 ( $3d_{5/2}$ )	19	O3 ( $5p_{3/2}$ )	29	Q1 ( $7s_{1/2}$ )
10	N1 ( $4s_{1/2}$ )	20	O4 ( $5d_{3/2}$ )	30	outer shells

Table 7: Atomic electron shells indexes.

## 5 Framework usage

This section, address how to use the main program provided within the PenRed package. This one allows the user to make simulations without programming. Instead, the user will use a configuration file which structure is explained following.

First, all configuration files must follow the PenRed’s internal data library format, which has been explained in section 3. Secondly, all configuration files must include some mandatory sections to specify the simulation characteristics. These sections are, sources, geometry, materials, tallies and global parameters, and all of them are described in the following sections.

### 5.1 Materials

The material properties, such as cross sections, density, components, etc. are described inside material files. These ones can be generated on runtime, specifying the composition in the configuration as will be discussed, or using an external program. Moreover, as the files follows the same structure as those used in the original PENELOPE FORTRAN code, material files can be generated using the tools and data bases provided at original PENELOPE package, which are also included in the PenRed package. The material data base is located in the folder

`dataBases/penmaterials`

where both, the program code to build materials and the database are stored. That code is a literal translation to C++ of the original FORTRAN code used by PENELOPE to create the materials. Thus, outputs of both C++ and FORTRAN versions must be perfectly equivalent and usable as input for PenRed and PENELOPE. To compile the material build program, the user must compile the “material.cpp” file. For instance, the following line can be used to compile this code using the C++ GNU GCC compiler,

```
g++ -o createMat material.cpp
```

where *createMat* is the executable name. Once the code has been compiled, execute it in the same folder where the database is (folder *pdfiles*) and follow the program instructions to create the material. As in the FORTRAN code, a set of predefined materials can be build via numerical codes, which are specified in the appendix A, in the tables 10 and 11.

### 5.1.1 Configuration

To specify a material in the configuration file, the user must adhere to the pattern shown in code 3. Here, *materials* denotes a constant text, *material-name* serves as a user-selected text identifier or name, and *parameter/path* and *value* represent the parameters and values to be specified for that material.

```
1 materials/material-name/parameter/path value
```

Code 3: Material configuration pattern

For instance, the code 4 shows the copper material configuration used in the first example provided in the package.

```
1 materials/cu/number 1
2
3 materials/cu/eabs/electron 1.0e3
4 materials/cu/eabs/positron 1.0e3
5 materials/cu/eabs/gamma 1.0e3
6
7 materials/cu/C1 0.05
8 materials/cu/C2 0.05
9
10 materials/cu/WCC 1.0e3
11 materials/cu/WCR 1.0e3
12
13 materials/cu/filename "Cu.mat"
```

Code 4: Complete material configuration example

As shown, the parameters required for each material specification are analogous to those needed by the PENELOPE FORTRAN version. Initially, the material *number* is employed to identify the material in the geometry system. This number must be greater than 0, with material 0 representing void, and lesser or equal to the total number of materials.

Other parameters are optional and can be set automatically by penRed default values, with only the material number, and the filename if the material composition is not provided, being mandatory parameters. However, it is advisable to select these parameters based on the simulation characteristics. In the following sections, all available configuration parameters for materials will be described.

#### 5.1.1.1 Composition

A material can be automatically created using the *elements* section. In this section, each path defines an element's atomic mass, and the corresponding value specifies the fraction by mass of this element in the material composition. This pattern is illustrated in Code 5.

```
1 materials/material-name/elements/Z fraction-by-mass-weight
```

Code 5: Material composition configuration pattern

For example, to define the PMMA material in the *8-fake-chamber* examples, the following configuration can be used:

```
1 materials/chamberTop/number 3
2
3 materials/chamberTop/elements/1 0.080538
4 materials/chamberTop/elements/6 0.599848
5 materials/chamberTop/elements/8 0.319614
6 materials/chamberTop/density 1.19500000E+00
7 materials/chamberTop/force-creation true
8
9 materials/chamberTop/eabs/electron 1.0e3
```

```

10 materials/chamberTop/eabs/positron 1.0e3
11 materials/chamberTop/eabs/gamma 1.0e3
12
13 materials/chamberTop/C1 0.00
14 materials/chamberTop/C2 0.00
15
16 materials/chamberTop/WCC 0.0
17 materials/chamberTop/WCR 0.0
18
19 materials/chamberTop/filename "pmma.mat"

```

Code 6: Material composition configuration pattern

In Code 6, the parameters for material file creation, in addition to the already discussed *elements* subsection, are:

- **density:** Defines the material density in  $\text{g/cm}^3$ .
- **force-creation:** This is an optional parameter that enables/disables forcing the creation of the material file. If the file already exists, the program will skip its creation and process the existing one instead, as default behaviour.
- **filename:** This parameter is mandatory if the **elements** section is not defined. It specifies the material file where the material data will be read, or written if it is created during execution.

#### 5.1.1.2 Compoistion databases

Instead of manually specifying the composition of a material, users can load a material composition from the available databases. To view the available databases and included materials, use the *compositionsDB* utility (Section 8.10).

To specify a material in a database, the following values must be specified:

- **DB:** Name of the database.
- **DB-material:** Name of the DB material.

For example, the following code generate a material from the *ICRP\_AF* database:

```

1 materials/humeri/number 3
2
3 materials/humeri/DB "ICRP_AF"
4 materials/humeri/DB-material "Humeri_upper_cortical"
5 materials/humeri/filename "humeri.mat"

```

Code 7: Material composition DB configuration pattern

#### 5.1.1.3 Absorption energy

Absorption energy can be provided on both a global and local scale for each particle. When a particle reaches its absorption energy threshold, it will be absorbed within its respective material. Similarly, if secondary particles are generated with a energy below this threshold, they will be considered locally absorbed and will not be simulated. To set a default absorption energy value for all materials, use the following path:

```
material-eabs/particle-name value
```

Alternatively, to apply the absorption energy to a specific material, use the following path:

```
materials/material-name/eabs/particle-name value
```

Here, *material-name* represents the user-defined name of the material, *particle-name* (Table 3) refers to the particle to which the material-specific absorption energy will be applied, and *value* denotes the absorption energy in electronvolts (eV). If no global or local values are specified, a default value of 1 keV will be used.

#### 5.1.1.4 Limiting range

Instead of absorption energy, the minimum range to simulate each particle type can be defined. Thus, a particle whose energy corresponds to a smaller *range* than the one specified will be absorbed. The interpretation of the range depends on the particle:

- **gamma:** The range is defined as the particle mean free path (inverse attenuation coefficient), in cm.
- **electrons and positrons:** The range is interpreted as the Continuous Slowing Down Approximation (CSDA) range, in cm.

To specify this parameter, the following pattern must be used in the corresponding material configuration section,

```
materials/material-name/range/particle-name value
```

where *range* is a constant key and *particle-name* must be replaced by the particle name to which the range constraint is to be applied. Finally, the *value* specify the limiting range in cm.

In addition, global particle limiting ranges can be defined for all materials. These can be specified following the pattern,

```
material-ranges/particle-name value
```

outside the material section. There, *material-ranges* is a constant key, *particle-name* is the particle name to apply the limiting range and *value* specify the limiting range in cm.

Notice that default range values can be overwritten by local material ranges. For example, the configuration shown in the Code 8 sets the the default range for electron and gamma to 0.1 cm for all materials. Then, a *infinite* absorption energy is set to positrons to avoid their transport. Finally, in the *water* material, the default range value is overwritten for gammas to 0.05 cm. Because the electron transport parameters are not specified, all materials use the default ones.

```
1
2 material-ranges/electrons 0.1
3 material-ranges/gamma 0.1
4
5 materials/cu/number 1
6 materials/cu/eabs/positron 1.0e35
7 materials/cu/filename "Cu.mat"
8
9 materials/Fe/number 2
10 materials/Fe/eabs/positron 1.0e35
```

```

11 materials/Fe/filename "Fe.mat"
12
13 materials/water/number 3
14 materials/water/range/gamma 0.05
15 materials/water/eabs/positron 1.0e35
16 materials/water/filename "H2O.mat"

```

Code 8: Configuration example with different absorption limits

#### 5.1.1.5 Electron and positron transport parameters

Next, the  $C1$ ,  $C2$ ,  $WCC$  and  $WCR$  parameters are used to control the class II transport of electrons and positrons. To be able to select the appropriate parameters, the corresponding description follows, which has been extracted from the PENELOPE manual [3], where are further explained,

- **C1:** Average angular deflection,  $C1 \approx 1 - \langle \cos\theta \rangle$ , produced by multiple elastic scattering along a path length equal to the mean free path between consecutive hard elastic events. The maximum allowed value is 0.2, but a value of 0.05 is usually adequate.
- **C2:** Maximum average fractional energy loss between consecutive hard elastic events. As  $C1$ , the maximum allowed value is 0.2 and a value of about 0.05 is, usually, adequate.
- **WCC:** Cutoff energy loss, in eV, for hard inelastic collisions.
- **WCR:** Cutoff energy loss, in eV, for hard bremsstrahlung emission.

The election of these parameters determine the simulation trade off between speed and accuracy. For better accuracy,  $C1$  and  $C2$  should have small values (0.01). Larger values makes the simulation faster but less accurate. On the other hand, the cutoff energies  $WCC$  and  $WCR$  speed up the simulation when using larger values, but if these are too large, the tallied energy distributions could be distorted. To avoid this effect, the values of  $WCC$  and  $WCR$  should be lesser than the bin width used to tally the energy distributions. Finally, to ensure the reliability of the simulation, the user must ensure that the number of steps, or random hinges, per primary track is “statistically sufficient”, which should be achieved with more than 10 steps.

If the mentioned parameters are not specified by the user, PenRed will set the parameters automatically.

#### 5.1.1.6 Filename

Finally, the *filename* parameter specify the relative path to the material file. The path to the material file must be enclosed by quotes to be identified as text.

Notice that the configuration shown in the code 4 must be repeated for each used material, changing the corresponding *name* and parameter values.

## 5.2 Particle sources

Particle sources handle the creation of particles to be simulated. The created particles can be both primary or secondary depending on the selected source. In addition, multiple sources can be specified in the same configuration file. In this case, the sources will be



computed sequentially, simulating the first particle of a source after the simulations end of the last particle of the previous source.

To specify a particle source, the user must follow the pattern shown in code 9,

```
1 sources/type/name/parameter/path value
```

Code 9: Source configuration parameters

where *type* must be set to *generic* or *polarized*, whose are used to specify if the source samples generic particle states or gamma polarized states respectively. Via the field *name*, the user specifies a custom name for the particle source. Finally, *parameter/path* specifies the configuration parameter to set with the corresponding *value*. For example, the code 10 shows the basic parameters used by all generic sources type, which name has been set to *source1*.

```
1 sources/generic/source1/nhist 1.0e6 (mandatory)
2 sources/generic/source1/kpar "gamma" (mandatory)
3 sources/generic/source1/record-time true (optional)
4 sources/generic/source1/source-body 1 (optional)
5 sources/generic/source1/source-material 1 (optional)
```

Code 10: Source generic configuration parameters

That source will produce  $10^6$  primary gamma particles and enables time recording. So the parameter *nhist*, *kpar* and *record-time* sets the number of particles to generate by the source, the particles type (electron, gamma or positron) and if time recording is enabled or disabled respectively.

In addition we can specify if the particles creation must be restricted to a specific body or material using the parameters *source-body* and *source-material* respectively. Both parameters expects an integer index, which specify the source body or material. However, notice that only one restriction can be used for each source. If specified, when a particle is sampled in a different body or material, the spatial sampling will be repeated until the condition is fulfilled.

Note that each parameter requires a specific type. *nhist* requires a number, *kpar* a text (using the double quotes) and *record-time* a boolean (true or false). Now, we have the generic parameters, but the sampler has not knowledge about how to sampling the initial state of the particle, i.e. position, direction, energy, time etc. Therefore, we need to specify the samplers to use, which are classified in spatial, direction, energy, time and specific samplers. Accordingly, to specify what kind of sampler we are configuring, we will use the keywords *spatial*, *direction*, *energy*, *time* and *specific*. Each sampler type is used with a specific purpose, being spatial, direction, energy and time samplers in charge of determine the particle initial position, direction, energy and live time, respectively. These samplers are cataloged as generic samplers. Nevertheless, specific samplers can set the whole particle state or a portion of it. Therefore, specific samplers can be combined with generic ones depending on the sampler implementation or handle the whole state itself.

Although each sampler has its own parameters, whose can include custom paths, all of them require a parameter named *type*, which sets the sampler to use. For example, to select a mono-energetic energy sampler on the previous source, the line with the required type is shown in code 11. Notice that the source type has been specified after the source name, i.e. with the keyword *energy* and the type value specify which sampler to use among all the available energy samplers.

```
1 sources/generic/source1/energy/type "MONOENERGETIC"
```

Code 11: Mono-energetic sampler configuration example

In addition, mono-energetic sources only require a single parameter, the particle energy. The next line sets a sampling energy of 30 keV,

```
1 sources/generic/source1/energy/energy 3.0e7
```

Code 12: Mono-energetic sampler configuration example

The other samplers are configured analogously, but using their own parameters. To put it all together, a complete generic source configuration is shown in the code 13, i.e. with no specific sampler. In that code, the lines that begins with `#` are considered comments and ignored by the parser.

```
1 #-----
2 # Source 1
3 #####
4
5 sources/generic/source1/nhist 1.0e6
6 sources/generic/source1/kpar "gamma"
7 sources/generic/source1/record-time true
8
9 # Directional sampling
10 #####
11
12 sources/generic/source1/direction/type "CONE"
13
14
15 # Set theta
16 sources/generic/source1/direction/theta 0.0
17
18 # Set phi
19 sources/generic/source1/direction/phi 0.0
20
21 # Set oberture (alpha)
22 sources/generic/source1/direction/alpha 5.0
23
24
25 # Energy sampling
26 #####
27
28 sources/generic/source1/energy/type "MONOENERGETIC"
29
30 # Set energy
31 sources/generic/source1/energy/energy 3.0e7
32
33
34 # Spatial sampling
35 #####
36
37 sources/generic/source1/spatial/type "POINT"
38
39 # Set particle origin
40 sources/generic/source1/spatial/position/x 0.0
41 sources/generic/source1/spatial/position/y 0.0
42 sources/generic/source1/spatial/position/z -25.0
```

Code 13: Complete source configuration example

A source without specific sampler, like the previous one, requires at least spatial, energy and direction samplers, being the time sampler optional. If a specific sampler is used, the required generic samplers depends on the specific sampler itself, and may need some or any generic sampler. In the examples folder we can find configuration files with different sources. In addition, all the available samplers classified by type will be described following.

## 5.2.1 Spatial source samplers

### 5.2.1.1 Point

The spatial point sampler needs the  $(x, y, z)$  coordinates to fill the position of generation particles. These values are doubles and are specified in cm. Following there is a configuration example of this kind of source sampling corresponding to the 1-disc example.

```
1 # Spatial sampling
2 #####
3
4 sources/generic/source1/spatial/type "POINT"
5
6 # Set particle origin
7 sources/generic/source1/spatial/position/x 0.0
8 sources/generic/source1/spatial/position/y 0.0
9 sources/generic/source1/spatial/position/z -0.0001
```

Code 14: Spatial source POINT sampler

### 5.2.1.2 Box

This spatial sampler creates a box where particles are generated with uniform probability. The configuration file contains the information of the box origin introducing the  $(x, y, z)$  coordinates values, in cm. In addition, the configuration requires the size of the box in each axis  $(dx, dy, dz)$  in cm, which must be positive or zero. All these values are interpreted as doubles. An example of configuration file is shown below.

```
1 # Spatial sampling
2 #####
3
4 sources/generic/source1/spatial/type "BOX"
5
6
7 # Set box size
8 sources/generic/source1/spatial/size/dx 0.75
9 sources/generic/source1/spatial/size/dy 0.75
10 sources/generic/source1/spatial/size/dz 1.25
11
12 # Set particle origin
13 sources/generic/source1/spatial/position/x 0.0
14 sources/generic/source1/spatial/position/y 0.0
15 sources/generic/source1/spatial/position/z -0.0001
```

Code 15: Spatial source BOX sampler

### 5.2.1.3 Cylindrical shell

This spatial sampler generates particles within a cylindrical shell. It is defined by subtracting a smaller cylinder from a bigger one where it is included. The sampling region corresponds to the shaded area in the Figure 11.

Notice that the center of both cylinders coincide. The parameters to define the region are listed following:

- **type:** Must be set to “CYLINDER”.
- **size/rmin:** Inner cylinder radius in cm.
- **size/rmax:** External cylinder radius in cm.

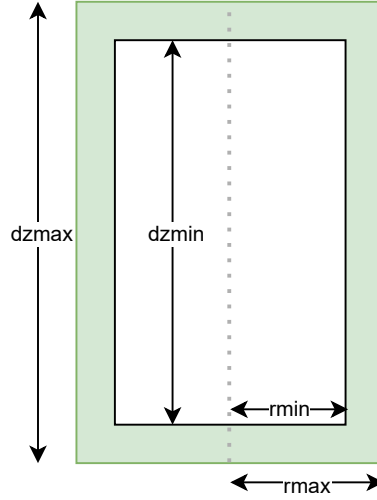


Figure 11: Cylindrical shell sampling region and values to define.

- **size/dzmin**: Inner cylinder height in cm.
- **size/dzmax**: External cylinder height in cm
- **position/x**:  $X$  coordinate of the cylinders center in cm.
- **position/y**:  $Y$  coordinate of the cylinders center in cm.
- **position/z**:  $Z$  coordinate of the cylinders center in cm.

For example, the Code 16 shows the configuration to sample particles in a thin layer of radioactive paint on a brachytherapy seed located at the origin.

```

1 # Spatial sampling
2 #####
3
4 sources/generic/source1/spatial/type "CYLINDER"
5
6 # Set cyl size
7 sources/generic/source1/spatial/size/rmin 0.025
8 sources/generic/source1/spatial/size/rmax 0.025175
9 sources/generic/source1/spatial/size/dzmin 0.28
10 sources/generic/source1/spatial/size/dzmax 0.295
11
12 # Set position
13 sources/generic/source1/spatial/position/x 0.0
14 sources/generic/source1/spatial/position/y 0.0
15 sources/generic/source1/spatial/position/z 0.0

```

Code 16: Spatial source CYLINDER sampler

#### 5.2.1.4 Measure (1D,2D,3D)

This family of samplers uses 1D, 2D, or 3D distributions to determine the spatial position of generated particles. These distributions can be obtained directly from the *Detection Spatial Distribution* tally (Section 5.4.17).

Regardless of the number of dimensions used, the samplers share the following common parameters:

- **filename:** Specifies the path to the file containing the spatial distribution.
- **dx:** Specifies a displacement, in cm, applied to the distribution along the  $X$  axis. This value is optional and defaults to 0.
- **dy:** Specifies a displacement, in cm, applied to the distribution along the  $Y$  axis. This value is optional and defaults to 0.
- **dz:** Specifies a displacement, in cm, applied to the distribution along the  $Z$  axis. This value is optional and defaults to 0.

The dimension-specific parameters are as follows:

- **3D:**
  - **Type:** The tally type must be set to “3D\_MEASURE”.
- **2D:**
  - **Type:** The tally type must be set to “2D\_MEASURE”.
  - **plane:** Specifies the plane to be sampled, using one of the following values: “xy”, “xz”, “yz”.
  - **constant-coordinate:** Specifies the value of the constant coordinate, in cm, according to the selected plane. For example, if the “xy” plane is selected, this parameter fixes the coordinate along the  $Z$  axis.
- **1D:**
  - **Type:** The tally type must be set to 1D\_MEASURE”.
  - **axis:** Specifies which axis will be sampled. The allowed values are “x”, “y”, and “z”.
  - **x:** Sets the constant value along the  $X$  axis. Defaults to 0.
  - **y:** Sets the constant value along the  $Y$  axis. Defaults to 0.
  - **z:** Sets the constant value along the  $Z$  axis. Defaults to 0.

An example configuration is shown in code 17 for the 2D case.

```

1 sources/generic/source1/spatial/type "2D_MEASURE"
2
3 sources/generic/source1/spatial/filename "source.dat"
4
5 sources/generic/source1/spatial/constant-coordinate 0.0
6 sources/generic/source1/spatial/plane "xy"
7
8
9 sources/generic/source1/spatial/dx 0.0
10 sources/generic/source1/spatial/dy 0.0
11 sources/generic/source1/spatial/dz 0.0

```

Code 17: 2D Measure sampler configuration example

## 5.2.2 Direction samplers

### 5.2.2.1 Cone

Particles generated by the source are sampled in a conical beam. In this sample the overture of the cone must be specified to sample the particles direction uniformly inside the corresponding solid angle. Therefore, the required parameters are the polar (*theta*) and azimuthal (*phi*) angles to determine the solid angle direction, and the semiaperture (*alpha*). Notice that all values are expected in degrees. To reproduce a monodirectional source, the overture must be set to *alpha* = 0.0. However, to obtain a isotropic source, the semiaperture must be set to *alpha* = 180.0. All values are expected to be doubles.

The following lines show the directional sampling of the configuration file of example 2-plane.

```
1
2 # Directional sampling
3 #####
4
5 sources/generic/source1/direction/type "CONE"
6
7
8 # Set theta
9 sources/generic/source1/direction/theta 0.0
10
11 # Set phi
12 sources/generic/source1/direction/phi 0.0
13
14 # Set oberture (alpha)
15 sources/generic/source1/direction/alpha 5.0
```

Code 18: Direction CONE sampler

### 5.2.2.2 Sphere

In this case the direction of the particles generated are sampled in a sphere section. The information needed in this directional sampler is the direction components in each axis (*u, v, w*), the polar overture introducing the minimum and maximum value of the polar angle (*theta0, theta1*), and the azimuthal overture introducing the minimum value of the phi angle (*phi0*) and its overture (*dphi*). All of this values types must be doubles. The Core 19 shows an example corresponding to a sphere section pointing to +Z axis with a symmetric overture from -30 to 30 degrees in all the azimuthal interval.

```
1
2 # Directional sampling
3 #####
4
5 sources/generic/source1/direction/type "SOLID_ANGLE"
6
7
8 # Set direction vector
9 sources/generic/source1/direction/u 0.0
10 sources/generic/source1/direction/v 0.0
11 sources/generic/source1/direction/w 1.0
12
13 # Set polar overture
14
15 sources/generic/source1/direction/theta0 -30.0
16 sources/generic/source1/direction/theta1 30.0
17
```

```

18
19 # Set azimuthal overture
20 sources/generic/source1/direction/phi0 0.0
21 sources/generic/source1/direction/dphi 360.0

```

Code 19: Direction SOLID\_ANGLE sampler

## 5.2.3 Energy samplers

### 5.2.3.1 Monoenergetic

This energy sampler generates a monoenergetic beam of particles with the specified energy in eV. The only required parameter is the *energy* value, which is interpreted as double and expected to be positive. An example of its usage can be found in the configuration file shown below. These lines correspond to the example 4-x-ray-tube, file `tube.in`.

```

1
2 # Energy sampling
3 #####
4
5 sources/generic/source1/energy/type "MONOENERGETIC"
6
7 # Set energy
8 sources/generic/source1/energy/energy 1.5e5

```

Code 20: Energy samplers MONOENERGETIC

### 5.2.3.2 Intervals

Intervals sampler generates energies within the specified spectral lines according to the probability assigned to each one. To define the spectral lines in the configuration file, first, the user must specify the number of intervals, *ninterval*, using a integer value. Then, the energy range of each interval,  $[lowE, topE]$ , are specified as follows. A single array named *lowE* will contain all the low energy boundaries of all energy intervals. In the same way, all values for top energy limits must be grouped in an array named *topE*. Finally, the probabilities of each interval are wrote in a third array named *probabilities*. Therefore, the first position of the array *lowE* specify the low boundary for the first interval or spectral line, the first position of the array *topE* the top limit and the first position of the *probabilities* array the corresponding probability. The same is applied to the second position, which specify the second interval or spectral line, and so on. Notice that the probabilities are not required to be normalised, as is shown in the following example code, which corresponds to the 3-detector example, file `detector1.in`. Regarding the variable types, the energy range values and probabilities are doubles and must be, at least, zero.

```

1
2 # Energy sampling
3 #####
4
5 sources/generic/source1/energy/type "INTERVALS"
6
7
8 # Set intervals
9 sources/generic/source1/energy/nintervals 2
10
11
12 # Set energies
13 sources/generic/source1/energy/lowE [1.17e6,1.33e6]
14 sources/generic/source1/energy/topE [1.17e6,1.33e6]

```

```

15
16 # Set probabilities
17 sources/generic/source1/energy/probabilities [50.0,50.0]

```

Code 21: Energy samplers INTERVALS

### 5.2.3.3 File spectrum

This energy sampler has been implemented to provide compatibility between the PenEasy [7] (v.2020-03-25) energy spectrum format and PenRed. Although this one is described in the PenEasy manual [8], it is also described following for the user convenience.

The format consists on a piecewise function where each entry in the spectrum contains two numbers. The first one, specify the starting energy of a channel, and the second its unnormalized probability. The probabilities must be non-negative and don't require to be normalized to unity. The program interprets a negative probability as the end of the spectrum. For example, a valid spectrum format is shown in the code 22, where the character `#` is interpreted as a comment. Notice that the spectrum file must contain only comments or valid spectrum values until the end of the spectrum. For example, blank lines are not allowed. However, all the text beyond the end of the spectrum is ignored by the sampler.

```

1 #Spectrum      (character '#' is interpreted as a comment)
2 1.20e3 10      1st channel: [1.20,1.61]keV
3 1.61e3 0.0     2nd channel: [1.61,5.00]keV, no emissions
4 5.00e3 20      monoenergetic line at 5.00 keV
5 5.00e3 0       3rd channel: [5.00,6.24]keV, no emissions
6 6.24e3 33.3    4th channel: [6.24,7.32]keV
7 7.32e3 5.02    5th channel: [7.32,8.00]keV
8 8.00e3 10.2    6th channel: [8.00,9.76]keV
9 9.76e3 15.2    7th channel: [9.76,10.0]keV
10 10.0e3 0.0     8th channel: [10.0,15.0]keV, no emissions
11 15.0e3 20      monoenergetic line at 15.0 keV
12 15.0e3 -1      end of the spectrum

```

Code 22: File energy spectrum format

This sampler only requires two configuration values. As the other samplers, the first one is the *type*, which must be set to *"FILE\_SPECTRUM"*. The other is the spectrum filename which is introduced as a string, as is shown in the code 23.

```

1
2 # Energy sampling
3 #####
4
5 sources/generic/source1/energy/type "FILE_SPECTRUM"
6 sources/generic/source1/energy/filename "spectrum.spc"

```

Code 23: File spectrum configuration

## 5.2.4 Time samplers

### 5.2.4.1 Decay

This sampler randomly generates event times based on the exponential decay characteristic of a radioactive element. The sampled time is constrained by user-specified initial and final times, and the exponential decay follows the radioisotope's half-life. The parameters required are:



- **halfLife:** The half-life of the radioactive element, in seconds. Must be a positive value.
- **time/time0:** The initial time for the decay interval, in seconds. Must be a positive value.
- **time/time1:** The final time for the decay interval, in seconds. Must be a positive value.

The decay follows an exponential distribution, parameterized by the specified half-life, which governs the probability distribution of event occurrences over time.

## 5.2.5 Specific Sources

### 5.2.5.1 Phase Space File

This type of source uses the phase space file created with the tally type “PSF” to run the simulation, from where the particles will be read. An example where this type of source is used is the example 5-accelerator-2, which configuration is shown in the code 24.

Regarding the required parameters, the *filename* is the relative path to of the file from where the particles will be read, and the parameter *E<sub>max</sub>* specify the maximum energy value of all particles in the PSF. Then, the two next parameters are related with VR techniques. The first one, *wght-window*, sets the interval to apply splitting and Russian Roulette. How both VR techniques are applied is explained following. First, Russian roulette is applied to particles with a weight value lower than the minimum value of the interval. Instead, splitting is applied to particles with a weight value larger than the maximum value of this interval. Therefore, particles with a weight value inside the interval remain unaltered. The second parameter, *split*, sets the number of splits per particle.

Finally there are two groups of optional parameters to be able to rotate and translate the particles in the PSF file. First, the rotation is specified as a ‘ZYX’ rotation using the Euler angles *omega*, *theta* and *phi*. Then, the user can apply a displacement using the *dx* parameter for translations in the *x* axis, and *dy* and *dz* for *y*, *z* axes respectively. Notice that the rotation is applied before the translation.

```

1 # Source 1
2 #####
3
4 sources/generic/source-psf/nhist 1.0e6
5 sources/generic/source-psf/specific/type "PSF"
6 sources/generic/source-psf/specific/filename "psf-merged.dat"
7 sources/generic/source-psf/specific/Emax 7e6
8
9 sources/generic/source-psf/specific/wght-window [5e-4, 1e-3]
10 sources/generic/source-psf/specific/nsplit 10
11
12 sources/generic/source-psf/specific/rotation/omega 0
13 sources/generic/source-psf/specific/rotation/theta 90
14 sources/generic/source-psf/specific/rotation/phi 0
15
16 sources/generic/source-psf/specific/translation/dx 0
17 sources/generic/source-psf/specific/translation/dy 0
18 sources/generic/source-psf/specific/translation/dz 10

```

Code 24: Phase Space File Source

### 5.2.5.2 Computed Tomography (CT)

This source has been developed to emit particles along a circumference by two different ways. The particles are emitted following discrete and regular steps specified by the user. On the one hand, reading the particles from a previously generated Phase Space File.

As this source uses internally the Phase Space File source (section 5.2.5.1), it inherits some parameters already defined in the PSF source. That parameters are: *filename*, *E<sub>max</sub>*, *wght-window*, *nsplit*. Moreover, the PSF translation and rotation can be applied to this source.

On the other hand, a generic source can be used with the corresponding directional, energy and spatial sampling. Additionally, as the output particles in a CT accelerator are, usually, secondary particles, a field named *nSecondaries* must be set to specify the number of secondary particles sampled by each history. Therefore, in a single history, *nSecondaries* particles will be sampled before increasing the history number.

Regarding the source type, it must be set to “CT” in the configuration file. Then, the specific parameters to configure the CT source are divided in three subsections, which are described following,

- **Angular information:** On the one hand, the parameters *phi-ini* and *phi-end* specify the initial and final angular position of the CT source. On the other hand, the parameter *angularStep* sets the angular step between two consecutive projections. All values must be set in degrees.
- **Position information:** To specify the source position, the [*CTx0*, *CTy0*, *CTz0*] is required. This corresponds to the CT origin, where the source will be translated. Finally, the CT radius, (*rad*), is needed to apply the final source translation to this point.
- **Time information:** The time between projections must be set using the parameters *tmin*, for the minimum time stamp of the psf particles, and *dt*, for the time interval.

An example of CT source using PSF generated previously is shown in 25, while an example in which a generic source is used is shown in the code 26.

```
1 # Source 1
2 #####
3
4 sources/generic/source-ct/nhist 1.5e8
5 sources/generic/source-ct/specific/type "CT"
6 sources/generic/source-ct/specific/filename "psf-merged.dat"
7 sources/generic/source-ct/specific/Emax 1.0e5
8
9 sources/generic/source-ct/specific/wght-window [5e-4, 1e-3]
10 sources/generic/source-ct/specific/nsplit 100
11
12 # Angular information
13 #####
14 sources/generic/source-ct/specific/phi-ini 180
15 sources/generic/source-ct/specific/phi-end 540
16 sources/generic/source-ct/specific/angularStep 1.0
17
18
19 # Position information
20 #####
21 sources/generic/source-ct/specific/CTx0 0
22 sources/generic/source-ct/specific/CTy0 0
23 sources/generic/source-ct/specific/CTz0 0
```

```

24 sources/generic/source-ct/specific/rad 52.0
25
26 # Time information
27 #####
28 sources/generic/source-ct/specific/tmin 0.0
29 sources/generic/source-ct/specific/dt 120.0

```

Code 25: PSF Computed Tomography Source

```

1 # Source 1
2 #####
3
4 sources/generic/source-ct/nhist 1e8
5 sources/generic/source-ct/specific/type "CT"
6
7 sources/generic/source-ct/specific/nSecondaries 80
8
9 sources/generic/source-ct/kpar "gamma"
10
11
12 # Directional sampling
13 #####
14 sources/generic/source-ct/direction/type "CONE"
15
16 # Set theta
17 sources/generic/source-ct/direction/theta 90.0
18
19 # Set phi
20 sources/generic/source-ct/direction/phi 180.0
21
22 # Set oberture (alpha)
23 sources/generic/source-ct/direction/alpha 11
24
25
26 # Energy sampling
27 #####
28
29 sources/generic/source-ct/energy/type "MONOENERGETIC"
30
31 # Set energy
32 sources/generic/source-ct/energy/energy 1.20e5
33
34 # Spatial sampling
35 #####
36
37 sources/generic/source-ct/spatial/type "POINT"
38
39 # Set particle origin
40 sources/generic/source-ct/spatial/position/x 0.0
41 sources/generic/source-ct/spatial/position/y 0.0
42 sources/generic/source-ct/spatial/position/z 0.0
43
44
45 # Angular information to do
46 #####
47 sources/generic/source-ct/specific/phi-ini 180
48 sources/generic/source-ct/specific/phi-end 540
49 sources/generic/source-ct/specific/angularStep 0.5
50
51
52
53 # Position information
54 #####

```

```

55 sources/generic/source-ct/specific/CTx0 19.15
56 sources/generic/source-ct/specific/CTy0 19.15
57 sources/generic/source-ct/specific/CTz0 14.8
58 sources/generic/source-ct/specific/rad 106.0
59
60
61
62 # Time information
63 #####
64
65 sources/generic/source-ct/specific/tmin 0.0
66 sources/generic/source-ct/specific/dt 120.0

```

Code 26: Generic Computed Tomography Source

### 5.2.5.3 PENNUC

The PENNUC source has been adapted from the original FORTRAN code published in [9] and developed by the CIEMAT, Laboratoire National Henri Becquerel, and Universitat de Barcelona. This one allows to simulate the decay of radioactive nuclides and use the resulting particles in a generic simulation. The decay characteristics for each nuclide, using the PENNUC format, can be downloaded from a public database <sup>4</sup>. In addition, the files `pdatconf.p14` and `pdrelax.p11` must be contained in the simulation folder to be able to process the decay data. Both files are available in the following database folder,

`dataBases/penmaterials/pdfiles/`

Moreover, a complete example of simulation using the PENNUC source can be found in the folder

`examples/quadrics/3-detector-2/`

To perform the spatial sampling, the PENNUC source requires a generic spatial sampler, for example, the box sampler. In addition, the source must be configured with the parameters shown in the code 27 and explained following.

- **type**: The source type must be set to **PENNUC**
- **nucleide\_filename**: Specify the file with the decay data downloaded from the nucleide database <sup>5</sup>.
- **age**: Optional parameter to enable or disable the age recording.
- **source-material**: Forces the particle to be generated only in a specific material.

```

1
2 # PENNUC sampling
3 #####
4
5 sources/generic/source1/specific/type "PENNUC" (mandatory)
6 sources/generic/source1/specific/nucleide_filename "Co-60.nuc" (mandatory)
7 sources/generic/source1/specific/age true (optional)
8 sources/generic/source1/specific/source-material 1 (optional)

```

Code 27: PENNUC source configuration

Notice that PENNUC will sample the direction isotropically, and only the spatial sampling step can be controlled, using a spatial sampler.

<sup>4</sup>[http://www.nucleide.org/DDEP\\_WG/DDEPdata.htm](http://www.nucleide.org/DDEP_WG/DDEPdata.htm)

<sup>5</sup>[http://www.nucleide.org/DDEP\\_WG/DDEPdata.htm](http://www.nucleide.org/DDEP_WG/DDEPdata.htm)

#### 5.2.5.4 Brachytherapy seeds source

This source has been implemented to simulate brachytherapy treatments, in which the seed information can be read from the DICOM plan of the patient provided by the user. To be able to use this source, first, a phase space file must be obtained to characterize the brachytherapy seed. That PSF will be moved and rotated according to the information provided in the DICOM plan to sample particles in each seed position. However, the PSF must be created accomplishing some conditions, which are listed below,

- The center of the seed geometry must be located at the (0,0,0).
- The seed geometry must look upwards (+Z axis).

This conditions must be satisfied to ensure that the location and rotation of the PSF is done properly for every seed position. Nevertheless, notice that the PSF can be translated and rotated via the configuration file. Therefore, if the original provided PSF does not accomplish the conditions, the required transformations can be specified in the configuration.

To use this source in the configuration file, the following parameters must be specified. The type of the source must be set to “BRACHY”. Then, a subsection named *psf* must be created to configure the PSF source used internally by the brachytherapy source. The PSF configuration is described in the section 5.2.5.1. In addition, the rotation of the PSF for each seed position, can be disabled, specifying *seed-rotation false*. This option is enabled by default. An example of a complete configuration is shown in the code 28.

```
1
2 #
3 # Source 1
4 #####
5
6 sources/generic/source-psf/nhist 5.0e7
7 sources/generic/source-psf/specific/type "BRACHY"
8 sources/generic/source-psf/specific/psf/filename "psf-merged.dat"
9 sources/generic/source-psf/specific/psf/Emax 1.40e6
10
11 sources/generic/source-psf/specific/psf/wght-window [5e-4, 1e-3]
12 sources/generic/source-psf/specific/psf/nsplit 10
13
14 sources/generic/source-psf/specific/seed-rotation false
```

Code 28: Brachytherapy source

Notice that the DICOM is not specified in the source configuration. This is because the DICOM image is taken directly from the geometry. Therefore, this source is only intended to be used with DICOM based geometries. In addition, if no DICOM plan file is provided in the specified DICOM, or the plan has not any seed, a warning will be shown in the log and the particles will be created at the (0,0,0) cm.

### 5.3 Geometries

Each simulation only allows to use a single geometry which configuration parameters are specified by the prefix *geometry* to be correctly identified. Like sources, geometries require to specify its type, and other specific parameters.

#### 5.3.1 Geometry visualization

All geometries compatible with the PenRed framework can be visualized using the viewer provided in the following repository,

<https://github.com/PenRed/GeometryViewer>

which includes the documentation to install and use the viewer.

### 5.3.2 Quadric

This geometry type is used in all the PENELOPE based examples which have been translated to be reproduced with PenRed. For instance, the code 29 shows the geometry configuration for the first simulation example (1-disc). The geometries for this type are defined in an external file following as is explained in the PENELOPE manual [3] in the Chapter 6 **Constructive quadric geometry**. Moreover, PenRed is compatible with the PENELOPE geometry files, allowing to use them directly from PENELOPE simulations.

```
1 geometry/type "PEN_QUADRIC"
2 geometry/input-file "disc.geo"
3 geometry/processed-geo-file "report.geo"
4
5 geometry/dsmax/1 1.0e-4
6 geometry/kdet/1 1
```

Code 29: Complete geometry configuration example

As other configuration examples, the type is specified by the *type* parameter. Then, the *input-file* parameter specifies the relative path to the file where the geometry has been defined. The other parameters *processed-geo-file*, *dsmax* and *kdet* are optional and described following. The first one, *processed-geo-file*, specifies a file to generate a geometry report. Secondly, *dsmax* specifies the maximum distance allowed for electrons and positrons to jump when class II transport is active. This one is used in thin regions to ensure a minimum number of steps in the specified body. Notice that the body where the *dsmax* is applied is specified via its alias in the parameter path, *geomtry/dsmax/\*alias\**. For instance, in the previous code, the *dsmax* is applied to the body with the alias "1". Finally, the parameter *kdet* specifies the detector identifier which the body belongs to. Again, the body is specified using its alias in the parameter path, *geometry/kdet/\*alias\**.

Optionally, it is possible to specify local absorption energies for any body and particle. In the code 30 we shown an example where the local energy absorption for each particle is set to 10 keV for the body with alias "2". Notice that the most restrictive absorption energy is applied, either the assigned to the corresponding material or the one assigned to the body.

```
1 geometry/eabs/2/electron 1.0e4
2 geometry/eabs/2/gamma 1.0e4
3 geometry/eabs/2/positron 1.0e4
```

Code 30: Configuration of geometry absorption energy example

### 5.3.3 Triangular Mesh Surfaces

This type of geometry uses triangular mesh surfaces to define the geometry system bodies. For this purpose, PenRed defines its own format for input files. However, with the provided Blender plugin (Section 8.1), several mesh formats can be easily transformed to the PenRed one.

Notice that this geometry is still under testing and could be changed in future versions.

### 5.3.3.1 Input file format

In the examples folder, several triangular mesh based geometry files can be found.

`examples/triangular_surfaces`

These geometry files must contain the following elements, in order,

- A line with a integer which indicates the total number of bodies in the geometry.
- Then, the specification of each body in the defined geometry follows. For each one, the information to be specified is,
  - **Body metadata:** Consists of a line with the following information, in order:
    - \* Material number
    - \* Number of triangles
    - \* Number of vertex
    - \* Body name
    - \* Parent body name
    - \* Number of vertex groups

An example is shown in the Code 31. Notice that lines beginning with `#` are considered as comments and are ignored when the input is processed. In all geometries, one body acts as the **world**, i.e. it has no parent and all other bodies are included inside it. To specify which body is considered as the world, its parent name must be set to **void**. Therefore, **void** can't be used as a body name.

```
1
2 #MAT  #NFACTS #NVERTEX  #NAME  #PARENT NAME  #N VERTEX GROUPS
3 002   40796   20400    Capsule   Phantom       5
```

Code 31: Body metadata for triangular mesh geometries.

- **Vertex Groups:** Each vertex group consists of a first line with the group name followed by the number of vertices included in this group, along with a list of the vertex indexes, one per line. An example is illustrated in Code 32, where three vertex groups are defined.

```
1 # VERTEX GROUPS
2 #NAME  #NVERTEX
3 back   0004
4 0002
5 0003
6 0006
7 0007
8 #NAME  #NVERTEX
9 front_down  0002
10 0001
11 0005
12 #NAME  #NVERTEX
13 front_up   0002
14 0000
15 0004
```

Code 32: Vertex group format.

Notice that this section is absent in files with zero vertex groups. An example with vertex groups can be found at the following path:

`src/lib/x-ray/source/baseAnode.geo`

Here, the vertex groups are utilized to construct an x-ray anode at runtime

- **Vertex list:** A list of all mesh vertex. Each one is specified in a single line with the following format,

`vertex_number X Y Z`

An example is shown in the Code 33.

```

1 # VERTEX LIST
2 # Index (X Y Z)
3 0000 +0.000000E+00 -8.000000E-02 +7.566000E-02
4 0001 +0.000000E+00 -8.000000E-02 -1.731400E-01
5 0002 +2.513000E-03 -7.996000E-02 +7.566000E-02
6 0003 +2.513000E-03 -7.996000E-02 -1.731400E-01
7 0004 +5.023000E-03 -7.984200E-02 +7.566000E-02
8 .
9 .
10 .

```

Code 33: Body vertex list in triangular mesh geometries.

Notice that vertex coordinates are relative to the geometry origin (0,0,0), not to the body center.

- **Triangle list:** A list with all mesh triangles. As vertex, each one is specified in a single line, which contains the three vertex indexes that defines the triangle. Notice that the order of the vertex is used to calculate the direction of the triangle normal vector. The normals are assumed to point outward from the body. Most 3D design softwares uses this schema by default and provide tools both to check the geometry normals and to recalculate them.

An input example is shown in the Code 34.

```

1 # FACES( triangles )
2 499 001 003
3 499 003 599
4 599 003 005
5 599 005 699
6 699 005 007
7 699 007 799
8 .
9 .
10 .

```

Code 34: Body triangle list in triangular mesh geometries.

### 5.3.3.2 Configuration

The mandatory configuration parameters to use this type of geometry are shown in the code 36 and are listed following,

- **type:** Must be set to **MESH\_BODY**.
- **input-file:** Path to the input file with the geometry description. This must follow the format described in the section 5.3.3.1.



In addition, the **report-file** parameter can be specified to print a geometry report.

Regarding specific parameters for each body, the user can specify the **dsmax** and **kdet** parameters as is done for quadric geometries.

If vertex groups have been defined in the geometry, the vertices belonging to those groups can be transformed via the configuration file. These transformations are organized in groups to streamline their ordering and must be defined using the following pattern inside the geometry configuration section:

**transforms/body-name/transformation-group/path value**

where *transforms* is a constant text identifier, *body-name* identifies a body inside the geometry by its alias, *transformation-group* is a user-defined name for this transformation group, *path* depends on the parameter to be set, and *value* is the value assigned to the parameter. The possible *paths* are listed below:

- **index:** Assigns an index to this transformation group. Transformation groups are applied following a growing index order.
- **vertex-group:** Specify which vertex group is affected by the transformation group. Note that each transformation group can be applied to only one vertex group.
- **transforms:** This subsection contains all the transformations to be applied to this group. Each one must be defined using the following pattern within this subsection,  
**transformation-name/path value**

where *transformation-name* is a user defined name for this transformation. Each transformation must contain the following elements:

- **index:** As groups, the transformations are ordered inside each group to be applied in ascending index order.
- **type:** Defines the transformation type. Actually, the available types are:
  - \* **TRANSLATION:** Apply a generic translation. Requires the following additional elements:
    - **ds:** Translation distance in cm.
    - **u:** *X* component of the translation direction.
    - **v:** *Y* component of the translation direction.
    - **w:** *Z* component of the translation direction.
  - \* **TRANSLATION\_X:** Translation in the *X* axis. Only requires the *ds* parameter.
  - \* **TRANSLATION\_Y:** Translation in the *Y* axis. Only requires the *ds* parameter.
  - \* **TRANSLATION\_Z:** Translation in the *Z* axis. Only requires the *ds* parameter.
  - \* **SCALE:** Scales the distance between each vertex and the group mass center a factor specified by the parameter *factor*.
  - \* **SCALE\_X:** Scales only in the *X* axis, with *Y* and *Z* remaining constant. Requires the parameter *factor*.
  - \* **SCALE\_Y:** Scales only in the *Y* axis, with *X* and *Z* remaining constant. Requires the parameter *factor*.
  - \* **SCALE\_Z:** Scales only in the *Z* axis, with *X* and *Y* remaining constant. Requires the parameter *factor*.

- \* **SCALE\_XY**: Scales only in the plane  $XY$ , with  $Z$  constant. Requires the parameter *factor*.
- \* **SCALE\_XZ**: Scales only in the plane  $XZ$ , with  $Y$  constant. Requires the parameter *factor*.
- \* **SCALE\_YZ**: Scales only in the plane  $ZY$ , with  $X$  constant. Requires the parameter *factor*.

An example of transformations applied to a body named *anode* is shown at Code 35.

```

1 transforms/anode/backEnlarge/index 0
2 transforms/anode/backEnlarge/vertex-group "back"
3 transforms/anode/backEnlarge/transforms/enlargeY/type "TRANSLATION_Y"
4 transforms/anode/backEnlarge/transforms/enlargeY/index 0
5 transforms/anode/backEnlarge/transforms/enlargeY/ds -1.0

```

Code 35: Vertex group transformation configuration example.

In this kind of geometries, to check if a particle crosses a body, this one is split in several regions containing a subset of the body triangles, allowing to check only the triangles belonging to crossed regions. In turn, this regions are grouped in super-regions, following the same approach. That clustering avoids checking many triangles and saves computational time.

To control how the clustering is done, the user can specify two parameters to control how these regions are constructed, which are described following,

- **RegionElements**: Specify the objective number of triangles in each region for the specified body.
- **SuperRegions**: Specify the objective number of regions in each super-region for the specified body.

These parameters can be different for each body. For example, in the code 36, the body named **Capsule\_cyl** has an objective number of triangles per region of 50 and will try to group 20 regions in each super-region. Notice that the real number of elements in each cluster could be different depending on the body structure. For example, regions with a small number of triangles are merged with boundary regions.

```

1
2 geometry/type "MESHBODY" (mandatory)
3 geometry/input-file "seedMesh.geo" (mandatory)
4 geometry/report-file "report.geo" (optional)
5
6 geometry/RegionElements/Capsule_cyl 50 (optional)
7
8 geometry/SuperRegions/Capsule_cyl 20 (optional)

```

Code 36: Triangular mesh geometry configuration example.

### 5.3.3.3 Limitations

Unlike in quadric geometries, the bodies surfaces in triangular mesh geometries can't cross the surface of another body. The equivalent in quadric geometries should be consider all bodies as modules. However, the surface of two different bodies can be in contact.

### 5.3.4 Voxel

Voxelized geometries consist of a 3D matrix of regular prism elements with a specific material and density factor for each one. The density factor is used to specify heterogeneities in the density of the materials. For instance, a voxel with a density factor of 1.10 will have a density 10% greater than the material's nominal density (specified in the material file or the configuration). Likewise, a voxel with a density factor of 0.8 is considered to have a density 20% lower than the nominal material density. To use the nominal material density in a voxel, this factor must be set to 1.

This type of geometry accepts the following parameters:

- **type**: Must be set to **VOXEL**.
- **filename**: Specify the path to the file containing the geometry definition.
- **ascii**: Specify if the geometry to read is in ASCII (*true*) or binary (*false*) format. This parameter is optional and defaults to binary.
- **nvoxels/nx, nvoxels/ny, nvoxels/nz**: These three parameters specify the number of voxels in the x, y, and z axes, respectively. These parameters are optional, as that information is already read from the geometry file. The only purpose of these parameters is to double-check the correctness of the specified file, especially in the binary case.
- **voxe-size/dx, voxe-size/dy, voxe-size/dz**: These three parameters specify the size of voxels (in cm) in the x, y, and z axes, respectively. As before, these parameters are optional and are only used to double-check the geometry file.
- **dsmax**: This optional section allows the user to set a *dsmax* value, in cm, for all voxels with a specific material index. This must be specified as follows:

`dsmax/mat-index value`

- **enclosure-margin**: This parameter defines the margin, in cm, of a homogeneous enclosure containing the voxelized geometry (Figure 12). It is intended to take into account border effects, such as backscatter.

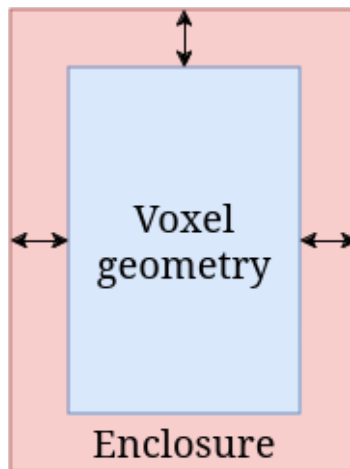


Figure 12: Voxelized geometry enclosure schema

- **enclosure-material**: Specify the enclosure material index.
- **print-ASCII**: If this boolean parameter is set to *true*, the processed geometry will be saved in a file named *voxelsASCII.rep* in ASCII format. This option is intended to be used with binary formats.

An example of configuration file for voxelized geometries is shown in the code 37.

```

1 geometry/type "VOXEL"
2 geometry/filename "3blocks.vx"
3 geometry/nvoxels/nx 60
4 geometry/nvoxels/ny 60
5 geometry/nvoxels/nz 80
6 geometry/voxel-size/dx 0.2
7 geometry/voxel-size/dy 0.2
8 geometry/voxel-size/dz 0.1
9
10 geometry/enclosure-margin 20
11 geometry/enclosure-material 5
12
13 geometry/dsmax/1 0.05
14 geometry/dsmax/2 0.02
15 geometry/dsmax/3 0.02
16 geometry/dsmax/4 0.50

```

Code 37: Configuration of voxelized geometry example

#### 5.3.4.1 Binary format

Voxelized binary geometries consist of binary data generated by the *pen\_voxelGeo* class. An example of how to create a geometry file using that class can be found in the test code:

*src/tests/geometry/voxels/read\_Dump.cpp*

This example creates a randomly filled voxel geometry, stores it in a file, and finally loads the file to compare it with the originally created geometry. Additionally, another example can be found in the utility *geo2voxel*, whose source file is located at:

*src/utilities/geometry/geo2voxel.cpp*

This utility instantiates a geometry with the type and configuration specified by the user and creates a voxelized geometry file according to that geometry.

As shown in the mentioned codes, the function to fill a voxel geometry class is named **setVoxels**, illustrated in code 38. The initialization of voxel materials and densities is not shown. The required parameters are:

- A three-element array (*nvox*) with the number of voxels in each axis (*nx, ny, nz*).
- The voxel sizes in each axis (*sizes*) (*dx, dy, dz*) in cm.
- A one-dimensional array with the material assignment of each voxel (*voxMats*), containing  $nx \times ny \times nz$  elements.
- A one-dimensional array with the density factor of each voxel, also containing  $nx \times ny \times nz$  elements
- A verbose level.

Once the voxel mesh has been created, the geometry file can be printed in binary format, ready for simulation, using the function `dump2File`. Additionally, the voxel geometry information can be printed in ASCII format using the function `printImage`.

```

1
2 unsigned nvox[3] = ...
3 double sizes[3] = ...
4 unsigned* voxMats = ...
5 double* voxDensFact = ...
6
7 pen_voxelGeo voxelgeo;
8 int err = voxelgeo.setVoxels(nvox, sizes, voxMats, voxDensFact, 3);
9 if(err != 0){
10     printf("Error using 'setVoxels': %d\n", err);
11     return -1;
12 }
13 //Print ASCII file
14 voxelgeo.printImage("voxelGeo.ascii");
15 //Dump binary file
16 voxelgeo.dump2File("voxelGeo.bin");

```

Code 38: Set voxels function and geometry file creation

#### 5.3.4.2 ASCII format

The ASCII format for voxelized geometries consists of the following elements, in order:

- Three positive, non-zero integer values representing the number of voxels in the x, y, and z axes.
- Three positive, non-zero numeric values specifying the size of a voxel in the x, y, and z axes.
- Two values for each voxel: first, an integer specifying the voxel material index; second, a numeric value corresponding to the density factor. Both values must be positive and greater than zero, as void material is not allowed inside a voxelized geometry.

The first lines of an example file in ASCII format for a voxelized geometry are shown in Code 39. This example represents a voxelized matrix of  $(100 \times 200 \times 120)$  voxels with a voxel size of  $(0.1 \times 0.1 \times 0.2)$  cm.

```

1
2 100 200 120
3 0.1 0.1 0.2
4 1 1.0
5 1 1.0
6 1 1.1
7 2 0.8
8 2 0.9
9 .
10 .
11 .

```

Code 39: ASCII voxelized geometry example.

#### 5.3.5 DICOM

Medical images are usually stored using the international standard of Digital Imaging and Communications in Medicine (DICOM). PenRed implements a DICOM geometry module

to convert the DICOM file to a voxel geometry automatically, which is ready to be simulated directly. As other PenRed components, DICOM geometries require a specific configuration structure to be used. This configuration includes the following parameters,

- **type:** Geometry type, must be set to “DICOM”.
- **directory:** Specifies the relative path where the DICOM images are stored. Notice that all the DICOM images found in that folder are expected to be from the same image.
- **calibration:** This optional parameter is only used for CT images. This must be specified as an array of numbers which belong to a polynomial calibration to convert from Hounsfield Units (HU) to density ( $g/cm^3$ ). If the calibration is not specified for CT images, the raw data will be used and the density must be assigned using other techniques, as we will see below.
- **default/material:** Specifies default material index for all voxels which material has not been assigned by other methods.
- **default/density:** Specifies default density for all voxels which density has not been assigned by any of the available methods.
- **intensity-ranges:** Provides a subsection to assign a material index and density value to all voxels inside the specified pixel value ranges. This subsection consists of the following parameters,
  - **material:** Material index to assign.
  - **density:** Density value ( $g/cm^3$ ) to assign.
  - **low:** Lower range pixel value to assign this material and density.
  - **top:** Upper range pixel value to assign this material and density.

So, all voxels with intensity values in the range  $[low, top)$  will be assigned with the material index *material* and the density value *density*. To differentiate between ranges, each one requires a unique name. This name must not be the material name, but is advisable for debug purposes. An example of this configuration is shown in code 40, where we define an interval named *Air*.

```

1 geometry/intensity-ranges/Air/material 1
2 geometry/intensity-ranges/Air/low -2000
3 geometry/intensity-ranges/Air/top -500
4 geometry/intensity-ranges/Air/density 0.001290

```

Code 40: DICOM intensity ranges configuration

- **ranges:** Subsection analogous to *intensity-ranges* but using density ranges instead of pixel values. Thus it could be used to specify the voxels material indexes via density ranges. Notice that it is required a calibration curve to previously convert HU units to densities. This subsection consists of the following parameters,
  - **material:** Material index to assign.
  - **density-low:** Minimum density value for this range.
  - **density-top:** Maximum density value for this range.
 All voxels with a density value between  $(density - low, density - top]$  will be assigned with the material index *material*.

- **contours:** Subsection that allows to the user to assign materials and densities according to contours stored inside the DICOM image. Each subsection of this type defines a single contour which name is specified in the corresponding parameters paths. Notice that the contour name must coincide with the contour name stored in the DICOM file. This subsection contains the following parameters,

- **material:** Default material to assign to this contour. This is an optional field. If it is not provided, the existing material assign will not be overwritten.
- **density:** Default density to assign to this contour ( $g/cm^3$ ). This is an optional field. If it is not provided, the voxel density will be not overwritten.
- **priority:** A priority value to control which contours are overwritten by other ones. Contours with lower priority values will be overwritten by contours with higher priority values.
- **intensity-ranges:** Material and density can also be assigned defining intensity ranges specific for each contour. This method overwrites the specified default material and density for the contour. To define the ranges, use the same pattern explained for global **intensity-ranges**.
- **ranges:** Materials can also be assigned defining density ranges specific for each contour. This method overwrites the specified default material for the contour. To define the ranges, use the same pattern explained for global **ranges**.

For example, to configure a contour named *target*, with air in some regions, the configuration file should contain something like the lines shown in code 41.

```

1 geometry/contours/target/material 1
2 geometry/contours/target/density 1.05
3 geometry/contours/target/priority 1.0
4
5 geometry/contours/target/intensity-ranges/Air/material 2
6 geometry/contours/target/intensity-ranges/Air/low -2000
7 geometry/contours/target/intensity-ranges/Air/top -500
8 geometry/contours/target/intensity-ranges/Air/density 0.001290

```

Code 41: DICOM contour configuration

To assign a uniform material and density in the contour, just don't specify any range.

- **enclosure-margin:** Like the voxel based geometry case, specify the distance between the DICOM mesh corner and the limit of an external enclosure, **in cm**.
- **enclosure-material:** Specify the enclosure material index.
- **print-ASCII:** This true/false optional configuration parameter, can be set to *true* to print the processed DICOM in ASCII format. The data will be stored in a file named *dicomASCII.rep*.

Notice that this geometry type presents several ways to assign material indexes and densities to voxels. Due to this characteristic, exists a method hierarchy where preferential methods overwrite the others. This preference is shown in the Figure 13.

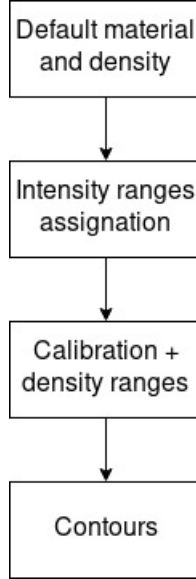


Figure 13: DICOM voxels material and density assign methods hierarchy.

Notice also that the allowed image modalities for DICOM geometry type are, by the moment, Computed Tomography (CT), Ultrasound (US), Radiotherapy Structure Set (RT-STRUCT) and Radio-therapy Plan (RTPLAN). Positron emission tomography (PET) is also accepted, but not for geometry construction purposes. Instead, PET images can be used to create a spatial sampling.

A DICOM example where voxel density and material are set using only intensity ranges can be found at the *example* folder. This one reproduces the GEANT IV DICOM simulation example using the DICOM image developed at [10]. So, the user must download that DICOM to reproduce the example.

#### 5.3.5.1 Constraints

As segmentation by density or intensity ranges can sometimes produce undesired results, such as small clusters of materials, thin erroneous material layers at some interfaces, etc., the user can specify constraints to control these effects.

The constraints are specified with the following pattern:

`constraints/constraint-name/path/to/element value`

where *constraint-name* is a user-defined name for the constraint, and the available elements to specify are listed below:

- **material:** This mandatory parameter specifies the material index to which the constraint will be applied.
- **min-volume:** Sets the minimum volume, in  $\text{cm}^3$ , for voxel clusters of this material. If a group of voxels of this material has a smaller volume, it will be overwritten with boundary materials.
- **max-volume:** Sets the maximum volume, in  $\text{cm}^3$ , for voxel clusters of this material. If a group of voxels of this material has a larger volume, it will be overwritten with boundary materials.



- **max-clusters:** Specifies the maximum number of disconnected voxel clusters of the specified material. If more clusters are found, the clusters with the smallest volume will be overwritten with boundary materials.

Note that the constraints are applied after the density and intensity range segmentation. Therefore, contour segmentation will overwrite the results of these constraints.

### 5.3.6 Combo

The geometry type “COMBO” provides the capability to merge different geometries of any type in a single one. There is no limit to the number of geometries that can be combined.

To determine which geometry exists in each region, each one must be assigned with an integer priority in the range  $[0, N)$ , where  $N$  is the number of geometries to combine. Then, the geometry with priority 0 will overwrite geometries with priorities  $[1, N)$ , the priority 1 will overwrite the priorities  $[2, N)$ , and so on.

Regarding geometries with higher priority numbers, these will overwrite only the void regions of geometries with lesser priority index. Therefore, a particle will only “see” the geometry with priority  $n$ , at the position  $(x, y, z)$ , if all geometries with priority indexes in the range  $[0, n)$  are void at  $(x, y, z)$ .

Notice that is not possible to define a “transparent” material to be overwrite by higher priority indexes as is done for void. This feature has not been implemented because its high impact on simulation speed.

The parameters to configure this type of geometry are listed below,

- **type:** Must be set to “COMBO”
- **geometries:** Is a subsection containing the information of each combined geometry. Each geometry is identified by a name. Therefore, to specify the parameters of geometry *accelerator* we must use the path,

`geometries/accelerator/path/to/parameter`

The parameters to be specified for each geometry are:

- **priority:** The priority integer index in the range  $[0, N)$ , where  $N$  is the number of combined geometries.
- **config:** A subsection containing all the required parameters to configure the geometry. Therefore, this subsection depends on the type of the geometry to be combined.

An example of configuration is shown in the Code 42, where three quadric geometries, named 1, 2, 3, are combined.

```

1 geometry/type "COMBO"
2
3 geometry/geometries/1/priority 2
4 geometry/geometries/2/priority 1
5 geometry/geometries/3/priority 0
6
7 geometry/geometries/1/config/type "PEN_QUADRIC"
8 geometry/geometries/1/config/input-file "1.geo"
9 geometry/geometries/1/config/processed-geo-file "report1.geo"
10
11 geometry/geometries/2/config/type "PEN_QUADRIC"
```

```

12 geometry/geometries/2/config/input-file "2.geo"
13 geometry/geometries/2/config/processed-geo-file "report2.geo"
14
15 geometry/geometries/3/config/type "PEN_QUADRIC"
16 geometry/geometries/3/config/input-file "3.geo"
17 geometry/geometries/3/config/processed-geo-file "report2.geo"

```

Code 42: Geometry “COMBO” type configuration example.

Although combining geometries is generally slower, it could simplify the optimization in some cases, for example allowing the usage of modules in combined quadric geometries. Therefore, in some particular cases, in addition to simplify the geometry construction, it could speed-up the simulation.

A more interesting usage case is the combination of different types of geometries. For example, the Code 43 shows the combination of a seed constructed using quadric geometries with a DICOM based geometry where the materials are assigned via contours. In this example, the seed geometry overwrites the DICOM.

```

1
2 geometry/type "COMBO"
3
4 geometry/geometries/seed/priority 0
5 geometry/geometries/DICOM/priority 1
6
7 geometry/geometries/seed/config/type "PEN_QUADRIC"
8 geometry/geometries/seed/config/input-file "seed.geo"
9 geometry/geometries/seed/config/processed-geo-file "report.geo"
10
11 geometry/geometries/DICOM/config/type "DICOM"
12 geometry/geometries/DICOM/config/directory "test_mamaHDR"
13
14 #Contour assign
15 geometry/geometries/DICOM/config/enclosure-margin 20
16 geometry/geometries/DICOM/config/enclosure-material 4
17
18 geometry/geometries/DICOM/config/default/material 4
19 geometry/geometries/DICOM/config/default/density 1.2000E-03
20
21 geometry/geometries/DICOM/config/contours/body/material 1
22 geometry/geometries/DICOM/config/contours/body/density 1.0
23 geometry/geometries/DICOM/config/contours/body/priority 2.2
24
25 geometry/geometries/DICOM/config/contours/heart/material 2
26 geometry/geometries/DICOM/config/contours/heart/density 1.5
27 geometry/geometries/DICOM/config/contours/heart/priority 2.6
28
29 geometry/geometries/DICOM/config/contours/skin/material 3
30 geometry/geometries/DICOM/config/contours/skin/density 1.1
31 geometry/geometries/DICOM/config/contours/skin/priority 2.3
32
33 geometry/geometries/DICOM/contours/left_lung/material 4
34 geometry/geometries/DICOM/contours/left_lung/density 1.2E-03
35 geometry/geometries/DICOM/contours/left_lung/priority 2.3
36
37 geometry/geometries/DICOM/contours/right_lung/material 4
38 geometry/geometries/DICOM/contours/right_lung/density 1.2E-03
39 geometry/geometries/DICOM/contours/right_lung/priority 2.3
40
41 geometry/geometries/DICOM/contours/left_ribs/material 5
42 geometry/geometries/DICOM/contours/left_ribs/density 1.92
43 geometry/geometries/DICOM/contours/left_ribs/priority 2.3
44

```

```

45 geometry/geometries/DICOM/contours/right_ribs/material 5
46 geometry/geometries/DICOM/contours/right_ribs/density 1.92
47 geometry/geometries/DICOM/contours/right_ribs/priority 2.3
48
49 geometry/geometries/DICOM/contours/ptv/material 6
50 geometry/geometries/DICOM/contours/ptv/density 1.0
51 geometry/geometries/DICOM/contours/ptv/priority 2.5

```

Code 43: Geometry “COMBO” with quadric and DICOM geometries.

## 5.4 Tallies

Configurations for tallies are similar to sources. The pattern to configure tallies is exemplified in the code 44, where *tallies* is a constant text, *tally-name* is a user defined name for the tally, and *parameter/path* and *value* sets the tally parameters, whose depends on tally type.

```

1 tallies/tally -name/parameter/path value

```

Code 44: Tally configuration pattern

As sources and geometries, exists several tally types, so the user must specify the parameter *type* on each created tally. In addition to tally type, a path to save the tally results can be specified via the *outputdir* parameter. Both parameters are common to all tallies, and are shown in the Code 45.

```

1 tallies/tally -name/type "TYPE.STRING"
2 tallies/tally -name/outputdir "path/to/results/dir/"

```

Code 45: Tally common parameters

Notice that, in the *outputdir* value, the remaining path after the last slash (“/” or “\” for unix and windows based systems respectively) will be appended as a prefix to the results filename instead of be interpreted as a directory. The Table 8 shows some cases to exemplify this behaviour, where the left column represents the value of *outputdir* specified in the configuration file, and the right column the final file path in the OS file system. If the specified path does not exists, i.e. one or more directories have been not created by the user, PenRed will try to append the prefix “*not-found*” to the filename and create the corresponding file in the execution directory, in order to avoid losing the simulation progress.

outputdir path	final path
dir1/	dir1/filename
dir1	dir1filename
dir1/dir2/	dir1/dir2/filename
dir1/dir2	dir1/dir2filename

Table 8: Tally *outputdir* behaviour examples.

An example of complete tally configuration is shown in the Code 46, which corresponds to a cylindrical dose distribution tally. This one requires limits for radial distance (*rmin* to *rmax*), number of radial bins *nbinsr*, limits for *z* axis (*zmin* to *zmax*) and the number of *z* bins (*nbinsz*).

```

1 tallies/cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"
2 tallies/cylDoseDistrib/print-xyz true
3 tallies/cylDoseDistrib/rmin 0.0
4 tallies/cylDoseDistrib/rmax 30.0

```

```

5 tallies/cylDoseDistrib/nbinsr 60
6 tallies/cylDoseDistrib/zmin 0
7 tallies/cylDoseDistrib/zmax 30.0
8 tallies/cylDoseDistrib/nbinsz 60

```

Code 46: Tally configuration pattern

An aspect to consider when we are choosing the limits, is that the behaviour of the intervals is  $[min, max)$ . Thus, in the previous example,  $rmax$  and  $zmax$  are out of their respective intervals.

The next subsections describe briefly the data measured in each tally and provide an example to use them at the configuration file.

#### 5.4.1 Radial and Cylindrical Dose Distribution

This tally measures the absorbed dose in (eV/g) for each radial bin in the range of  $[rmin, rmax)$ , in cm and can be configured to measure the depth absorbed dose distribution. Regarding the units, all results are expressed in (eV cm/g) per history. Below is shown the available parameters of this tally,

- **type “CYLINDRICAL\_DOSE\_DISTRIB”**: Type name of the tally.
- **print-xyz**: This variable can be set to **true** to print more information in the results file. If activated, two extra values per bin coordinate are printed, which corresponds to the coordinates low and the average values. For z coordinates, the average value is considered at the middle point of the bin. For the r coordinate, the average is weighted with a weight proportional to the radius r. If *print-xyz* is not specified, is set to **false** by default.
- **rmin**: Minimum value of the radial coordinate. Must be greater than zero and lower than  $rmax$ .
- **rmax**: Maximum value of the radial coordinate.
- **nbinsr**: Number of radial bins. Must be at least 1.
- **zmin**: Minimum value of the depth coordinate. Must be lower than  $zmax$  when *nbinsz* is set greater to zero.
- **zmax**: Maximum value of the depth coordinate. Must be set to  $zmin$  when depth absorbed dose is not measured.
- **nbinsz**: Number of depth bins. Must be set to zero when depth absorbed dose is not measured, otherwise, must be at least 1.
- **nbinsPhi**: Number of angular ( $\phi$ ) bins. By default, this parameter is set to 1.

Regarding the value types, the limits of radial and depth intervals must be doubles, while the number of bins in each case must be integers. The output filename of this tally ends with *cylindricalDoseDistrib.dat* but, if we disable the depth absorbed dose measure in the configuration, the output filename will end with *radialDoseDistrib.dat*.

An example of configuration for this tally is shown in the code 47, which has been extracted from the file *disc.in* of the example 1-disc.

```

1 tallies/cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"
2 tallies/cylDoseDistrib/print-xyz true (optional)
3 tallies/cylDoseDistrib/rmin 0.0 (mandatory)
4 tallies/cylDoseDistrib/rmax 0.01 (mandatory)
5 tallies/cylDoseDistrib/nbinsr 50 (mandatory)
6 tallies/cylDoseDistrib/zmin 0.0 (mandatory)
7 tallies/cylDoseDistrib/zmax 0.005 (mandatory)
8 tallies/cylDoseDistrib/nbinsz 100 (mandatory)
9 tallies/cylDoseDistrib/nbinsPhi 1 (optional)

```

Code 47: Cylindrical dose distribution tally configuration

### 5.4.2 Emerging Particles Distribution

This tally measures the energy distribution of particles that left the geometry within a specific energy range of  $[emin, emax)$ , expressed in particles/(eV history). We consider two cases in this tally. First, down bound emerging particles, which  $W$  direction is lower or equal to zero. Then, the up bound case when the  $W$  value is greater than zero. Also, the tally measures the number of particles that left the geometry per steradian determined by the theta and phi angular values, in degrees, of the particle deviation. In this case, measurements are in particles/(sr history).

The tally information needed in the configuration file is shown below,

- **type “EMERGING\_PART\_DISTRIB”**: Type name of the tally.
- **emin**: Minimum energy value. Must be lower than *emax*.
- **emax**: Maximum energy value.
- **nBinsE**: Number of energy bins. Must be at least 1.
- **nBinsTheta**: Number of polar bins. Must be greater than zero.
- **nBinsPhi**: Number of azimuthal bin, Must be greater than zero.

Regarding the values types, the energy limits must be doubles, while the bin numbers must be integers. The output files of this tally end as: *emergin-downbound.dat*, *emergin-upbound.dat*, *emergin-angle.dat*, with information for theta and phi values and *emergin-polar-angle.dat* with only angular information of theta values.

The code 48 is an example of this tally, which has been extracted from the configuration file *detector1.in* of the example 3-detector-1:

```

1 tallies/EmergingPartDistrib/type "EMERGING_PART_DISTRIB"
2 tallies/EmergingPartDistrib/emin 0.0 (mandatory)
3 tallies/EmergingPartDistrib/emax 1.45e6 (mandatory)
4 tallies/EmergingPartDistrib/nBinsE 280 (mandatory)
5 tallies/EmergingPartDistrib/nBinsTheta 45 (mandatory)
6 tallies/EmergingPartDistrib/nBinsPhi 18 (mandatory)

```

Code 48: Emerging particles tally configuration

### 5.4.3 Energy Deposition Body

This tally measures the energy, in eV per history, deposited in each body. To use it, only the *type* parameter, to select the tally itself, is required,

- **type “EDEP\_BODY”**: Type name of the tally.

The output filename ends with *bodyEnergyDeposition.dat*.

A configuration example of this tally is shown in the code 49, which belongs to the file *plane.in* of the example 2-plane.

```
1 tallies/bodyEDep/type "EDEP_BODY"
```

Code 49: Energy deposition in body tally configuration

#### 5.4.4 Energy Deposition Material

This tally measures the energy, in eV per history, deposited in each material. To use it, only the *type* parameter, to select the tally itself, is required,

- **type “EDEP\_MAT”**: Type name of the tally.

The corresponding output file ends with *materialEnergyDeposition.dat*

The code 50 shows a configuration example obtained from the configuration file *detector1.in* belonging to the example 3-detector-1.

```
1 tallies/matEDep/type "EDEP_MAT"
```

Code 50: Energy deposition in material tally configuration

#### 5.4.5 Impact Detector

This tally is capable to measure different magnitudes: fluence spectrum, particle energy spectrum and particle age. Each one is described following,

- **Fluence**: integrates the spectral fluence over the detector volume in cm/eV. The output file contains the fluence for each particle type and the total fluence in the specified detector. The filename ends with *fluenceTackLength-num.dat*, where *num* is the corresponding detector number.
- **Energy spectrum**: reports the energy spectrum of particles which enter in the specified detector volume. Particles created inside the detector, for example secondary particles, are not considered in this tally. Units are expressed in 1/(eV history). The output file contains the energy spectrum for each particle type and the total spectrum in the specified detector is also measured. The filename ends with *spectrum-impdet-num.dat*, where *num* is the number of the assigned detector.
- **Age**: reports the age distribution of particles which impact to the considered detector, in units of 1/(seconds history). The output file that ends with *age-impdet-num.dat*, contains the probability distribution along each time interval over all simulated particles. As in the previous cases, *num* is the number that identifies the detector.
- **Energy deposition**: reports the energy deposited spectrum measured in the considered detector. Units are expressed in 1/(eV history). The output file ends with *energyDeposition-impdet-num.dat*, where *num* is the number of the detector. This file contains the probability distribution along each energy interval for the specific detector over all simulated particles.

Regarding the configuration parameters, these are listed below:

- **type “IMPACT\_DET”**: Type name of the tally.
- **detector**: Detector index where the measurements of this tally are taken.

- **fluence**: Must be set to **true** to obtain fluence measurements, otherwise set it to **false**. If this information is not specified, **false** value will be assigned by default.
- **emin**: Minimum energy value for tallied particles in fluence, energy spectrum and energy deposition measurements.
- **emax**: Maximum energy value for tallied particles in fluence, energy spectrum and energy deposition measurements. Must be greater than *emin*.
- **nbin-energy**: Number of energy bins, which is used for fluence, energy spectrum and energy deposition. Must be at least 1.
- **spectrum**: Must be set to **true** to obtain spectrum measurements, otherwise set it to **false**. If this information is not specified, **false** is set as default value.
- **age**: Must be set to **true** to obtain age measurements, otherwise set it to **false**. If this information is not specified, **false** is set as default value.
- **energy-dep**: Must be set to **true** to obtain energy deposition measurements, otherwise set it to **false**. If this information is not specified, **false** value will be assigned by default.
- **linearScale-spc**: Determines if energy spectrum measurements will be tallied using a liner scale (**true**) or a logarithmic scale (**false**). When this parameter is not specified, linear scale is selected by default (**true**).
- **linearScale-edep**: By default is set to **true**, meaning that the energy deposition will be tallied using a linear scale. Instead, if set to **false**, the tally will be created following a logarithmic scale.
- **linearScale-age**: Determines if the output measurements for the age spectrum are expressed using a linear scale (**true**) or a logarithmic scale (**false**). If not specified, a linear scale is selected by default (**true**).
- **nbin-age**: Number of age interval bins. Is only required if **age** is active and must be at least 1.
- **age-min**: Minimum value of the age interval.
- **age-max**: Maximum value of the age interval. Must be greater than *age-min*.

Regarding the parameter types, the interval limits in each case must be doubles and the bin numbers must be specified as integers.

To exemplify the tally usage, next are two examples to show diferent configurations for this tally. First, code 51 shows a part of the configuration file *plane.in* of the 2-plane example, where we the fluence, spectrum and age information are tallied. If fluence, spectrum, age or energy deposition are activated, their corresponding fields are mandatory, otherwise are neither used or expected.

```

1 tallies/ImpactDetector/type "IMPACT_DET"
2 tallies/ImpactDetector/detector 1 (mandatory)
3 tallies/ImpactDetector/fluence true (optional)
4 tallies/ImpactDetector/emin 1.0e5 (mandatory/optional)
5 tallies/ImpactDetector/emax 3.5e7 (mandatory/optional)
6 tallies/ImpactDetector/nbin-energy 100 (mandatory/optional)
7 tallies/ImpactDetector/linearScale-fln true (optional)
8 tallies/ImpactDetector/spectrum true (optional)

```



```

9 tallies/ImpactDetector/age true (optional)
10 tallies/ImpactDetector/linearScale-age false (mandatory/optional)
11 tallies/ImpactDetector/nbin-age 100 (mandatory/optional)
12 tallies/ImpactDetector/age-min 1.0e-9 (mandatory/optional)
13 tallies/ImpactDetector/age-max 1.0e-8 (mandatory/optional)

```

Code 51: Impact detector tally configuration example 1

Secondly, code 52 shows some lines of the configuration file *detector1.in* that belongs to the example 3-detector-1. In this example, only the energy deposition information is tallied.

```

1 tallies/ImpactDetector/type "IMPACT_DET"
2 tallies/ImpactDetector/detector 1 (mandatory)
3 tallies/ImpactDetector/emax 0.0e0 (mandatory/optional)
4 tallies/ImpactDetector/emax 1.45e6 (mandatory/optional)
5 tallies/ImpactDetector/nbin-energy 280 (mandatory/optional)
6 tallies/ImpactDetector/energy-dep true (optional)
7 tallies/ImpactDetector/linearScale-edep true (optional)

```

Code 52: Impact detector tally configuration example 2

### 5.4.6 Spatial Dose Distribution

This tally measures the 3D absorbed dose distribution along the intervals  $[xmin, xmax)$ ,  $[ymin, ymax)$ ,  $[zmin, zmax)$  in cm. The units of dose values are eV/g per history. For each coordinates, the user must select the number of bins used to report the data in each axis  $(nx, ny, nz)$ . In addition, the tally reports the depth dose distribution along the  $z$  coordinate in eV/(g/cm<sup>2</sup>).

The available tally parameters to configure are explained below,

- **type "SPATIAL\_DOSE\_DISTRIB"**: Type name of the tally.
- **xmin**: Minimum value of coordinate  $x$ .
- **xmax**: Maximum value of coordinate  $x$ . Must be greater than  $xmin$ .
- **nx**: Number of  $x$  bins. Must be, at least, 1.
- **ymin**: Minimum value of coordinate  $y$ .
- **ymax**: Maximum value of coordinate  $y$ . Must be greater than  $ymin$ .
- **ny**: Number of  $y$  bins. Must be at least 1.
- **zmin**: Minimum value of coordinate  $z$ .
- **zmax**: Maximum value of coordinate  $z$ . Must be greater than  $ymin$ .
- **nz**: Number of  $z$  bins. Must be at least 1.
- **print-depthDose**: Enables/disables printing depth dose information. Notice that this depth dose is discretized only in the Z axis, i.e. scores all energy depositions regardless the X and Y component. Therefore, in voxelized geometries, the enclosure is considered in the calculus. By default is disabled.

Considering the parameters type, those who specify coordinate values must be doubles, while the number of bins for each coordinate must be specified as integer. For plotting purposes, two values per bin coordinate are given: the low and the middle point of each



bin. Output filenames end with *spatialDoseDistrib-3D.dat* for 3D absorbed dose distribution and *depth-dose.dat* for depth dose distribution. To exemplify the configuration, the code 53 shows an example extracted from the configuration file *disc.in* that belongs to the example 1-disc-novr.

```

1 tallies/SpatialDoseDistrib/type "SPATIAL_DOSE_DISTRIB"
2 tallies/SpatialDoseDistrib/print-xyz true (optional)
3 tallies/SpatialDoseDistrib/xmin 0.0 (mandatory)
4 tallies/SpatialDoseDistrib/xmax 1.0 (mandatory)
5 tallies/SpatialDoseDistrib/nx 1 (mandatory)
6 tallies/SpatialDoseDistrib/ymin 0.0 (mandatory)
7 tallies/SpatialDoseDistrib/ymax 1.0 (mandatory)
8 tallies/SpatialDoseDistrib/ny 1 (mandatory)
9 tallies/SpatialDoseDistrib/zmin 0.0 (mandatory)
10 tallies/SpatialDoseDistrib/zmax 0.005 (mandatory)
11 tallies/SpatialDoseDistrib/nz 100 (mandatory)
12 tallies/SpatialDoseDistrib/print-depthDose false (optional)

```

Code 53: Spatial Dose Distribution tally configuration

#### 5.4.7 Angular Detector

This tally reports the angular energy spectrum in a specified detector. This information is tallied in the energy interval  $[emin, emax)$ , specified in eV, and the angular intervals  $[theta1, theta2)$ ,  $[phi1, phi2)$ , specified in degrees. The energy spectra of particles is tallied in units of  $1/(eV \text{ sr particle})$ .

Below, we describe the parameters used to configure this tally,

- **type “ANGULAR\_DET ”**: Tally type name.
- **detector**: Detector index where the angular detector will be calculated.
- **emin**: Minimum energy value.
- **emax**: Maximum energy value. Must be greater than *emin*.
- **theta1**: Minimum value of polar angle.
- **theta2**: Maximum value of polar angle. The polar interval must be in the range (0, 180) and *theta2* must be greater than *theta1*.
- **phi1**: Minimum value of azimuthal angle.
- **phi2**: Maximum value of azimuthal angle. The azimuthal interval must be in the range (0, 360) or (−180, 180) and *phi2* must be greater than *phi1*.
- **nBinsE**: Number of energy bins.
- **linearScale**: Determines if the output measurements are expressed in linear scale (**true**) or in logarithmic scale (**false**). If scale is not specified, linear scale will be set.

The expected parameter types are integers for *detector* and the number of bins, and doubles for the energy and angular limits. The output filename for this tally ends with *spc-angdet-num.dat* where *num* is the number of the assigned detector.

The code 54 belongs to the configuration of the example 1-disc-novr, where this tally is used. The configuration file is named *disc.in*.

```

1 tallies/AngularDetector/type "ANGULAR_DET"
2 tallies/AngularDetector/detector 1 (mandatory)
3 tallies/AngularDetector/emin 0.0 (mandatory)
4 tallies/AngularDetector/emax 40.5e3 (mandatory)
5 tallies/AngularDetector/theta1 90.0 (mandatory)
6 tallies/AngularDetector/theta2 180.0 (mandatory)
7 tallies/AngularDetector/phi1 0.0 (mandatory)
8 tallies/AngularDetector/phi2 360.0 (mandatory)
9 tallies/AngularDetector/nBinsE 200 (mandatory)
10 tallies/AngularDetector/linearScale true (optional)

```

Code 54: Angular detector tally configuration

#### 5.4.8 Particle Generation

This tally reports information about the number of primary and secondary particles simulated. First, counts the number of primary particles that escape up bound and down bound and the number of absorbed particles. In addition, calculates the probabilities for secondary particles to go up bound, down bound and be absorbed.

This tally doesn't require to specify any parameter in the configuration file, only the tally type,

- **type "SECONDARY\_GEN"**: Tally type name.

The output filename ends with *particleGeneration.dat*.

An example of the configuration file of this tally is code 55. This line is the same for all examples where this tally will be used.

```

1 tallies/secondary/type "SECONDARY_GEN"

```

Code 55: Particle generation tally configuration

#### 5.4.9 Spherical Dose Distribution

This tally reports the absorbed dose distribution, in (eV/g) per history, in a spherical distribution, i.e. the bins follows spherical coordinates. By default, a single bin is used for each angular coordinate and only a radial distribution is tallied in the range  $[rmin, rmax)$ , specified in cm. The number of radial bins are denoted by *nbin* and, optionally, the user can activate the boolean *print-xyz* (true) to print different values for radial coordinate in the output file: the low value and the average value. This average value is weighted proportionally to  $r^2$ .

Tally configuration parameters are listed following:

- **type "SPHERICAL\_DOSE\_DISTRIB "**: Type name of the tally.
- **print-xyz**: Can be set to **true** to print different values for radial coordinate in the output file i.e the low value and the average value. This average value is weighted proportionally to  $r^2$ . If this parameter is not specified, will be set to **false** by default.
- **rmin**: Minimum value of radial coordinate. Must be greater than zero.
- **rmax**: Maximum value of radial coordinate. Must be greater than *rmin*.
- **nr**: Number of radial bins. Must be at least 1.
- **ntheta**: Number of polar ( $\theta$ ) bins. By default, this parameter is 1.

- **nphi**: Number of azimuth ( $\phi$ ) bins. By default, this parameter is 1.

Regarding the parameter types, all limits are expected to be doubles and the number of bins integers. The output filename ends with *sphericalDoseDistrib.dat*. Any of the provided examples use this tally. However, a configuration example is shown in the code 56.

```

1 tallies/SphericalDose/type "SPHERICALDOSEDISTRIB"
2 tallies/SphericalDose/print-xyz true (optional)
3 tallies/SphericalDose/rmin 0.0 (mandatory)
4 tallies/SphericalDose/rmax 0.06 (mandatory)
5 tallies/SphericalDose/nr 65 (mandatory)
6 tallies/SphDoseDistrib/ntheta 1 (optional)
7 tallies/SphDoseDistrib/nphi 1 (optional)

```

Code 56: Spherical dose distribution tally configuration

#### 5.4.10 Phase Space File (PSF)

This tally generates a particle phase space file that stores all particles impacting the specified detector. Additionally, the stored particles can be filtered by energy range and particle type.

The available tally parameters for the configuration file are listed below:

- **type “PSF”**: Type name of the tally.
- **detector**: Detector index where this tally will be calculated.
- **emin**: Minimum energy value.
- **emax**: Maximum energy value. Must be greater than *emin*.
- **particles**: Section to enable or disable particle types recording. In this section, the user can specify the following parameters:
  - **default**: This optional boolean value enables (true) or disables (false) the particle recording for all particle types. Defaults to true.
  - **particle-name**: This optional boolean parameter, where *particle-name* must be replaced by a valid particle name (e.g., *gamma*), disables (false) or enables (true) recording of the specified particle type. If defined, it overrides the *default* parameter.

```

1 tallies/psf/type "PSF" (mandatory)
2 tallies/psf/detector 1 (mandatory)
3 tallies/psf/emin 0.0 (mandatory)
4 tallies/psf/emax 6.1e6 (mandatory)
5 tallies/psf/particles/default false (optional)
6 tallies/psf/particles/gamma true (optional)

```

Code 57: Example of phase space file tally configuration where only gamma particles are recorded

This tally can be used with multiple threads. In that case, each thread will temporarily store its PSF particles in an independent file. When the simulation finishes, all files will be concatenated automatically, ordered by thread ID in ascending order.

#### 5.4.11 Kerma track length estimator

This tally is based on the equivalence of particle fluence and the total photon path length per unit volume. A complete description of the estimator can be found at [11]. The estimator could be tallied using three type of meshes: Voxel or cartesian based, cylindrical and spherical meshes. The required configuration for this tally is listed below.

- **type “KERMA\_TRACK\_LENGTH”**: Type name of the tally.
- **emin**: Photon minimum energy to be considered (eV).
- **emax**: Photon maximum energy to be considered (eV).
- **dataFiles**: One data filename with the  $\mu_{en}/\rho$  ( $cm^2/g$ ) coefficients per tallied material is required. However, if the file does not exists, the tally will create the corresponding  $\mu_{en}/\rho$  data automatically. Then, the data will be stored in the specified file.

Each filename is specified following this pattern:

```
path/to/tally/dataFiles/material-number "filename"
```

The data files format consists of two columns. The first one stores the energy in  $eV$  and the second the corresponding  $\mu_{en}/\rho$  value in ( $cm^2/g$ ). The provided points must contain the energy range  $[emin, emax]$ . However, is not required a constant energy distance between points. Furthermore, there are no limit to the number of points in each data file. Notice that materials with no  $\mu_{en}/\rho$  data filename provided will be ignored by the tally. To obtain the  $\mu_{en}/\rho$  coefficients manually, the utility **mutren** can be used. This one is provided in the PenRed package and described in the section 8.2. On the other hand, the user can let it be calculated automatically, as discussed before.

- **cartesian**: If this optional section exists, the tally will record the estimator using a regular voxel mesh. The required parameters are listed below:
  - **cartesian/nx**: Number of bins on the  $X$  axis.
  - **cartesian/xmin**: Init mesh point on  $X$  axis in  $cm$ .
  - **cartesian/xmax**: Limit mesh point on  $X$  axis in  $cm$ .
  - **cartesian/ny**: Number of bins on the  $Y$  axis.
  - **cartesian/ymin**: Init mesh point on  $Y$  axis in  $cm$ .
  - **cartesian/ymax**: Limit mesh point on  $Y$  axis in  $cm$ .
  - **cartesian/nz**: Number of bins on the  $Z$  axis.
  - **cartesian/zmin**: Init mesh point on  $Z$  axis in  $cm$ .
  - **cartesian/zmax**: Limit mesh point on  $Z$  axis in  $cm$ .
- **cylindrical**: If this optional section exists, the tally will record the estimator using a cylindrical mesh. The required parameters are listed below:
  - **cylindrical/nr**: Number of radial bins.
  - **cylindrical/rmin**: Cylinder minimum radius in  $cm$ .
  - **cylindrical/rmax**: Cylinder maximum radius in  $cm$ .

- **cylindrical/nphi**: Number of angular ( $\phi$ ) bins.
- **cylindrical/nz**: Number of  $Z$  bins.
- **cylindrical/zmin**: Init mesh point on  $Z$  axis in *cm*.
- **cylindrical/zmax**: Limit mesh point on  $Z$  axis in *cm*.
- **spherical**: If this optional section exists, the tally will record the estimator using a spherical mesh. The required parameters are listed below:
  - **spherical/nr**: Number of radial bins
  - **spherical/rmin**: Sphere minimum radius in *cm*.
  - **spherical/rmax**: Sphere maximum radius in *cm*.
  - **spherical/ntheta**: Number of polar angular ( $\theta$ ) bins.
  - **spherical/nphi**: Number of azimuth angular ( $\phi$ ) bins.

Notice that more than one mesh type could be active at the same tally. In these cases, the tally will generate one report for each mesh type. In code 58 we show a configuration example for that tally.

```

1 tallies/kermaTrackLength/type "KERMA-TRACKLENGTH" (mandatory)
2 tallies/kermaTrackLength/emin 1.0e3 (mandatory)
3 tallies/kermaTrackLength/emax 1.5e6 (mandatory)
4 tallies/kermaTrackLength/dataFiles/1 "mu-Water1000.mat" (mandatory)
5
6 #Optional section:
7 tallies/kermaTrackLength/cylindrical/nr 10 (mandatory)
8 tallies/kermaTrackLength/cylindrical/rmin 0.1 (optional)
9 tallies/kermaTrackLength/cylindrical/rmax 10.0 (mandatory)
10 tallies/kermaTrackLength/cylindrical/nphi 10 (mandatory)
11 tallies/kermaTrackLength/cylindrical/nz 10 (mandatory)
12 tallies/kermaTrackLength/cylindrical/zmin -10.0 (mandatory)
13 tallies/kermaTrackLength/cylindrical/zmax 10.0 (mandatory)
14
15 #Optional section:
16 tallies/kermaTrackLength/cartesian/nx 10 (mandatory)
17 tallies/kermaTrackLength/cartesian/xmin -10.0 (mandatory)
18 tallies/kermaTrackLength/cartesian/xmax 10.0 (mandatory)
19 tallies/kermaTrackLength/cartesian/ny 10 (mandatory)
20 tallies/kermaTrackLength/cartesian/ymin -10.0 (mandatory)
21 tallies/kermaTrackLength/cartesian/ymax 10.0 (mandatory)
22 tallies/kermaTrackLength/cartesian/nz 10 (mandatory)
23 tallies/kermaTrackLength/cartesian/zmin -10.0 (mandatory)
24 tallies/kermaTrackLength/cartesian/zmax 10.0 (mandatory)
25
26 #Optional section:
27 tallies/kermaTrackLength/spherical/nr 10 (mandatory)
28 tallies/kermaTrackLength/spherical/rmin 0.2 (optional)
29 tallies/kermaTrackLength/spherical/rmax 10.0 (mandatory)
30 tallies/kermaTrackLength/spherical/ntheta 10 (mandatory)
31 tallies/kermaTrackLength/spherical/nphi 10 (mandatory)

```

Code 58: Kerma track length tally configuration

Notice that the elements inside an optional section are mandatory only if the section exists. For example, if the section `tallies/kermaTrackLength/cartesian` doesn't exist, any of the cartesian parameters are required, such as *cartesian/nx* or *cartesian/xmin*.

Regarding the parameter types, all bin number are expected to be integers, while the corresponding limits are expected to be doubles.

#### 5.4.12 CT tally

This tally has been created to collect the photons that arrive to a virtual detector. This tally provides a file with CT sinogram report and a material energy deposition report by projection. The tally information needed in the configuration file is described below.

- **type “CT\_SINOGRAM”**: Type name of the tally.
- **emin**: Photon minimum energy to be collect (eV).
- **emax**: Photon maximum energy to be collect (eV).
- **npixels**: Number of pixels of the virtual detector.
- **pixel-depth**: Size of pixel deph (cm).
- **r-inner**: Radius of the virtual detector. Corresponds to the isocenter-detector distance.
- **xOrigin, yOrigin, zOrigin**: x,y and z values of the CT origin. Corresponds to the position of the CT isocenter.
- **phi-ini**: Initial angular position of the CT detector, in degrees. Must be set in front of the CT source with an angular increment of 180 degrees.
- **phi-end**: Final angular position of the CT detector, in degrees. Must be set in front of the CT source with an angular increment of 180 degrees.
- **angularStep**: Angular step between CT projection. Must be set with the same value of the CT source angular step.
- **aperture**: Angle subtended by the detector with respect to the isocenter.

Usually, the angular aperture is provided by CT manufacturers as the angle subtended with respect the CT source instead of with respect to the isocenter. The change to the aperture needed by the configuration file can be done easily if the distance source-detector is twice the distance isocenter-detector. In this case, the user can apply the theorem of the central angle: The central angle subtended by two points on a circle is twice the inscribed angle subtended by those points. This theorem only holds when the source point is in the major arc. If this point is in the minor arc, it is, between the detector limits, the relationship between these two angles is that the inscribed angle is the supplement of half the central angle.

- **particle**: particle type that will be collect by the detector.
- **tmin**: Minimum time stamp of the particles of the first projection.
- **timeInterval**: Time interval between projections.
- **scattered**: This parameter offers the possibility to collect all the photons in the energy range if it is set to true or collect only photons which have not been suffer scatter. This is an optional parameter which is set to true by default.

```

1 tallies/CTsinogram/type "CT.SINOGRAM" (mandatory)
2 tallies/CTsinogram/emin 0.0 (mandatory)
3 tallies/CTsinogram/emax 1.0e5 (mandatory)
4 tallies/CTsinogram/npixels 700 (mandatory)
5 tallies/CTsinogram/pixel-depth 4 (mandatory)
6 tallies/CTsinogram/r-inner 52.0 (mandatory)
7 tallies/CTsinogram/xOrigin 0.0 (mandatory)
8 tallies/CTsinogram/yOrigin 0.0 (mandatory)
9 tallies/CTsinogram/zOrigin 0.0 (mandatory)
10 tallies/CTsinogram/phi-ini 0 (mandatory)
11 tallies/CTsinogram/phi-end 360 (mandatory)
12 tallies/CTsinogram/angularStep 0.5 (mandatory)
13 tallies/CTsinogram/aperture 44.0 (mandatory)
14 tallies/CTsinogram/particle "gamma" (mandatory)
15 tallies/CTsinogram/tmin 0.0 (mandatory)
16 tallies/CTsinogram/timeInterval 120.0 (mandatory)
17 tallies/CTsinogram/scattered true (mandatory)

```

Code 59: CT tally configuration

#### 5.4.13 DICOM kerma tally (Dose Volume Histogram)

This tally has been implemented to obtain the Dose Volume Cumulative Histogram (DVH) for each contoured structure of a DICOM image. Therefore, this tally requires a DICOM based geometry to be used. In addition, the specified DICOM must include a RTSTRUCT file with the contour information.

To obtain the absorbed dose in Gy for each volume, this tally calculates the kerma, using the kerma track length tally described in the section 5.4.11. However, because the voxelized structure of DICOM images, it uses only the cartesian scoring method. The tally information needed in the configuration file is described following:

- **type “DICOM KERMA TRACK LENGTH”**: Type name of the tally.
- **kerma/emin**: Photon minimum energy to be considered (eV).
- **kerma/emax**: Photon maximum energy to be considered (eV).
- **kerma/dataFiles**: One data file with the  $\mu_{en}/\rho$  ( $cm^2/g$ ) coefficients per tallied material is required. This parameter is detailed in the Kerma track length estimator tally, section 5.4.11.
- **prescribedDose**: Treatment prescribed dose, in Gy. If this parameter has not been specified, it is set to 1 Gy.
- **MCkerma2Dose**: Conversion factor from kerma MC simulation units (eV/g per history) to absolute dose units, which must be set in Gy/(eV/g per history). The *MCkerma2Dose* factor is calculated using the equation 2.

$$MCkerma2Dose = \frac{1}{100} \frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \quad (2)$$

where  $S_{k,PLAN}$  is the *source air-kerma strength* [12] which can be found in the DICOM plan in U units ( $1U = cm^2 \cdot cGy/h$ ). The  $S_{k,MC}$  value is the *source air-kerma strength* obtained for the source with Monte Carlo simulation. The last one can be calculated with the following relation,

$$S_{k,MC} = K_{MC}(d)d^2 \left[ \frac{eV \cdot cm^2}{g \cdot history} \right] \quad (3)$$

where  $K_{MC}$  is the kerma in air in a thin air cell, such as a ring with  $0.1cm$  width and  $0.1cm$  depth, located on the seed transversal axis at a reference distance  $d$ , which is usually set to  $d = 10$  cm or  $d = 100$  cm. Also, the source must be located in a vacuum media. This configuration example is shown in the figure 14.

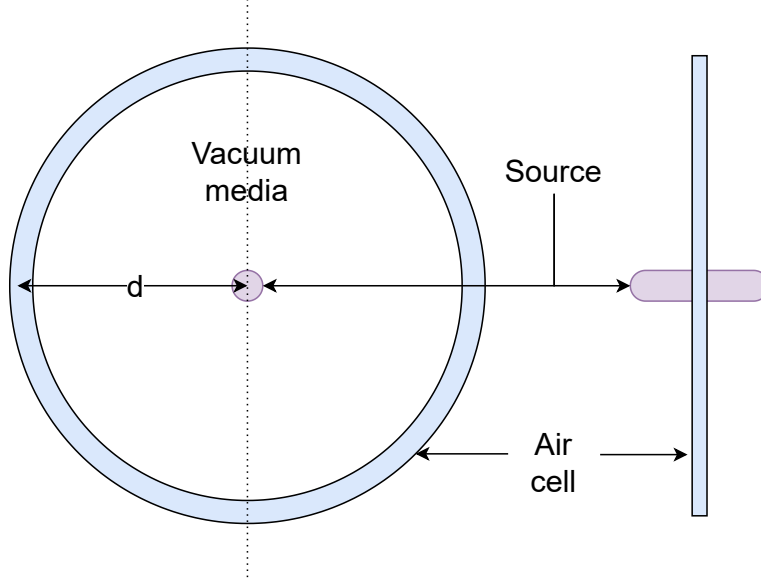


Figure 14: Geometry schema to calculate the  $S_{k,MC}$  factor using a ring air cell situated at a reference distance  $d$ . The left image shows the central  $Z$  plane. The right image shows the central  $X$  or  $Y$  plane.

Thus, the resulting units for the  $S_k$  quotient are,

$$\begin{aligned} & \frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[ \frac{g \cdot history}{eV \cdot cm^2} \right] \left[ \frac{cm^2 \cdot cGy}{h} \right] [h] \\ & \frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[ \frac{cGy \cdot history}{(eV/g)} \right] \end{aligned} \quad (4)$$

Converting to Gy, we obtain the units of the equation 2.

$$MCkerma2Dose = \frac{1}{100} \frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[ \frac{Gy \cdot history}{(eV/g)} \right] \quad (5)$$

If the conversion factor is not specified, it will be set to 1.

- **DVH-maxDose:** Maximum dose value for the DVHs. If it is not specified, its value is set to 3 times the prescribedDose.
- **DVH-bins:** Number of bins for the DVHs. If this parameter has not been specified, it is set to 200.



```

1 tallies /DICOMkerma/type "DICOMKERMA_TRACKLENGTH" (mandatory)
2
3
4 tallies /DICOMkerma/kerma/emin 2.5e3 (mandatory)
5 tallies /DICOMkerma/kerma/emax 1.39950E+06 (mandatory)
6 tallies /DICOMkerma/kerma/dataFiles/1 "prostate-mutren.dat" (mandatory)
7 tallies /DICOMkerma/kerma/dataFiles/2 "uretra-mutren.dat" (mandatory)
8 tallies /DICOMkerma/kerma/dataFiles/3 "rectum-mutren.dat" (mandatory)
9 tallies /DICOMkerma/kerma/dataFiles/4 "bladder-mutren.dat" (mandatory)
10 tallies /DICOMkerma/kerma/dataFiles/5 "body-mutren.dat" (mandatory)
11 tallies /DICOMkerma/kerma/dataFiles/6 "bone-mutren.dat" (mandatory)
12
13 tallies /DICOMkerma/prescribedDose 15 (optional)
14 tallies /DICOMkerma/MCkerma2Dose 0.06831 (optional)
15 tallies /DICOMkerma/DVH-maxDose 40 (optional)
16 tallies /DICOMkerma/DVH-bins 1000 (optional)

```

Code 60: DICOM kerma tally configuration

Notice that the DICOM is not specified in the source configuration. This is because the DICOM image is taken directly from the geometry.

#### 5.4.14 Tracks

The *Track* tally stores the simulated particle tracks in separate files by particle type. Note that a single history could produce long tracks, especially for electrons, and the generated files can use a large amount of disk space if the number of tracked histories is large. It is recommended not to track more than 50 histories, depending on the energy, simulated particles, and geometry. The tally parameters are:

- **type**: Must be set to **TRACK**.
- **nhists**: Specify the number of histories to be tracked.

Take into account that only particles reaching the geometry system will be tracked. That is, if a particle is sampled in a void region and never reaches a non-void body, it will not appear in the corresponding track file.

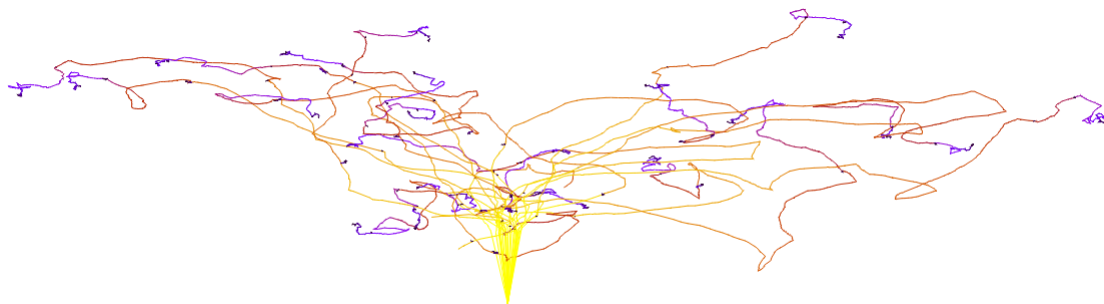


Figure 15: Example of electron tracks in a water medium. The track color shows changes in the electron energy.

In the generated files, particle tracks are separated by two blank lines, and each line contains the whole state of a particle at the specified point.

#### 5.4.15 Primary and Scatter Separated (PSS)

This one is a wrapper tally which score the contribution of primary, scattered and multi-scattered radiation in three independent counters. To be able to combine it with different counting procedures, it delegates the counting process to another existing tally type. However, it has been designed for tallies scoring energy/dose deposition and will not be compatible with all existing tallies.

To determine if a particle is considered primary, scattered or multi-scattered radiation, a counter or bookkeeping (*bkp*) is used for each generated particle. However, only interactions and productions of the selected particle type will increase the counter. For example, for photons as the main particle, the following rules applies:

- Primary photons ( $\gamma_1$ ) start with  $bkp = 0$ . Any photon interaction (e.g., Compton) occurring outside the source generates  $bkp+ = 1$  (e.g.,  $bkp_{\gamma_1} = 1$ )
- Any secondary particle (e.g., Compton electrons),  $e_1$ , inherits the generation of the incident photon (e.g.,  $bkp_{e_1} = 0$ ).
- Any photon produced by these secondary particles (e.g., bremsstrahlung photon),  $\gamma_2$ , needs  $bkp+ = 1$  (e.g.,  $bkp_{\gamma_2} = 1$ ).
- The source materials (e.g., active core and encapsulation) don't play any role in the  $bkp$  of primary photons. However, once the primary particle is scattered once in the phantom (e.g., water), the source starts acting like any other material, adding +1 to the scattered photons'  $bkp$ .

The configuration of this tally contains the following parameters,

- **type "PSS"**: Type name of the tally.
- **PSS-particle**: Specify which particle must be considered as main radiation. If it is not specified, photons are selected by default.
- **PSS-source-mats**: Consists of an array with the material indexes considered as source materials. If it is not specified, no material will be considered as source.
- **PSS-tally**: Specify the tally type to be used to perform the scoring. For example **SPATIAL\_DOSE\_DISTRIB** could be used. As discussed, not all tallies are compatible with the PSS. The following tally types are compatible,

- "ANGULAR\_DET"
- "CYLINDRICAL\_DOSE\_DISTRIB"
- "DICOM\_DOSE\_DISTRIB"
- "DICOM\_KERMA\_TRACK\_LENGTH"
- "EDEP\_BODY"
- "EDEP\_MAT"
- "EMERGING\_PART\_DISTRIB"
- "KERMA\_TRACK\_LENGTH"
- "SPATIAL\_DOSE\_DISTRIB"
- "SPHERICAL\_DOSE\_DISTRIB"

and the non compatible types are,

- "CT\_SINOGRAM"
- "IMPACT\_DET"
- "PSF"
- "PSS"
- "SECONDARY\_GEN"
- "TRACK"

If the specified tally is not in either of the two lists a warning will be shown and the correct behaviour of the tally is not guaranteed.

In addition to the **PSS** parameters, the specific parameters of the selected tally must be specified to configure them, as is shown in the Code 61. In this example, a cylindrical dose tally is used and materials 1 and 2 are considered as source materials.

```

1 tallies/cylDoseDistribPSS/type "PSS"
2 tallies/cylDoseDistribPSS/PSS-source-mats [1,2]
3 tallies/cylDoseDistribPSS/PSS-tally "CYLINDRICAL_DOSE_DISTRIB"
4 tallies/cylDoseDistribPSS/print-xyz true
5 tallies/cylDoseDistribPSS/rmin 0.0
6 tallies/cylDoseDistribPSS/rmax 0.01
7 tallies/cylDoseDistribPSS/nbinsr 50
8 tallies/cylDoseDistribPSS/zmin 0.0
9 tallies/cylDoseDistribPSS/zmax 0.005
10 tallies/cylDoseDistribPSS/nbinsz 100

```

Code 61: PSS tally configuration example combined with a cylindrical dose distribution

#### 5.4.16 Detection Spatial Distribution

This tally records the energy spectrum of particles reaching a specified detector within a 3D grid. The available configuration parameters are as follows:

- **type**: Must be set to "DETECTION\_SPATIAL\_DISTRIB"
- **spatial/nx**: Specifies the number of spatial bins to use along the  $X$  axis. This value is optional and defaults to 1.
- **spatial/xmin**: Sets the origin of the spatial grid along the  $X$  axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/xmax**: Sets the limit of the spatial grid along the  $X$  axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/ny**: Specifies the number of spatial bins to use along the  $Y$  axis. This value is optional and defaults to 1.
- **spatial/ymin**: Sets the origin of the spatial grid along the  $Y$  axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/ymax**: Sets the limit of the spatial grid along the  $Y$  axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/nz**: Specifies the number of spatial bins to use along the  $Z$  axis. This value is optional and defaults to 1.

- **spatial/zmin**: Sets the origin of the spatial grid along the  $Z$  axis, in cm. This is required only if *spatial/nz* is specified.
- **spatial/zmax**: Sets the limit of the spatial grid along the  $Z$  axis, in cm. This is required only if *spatial/nz* is specified.
- **detector**: Specifies the detector index where the data will be collected. The detector must be previously defined in the geometry configuration.
- **energy/nbins**: Specifies the number of energy bins to register the spectrum. This value is optional and defaults to 1.
- **energy/emin**: Sets the lower limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **energy/emax**: Sets the upper limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **particle**: Specifies the particle to be registered by name (text identifier).
- **printBins**: This optional boolean value enables/disables printing the spatial and energy bin indexes in the results file. The default is disabled.
- **printCord**: This optional boolean value enables/disables printing the bin coordinates, both spatial and energetic, in the results file. The default is enabled.

A configuration sample is shown in the Code 62, corresponding to the example *quadrics/9-measure-source/fromSource*.

```

1 tallies/SpatialDetector1/type "DETECTION_SPATIAL_DISTRIB"
2 tallies/SpatialDetector1/spatial/nx 100
3 tallies/SpatialDetector1/spatial/xmin -8.0
4 tallies/SpatialDetector1/spatial/xmax 8.0
5 tallies/SpatialDetector1/spatial/ny 50
6 tallies/SpatialDetector1/spatial/ymin -4.0
7 tallies/SpatialDetector1/spatial/ymax 4.0
8 tallies/SpatialDetector1/detector 1
9 tallies/SpatialDetector1/particle "gamma"
10 tallies/SpatialDetector1/energy/nbins 100
11 tallies/SpatialDetector1/energy/emin 10e3
12 tallies/SpatialDetector1/energy/emax 1.5e5

```

Code 62: Detection spatial distribution tally configuration

#### 5.4.17 Emerging Spherical Distribution

This tally records the energy spectrum of particles emerging from the geometry system within a spherical grid. In addition, the last interaction position of emerging particles is recorded in a spatial 3D grid. The available configuration parameters are as follows:

- **type**: Must be set to “EMERGING\_SPHERICAL\_DISTRIB”
- **spatial/nx**: Specifies the number of bins to use along the  $X$  axis within the spatial distribution. This value is optional and defaults to 1.
- **spatial/xmin**: Sets the origin of the spatial grid along the  $X$  axis, in cm. This is required only if *spatial/nx* is specified.

- **spatial/xmax**: Sets the limit of the spatial grid along the  $X$  axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/ny**: Specifies the number of spatial bins to use along the  $Y$  axis within the spatial distribution. This value is optional and defaults to 1.
- **spatial/ymin**: Sets the origin of the spatial grid along the  $Y$  axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/ymax**: Sets the limit of the spatial grid along the  $Y$  axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/nz**: Specifies the number of spatial bins to use along the  $Z$  axis within the spatial distribution. This value is optional and defaults to 1.
- **spatial/zmin**: Sets the origin of the spatial grid along the  $Z$  axis, in cm. This is required only if *spatial/nz* is specified.
- **spatial/zmax**: Sets the limit of the spatial grid along the  $Z$  axis, in cm. This is required only if *spatial/nz* is specified.
- **theta/nbins**: Specifies the number of polar bins. This value is optional and defaults to 1.
- **theta/min**: Sets the minimum polar angle to tally, in degrees. This value defaults to 0.
- **theta/max**: Sets the maximum polar angle to tally, in degrees. This value defaults to 180.
- **phi/nbins**: Specifies the number of azimuthal bins. This value is optional and defaults to 1.
- **phi/min**: Sets the minimum azimuthal angle to tally, in degrees. This value defaults to 0.
- **phi/max**: Sets the maximum azimuthal angle to tally, in degrees. This value defaults to 360.
- **energy/nbins**: Specifies the number of energy bins to register the spectrum. This value is optional and defaults to 1.
- **energy/min**: Sets the lower limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **energy/max**: Sets the upper limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **particle**: Specifies which particles will be registered by name (text identifier). The particle name is specified in the path and a boolean value is set to enable/disable it:  
particle/\${particle-name} value

By default, all particles are disabled.

- **printBins**: This optional boolean value enables/disables printing the spatial and energy bin indexes in the results file. The default is disabled.

- **printCord:** This optional boolean value enables/disables printing the bin coordinates, both spatial and energetic, in the results file. The default is enabled.

A configuration sample is shown in the Code 63.

```

1 tallies/angularEmerging/type "EMERGING_SPHERICAL_DISTRIB"
2 tallies/angularEmerging/energy/min 5.0e3
3 tallies/angularEmerging/energy/max 50e3
4 tallies/angularEmerging/energy/nbins 90
5 tallies/angularEmerging/theta/nbins 720
6
7 tallies/angularEmerging/spatial/nx 1
8 tallies/angularEmerging/spatial/xmin -10.0
9 tallies/angularEmerging/spatial/xmax 10.0
10 tallies/angularEmerging/spatial/ny 1
11 tallies/angularEmerging/spatial/ymin -10.0
12 tallies/angularEmerging/spatial/ymax 10.0
13 tallies/angularEmerging/spatial/nz 200
14 tallies/angularEmerging/spatial/zmin 0.0
15 tallies/angularEmerging/spatial/zmax 100.0
16
17 tallies/angularEmerging/particle/gamma true

```

Code 63: Emerging spherical distribution tally configuration

## 5.5 Variance Reduction

PenRed implements the very same variance reduction (VR) techniques as the original FORTRAN code and some more. These are, for generic simulations, interaction forcing (*IF*), x-ray and bremsstrahlung splitting. Then, phase space file based simulations adds particle splitting and Russian Roulette techniques. Note that variance reduction for phase space file will be configured via the corresponding particle source parameters, as we saw on section 5.2. In addition, PenRed implements generic splitting and Russian roulette as VR modules and provide the capabilities to add new ones, as is explained in the section 5.5.

We have two types of VR techniques. First, the context specific techniques, which are implemented inside the particle class, such as interaction forcing or bremsstrahlung splitting. On the other hand, we have the VR techniques implemented as independent modules, such as x-ray and generic splitting or Russian roulette.

### 5.5.1 Context specific

To use the first kind of techniques, the parameters in the configuration file use the prefix *VR*, as shows code 64. There, *vr-technique* can be interaction forcing (*IForcing*) or bremsstrahlung splitting (*bremss*). Then, on *objectToApply*, the user can select *bodies* or *materials* to apply the variance reduction technique to a single body or to an entire material respectively. Next, *name* is a user defined text identifier for this VR technique. Finally, *parameter/path* and *value* depends on the VR type.

```

1 VR/vr-technique/objectToApply/name/parameter/path/path value

```

Code 64: Variance reduction configuration pattern

First, lets see the interaction forcing (IF) configuration. The code 65 shows a complete interaction forcing configuration used in the example *4-x-ray*. As we can see, this example enables interaction forcing in a single body, which alias is specified by the parameter *body* as a string. Then, specify the kind of particle to force using the parameter *particle* and the interaction numerical identifier to force (*interaction*). That identifier can be found on

section 4.4. The next parameter, *factor*, sets the interaction forcing amplification factor and, finally, *min-weight* and *max-weight* limits the weight window where apply this VR technique. Notice that a negative *factor* value will be interpreted as in the PENELOPE main program, i.e., is assumed to mean that a particle with energy  $E = EPMAX$  should interact, on average,  $+factor$  times in the course of its slowing down to rest, for electrons and positrons, or along a mean free path, for photons.

```
1 VR/IForcing/bodies/VR1/body "1"
2 VR/IForcing/bodies/VR1/particle "electron"
3 VR/IForcing/bodies/VR1/interaction 2
4 VR/IForcing/bodies/VR1/factor 400
5 VR/IForcing/bodies/VR1/min-weight 0.1
6 VR/IForcing/bodies/VR1/max-weight 2
```

Code 65: Interaction forcing configuration for bodies

To set the same interaction forcing on materials, simply substitute *bodies* by *materials*, *body* parameter by *mat-index* and set the material index where apply VR as a integer value. For example, to apply this IF on material 2 the configuration should look like code 66.

```
1 VR/IForcing/materials/VR1/mat-index 2
2 VR/IForcing/materials/VR1/particle "electron"
3 VR/IForcing/materials/VR1/interaction 2
4 VR/IForcing/materials/VR1/factor 400
5 VR/IForcing/materials/VR1/min-weight 0.1
6 VR/IForcing/materials/VR1/max-weight 2
```

Code 66: Interaction forcing configuration for materials

Next technique, bremsstrahlung splitting, requires one or two parameters depending on whether the VR is specified for bodies or materials respectively. Both patterns are shown in the code 67, where *body-alias* must be substituted by the body alias where we want to apply the splitting, *splitting-factor* specify the number of times bremsstrahlung photons will be cloned and *imat* is the material index where apply splitting.

```
1 VR/bremss/bodies/body-alias/splitting splitting-factor
2
3 VR/bremss/materials/mat-index imat
4 VR/bremss/materials/splitting splitting-factor
```

Code 67: Bremsstrahlung splitting configuration for bodies and materials

### 5.5.2 Specific VR techniques

A specific VR techniques can be used only on a specific particle state type. All of them, when used in the PenRed provided main program, follows the configuration pattern code 68.

```
1 VR/type/vr-name/parameter/path value
```

Code 68: Specific VR technique parameter configuration pattern

Actually, PenRed only provides specific VR techniques for photons. Thus, the only possible value for the specific VR type is *photon*.

```
1 VR/photon/vr-name/parameter/path value
```

Code 69: Photons VR technique parameter configuration pattern

The parameter *vr-name* specify a name for this VR instance and is only used for identification purposes.

### 5.5.2.1 X-Ray splitting

X-Ray splitting uses a pattern similar to bremsstrahlung case, an example can be found in the example *1-disc-vr* and all the available parameters are summarised in the code 70.

```
1 VR/photon/vr-name/type "XRAY_SPLITTING" (mandatory)
2
3 VR/photon/vr-name/bodies/body-alias/splitting splitting-factor (optional)
4
5 VR/photon/vr-name/materials/mat-index imat (optional)
6 VR/photon/vr-name/materials/splitting splitting-factor (optional)
```

Code 70: X-ray splitting configuration for bodies and materials

First, the VR type is specified by the parameter *type*. Then, the other parameters *VR/photon/name/bodies/...* and *VR/photon/name/materials/...* are analogous to the bremsstrahlung case.

### 5.5.3 Generic VR techniques

Generic VR techniques can be used on all particle state types. All of them, when used in the PenRed provided main program, follows the configuration pattern shown in the code 71.

```
1 VR/generic/vr-name/parameter/path value
```

Code 71: Specific VR technique parameter configuration pattern

where *vr-name* specify the VR instance name for identification purposes.

#### 5.5.3.1 Splitting

The splitting VR technique consists of cloning the particle and reducing its weight by a factor equal to the number of clones. Thus, if a particle with a weight *WGHT* is split in 10 clones, counting itself, the weight of all the resulting particles will be *WGHT/10*. An example of configuration can be found in the example *8-fake-chamber*. Also, all the parameters are shown in the code 72 and explained below,

- **type**: Specify the VR type, must be set to *"SPLITTING"*.
- **minWght**: Specify the minimum particle weight to apply splitting. Particles with lesser weight than the specified will not be split.
- **maxWght**: Specify the maximum particle weight to apply splitting. Particles with greater weight than the specified will not be split.
- **materials/mat-name**: Subsection to specify splitting parameters for materials. Notice that **mat-name** is only used as identifier in the VR instance, and will not be identified with the name used in the geometry configuration. Instead, the material is specified by the integer index **mat-index**. Finally, the number of cloned particles in this material is specified by the **splitting** parameter. This parameter can be repeated for different material indexes.
- **bodies/body-alias/splitting**: There, *body-alias* must identify a valid body in the configured geometry. The specified value of **splitting** will be used in this body. Notice that *bodies* configuration overwrite materials ones. This parameter can be repeated for different bodies.



```

1 VR/generic/vr-name/type "SPLITTING" (mandatory)
2 VR/generic/vr-name/minWght 0.05 (mandatory)
3 VR/generic/vr-name/maxWght 21.0 (mandatory)
4
5 VR/generic/vr-name/materials/mat-name/mat-index 2 (optional)
6 VR/generic/vr-name/materials/mat-name/splitting 20 (optional)
7
8 VR/generic/name/bodies/body-alias/splitting 20 (optional)

```

Code 72: Splitting configuration for bodies and materials

Notice that the splitting is only applied when a particle enter to the material/body after crossing an interface. Therefore, is not applied to secondary particles generated inside the material and is not triggered in bodies if the geometry has not an interface in their boundaries. To force a interface, the user can create a detector or use different materials.

### 5.5.3.2 Russian roulette

Russian roulette technique kills the particles with a probability specified in the configuration (**prob**). If the particle survive, its weight is multiplied by a factor  $1/prob$ . All the parameters of the Russian roulette technique are equivalent to the *splitting* class parameters (section 5.5.3.1). The only difference is that the **splitting** parameter is substituted by the **prob** parameter, which specify the particle survival probability. The parameters are summarised in the code 73. In addition, the **type** parameter must be set to "RUSSIAN\_ROULETTE" instead of "SPLITTING". An example of configuration can be found in the example 8-*fake-chamber*.

```

1 VR/generic/vr-name/type "RUSSIAN_ROULETTE" (mandatory)
2 VR/generic/vr-name/minWght 0.05 (mandatory)
3 VR/generic/vr-name/maxWght 21.0 (mandatory)
4
5 VR/generic/vr-name/materials/mat-name/mat-index 2 (optional)
6 VR/generic/vr-name/materials/mat-name/prob 20 (optional)
7
8 VR/generic/name/bodies/body-alias/prob 20 (optional)

```

Code 73: Russian roulette configuration for bodies and materials

Like in the *splitting* case, Russian roulette only triggers when a particle cross an interface.

## 5.6 Simulation parameters

Simulation parameters are main specific, unlike source, tally, geometry or material configurations. This section will explain the available parameters for the "pen.main" program. All parameters follows the pattern in the code 74.

```

1 simulation/parameter/path value

```

Code 74: Simulation parameters configuration pattern

### 5.6.1 Dump

PenRed main program provides the capability to dump the current state of the whole simulation. This dump can be used to resume a crashed simulation, store the final results in binary format, or add the results of simulations with the same configuration. To configure this feature, the user can use a set of configuration parameters, which are explained following.

- **dump-interval**: Specify the time, in seconds, between successive dumps

- **dump2read**: Expects a string with the name of a dump file to be resumed. This file will be read before the simulation beginning to continue a previous simulation. Notice that, if multiple threads are used, the simulation must be resumed with the same number of threads as the original one. Each thread will try to read the specified file name with a prefix *thN*, where *N* specify the thread number. For example, if the filename is “**dump.dat**”, the file corresponding to the thread number 2 must be named “**th2dump.dat**”.
- **dump2write**: Specify the file name used to save the dumps. Each thread will append the *thN* prefix to the filename automatically, where *N* specify the thread number. If is not specified, dump files will be named with the default value **dump.dat**.
- **dump2ascii**: Is a bool value that must be used with the **dump2read** parameter. If enabled, the simulation will not be resumed. Instead, the program will load the state stored in the specified dump file and extract the tally contents using the usual ASCII data reports. Is disabled by default.
- **finalDump**: This bool value can be used to force a dump in each thread at the end of the simulation. Is disabled by default.
- **ascii-results**: A bool value that enables or disables the result reports in ASCII format. If disabled, the result reports will be write in a dump file named **results.dump**. In addition, no partial results will be printed during the simulation. Is enabled by default.
- **partial-results**: If **ascii-results** is enabled, this bool value can enable or disable the ASCII printing of partial results. Is disabled by default.

An example of dump configuration can be found in the code 75, where the simulation will resume the stored state at dump file *dump.dat* and store the new generated dumps to *dump2.dat*. New dumps will be generated every 3600 s and the final dump is disabled.

```

1 simulation/dump-interval 3600
2 simulation/dump2read  "dump.dat"
3 simulation/dump2write "dump2.dat"
4 simulation/finalDump  false
5 simulation/dump2ascii false
6 simulation/ascii-results true

```

Code 75: Dump configuration parameters

### 5.6.1.1 Adding dumps

Another feature related with dumped simulations is the possibility to add dump files. This can be done via the command line with the option **--addDumps**. To use this option, the user must provide the usual configuration file of the simulation, and all the dump files to be added as follows,

```
./pen_main config.in --addDumps dumpFile1 dumpFile2 dumpFile3
```

where can be specified as many files as required. Notice that no simulation will be carried on when this option is specified in the command line. Also, all added dump files must have been generated using the same configuration. As result, a new dump file named **mergedDump.dump** will be generated with the added information from all specified files. Furthermore, if **ascii-results** is enabled, the corresponding ASCII reports will be saved too.

### 5.6.2 Threading

Another type of simulation parameters are those that control the multi-threading capabilities. These parameters are described following.

- **nthreads**: Expects an integer value that specify the number of threads to be used. If this parameter is set to 0 or less, the number of threads will be calculated automatically according to the value returned by the function,

`std::thread::hardware_concurrency()`

By default a single thread is used. In MPI executions, this value is used to specify the number of threads in each MPI rank.

- **thread-affinity**: This parameter expects a boolean value to enable or disable CPU affinity. Actually, this feature can be used only if threads are implemented via *pthread*s package, i.e., in most Unix environments. Is disabled by default.
- **simulation/rank/N/threads**: This parameter specify the number of threads to be used in the MPI rank number  $N$ , which value must be set to the desired rank. Notice that this one overwrites the **nthreads** parameter. An example is shown in the code 77.

The number of histories to simulate on each source will be distributed among all specified threads.

```
1 simulation/threads 2
2 simulation/thread-affinity true
```

Code 76: Number of threads specification

Notice that MPI can be combined with multi-threading. However, the number of MPI processes is not specified in the configuration file. Instead, this is specified via the *mpirun* parameters (see section 2). So, when both kinds of parallelism are combined, the number of threads spawned in each MPI rank is calculated according to the parameters specified in the configuration file. This approach “suggests” to the user to create a single MPI process for each node in a distributed memory infrastructure and use threads instead of more MPI processes. With this method, each node will use less memory because data bases will be shared by all threads in the same node. Furthermore, the memory access will be more efficient due to the memory sharing between threads.

In addition, it is possible to specify a different number of threads for each MPI process. For example, the configuration in the code 77 specify two threads for each MPI rank except for the ranks 1 and 2, which will use 4 and 8 threads respectively.

```
1 simulation/threads 2
2 simulation/rank/1/threads 4
3 simulation/rank/2/threads 8
```

Code 77: Number of threads specification

### 5.6.3 Seeds

The user can specify the initial random generator seeds using *seed1* and *seed2* parameters. Another option consists of selecting a seed pair provided by *rand0* [13] function via the *seedPair* parameter. Notice that on multi-threading and/or MPI simulations, only the parameter *seedPair* can be used to set initial seeds. This restriction is necessary to ensure that each random number chain is truly independent.

```

1 simulation/seed1 1
2 simulation/seed2 1
3 simulation/seedPair 12

```

Code 78: Initial seeds

There are 1001 possible seed pairs to be used.

#### 5.6.4 Simulation time

It is possible to limit the maximum allowed simulation time. If this limit is reached, PenRed will produce the corresponding dump files with the simulation state and finish its execution. To specify that time, only one parameter is required,

- **max-time:** Maximum simulation time in seconds.

For example, the Code 79 shows a configuration line specifying a maximum simulation time of 5 minutes.

```

1 simulation/max-time 300

```

Code 79: Maximum simulation time

### 5.7 Logs

By default, logs are saved in two separate files: one for the configuration process, named *config.log*, and another for the simulation, named *simulation.log*. These log files can be modified using the following configuration parameters:

- **log/configuration:** Expects a **string** value, specifying the file where the configuration log will be saved. The default is *config.log*.
- **log/simulation:** Expects a **string** value, specifying the file where the simulation log will be saved. The default is *simulation.log*.

In both cases, if an empty string is provided as the value:

```

log/configuration ""
log/simulation ""

```

the configuration and/or simulation logs will be redirected to the standard output.

### 5.8 Image output

The provided main program allows to export some of the simulation outputs as images. Which outputs can be exported as images depends on each component implementation, for example, many tallies uses this capability. To enable the image exporter, the field *images* must be included in the *simulation* section followed by the format to enable/disable, as is shown in the code 80. Actually, the two available image formats are *MHD* [14], which is supported by many viewers and can be read with the ITK library, and *GNU*, which consists of a format ready to be plotted as a matrix with the Gnuplot [15] tool. Each format can be enabled or disabled independently of the others. For instance, the code 80 shows how to enable both formats

```

1 simulation/images/mhd true
2 simulation/images/gnu true

```

Code 80: Image export

## 5.9 Load balance parameters

As simulation parameters, load balance parameters are main dependent. This optional feature can be enabled during the cmake configuration with the option `WITH_LB`. Actually, the only mandatory parameter to configure the load balance system is `balance-interval` (code 81). This one specify the minimum balance time interval in seconds. Specifying this simple parameter, PenRed simulation threads and MPI processes, if enabled, will be balanced automatically.

In addition, PenRed simulations can be balanced using a centralised server to provide a method to balance simulations across the internet. To start a balance server, the PenRed package provides a balance server example code in `src/utilities/LB/LBserver.cpp`. In addition, in the configuration, the user must specify the balance server hostname or IP as string (`loadBalance/host`) and port (`loadBalance/port`). Also, a worker number identification must be specified (`loadBalance/worker`).

Finally, if the SSL support has been enabled at the cmake configuration with the parameter `WITH_SSL`, the communications between workers and the balance server will be secured with the SSL protocol. This feature requires a set of certificates listed below.

- **CA-cert**: Certification chain file of the trusted CA.
- **cert-file**: Filename of the worker certificate file.
- **key-file**: Filename of the worker key file.
- **key-password**: Worker key password.
- **hostname**: Expected server certificate hostname.

```
1 loadBalance/balance-interval 500    (mandatory)
2 loadBalance/host "server-hostname" (optional)
3 loadBalance/port 5555               (mandatory if enabled "host")
4 loadBalance/worker 0                (mandatory if enabled "host")
5 loadBalance/CA-cert                 (optional)
6 loadBalance/cert-file               (mandatory if enabled "CA-cert")
7 loadBalance/key-file                (mandatory if enabled "CA-cert")
8 loadBalance/key-password             (optional)
9 loadBalance/hostname                (optional)
```

Code 81: Load balance parameters

## 6 Python wrapper

A Python wrapper named (**pyPenred**) is included in the penRed library and can be compiled by the user to create a package that can be installed via `pip`. This one is still experimental and could change in future versions. To install it, execute the provided scripts: `installPyPenred.sh` for Unix systems or `installPyPenred.bat` for Windows systems.

These scripts compile the *penRed* code with the **BUILD\_PYTHON\_MODULES** flag enabled, creating the Python bindings using the **pybind11** package. If **pybind11** is not already installed on the system, the CMake script will automatically download it from GitHub. Therefore, the only additional required dependencies are the Python development packages for C++, which must be pre-installed.

After compilation, the script will install the Python package located in:

src/bindings/python/pyPenred

using the `pip install .` command. Hence, `pip` must also be installed on your system. Once the installation is successful, the package will be available for use within a Python environment.

## 6.1 Configuration format

To allow users to edit their configurations directly within the Python environment, a straightforward method for converting the classic `penRed` configuration format to Python dictionaries has been defined. This method simply involves converting each slash in the parameter paths into a dictionary level.

For example, Codes 82 and 83 illustrate the same configuration chunk for the *2-plane* example. The first code block shows the traditional `penRed` configuration file format, while the second is a YAML file that can be loaded into a Python dictionary for use with `pyPenred`. Additionally, classic `penRed` configuration files can also be loaded by `pyPenred` and converted into Python dictionaries. This feature will be described in detail in the usage section. Furthermore, the utility `internalData2YAML` (Section 8.11) allows for the conversion of any `penRed` configuration file into YAML format.

```
1 # Source #
2 sources/generic/source1/nhist 1.0e9
3 sources/generic/source1/kpar "gamma"
4 sources/generic/source1/record-time true
5
6 # Directional sampling
7 sources/generic/source1/direction/type "CONE"
8 sources/generic/source1/direction/theta 0.0
9 sources/generic/source1/direction/phi 0.0
10 sources/generic/source1/direction/alpha 5.0
11
12 # Energy sampling
13 sources/generic/source1/energy/type "MONOENERGETIC"
14 sources/generic/source1/energy/energy 3.0e7
15
16 # Spatial sampling
17 sources/generic/source1/spatial/type "POINT"
18 sources/generic/source1/spatial/position/x 0.0
19 sources/generic/source1/spatial/position/y 0.0
20 sources/generic/source1/spatial/position/z -25.0
21
22 # GEOMETRY PARAMETERS #
23
24 geometry/type "PEN_QUADRIC"
25 geometry/input-file "plane.geo"
26 geometry/processed-geo-file "report.geo"
27 geometry/kdet/1 1
28
29 # MATERIALS #
30
31 materials/water/number 1
32 materials/water/eabs/electron 1.0e5
33 materials/water/eabs/positron 1.0e5
34 materials/water/eabs/gamma 1.0e4
35 materials/water/C1 0.05
36 materials/water/C2 0.05
37 materials/water/WCC 2.0e3
38 materials/water/WCR 2.0e3
39 materials/water/filename "water.mat"
40
```

```

41 # SIMULATION PARAMETERS #
42
43 simulation/threads 2
44 simulation/max-time 600

```

Code 82: Configuration with the usual penRed format for the example *2-plane*.

```

1 geometry:
2   input-file: "plane.geo"
3   kdet:
4     1: 1
5   processed-geo-file: "report.geo"
6   type: "PEN.QUADRIC"
7 materials:
8   water:
9     C1: 5.00000E-02
10    C2: 5.00000E-02
11    WCC: 2.00000E+03
12    WCR: 2.00000E+03
13    eabs:
14      electron: 1.00000E+05
15      gamma: 1.00000E+04
16      positron: 1.00000E+05
17    filename: "water.mat"
18    number: 1
19 simulation:
20   max-time: 600
21   threads: 2
22 sources:
23   generic:
24     source1:
25       direction:
26         alpha: 5.00000E+00
27         phi: 0.00000E+00
28         theta: 0.00000E+00
29         type: "CONE"
30       energy:
31         energy: 3.00000E+07
32         type: "MONOENERGETIC"
33       kpar: "gamma"
34       nhist: 1.00000E+09
35       record-time: true
36       spatial:
37         position:
38           x: 0.00000E+00
39           y: 0.00000E+00
40           z: -2.50000E+01
41         type: "POINT"

```

Code 83: Configuration file in YAML format for the example *2-plane*.

## 6.2 Usage

Currently, the *pyPenred* wrapper does not implement all the features of the C++ penRed library. For example, it lacks support for MPI and the capability to sum dumps. However, all core simulation components, such as tallies, geometries, and samplers, are fully functional. As a result, simulations configured for penRed can also be executed using *pyPenred*.

The easiest way to run a simulation is by using the *runFromFile* function. This function starts the simulation and periodically reports its status at regular intervals. It accepts four optional parameters for configuration:

- **configFile**: A string specifying the path to the configuration file. Defaults to *config.in*.
- **statusTime**: Time interval, in seconds, for reporting the simulation status.
- **configLog**: A string specifying the path to the configuration log file. Defaults to *config.log*.
- **simulationLog**: A string specifying the path to the simulation log file. Defaults to *simulation.log*.

For example, if the configuration file *config.in* is located in the current directory, the following Python script can be used to configure and start the simulation:

```
1 import pyPenred
2 pyPenred.runFromFile("config.in")
```

Code 84: How to run a simulation from pyPenred and a penRed configuration file.

In addition to the usual format used by penRed for configurations, this function accepts also *YAML* files. These *YAML* files must follow the structure shown at the Section 6.1.

The previous example is quite simplistic, as it does not offer much control to the user. However, the definition of the *runFromFile* function (Code 85) can serve as a foundation for building more complex workflows.

```
1 def runFromFile(configFile = "config.in",
2                 statusTime = 120,
3                 configLog = "config.log",
4                 simulationLog = "simulation.log"):
5
6     #Set logs
7     simulation.setConfigurationLog(str(configLog))
8     simulation.setSimulationLog(str(simulationLog))
9
10    #Create simulation object
11    simu = simulation.create()
12
13    #Try to get the configuration from yaml or penred internal format
14    try:
15        f = open(configFile)
16        d = yaml.load(f, Loader=yaml.SafeLoader)
17        confSetErr = simu.configure(d)
18    except:
19        confSetErr = simu.configFromFile(configFile)
20
21    if(confSetErr != 0):
22        print("Invalid configuration file format. See config log\n")
23        return 1
24
25    print("Configuration set\n")
26
27    #Start the simulation asynchronously
28    err = simu.simulate(True)
29    if(err != 0):
30        print("Error on simulation configuration. See config.log\n")
31        return 2
32
33    #Simulation started, check status every 30 seconds
34    print("Simulation started\n")
35    while simu.isSimulating():
36        try:
```



```

37         time.sleep(statusTime)
38     except:
39         time.sleep(120)
40     status = simu.stringifyStatus()
41     for e in status:
42         print(e)
43
44     print("Simulation finished")

```

Code 85: Function *runFromFile*.

First of all, the function sets the log files for both the configuration and simulation steps:

```

1  #Set logs
2  simulation.setConfigurationLog(str(configLog))
3  simulation.setSimulationLog(str(simulationLog))

```

Code 86: Function *runFromFile* setting logs.

Next, a simulation object is created, which handles the simulation configuration and execution.

```

1  #Create simulation object
2  simu = simulation.create()

```

Code 87: Function *runFromFile* creating simulation object.

To read the configuration from the specified file, the function first attempts to parse it as a YAML format, using the resulting dictionary to configure the simulation object via the method *configure*. If the parsing fails, it then tries to parse the same file using the penRed internal format using the method *configFromFile*. Note that while the parsed configuration is stored in the simulation object, the configuration step has not yet begun. This means that sources, tallies, geometries, and other simulation components are not created at this stage. Consequently, any component-specific configuration errors will not be detected at this point.

```

1  #Try to get the configuration from yaml or penred internal format
2  try:
3      f = open(configFile)
4      d = yaml.load(f, Loader=yaml.SafeLoader)
5      confSetErr = simu.configure(d)
6  except:
7      confSetErr = simu.configFromFile(configFile)

```

Code 88: Function *runFromFile* parsing and storing configuration.

Once the configuration is parsed and saved in the simulation object, the simulation process can be initiated. The *pyPenred* wrapper allows for running the simulation in both blocking and asynchronous modes through the *simulate* method of the *simulation* object. This method takes one optional boolean parameter to enable (true) or disable (false) the asynchronous simulation mode. In Code 90, the simulation is executed in asynchronous mode to retrieve its status periodically.

When the *simulate* method is invoked, the stored configuration is utilized to construct all the simulation elements. As a result, this is the point at which potential configuration errors may arise, and they will be logged in the configuration log file.

```

1  #Start the simulation asynchronously
2  err = simu.simulate(True)
3  if(err != 0):
4      print("Error on simulation configuration. See config.log\n")
5      return 2

```

Code 89: Function *runFromFile* starting simulation.

Finally, the simulation status is checked and reported periodically. The method *isSimulating* returns *True* if a simulation is currently running, or *False* otherwise. It is important to note that the state of the simulation object cannot be changed during a simulation, including the stored configuration. The other used method, *stringifyStatus*, returns a string representation of the current simulation status.

```

1  #Simulation started , check status every 30 seconds
2  print("Simulation started\n")
3  while simu.isSimulating():
4      try:
5          time.sleep(statusTime)
6      except:
7          time.sleep(120)
8          status = simu.stringifyStatus()
9          for e in status:
10             print(e)
11
12  print("Simulation finished")

```

Code 90: Function *runFromFile* creating simulation object.

In the previous example, the entire configuration was set using a single dictionary or configuration file. However, it is also possible to override specific component configurations using the following methods from the *simulation* object:

- **configContext**: Sets the context configuration.
- **configSource**: Sets the source configuration.
- **configGeometry**: Sets the geometry configuration.
- **configTally**: Sets the tally configuration.
- **configVR**: Sets the variance reduction configuration.
- **setSimConfig**: Sets the simulation parameters.

Each of these methods accepts a single dictionary as an argument. Note that when configuring a specific component, the global dictionary key used for the entire simulation should be omitted. For instance, the dictionary passed to *configGeometry* should not include a top-level key like *geometry*. Instead, it should directly contain the component-specific settings.

## 6.3 API

Information about the package's functions and classes can be accessed using the *pydoc* utility. For example, to list the functions and methods of the *simulation* module, run:

```
python -m pydoc pyPenred.simulation
```

## 7 Examples

PenRed has been tested against the original PENELOPE FORTRAN code, on which it is based, using, among other methods, the provided PENELOPE examples. Some parts of the configuration files of these examples have been shown in previous sections to exemplify

the different samplers options, source, materials, geometry and tally definitions. At this section, all of the examples included in the distributed package are briefly described.

To execute the examples, geometry file, configuration file and material files are required. They must be at the same directory. Moreover, the executable file created after the program compilation is needed. As we have explained at section 2, the executable is compiled at *src/compiled/mains/pen\_main*. To run the simulation, the executable must be copied at the same folder where the example will be run with the other required files. After that, user can start the execution as follows,

```
./pen_main path/to/configuration/file
```

## 7.1 1-disc

This example presents a point source of electrons and a homogeneous disc phantom of *Cu*. The source is a monoenergetic beam with energy of  $40\text{KeV}$  and it is located at  $(x, y, z) = (0, 0, -0.0001)$  cm. The phantom size is a radius of 0.01 cm at plane *XY* and height of 0.005 cm at *z* axis with the base at  $z = 0$  cm.

### 7.1.1 1-disc-no-vr

This is the first version of the disc example, without variance reduction techniques. The required material files to execute this example is only *Cu.mat* file, and the configuration file is named *disc.in*. The geometry is read from the geometry file named *disc.geo*. The output of this example execution are the cylindrical and spatial dose distribution, the emerging particles distribution, the impact detector tallies for fluence and energy spectrum and the material energy deposition information.

### 7.1.2 1-disc-vr

The second version of the disc example includes variance reduction techniques. The material and geometry file are the same than the first version and, although the configuration file has the same name, there are some differences between these two versions. This second one is an example of the interaction forcing, x-ray and bremsstrahlung splitting capabilities of the PenRed code. At the configuration file is specified which is the body that suffers the variance reduction techniques and the kind of particle and interaction to force, electrons in this case. The output of this example execution is the same than at the first version. Notice that, when variance reduction techniques are used, the number of histories of the simulation is reduced because of the splitting applied to them.

## 7.2 2-plane

This example is a semi-infinite water plane with a spherical detector inside. The source is a photon monoenergetic beam of 30 MeV. The detector located in the water plane is defined as an impact detector for fluence measurements. No VR is applied in the plane example. The single material required to execute this example is the *Water.mat* file. The geometry information is read from the *plane.geo* file. The output of this example are the emerging particles distribution, cylindrical and spatial dose distribution, the impact detector tallies corresponding to fluence and energy spectrum measures and output files for material and body energy deposition.

### 7.3 3-detector

The geometry of this example consists of a NaI cylindrical detector with 5.08 cm of diameter and 5.08 cm height, with 1.27 cm Fe backing. A point-like Co-60 gamma-ray source emits a photon pencil beam in the  $-Z$  direction with equiprobable energies 1.17 and 1.33 MeV. The photons impinge on the NaI crystal from above. No VR is applied in this example. Materials required to execute this example are *NaI.mat* and *Fe.mat* files and the geometry is defined in the *detector.geo* file. In this case, the output files correspond to the material and energy deposition and to the emerging particles distribution.

### 7.4 4-x-ray-tube

The fourth example corresponds to a simple x-ray generator. It consists of a wolframium anode, a filter of aluminium and a silicon detector. The source emits a monoenergetic electron beam of 150 KeV directed to the anode to produce bremsstrahlung photon beam. For this purpose VR techniques are applied to produce splitting in this body. The filter is added to increase the average energy of the resulting beam, and the detector is defined to measure fluence, energy spectrum, and particle age. Materials required for this example are *W.mat*, *Al.mat*, *Si.mat*. The geometry information is in the *tube.geo* file. The output files are the emerging particles distribution, body and material energy deposition and detector measurements of spectrum, fluence and particle age.

### 7.5 5-accelerator

The accelerator example simulates a simple electron accelerator and calculates the dose distribution in a water phantom in two steps described following.

#### 7.5.1 5-accelerator-1

In the first step, a phase space file (PSF) is generated at a plane beyond the bottom of the accelerator head, using a planar impact detector. The geometry description of the *accel.geo* file consists of a tungsten target in which impinge an electron beam of 6 MeV, a collimator of the same material and a water phantom. The whole geometry is defined inside an air enclosure. Materials needed to run this simulation are *W.mat*, *H2O.mat* and *Air.mat*. The output files are the body and material energy deposition, the emerging particles distribution and the impact detector spectrum. Moreover, a *psf-merged.dat* file is generated with the PSF information.

#### 7.5.2 5-accelerator-2

In the second step, the initial particle states from that PSF is read and the dose distribution in the water phantom is obtained. The same materials and geometry file are used in this second step. In this case the output files are the emerging particle distribution, body and material energy deposition and the spatial dose distribution in the water phantom.

#### 7.5.3 5-accelerator-3

This example has been created to test the phase space file translation and rotation capabilities. Two variants of this example are created, 5-accelerator-orig and 5-accelerator-rot\_trans. The geometry used, *accel.geo*, consists of the same geometry of the previous example 5-accelerator-2 but with some modifications. In this case the water phantom is composed by different water slices with 2 cm of thickness instead of a solid water cube.

In the first variation of this example `5-accelerator-orig`, the phase space file and the geometry of the problem have not been rotated nor translated. An `EDEP_BODY` tally type has been set to obtain the deposited energy in each body of the geometry, including the different water slices. Moreover, three impact detector have been created with the tally type `IMPACT_DET` to obtain fluence and spectrum.

In the second variation, `5-accelerator-rot_trans`, the same geometry is used, but a rotation and translation is applied.

- Rotation:
  - $\omega = 20$  degrees
  - $\theta = 45$  degrees
  - $\phi = 20$  degrees
- Translation:
  - y shift = 10 cm

The same rotation and translation have been applied to the phase space file in the `config.in` file. The same `EDEP_BODY` and `IMPACT_DET` tallies used in the first variation have been set

## 7.6 6-polarisation

This example reproduces the Namito et al.'s (1993) scattering experiment with polarised photons. The source of this example is a monoenergetic polarised photon beam of 40 KeV, defined using the stokes parameters. Interaction forcing is used in this example. The geometry file corresponds to *gpol.geo* and the material files are *Cu.mat* and *Vacuum.mat*. The output files of the execution of this example are the emerging particle distribution, the material energy deposition and the energy spectrum.

## 7.7 7-aba

This example consist of a  $^{60}\text{Co}$  gamma rays source on a three layer cylinder phantom of water-Al-water. The source has equiprobable energies of 1.17 and 1.33 MeV. No VR is applied in this example. The geometry description of the cylinders is defined in the *3discs.geo*, and the material files needed are *water.mat* and *Al.mat*. The output files of this example corresponds to the emerging particles distribution, the impact detector energy deposition, the spatial dose distribution and the material and body energy deposition.

## 7.8 8-fake-chamber

This example describes an ionization chamber geometry inside a water phantom of  $30 \times 30 \times 30 \text{ cm}^3$ . The source is a monoenergetic photon beam of 4 MeV. The source-surface distance between the source and the surface of the water phantom is 90 cm. The ionization chamber is located 10 cm from the water surface. The geometry file of this example is the *chamber.geo* and the materials used for its simulation are *air.mat*, *pmma.mat*, *water.mat*. The same files are used for both versions of this example, without and with VR respectively.

### 7.8.1 8-fake-chamber-novr

In the first version of this example no VR is applied. The output files are the spatial dose distribution and the material energy deposition.

### 7.8.2 8-fake-chamber-vr

In the second version of this example VR is applied. Splitting and russian roulette are used. The output files are the same that in the first version, the spatial dose distribution and the material energy deposition.

## 8 Utilities

This section describes the utilities provided with the PenRed package that can be useful for the user. The code of each utility described in this section can be found in the corresponding folder in the directory

```
src/utilities/
```

### 8.1 Blender Geometry Plugin

This utility consists of a Python plugin intended for use within the Blender<sup>6</sup> software. The plugin leverages the Blender UI and tools for 3D design, aiming to simplify the construction of both Quadric (see Section 5.3.2) and Triangular Mesh Surface (see Section 5.3.3) based geometries. The plugin can be found in the folder,

```
src/utilities/Blender
```

and, actually, is tested for the Blender version LTS 3.6.9 <sup>7</sup>.

When constructing a geometry in Blender for export to PenRed, it is important to note that the default Blender units are already considered as centimeters. Therefore, no further transformations are necessary. The specific considerations for each geometry type will be discussed in the following sections. However, this manual does not contain an introduction to Blender usage, as it is already well-documented, and users can find many tutorials on the internet.

Therefore, to use this plugin effectively, we assume that the user has a basic understanding of how to use Blender. The key features to be familiar with include navigating in the 3D viewport, creating objects and adjusting their properties, establishing relationships between objects, editing meshes, and installing addons.

**Notice that the plugin utilizes Blender's parentage mechanics to determine which objects are included within others. Therefore, the user must construct the object tree according to the geometry requirements.**

Regardless the geometry type to be constructed, the penRed exporter can be found in the *File* menu, as is shown in the Figure 16.

On the save window, we can found a switch menu to select the geometry format to export, and a integer input box to specify the number of decimals to round the exported vaules (Figure 17).

---

<sup>6</sup><https://www.blender.org/>

<sup>7</sup><https://www.blender.org/download/lts/3-6/>

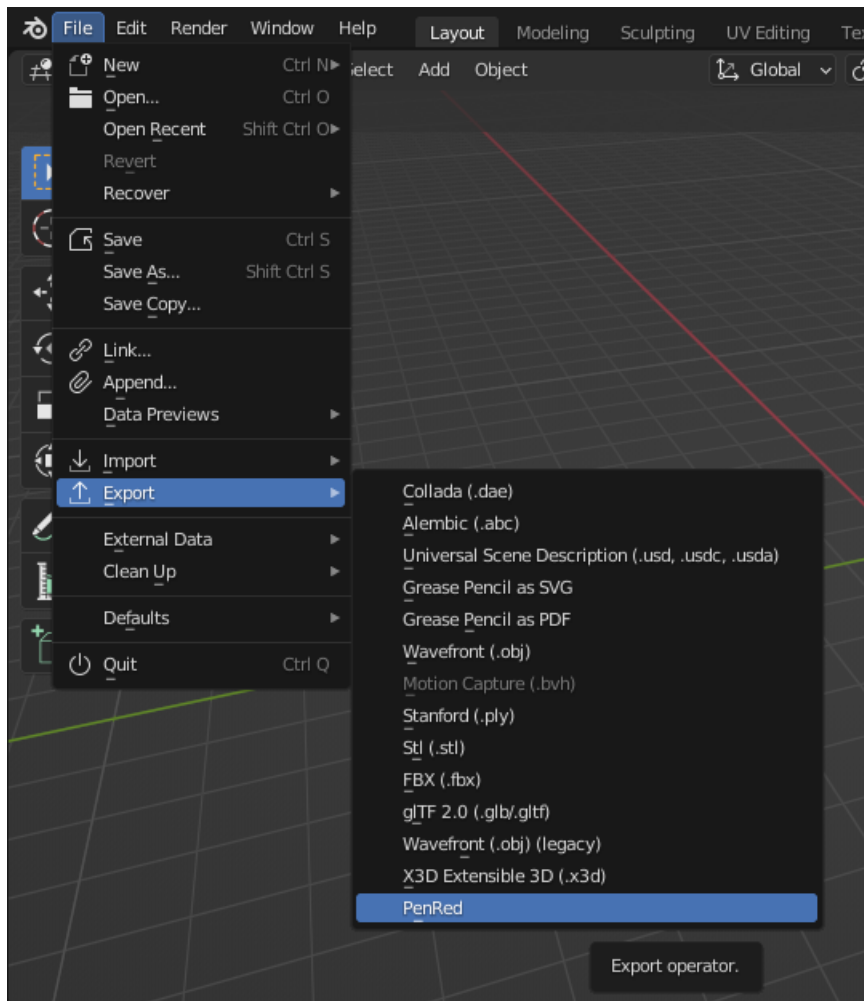


Figure 16: PenRed geometry export in Blender.

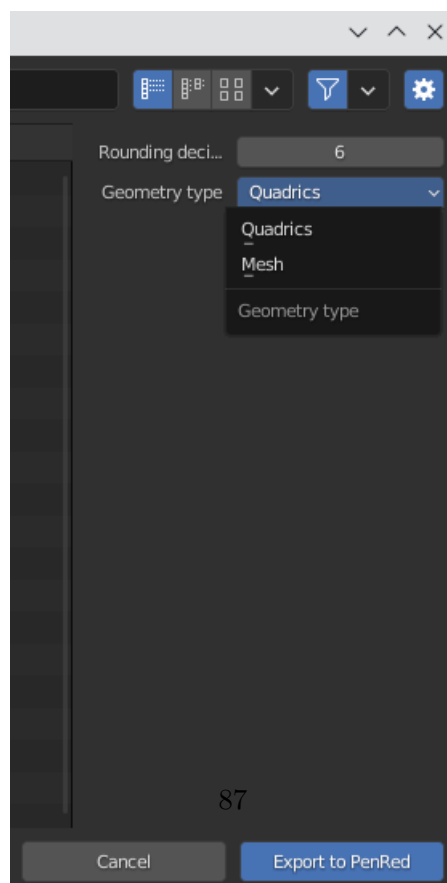


Figure 17: PenRed geometry export window in Blender.

Once all parameters and the filename has been specified, the user can export the geometry with the export button.

### 8.1.1 Quadric Geometries

This plugin is **not** capable of transforming a generic mesh surface into quadric geometries. Instead, it provides a set of predefined objects for constructing the geometry. These objects are listed in a submenu named *Quadric* in the *Add* menu, as shown in Figure 18.

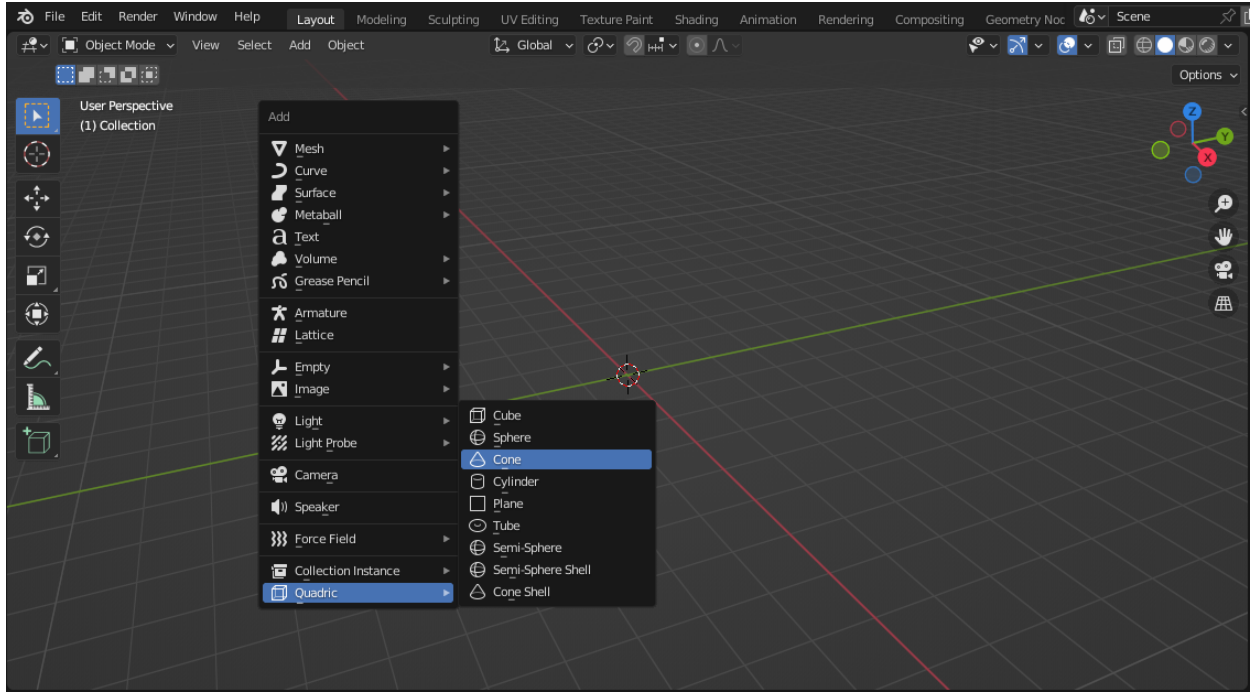


Figure 18: Predefined objects for quadric geometries

Notice that only the objects listed in the *Quadric* submenu can be exported as Quadric surfaces. Other types of objects will be ignored by the quadrics exporter.

All predefined objects have common parameters that need to be specified. These parameters can be defined in the *Quadric properties* submenu, which is located within the *Object properties* menu (Figure 19). The parameters are as follows:

- **Material:** Specify the material number to be assigned to the object.
- **Module:** If enabled (disabled), this object will be defined as a module (body).

The only objects that require additional parameters are the *CONE* and the *CONE\_SHELL*. For both, the top radius ( $r_2$ ) and bottom radius ( $r_1$ ) must be specified. Once these parameters are introduced, the user must apply the changes using the *remesh* button or revert to the original size with the *reset* button (see Figure 20).



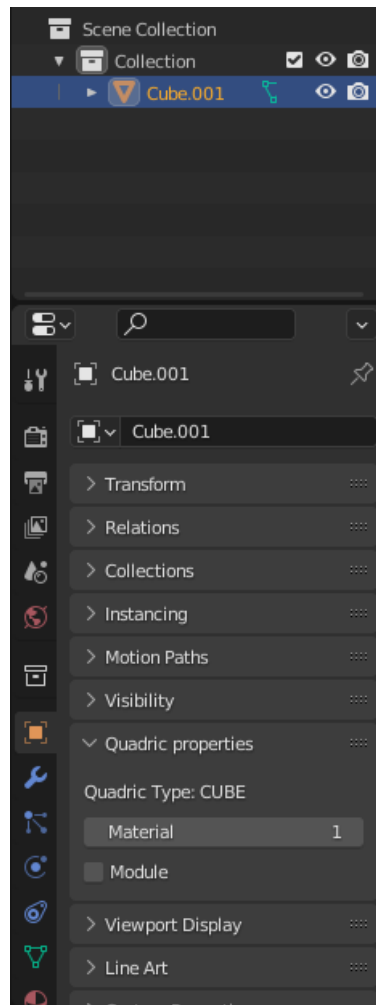


Figure 19: Common quadric parameters.

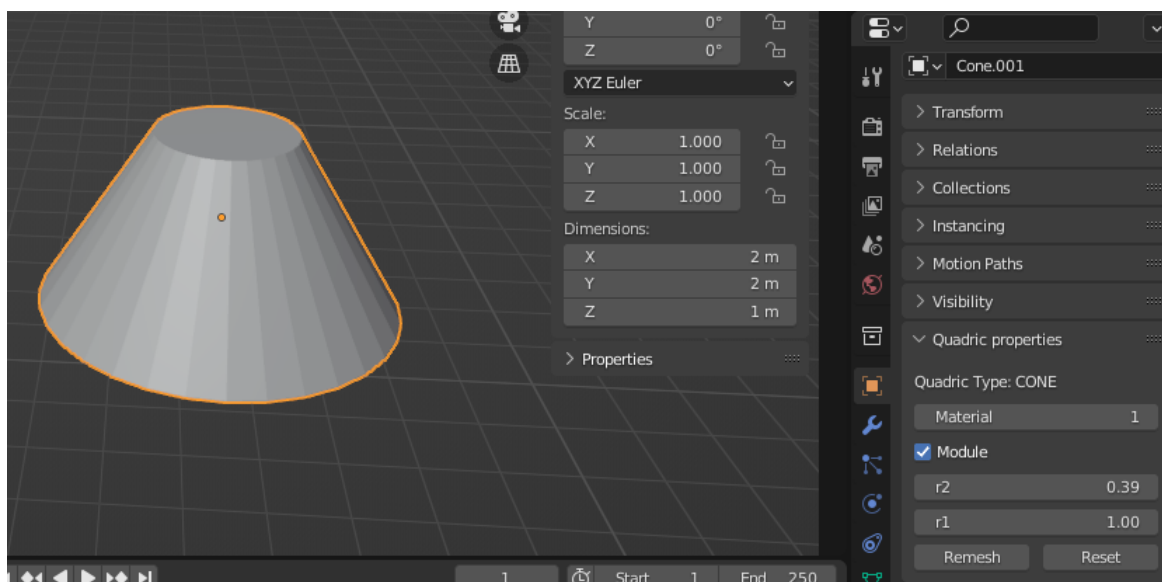


Figure 20: Cone parameters.

Note that, upon export, the body or module is positioned according to the object origin. Therefore, the user should avoid changing the relative position of the origin with respect to the object mesh. However, all objects can still be scaled, moved, and rotated according to the user's needs.

### 8.1.2 Triangular Mesh Surfaces

Unlike the quadric geometry exporter (Section 8.1.1), the exporter for triangular mesh surfaces can convert any mesh constructed with Blender to the PenRed format. Therefore, there is no need to use the predefined quadric objects. Additionally, Blender can be used to import meshes constructed with other programs and export them to the PenRed format.

The only parameter required for each body is its material, which can be found in the *Quadric properties* submenu within the *Object properties* menu (see Figure 19). Note that the module option is ignored for this type of geometry.

Furthermore, if vertex groups are defined for any object, the plugin will also export them according to the format described in Section 5.3.3. Subsequently, these vertex groups can be transformed during the simulation configuration at runtime.

## 8.2 Mutren

The Mutren utility is based on the original `mutren.f` code of the PENELOPE package. This one, calculates the  $\mu_{en}$  coefficients of the specified material for the specified energies. It consists of a command line program which takes the following parameters as input,

```
./mutren material-file energy-spectrum-file tolerance sim-time
```

Each parameter is described following,

- **material-file:** Path to the input material file.
- **energy-spectrum-file:** Path to the energy spectrum file. The format of this file consists of rows where each one contains a single energy in eV.
- **tolerance:** Relative error to stop the simulation, usually set to 0.1%
- **sim-time:** Allowed time to simulate each provided energy in seconds.

The program will calculate the coefficients for each provided energy and store them in a file named `mutren.dat`. Notice that the simulation of each energy point will finish if the objective tolerance has been achieved or if the simulation time reaches the specified limit (**sim-time**).

## 8.3 iaeaPSF

PenRed writes and reads the PSF in its own binary format. However, the International Atomic Energy Agency (IAEA) provides a standard PSF format. To be able to run IAEA PSF with PenRed, in the utilities folder, user can found iaeaPSF folder which includes the IAEA package of routines needed to obtain *iaea2PenRed* and *penRed2iaea* conversion tools. These tools can be found at

```
src/compiled/iaeaPSF/
```

The compilation can be set ON or OFF in the ‘compile.sh’ file. On the one hand, the *iaea2PenRed* program allows to convert PSF in IAEA format to PSF in internal binary PenRed format. To do it, user needs both header and phase space file of the IAEA standard format, “.IAEAheader” and “.IAEAphsp” respectively. To execute this tool the command line needed is

`./iaea2PenRed input filename (without extension) and output file name`

On the other hand, the *penRed2iaea* program converts a PenRed PSF in to IAEA PSF. In this case the comand line to execute it is

`./penRed2iaea input filename and output file name (without extension)`

providing two new files with extensions “.IAEAheader” and “.IAEAphsp”, respectively.

## 8.4 geo2voxel

This utility converts any geometry type in a voxelized one. As program arguments, the *geo2voxel* requires a configuration file which format is the same as *pen\_main*, i.e. the format described in the section 3. This configuration file, must include all the configuration parameters to load a geometry of any type, following the descriptions provided in the section 5.3, and the parameters listed below,

- **voxelized/nx**: Number of voxels in the  $X$  axis.
- **voxelized/ny**: Number of voxels in the  $Y$  axis.
- **voxelized/nz**: Number of voxels in the  $Z$  axis.
- **voxelized/dx**: Voxel size in the  $X$  axis, in cm.
- **voxelized/dy**: Voxel size in the  $Y$  axis, in cm.
- **voxelized/dz**: Voxel size in the  $Z$  axis, in cm.
- **voxelized/ox**: Voxelized geometry origin in the  $X$  axis, in cm.
- **voxelized/oy**: Voxelized geometry origin in the  $Y$  axis, in cm.
- **voxelized/oz**: Voxelized geometry origin in the  $Z$  axis, in cm.
- **voxelized/granularity**: Granularity used to determine the material and density of each voxel.
- **materials/material-name/ID**: Identification number of the material. There the *material-name* must be specified by the user and it is used only to distinguish different materials in the configuration file, i.e. has no influence in the converted geometry. This parameter must be repeated for different materials.
- **materials/material-name/density**: Density value of each material of the geometry, in  $\text{g}/\text{cm}^3$ . This parameter must be repeated for different materials. This density is assigned to the material number with the same *material-name*.

where *granularity* and the number of bins are expected to be integers, and the remaining parameters doubles. The origin of the voxelized geometry ( $ox, oy, oz$ ) is interpreted as the position of the left bottom corner of the lower  $Z$  voxelized geometry plane i.e., is not considered as the center of the first voxel of the mesh. Then, the granularity specify how many points per axis are generated inside each voxel to determine the material and density of each one. The density is obtained from the mean value of all generated points, while the material assignation will be done according to the material index with more points inside the voxel.

An example of configuration to generate a voxelized geometry from a quadric geometry file is shown in the code 91, where  $4^3$  points are generated in each voxel to determine the material and density and the center of the voxelized mesh has been set to the  $(0,0,0)$  of the quadric geometry.

```

1 geometry/type "PEN.QUADRIC"
2 geometry/input-file "disc.geo"
3 geometry/processed-geo-file "report.geo"
4
5 voxelized/nx 300
6 voxelized/ny 300
7 voxelized/nz 120
8
9 voxelized/dx 0.1
10 voxelized/dy 0.1
11 voxelized/dz 0.2
12
13 voxelized/ox -15.0
14 voxelized/oy -15.0
15 voxelized/oz -12.0
16
17 voxelized/granularity 4
18
19 materials/water/ID 1
20 materials/water/density 0.99821

```

Code 91: Complete `geo2voxel` configuration example to convert a quadric geometry.

## 8.5 geo2mesh

The 'geo2mesh' utility converts any geometry file compatible with PenRed into a simplified Wavefront file that can be used with the PenRed-GUI program. To accomplish this, the program creates a regular voxelized mesh with material information, allowing for the creation of the final triangular surface boundary. To use this utility, the user needs an input configuration file with the following parameters:

- **geometry:** This section contains information to configure the geometry, and its contents vary depending on the geometry type. It is structured as the used for a simulation.
- **nx:** The number of voxels used in the X-axis.
- **ny:** The number of voxels used in the Y-axis.
- **nz:** The number of voxels used in the Z-axis.
- **dx:** The voxel size in the X-direction, measured in centimeters.
- **dy:** The voxel size in the Y-direction, measured in centimeters.

- **dz:** The voxel size in the Z-direction, measured in centimeters.
- **ox:** The X-coordinate of the mesh origin, measured in centimeters.
- **oy:** The Y-coordinate of the mesh origin, measured in centimeters.
- **oz:** The Z-coordinate of the mesh origin, measured in centimeters.
- **granularity:** The number of points created within each voxel along each axis to assign the material number. It is advisable to use lower values to minimize computational costs. Typically, a value of 2 or 3 is sufficient.
- **smooth/steps:** The number of smoothing steps to apply to the mesh. This parameter is optional and is set to 50 by default.
- **smooth/factor:** The strength of the smoothing filter. This is also optional and is set to 0.2 by default.

Notice that the mesh origin is located at the first corner in the direction of increasing coordinates of the first mesh voxel. To use the configuration file, run the program as follows:

```
./geo2mesh config-file
```

the program will generate two files: one containing vertex information named 'geo.obj' and another with material information named 'geoMats.mtl'. The code 92 shows an example of configuration file to convert the 'tube.geo' geometry of the quadric example to a mesh.

```

1
2 #####
3 #  GEOMETRY PARAMETERS  #
4 #####
5
6 geometry/type "PEN_QUADRIC"
7 geometry/input-file "tube.geo"
8 geometry/processed-geo-file "report.geo"
9
10 #####
11 #  GEO2MESH PARAMETERS  #
12 #####
13
14 ox -10
15 oy -10
16 oz -10
17
18 dx 0.1
19 dy 0.1
20 dz 0.1
21
22 nx 200
23 ny 200
24 nz 200
25
26 granularity 3
27
28 smooth/steps 20
29 smooth/factor 0.2

```

Code 92: Complete `geo2mesh` configuration example to convert a quadric geometry.

## 8.6 range

The **range** utility prints the electron, positron and photon ranges for the specified material and energies. The corresponding code is located in the folder **penMats**. The program must be executed via the command line as follows,

```
./range material-file E1 E2 E3
```

where *range* is the executable name, *material-file* is the material file with the material information, the same as the used for simulations, and the following arguments ( $E1, E2, E3$ ) are a list of energies to calculate the corresponding ranges. Notice that in the execution example we have used 3 energies. However, any number of energies can be used.

The interpretation of the returned range depends on the particle type:

- **gamma:** The range is defined as the particle mean free path (inverse attenuation coefficient), in cm.
- **electrons and positrons:** The range is interpreted as the CSDA range, in cm.

## 8.7 PSF spectrum

The **psf\_spectre** utility extracts the energy spectrum distribution of the particles registered in a PSF file, and the corresponding source code can be found in the **psf/psf\_spectre.cpp** file. This utility is used via the command line as follows,

```
./psf_spectre PSFfilename emin emax nbins
```

where *psf\_spectre* is the utility executable, *PSFfilename* the path to the PSF file to be processed, *emin* the minimum energy, in eV, of the extracted spectrum, *emax* the corresponding spectrum maximum energy and *nbins* the number of spectrum bins.

## 8.8 PSF to ASCII

The **psf2ascii** utility converts a binary PSF file into a ASCII file format. The source code can be found in **psf/psf2ascii.cpp** and must be used via the command line as follows,

```
./psf2ascii inputFile outputFile
```

where *psf2ascii* is the executable, *inputFile* the binary PSF file path and *outputFile* the filename of the generated ASCII file.

## 8.9 Registers

The **registers** folder contains a set of sources which compiled programs show all the available types of a specific component. The shown types correspond to the ones used in the configuration files to specify the component *type* parameter. All the available registers are summarised in the Table 9. This ones are executed with no arguments via the command line.

Executable	Registered information
regGenericVR	Generic VR types
regPhotonVR	Specific photon VR types
regGeometries	Geometry types
regSamplers	Particle sampler types
regTallies	Tally types

Table 9: Available register programs with the corresponding registered information.

## 8.10 CompositionsDB

The *compositionsDB* utility provide access to the composition databases (DB). If no arguments are provided, the names of all available databases are listed (Code 93):

```

1 ./compositionsDB
2 usage: ./compositionsDB [composition-DB] [material-name]
3 Registered data bases:
4 + ICRP_AF
5 + ICRP_AM
```

Code 93: Composition utility DB listing.

For now, the two available databases are the ICRP compositions for the adult female and male phantoms, *ICRP\_AF* and *ICRP\_AM*, respectively [16]. If a database is specified, all materials belonging to it are listed. (Code 94):

```

1 ./compositionsDB ICRP_AF
2 usage: ./compositionsDB [composition-DB] [material-name]
3 Registered compositions in database 'ICRP_AF':
4 + Adrenal_left
5 + Adrenal_right
6 + ET1(0-8)
7 + ET1(8-40)
8 + ET1(40-50)
9 + ET1(50-surface)
10 + ET2(-15-0)
11 + ET2(0-40)
12 + ET2(40-50)
13 + ET2(50-55)
14 + ET2(55-65)
15 + ET2(65-surface)
16 + Oral_mucosa_tongue
17 + Oral_mucosa_mouth_floor
18 + Oral_mucosa_lips_and_cheeks
19 + Trachea
20 + BB(-11--6)
21 + BB(-6-0)
22 + BB(0-10)
23 .
24 .
25 .
```

Code 94: Composition utility: Listing ICRP\_AF materials.

Finally, if both the database and material name are provided, the density and composition of the specified material are shown. The composition is specified by pairs (atomic number, fraction by weight), as is shown in the Code 95:

```

1 ./compositionsDB ICRP_AF Humeri_upper_cortical
2 usage: ./compositionsDB [composition-DB] [material-name]
3
```

```

4 + Material 'Humeri_upper_cortical' ( 1.90400E+00 g/cm**3):
5   - 1 0.036000
6   - 6 0.159000
7   - 7 0.042000
8   - 8 0.448000
9   - 11 0.003000
10  - 12 0.002000
11  - 15 0.094000
12  - 16 0.003000
13  - 20 0.213000

```

Code 95: Composition utility DB: Listing material composition.

The user can specify multiple material names in a single execution, obtaining the corresponding composition for each one.

### 8.11 internalData2YAML

This utility converts a file with the penRed internal data format to a YAML format file. In the process, each slash is converted to a new level in the YAML structure.

```

1 usage: ./internalData2YAML internalData-filename outputYAML

```

Code 96: internalData2YAML usage.



## A Predefined materials

ID	Name	ID	Name	ID	Name
1	Hydrogen	34	Selenium	67	Holmium
2	Helium	35	Bromine	68	Erbium
3	Lithium	36	Krypton	69	Thulium
4	Beryllium	37	Rubidium	70	Ytterbium
5	Boron	38	Strontium	71	Lutetium
6	Amorphous carbon	39	Yttrium	72	Hafnium
7	Nitrogen	40	Zirconium	73	Tantalum
8	Oxygen	41	Niobium	74	Tungsten
9	Fluorine	42	Molybdenum	75	Rhenium
10	Neon	43	Technetium	76	Osmium
11	Sodium	44	Ruthenium	77	Iridium
12	Magnesium	45	Rhodium	78	Platinum
13	Aluminium	46	Palladium	79	Gold
14	Silicon	47	Silver	80	Mercury
15	Phosphorus	48	Cadmium	81	Thallium
16	Sulfur	49	Indium	82	Lead
17	Chlorine	50	Tin	83	Bismuth
18	Argon	51	Antimony	84	Polonium
19	Potassium	52	Tellurium	85	Astatine
20	Calcium	53	Iodine	86	Radon
21	Scandium	54	Xenon	87	Francium
22	Titanium	55	Cesium	88	Radium
23	Vanadium	56	Barium	89	Actinium
24	Chromium	57	Lanthanum	90	Thorium
25	Manganese	58	Cerium	91	Protactinium
26	Iron	59	Praseodymium	92	Uranium
27	Cobalt	60	Neodymium	93	Neptunium
28	Nickel	61	Promethium	94	Plutonium
29	Copper	62	Samarium	95	Americium
30	Zinc	63	Europium	96	Curium
31	Gallium	64	Gadolinium	97	Berkelium
32	Germanium	65	Terbium	98	Californium
33	Arsenic	66	Dysprosium	99	Einsteinium

Table 10: Elements predefined materials extracted from [3]. Material ID corresponds to the element atomic number.

<b>ID</b>	<b>Name</b>
100	Acetone
101	Acetylene
102	Adenine
103	Adipose tissue (ICRP)
104	Air, dry (near sea level)
105	Alanine
106	Aluminum oxide
107	Amber
108	Ammonia
109	Aniline
110	Anthracene
111	B-100 bone-equivalent plastic
112	Bakelite
113	Barium fluoride
114	Barium sulfate
115	Benzene
116	Beryllium oxide
117	Bismuth germanium oxide
118	Blood (ICRP)
119	Bone, compact (ICRU)
120	Bone, cortical (ICRP)
121	Boron carbide
122	Boron oxide
123	Brain (ICRP)
124	Butane
125	N-butyl alcohol
126	C-552 air-equivalent plastic
127	Cadmium telluride
128	Cadmium tungstate
129	Calcium carbonate
130	Calcium fluoride
131	Calcium oxide
132	Calcium sulfate
133	Calcium tungstate
134	Carbon dioxide
135	Carbon tetrachloride
136	Cellulose acetate, cellophane
137	Cellulose acetate butyrate
138	Cellulose nitrate
139	Ceric sulfate dosimeter solution
140	Cesium fluoride
141	Cesium iodide
142	Chlorobenzene
143	Chloroform
144	Concrete, portland
145	Cyclohexane
146	1,2-dichlorobenzene
147	Dichlorodiethyl ether
148	1,2-dichloroethane

149	Diethyl ether
150	N,n-dimethyl formamide
151	Dimethyl sulfoxide
152	Ethane
153	Ethyl alcohol
154	Ethyl cellulose
155	Ethylene
156	Eye lens (ICRP)
157	Ferric oxide
158	Ferroboration
159	Ferrous oxide
160	Ferrous sulfate dosimeter solution
161	Freon-12
162	Freon-12b2
163	Freon-13
164	Freon-13b1
165	Freon-13i1
166	Gadolinium oxysulfide
167	Gallium arsenide
168	Gel in photographic emulsion
169	Pyrex glass
170	Glass, lead
171	Glass, plate
172	Glucose
173	Glutamine
174	Glycerol
175	Graphite
176	Guanine
177	Gypsum, plaster of Paris
178	N-heptane
179	N-hexane
180	Kapton polyimide film
181	Lanthanum oxybromide
182	Lanthanum oxysulfide
183	Lead oxide
184	Lithium amide
185	Lithium carbonate
186	Lithium fluoride
187	Lithium hydride
188	Lithium iodide
189	Lithium oxide
190	Lithium tetraborate
191	Lung (ICRP)
192	M3 wax
193	Magnesium carbonate
194	Magnesium fluoride
195	Magnesium oxide
196	Magnesium tetraborate
197	Mercuric iodide
198	Methane

199	Methanol
200	Mixed wax
201	Ms20 tissue substitute
202	Muscle, skeletal (ICRP)
203	Muscle, striated (ICRU)
204	Muscle-equivalent liquid, with sucrose
205	Muscle-equivalent liquid, without sucrose
206	Naphthalene
207	Nitrobenzene
208	Nitrous oxide
209	Nylon, du Pont elvamide 8062
210	Nylon, type 6 and type 6/6
211	Nylon, type 6/10
212	Nylon, type 11 (rilsan)
213	Octane, liquid
214	Paraffin wax
215	N-pentane
216	Photographic emulsion
217	Plastic scintillator (vinyltoluene based)
218	Plutonium dioxide
219	Polyacrylonitrile
220	Polycarbonate (makrolon, lexan)
221	Polychlorostyrene
222	Polyethylene
223	Polyethylene terephthalate (mylar)
224	Polymethyl methacrilate (lucite, perspex, plexiglass)
225	Polyoxymethylene
226	Polypropylene
227	Polystyrene
228	Polytetrafluoroethylene (teflon)
229	Polytrifluorochloroethylene
230	Polyvinyl acetate
231	Polyvinyl alcohol
232	Polyvinyl butyral
233	Polyvinyl chloride
234	Polyvinylidene chloride (saran)
235	Polyvinylidene fluoride
236	Polyvinyl pyrrolidone
237	Potassium iodide
238	Potassium oxide
239	Propane
240	Propane, liquid
241	N-propyl alcohol
242	Pyridine
243	Rubber, butyl
244	Rubber, natural
245	Rubber, neoprene
246	Silicon dioxide
247	Silver bromide
248	Silver chloride

249	Silver halides in photographic emulsion
250	Silver iodide
251	Skin (ICRP)
252	Sodium carbonate
253	Sodium iodide
254	Sodium monoxide
255	Sodium nitrate
256	Stilbene
257	Sucrose
258	Terphenyl
259	Testes (ICRP)
260	Tetrachloroethylene
261	Thallium chloride
262	Tissue, soft (ICRP)
263	Tissue, soft (ICRU four-component)
264	Tissue-equivalent gas (methane based)
265	Tissue-equivalent gas (propane based)
266	Tissue-equivalent plastic (A-150)
267	Titanium dioxide
268	Toluene
269	Trichloroethylene
270	Triethyl phosphate
271	Tungsten hexafluoride
272	Uranium dicarbide
273	Uranium monocarbide
274	Uranium oxide
275	Urea
276	Valine
277	Viton fluoroelastomer
278	Water, liquid
279	Water vapour
280	Xylene

---

Table 11: Predefined compounds extracted from [3].

## References

- [1] V. Giménez-Alventosa, V. Giménez Gómez, S. Oliver, Penred: An extensible and parallel monte-carlo framework for radiation transport based on penelope, *Computer Physics Communications* 267 (2021) 108065. doi:<https://doi.org/10.1016/j.cpc.2021.108065>.  
URL <https://www.sciencedirect.com/science/article/pii/S0010465521001776>
- [2] Salvat F., Penelope. a code system for monte carlo simulation of electron and photon transport, Issy-Les-Moulineaux: OECD Nuclear Energy Agency (2014).
- [3] F. Salvat, PENELOPE-2018: A code System for Monte Carlo Simulation of Electron and Photon Transport, OECD/NEA Data Bank, Issy-les-Moulineaux, France, 2019, available from <http://www.nea.fr/lists/penelope.html>.
- [4] Dcmtk, <https://github.com/DCMTK/dcmtk>, accessed: 2019-09-28.

- [5] R. L. Graham, T. S. Woodall, J. M. Squyres, Open mpi: A flexible high performance mpi, in: R. Wyrzykowski, J. Dongarra, N. Meyer, J. Waśniewski (Eds.), *Parallel Processing and Applied Mathematics*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 228–239.
- [6] W. Gropp, Mpich2: A new start for mpi implementations, in: *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer-Verlag, Berlin, Heidelberg, 2002, p. 7.
- [7] J. Sempau, A. Badal and L. Brualla, A PENELOPE-based system for the automated Monte Carlo simulation of clinacs and voxelized geometries—application to far-from-axis fields, *Med. Phys.* 38 (2011) 5887–5895.  
URL <http://dx.doi.org/10.1118/1.3643029>
- [8] J. Sempau, Penelope/peneasy user manual, Tech. rep., Version 2019-01-01.
- [9] E. García-Toraño, V. Peyres, F. Salvat, Pennuc: Monte carlo simulation of the decay of radionuclides, *Computer Physics Communications* 245 (2019) 106849. doi:<https://doi.org/10.1016/j.cpc.2019.08.002>.  
URL <https://www.sciencedirect.com/science/article/pii/S0010465519302267>
- [10] V. Giacometti, S. Guatelli, M. Bazalova-Carter, A. Rosenfeld, R. Schulte, Development of a high resolution voxelised head phantom for medical physics applications, *Physica Medica* 33 (2017) 182 – 188. doi:<https://doi.org/10.1016/j.ejmp.2017.01.007>.  
URL <http://www.sciencedirect.com/science/article/pii/S1120179717300078>
- [11] J. F. Williamson, Monte carlo evaluation of kerma at a point for photon transport problems, *Medical Physics* 14 (4) (1987) 567–576. arXiv:<https://aapm.onlinelibrary.wiley.com/doi/pdf/10.1118/1.596069>, doi:10.1118/1.596069.  
URL <https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.596069>
- [12] N. Ravinder et al, Dosimetry of interstitial brachytherapy sources: recommendations of the AAPM radiation therapy committee task group no 43, *Med. Phys.* 22 (1995) 209–34. doi:10.1118/1.597458.
- [13] A. Badal, J. Sempau, A package of linux scripts for the parallelization of monte carlo simulations, *Computer Physics Communications* 175 (6) (2006) 440 – 450. doi:<https://doi.org/10.1016/j.cpc.2006.05.009>.  
URL <http://www.sciencedirect.com/science/article/pii/S001046550600230X>
- [14] Mhd/raw image format, <https://itk.org/Wiki/ITK/MetaIO/Documentation>, accessed: 2022-07-15.
- [15] Gnuplot, <http://www.gnuplot.info/>, accessed: 2022-07-15.
- [16] M. Zankl, Adult male and female reference computational phantoms (icrp publication 110), *Japanese Journal of Health Physics* 45 (4) (2010) 357–369.