Geant4-based Architecture for Medicine-Oriented Simulations (GAMOS)

Version 2.0.1

User's Guide

Pedro Arce

August 28, 2009

Contents

Chapter 1

Introduction

1.1 About this document

This manual refers to GAMOS version 2.0.1. It is written in LaTeX and it is maintained at the following address

http://fismed.ciemat.es/GAMOS/doc/gamos_uguide_2.0.1.pdf

We will use through this manual many terms common to the detector simulation terminology and specifically to the Geant4[1] terminology. If you are new to it, please read before, for example, the Geant4 documentation [2]. We have tried though to make this manual self-consistent, and we hope that, unless you need a deep knowledge of the Geant4 software, you will not need to refer to any further documentation.

If you find that some of the instructions given here do not give the expected result, please send a mail to pedro.arce@ciemat.es detailing the problem, the GAMOS version and providing as much information as possible. We will also warmly welcome any kind of comment or suggestion you would like to send us about this guide, or about the GAMOS functionalities or user interface.

1.2 Structure of GAMOS

The acronym GAMOS stands for **"Geant4-based Architecture for Medicine-Oriented Simulations"**. It is therefore a detector simulation software and it is based on the Geant4 toolkit [1]. The objective of GAMOS is to provide a software framework that serves the unexperienced user to simulate his/her medical physics project without having to code in C++ and with a minimal knowledge of Geant4, and at the same time, let an advanced user add new functionalities and easily integrate it with the rest of GAMOS functionality.

We have also tried to provide you with several tools that help you to understand in detail your simulation (controlling the verbosity, making his-

tograms about many variables, scoring different quantities, etc.), as well as other tools to help you in optimising your simulation. GAMOS is composed of a core software that covers the main functionality of a Geant4 simulation and a set of example of medicine applications.

GAMOS source code is organized in the following subsystems, each one subdivided in the following packages:

- **GamosCore**: core software covering main Geant4 functionality
  - **GamosBase**: parameter manager, analysis manager, filters, classifiers and other basic functionalities
  - **GamosFactories**: factories to convert the different simulation components (geometry, physics, generator, user actions, ...) in plug-in's
  - **GamosGeometry**: geometry-related utilities and support for text detector description
  - **GamosMovement**: support for displacements and rotations of volumes during a job
  - **GamosSD**: classes for sensitive detectors, hits and digitization
  - **GamosGenerator**: utilities to support single particle and isotope generators as well as different initial particle distributions
  - **GamosPhysics**: example of common physics list for medicine applications
    - **GamosPhysicsList**: example of electromagnetic physics list, supporting standard, low-energy and Penelope classes, and example of hadronic physics list (meant for hadrontherapy)
    - **GamosOtherPhysicsLists**: other examples of electromagnetic and hadronic physics lists
    - **GamosCuts**: management of production cuts and user limits, including tools to automatically optimise them
    - **GamosVarianceReduction**: implementation of several bremsstrahlung splitting techniques
  - **GamosUserActionMgr**: user action manager to allow several user actions of the same type, selectable by user commands
  - **GamosScoring**: scoring manager and messenger and examples of scoring plug-in classes (scorers, filters and printers)
  - **GamosAnalysis**: utilities that can help the advanced user to analyse results
  - **GamosReadDICOM**: code to read in DICOM files containing patient data

  - **GamosUtilsUA**: user action utilities (tracking verbosity control, track counting, process counting, time study, ...)
  - **GamosUtils**: general C++ utilities
  - **GamosApplication**: GAMOS run manager and the "main" program

- **text_read, text_build**: packages to read geometry from text files (to be moved to Geant4 official release
- **PET**: example of PET simulation
  - **PETGeometry**: example to simulate the most common PET devices by defining its properties in an input text file
  - **PETAnalysis**: PET event classifier and PET histograms
- **RadioTherapy**: example of RadioTherapy simulation, including writing and reading phase space files

Each package has the following subdirectories:

- **src**: the source code
- **include**: the header files

You do not need to follow this file distribution if you want to create a new package, but we recommend you to do so.

There are also several directories containing examples of the different GAMOS functionalities as well of how to extend them. Please refer to the examples chapter in this guide or the README files in each example for a detailed explanation.

In the directory **tutorials** you can find three step-by-step tutorials: PET, Radiotherapy and plug-in's. They include several exercises with increasing difficulty, and the exercise outputs as well as the exercise solutions are provided. We recommend you to follow one or several of these tutorials to become acquainted with GAMOS.

1.3 The plug-in concept

To provide the user with a big flexibility in choosing different simulation components (geometry, physics, user actions, histograms, ...) and combining them to his/her will in a simple way, GAMOS is based on the plug-in concept. This means that the "main" program runs without predefined components and the user tells it which components are being loaded at run time (without needing to recompile) by simply listing them in a text input file. This mechanism also lets the user define a new component that was

not foreseen by GAMOS and easily tell GAMOS to use it together with any other of his/her own components or GAMOS components.

For each of the simulation component types we will describe in the corresponding section which is the command to select it and how to transform a new user component into a GAMOS plug-in.

For the implementation of plug-in's GAMOS has chosen the CERN library SEAL [3].

A common example to better understand the plug-in concept is the plug-in's that are installed on your computer when you open some Internet page with your web browser. Your web browser can use these plug-in's to get an extra functionality (viewing videos, animated figures, ...) without your having to recompile it and even if the web browser designers had never before heard of the new plug-in.

Chapter 2

Getting started

This chapter explains the practical details to obtain the GAMOS code, compile and run it.

2.1 Getting the code and installing it

GAMOS has been tested in several Linux distributions (Scientific Linux, Fedora Core and Ubuntu [1]), as well as Mac OS 8.10.

You can download GAMOS from
http://fismed.ciemat.es/GAMOS

GAMOS depends on Geant4 and on a number of other libraries (CLHEP [4] and, optionally, ROOT[5]). To download and install everything you can follow the instruction in the 'Code download' area. As explained there you need to get first the installation scripts and uncompress them in the **scripts** directory (you may do it automatically by downloading the scripts installation utility from the web page). After that you just need to type the command

sh installGamos.csh MY_INSTALLATION_DIR

where **MY_INSTALLATION_DIR** is the directory where you want to install GAMOS.

This command will download the GAMOS code as well as CLHEP, Geant4 and ROOT packages, and it will compile them all. It will first make sure that you have a C++ compiler. Then it will check if you have the X11 and the OpenGL libraries installed and if not it will install GAMOS without OGLIX visualisation. After that follow the instructions below on how to run an example.

[1] do not use the test version Ubuntu 8.10, as it has been publicly reported to have problems with plu-in's

2.2 Running an example

If you have done a standard installation, you will have your code compiled and ready to run.

Before running any example you have to set some configuration variables, mainly where you have installed GAMOS and the depending libraries. This is all done in the file

MY_GAMOS_DIR/config/confgamos.csh

or

MY_GAMOS_DIR/config/confgamos.sh

depending on your shell flavour.

Therefore before running you have to source this file:

source MY_GAMOS_DIR/config/confgamos.csh

or

source MY_GAMOS_DIR/config/confgamos.sh

Remember to type this command every time you start a new session to run GAMOS.

To run your application inside GAMOS you do not have to write a "main" program, as GAMOS provides a unique "main" that serves to run any application. When you run the GAMOS "main" it will load and call the components you want (geometry, physics, generator, hits building, histograms, ...) by simply defining them in the input command file. Therefore to run your application simply type

gamos MY_INPUT_FILE

where **MY_INPUT_FILE** is a typical Geant4 macro file that includes Geant4 and GAMOS commands.

The minimum set of commands that you need are those to select a geometry, a physics list and a generator, to initialize Geant4 and to run N events. In this case your input file may look like this one:

```
/tracking/verbose 1
/gamos/setParam GmGeometryFromText:FileName geom.txt
/gamos/geometry GmGeometryFromText
/gamos/physicsList GmEMPhysics
/gamos/generator GmGenerator
/gamos/generator/addSingleParticleSource MY_ELEC e- 1. MeV

/run/initialize

/run/beamOn 10
```

This will create the geometry of a simple geometry described in the file geom.txt lying in the current directory or in the MY_GAMOS_DIR/data

directory, set the physics as the standard electromagnetic Geant4 physics and run 10 events with an electron of 1 MeV as primary particle.

You can then add any of the command described in this document, or any Geant4 command or any command you created yourself.

Beware that Geant4 is a state machine, and the list of available commands depends on the current state. The main state change is triggered by the **/run/initialize** command, which changes the state from *G4State_PreInit* to *G4State_Idle*. You may get a full list of the available commands at any moment with the command **/control/manual**.

To run your first example you can use the one at the directory examples/test. Simply type the commands:

cd MY_GAMOS_DIR/examples/test; gamos test.in

2.3 Compiling GAMOS

If you installed GAMOS as explained in the previous section, the compilation will be done automatically. Then you may run your application in GAMOS by writing your user commands without any need of compiling ever more.

Only if you want to extend the GAMOS functionality by providing new code, you will have to follow the instructions in this section.

GAMOS uses the GNU make tool to manage the compilation and generation of executables. It uses a set of configuration files based on the Geant4 ones. Therefore, if you are familiar to Geant4, you will find no difficulty in compiling GAMOS.

After untarring the installation file, you will have a directory called **MY_INSTALLATION_DIR/GAMOS.**2.0.1. This is the directory where the GAMOS code is, the rest are the external libraries used by GAMOS and the configuration utilities.

Before compiling, you have to define a few variables, mainly the location of the different external packages. All this is done by sourcing the file

source MY_GAMOS_DIR/config/confgamos.csh

To compile any directory of GAMOS, and all the directories below, you just have to go to that directory and type **make**. This will compile the .cc files found in the **src** directory, build the library and the plug-in's, and in the case of the directory GamosCore/GamosApplication it will also create the **gamos** executable. You may need to type the Linux command **rehash** to refresh your environmental variables in case there was no executable file before starting the compilation.

2.3.1 Compiling your new code

If you have created a new directory with your code you have to compile it following the Geant4 way. The implementation files should have the suffix **.cc** and should be in a subdirectory called **src**. For the declaration files,

you have a greater freedom; the Geant4 way is that they have the suffix **.hh** and lie in a subdirectory called **include**, but you can do it your own way (and after that, you have to be consistent in the GNUmakefile, as explained below).

You then have to build a **GNUmakefile**, that will steer the compilation and library building when you type the command **make**. For building it you may follow the examples in the **GamosCore/GamosXXX** directories. We take as example the file **GamosCore/GamosGeometry/GNUmakefile**:

```
name := GamosGeometry
G4TARGET := $(name)
G4EXLIB := true

.PHONY: all
all: lib plugin

include $(GAMOSINSTALL)/config/binmake.gmk
include $(GAMOSINSTALL)/config/general.gmk

EXTRALIBS += -lG4geomtextread -lG4geomtextbuild -lGamosBase
  -lGamosUtils -lGamosFactories -lGamosUserActionMgr
```

Let's go one by one through the lines:
In the first one you define the name of your library:

```
name := GamosGeometry
```

The following two lines are used internally by the GAMOS scripts and are mandatory:

```
G4TARGET := $(name)
G4EXLIB := true
```

Then you define what you want to do when you type **make**:

```
.PHONY: all
all: lib plugin
```

There are several possibilities:

- **lib** Compile and build the library.
- **plugin** Build the plug-in. Use it if and only if you are creating a new plug-in.

- **plugin_check** Check that you are linking with all the libraries that will be needed. This check is not mandatory, but if you do not do it and you are missing some library, when you run you will get an error message.

- **bin** Build the executable. You will probably never need this, as GAMOS is based on dynamic loading and plug-in's.

The following two lines are needed for configuration

```
include $(GAMOSINSTALL)/config/binmake.gmk
include $(GAMOSINSTALL)/config/general.gmk
```

If you edit the file **general.gmk**, you will see that it is just including a different file for each package. Therefore, instead of using it, you may include all or only a subset of them if you do not need them all.

Finally, you define the GAMOS libraries that will be linked to yours, i.e. the libraries of each of the files that you have included in your code [2]

```
EXTRALIBS += -lG4geomtextread -lG4geomtextbuild -lGamosBase
  -lGamosUtils -lGamosFactories -lGamosUserActionMgr
```

If you have doubts about which GAMOS libraries to include, you may include them all, as in **GamosCore/GamosApplication/GNUmakefile**.

If you have built several levels of directories you may want to have the possibility of typing **make** at the top most directory to trigger the compilation of all the directories. To do this you just have to add a **GNUmakefile** at the top level directory similar to the one at **$(GAMOSINSTALL)/source/GamosCore/GNUmakefile**, that we reproduce here:

```
include $(GAMOSINSTALL)/config/architecture.gmk

SUBDIRS  = GamosUtils GamosFactories GamosBase GamosUserActionMgr
  GamosAnalysis GamosGeometry GamosMovement GamosPhysics
  GamosGenerator GamosSD GamosUtilsUA GamosReadDICOM
  GamosScoring GamosApplication

include $(GAMOSINSTALL)/config/globlib.gmk
```

You only have to change the line starting by **SUBDIRS =**, to list the name of your subdirectories.

[2] For the external packages the set of libraries is fixed and defined in the configuration files

Chapter 3

Geometry

You can describe your detector in three different ways: by defining your setup in a text file, by using one of the geometry examples provided or in the standard Geant4 way, by writing your C++ class inheriting from G4VUserDetectorConstruction.

3.1 Building your geometry with a text file

You can define your geometry in a simple text file as described in the following subsection.

You can also use as example the one at

MY_GAMOS_DIR/data/g4geom.*.txt

Once your file is ready, you have to tell GAMOS to use your geometry definition, first telling the name of your file with the command

/gamos/setParam GmGeometryFromText:FileName MY_FILENAME[1]

and then telling GAMOS to use the constructor of geometry from text file

/gamos/geometry GmGeometryFromText

3.1.1 Description of geometry text file format

The description of the geometry is based on tags. A tag is a word that appears at the first one in a line and sets what the line means.

There are no constraints on the order of the tags in the file, except some logical restrictions, e.g. a volume cannot be positioned or given attributes if it has not been defined (e.g. no ':PLACE', ':VIS', ':COLOUR', ':CHECK_OVERLAPS' tags before ':VOLU' tag).

We will explain in this section the tags used to describe the geometry, also explaining the meaning of each of the words that follow the tag, and an example of each tag. Tags can be given with any combination of upper case

[1]The default path to look for this file is defined by the variable GAMOS_SEARCH_PATH. For details see section on *"Managing the input data files"*

and lower case letters. Each tag has a fixed number of arguments, known by the parser; therefore you may write all arguments in a line or in several lines at your will.

Isotope

:ISOT

1. Name
2. Z
3. N
4. A

Example:

```
 :ISOT  Cl35      17    18     35.
```

Element made of one unique isotope

:ELEM

1. Name
2. Symbol
3. Z
4. A

Example:

```
:ELEM Hydrogen H  1.  1.
```

Element composed of several isotopes

:ELEM_FROM_ISOT

1. Name
2. Symbol
3. Number of components

One line per isotope with

1. isotope name

2. fraction of number of atoms per volume

Example:

```
:ELEM_FROM_ISOT  Chlorine   Cl    2
    Cl35  0.4
    Cl36  0.6
```

Material made of one element

:MATE

1. Name
2. Z
3. A
4. Density

Example:

```
:MATE  Iron      26.    55.85     7.87
```

Material made of a mixture of elements or materials

:MIXT

1. Name
2. Density
3. Number of components

One line per material or element with

1. material name
2. proportion of material in the mixture.

The components can be either all elements or all materials, but both types cannot appear in the same mixture.

There are three mixture tags, depending of the way the proportions are defined:

- Proportions by weight fractions

  :MIXT_BY_WEIGHT

  This tag is equivalent to the ":MIXT" tag

- Proportions by number of atoms

  :MIXT_BY_NATOMS

- Proportions by volume

  :MIXT_BY_VOLUME

The first two tags can be used to build material mixtures out of elements or materials, but the last tag can only be applied with material components (elements do not have density).

Example:

```
:MIXT Fiber_Lead      9.29    2
          Lead          0.9778
          Polystyrene   0.0222
```

Geant4 internal database of materials and elements

Geant4 provides a list of predefined materials, whose compositions correspond to the NIST definition [13]. Among them you can find all single elementary materials, from Z = 1 (Hydrogen) to Z = 98 (Californium). You can use those materials when building a volume without the need to redefine them on your ASCII file. It is just enough that the material name you assign to a volume corresponds to the name of one of these predefined materials (they all start with "G4_"). The Geant4 materials have the mean excitation energy set explicitly, instead of allowing an automatic calculation from its components. You may override those materials if you want by redefining them in your ASCII file.

Also Geant4 provides the definition of all elements from Z = 1 (Hydrogen) to Z = 107 (Bohrium). Their names are the usual symbol in the periodic table of elements (no "G4_"). These elements take into account the isotope composition.

Apart from the elementary materials, many other are available, usually related to medical physics applications. you may find the details of their composition in the Geant4 file
source/materials/src/G4NistMaterialBuilder.cc:

The full list is the following:
G4_A-150_TISSUE, G4_ACETONE,
G4_ACETYLENE, G4_ADENINE,
G4_ADIPOSE_TISSUE_ICRP, G4_AIR,
G4_ALANINE, G4_ALUMINUM_OXIDE,
G4_AMBER, G4_AMMONIA,
G4_ANILINE, G4_ANTHRACENE,
G4_B-100_BONE, G4_BAKELITE,
G4_BARIUM_FLUORIDE, G4_BARIUM_SULFATE,

G4_BENZENE, G4_BERYLLIUM_OXIDE,
G4_BGO, G4_BLOOD_ICRP,
G4_BONE_COMPACT_ICRU, G4_BONE_CORTICAL_ICRP,
G4_BORON_CARBIDE, G4_BORON_OXIDE,
G4_BRAIN_ICRP, G4_BUTANE,
G4_N-BUTYL_ALCOHOL, G4_C-552,
G4_CADMIUM_TELLURIDE, G4_CADMIUM_TUNGSTATE,
G4_CALCIUM_CARBONATE, G4_CALCIUM_FLUORIDE,
G4_CALCIUM_OXIDE, G4_CALCIUM_SULFATE,
G4_CALCIUM_TUNGSTATE, G4_CARBON_DIOXIDE,
G4_CARBON_TETRACHLORIDE, G4_CELLULOSE_CELLOPHANE,
G4_CELLULOSE_BUTYRATE, G4_CELLULOSE_NITRATE,
G4_CERIC_SULFATE, G4_CESIUM_FLUORIDE,
G4_CESIUM_IODIDE, G4_CHLOROBENZENE,
G4_CHLOROFORM, G4_CONCRETE,
G4_CYCLOHEXANE, G4_1,
2-DICHLOROBENZENE, G4_DICHLORODIETHYL_ETHER,
G4_1,2-DICHLOROETHANE,
G4_DIETHYL_ETHER, G4_N,
N-DIMETHYL_FORMAMIDE, G4_DIMETHYL_SULFOXIDE,
G4_ETHANE, G4_ETHYL_ALCOHOL,
G4_ETHYL_CELLULOSE, G4_ETHYLENE,
G4_EYE_LENS_ICRP, G4_FERRIC_OXIDE,
G4_FERROBORIDE, G4_FERROUS_OXIDE,
G4_FERROUS_SULFATE, G4_FREON-12,
G4_FREON-12B2, G4_FREON-13,
G4_FREON-13B1, G4_FREON-13I1,
G4_GADOLINIUM_OXYSULFIDE, G4_GALLIUM_ARSENIDE,
G4_GEL_PHOTO_EMULSION, G4_Pyrex_Glass,
G4_GLASS_LEAD, G4_GLASS_PLATE,
G4_GLUCOSE, G4_GLUTAMINE,
G4_GLYCEROL, G4_GUANINE,
G4_GYPSUM, G4_N-HEPTANE,
G4_N-HEXANE, G4_KAPTON,
G4_LANTHANUM_OXYBROMIDE, G4_LANTHANUM_OXYSULFIDE,
G4_LEAD_OXIDE, G4_LITHIUM_AMIDE,
G4_LITHIUM_CARBONATE, G4_LITHIUM_FLUORIDE,
G4_LITHIUM_HYDRIDE, G4_LITHIUM_IODIDE,
G4_LITHIUM_OXIDE, G4_LITHIUM_TETRABORATE,
G4_LUNG_ICRP, G4_M3_WAX,
G4_MAGNESIUM_CARBONATE, G4_MAGNESIUM_FLUORIDE,
G4_MAGNESIUM_OXIDE, G4_MAGNESIUM_TETRABORATE,
G4_MERCURIC_IODIDE, G4_METHANE,
G4_METHANOL, G4_MIX_D_WAX,

G4_MS20_TISSUE, G4_MUSCLE_SKELETAL_ICRP,
G4_MUSCLE_STRIATED_ICRU, G4_MUSCLE_WITH_SUCROSE,
G4_MUSCLE_WITHOUT_SUCROSE, G4_NAPHTHALENE,
G4_NITROBENZENE, G4_NITROUS_OXIDE,
G4_NYLON-8062, G4_NYLON-6/6,
G4_NYLON-6/10, G4_NYLON-11_RILSAN,
G4_OCTANE, G4_PARAFFIN,
G4_N-PENTANE, G4_PHOTO_EMULSION,
G4_PLASTIC_SC_VINYLTOLUENE, G4_PLUTONIUM_DIOXIDE,
G4_POLYACRYLONITRILE, G4_POLYCARBONATE,
G4_POLYCHLOROSTYRENE, G4_POLYETHYLENE,
G4_MYLAR, G4_PLEXIGLASS,
G4_POLYOXYMETHYLENE, G4_POLYPROPYLENE,
G4_POLYSTYRENE, G4_TEFLON,
G4_POLYTRIFLUOROCHLOROETHYLENE, G4_POLYVINYL_ACETATE,
G4_POLYVINYL_ALCOHOL, G4_POLYVINYL_BUTYRAL,
G4_POLYVINYL_CHLORIDE, G4_POLYVINYLIDENE_CHLORIDE,
G4_POLYVINYLIDENE_FLUORIDE, G4_POLYVINYL_PYRROLIDONE,
G4_POTASSIUM_IODIDE, G4_POTASSIUM_OXIDE,
G4_PROPANE, G4_lPROPANE,
G4_N-PROPYL_ALCOHOL, G4_PYRIDINE,
G4_RUBBER_BUTYL, G4_RUBBER_NATURAL,
G4_RUBBER_NEOPRENE, G4_SILICON_DIOXIDE,
G4_SILVER_BROMIDE, G4_SILVER_CHLORIDE,
G4_SILVER_HALIDES, G4_SILVER_IODIDE,
G4_SKIN_ICRP, G4_SODIUM_CARBONATE,
G4_SODIUM_IODIDE, G4_SODIUM_MONOXIDE,
G4_SODIUM_NITRATE, G4_STILBENE,
G4_SUCROSE, G4_TERPHENYL,
G4_TESTES_ICRP, G4_TETRACHLOROETHYLENE,
G4_THALLIUM_CHLORIDE, G4_TISSUE_SOFT_ICRP,
G4_TISSUE_SOFT_ICRU-4, G4_TISSUE-METHANE,
G4_TISSUE-PROPANE, G4_TITANIUM_DIOXIDE,
G4_TOLUENE, G4_TRICHLOROETHYLENE,
G4_TRIETHYL_PHOSPHATE, G4_TUNGSTEN_HEXAFLUORIDE,
G4_URANIUM_DICARBIDE, G4_URANIUM_MONOCARBIDE,
G4_URANIUM_OXIDE, G4_UREA,
G4_VALINE, G4_VITON,
G4_WATER, G4_WATER_VAPOR,
G4_XYLENE, G4_GRAPHITE,
G4_lH2, G4_lN2,
G4_lO2, G4_lAr,
G4_lKr, G4_lXe,
G4_PbWO4, G4_Galactic

Solid

:SOLID

1. solid name
2. solid type name
3. ... List of solid parameters

The meaning and order of the solid parameters is the same as in the corresponding Geant4 solid constructor. All the Geant4 CSG and "specific" solids are implemented. The list of solid types and the corresponding parameters is the following (for better understanding of the solid parameters meaning we refer to the Geant4 user's manual [7]):

BOX: box

1. X Half-length
2. Y Half-length
3. Z Half-length

TUBE: tube

1. Inner radius
2. Outer radius
3. Half length in z

TUBS: tube section

1. Inner radius
2. Outer radius
3. Half length in z
4. Starting phi angle
5. Delta angle of the segment

CONE: cone

1. Inner radius at -fDz
2. Inner radius at +fDz
3. Outer radius at -fDz

4. Outer radius at +fDz
5. Half length in z (=fDz)

CONS: cone section

1. Inner radius at -fDz
2. Inner radius at +fDz
3. Outer radius at -fDz
4. Outer radius at +fDz
5. Half length in z (=fDz)
6. Starting angle of the segment
7. Delta angle of the segment

TRD: trapezoid

1. Half-length along x at the surface positioned at -dz
2. Half-length along x at the surface positioned at +dz
3. Half-length along y at the surface positioned at -dz
4. Half-length along y at the surface positioned at +dz
5. Half-length along z axis

PARA: paralellepiped

1. Half-length along x at the surface positioned at -dz
2. Half-length along x at the surface positioned at +dz
3. Half-length along y at the surface positioned at -dz
4. Angle formed by the y axis and by the plane joining the centre of the faces G4Parallel to the z-x plane at -dy and +dy
5. Polar angle of the line joining the centres of the faces at -dz and +dz in z
6. Azimuthal angle of the line joining the centres of the faces at -dz and +dz in z Half-length along y at the surface positioned at +dz

TRAP: generic trapezoid

1. Half-length along the z-axis (=pDz)

2. Polar angle of the line joining the centres of the faces at -/+pDz
3. Azimuthal angle of the line joining the centre of the face at -pDz to the centre of the face at +pDz
4. Half-length along y of the face at -pDz (=pDy1)
5. Half-length along x of the side at y=-pDy1 of the face at -pDz
6. Half-length along x of the side at y=+pDy1 of the face at -pDz
7. Angle with respect to the y axis from the centre of the side at y=-pDy1 to the centre at y=+pDy1 of the face at -pDz
8. Half-length along y of the face at +pDz (=pDy2)
9. Half-length along x of the side at y=-pDy2 of the face at +pDz
10. Half-length along x of the side at y=+pDy2 of the face at +pDz
11. Angle with respect to the y axis from the centre of the side at y=-pDy2 to the centre at y=+pDy2 of the face at +pDz

or alternatively, if your trapezoid is a simpler one, you can use the parameters

1. Length along z
2. Length along y
3. Length along x at the wider side
4. Length along x at the narrower side

SPHERE: sphere

1. Inner radius
2. Outer radius
3. Starting angle of the segment
4. Delta angle of the segment
5. Theta starting angle of the segment
6. Theta delta angle of the segment

ORB: full solid sphere

1. Outer radius

TORUS: torus

1. Inside radius
2. Outside radius
3. Swept radius of torus
4. Starting Phi angle (fSPhi+$fDPhi <= 2PI$, $fSPhi > -2PI$)
5. Delta angle of the segment

POLYCONE: polycone

1. Initial phi starting angle
2. Total phi angle
3. Number of z planes or Number of rz points

For each z plane:

1. Position of z plane
2. Tangent distance to outer surface
3. Half-length along the z-axis

For each rz corner:

1. R coordinate of these corners
2. Z coordinate of these corners

The software will know which if the numbers refer to plane or rz points by looking at the number of parameters provided and comparing it with the number expected

Example:

```
:SOLID polyc POLYCONE 0 360 6
3 -2
3.5 -2
3.5 0.75
3.75 1
3.75 2
3 2
```

or equivalently

```
:SOLID polyc POLYCONE 0 360 4
-2 3 3.5
0.75 3 3.5
1. 3. 3.75
2. 3. 3.75
```

POLYHEDRA: polyhedra

1. Initial phi starting angle
2. Total phi angle
3. Number of sides
4. Number of z planes or Number of rz points

For each z plane:

1. Position of z plane
2. Tangent distance to outer surface
3. Half-length along the z-axis

For each rz corner:

1. R coordinate of these corners
2. Z coordinate of these corners

The software will know which if the numbers refer to plane or rz points by looking at the number of parameters provided and comparing it with the number expected

Example:

```
:SOLID polyh POLYHEDRA 20. 180. 3 4
1900. 32.
1800. 30
1800. 0.
1900. 0.
```

or equivalently

```
:SOLID polyh POLYHEDRA 20. 180. 3 2
1800. 0. 30.
1900. 0. 32.
```

ELLIPTICAL_TUBE: elliptical tube

1. Half length X
2. Half length Y
3. Half length Z

ELLIPSOID: ellipsoid

1. Semiaxis in X
2. Semiaxis in Y
3. Semiaxis in Z
4. Lower cut plane level, z
5. Upper cut plane level, z

ELLIPTICAL_CONE: elliptical cone

1. Semiaxis in X
2. Semiaxis in Y
3. Height of elliptical cone
4. Upper cut plane level

HYPE: hyperbolic profile

1. Inner radius
2. Outer radius
3. Inner stereo angle
4. Outer stereo angle
5. Half length in Z

TET: tetrahedron

1. Anchor point
2. Point 2
3. Point 3
4. Point 4
5. Flag indicating degeneracy of points

TWISTED_BOX: box twisted along one axis

1. Twist angle
2. Half x length
3. Half y length
4. Half z length

TWISTED_TRAP: trapezoid twisted along one axis

1. Twisted angle
2. Half x length at y=-pDy
3. Half x length at y=+pDy
4. Half y length (=pDy1)
5. Half z length (=pDz)
6. Polar angle of the line joining the centres of the faces at -/+pDz
7. Half y length at -pDz (=pDy2)
8. Half x length at -pDz, y=-pDy1
9. Half x length at -pDz, y=+pDy1
10. Half y length at +pDz
11. Half x length at +pDz, y=-pDy2
12. Half x length at +pDz, y=+pDy2
13. Angle with respect to the y axis from the centre of the side

TWISTED_TRD: twisted trapezoid with the x and y dimensions varying along z

1. Half x length at the surface positioned at -pDz
2. Half x length at the surface positioned at +pDz
3. Half y length at the surface positioned at -pDz
4. Half y length at the surface positioned at +pDz
5. Half z length (=pDz)
6. Twisted angle

TWISTED_TUBS: tube section twisted along its axis

1. Twisted angle
2. Inner radius at end-cap
3. Outer radius at end-cap
4. Half z length
5. Phi angle of a segment

Boolean solids

The three types of Geant4 boolean solids are supported: union, subtraction and intersection. The same tag should be used as for normal solids, but putting as solid type the type of boolean operation. The parameters are

1. Solid name
2. Solid boolean operation (UNION/SUBTRACTION/INTERSECTION)
3. First component solid name
4. Second component solid name
5. Name of relative rotation matrix
6. Relative X position
7. Relative Y position
8. Relative Z position

Example:

```
:SOLID myunion UNION solid1 solid2 RM30 -11.8 12.5 0.
```

Volume

There are two ways to define a volume. You can build if from a previously declared solid associating a material to it,

:VOLU

1. Volume name
2. Solid name
3. Material name

Example:

```
:VOLU HALL HALL Air
```

or you can skip the definition of the solid and in one unique line define the solid and the material (valid also for boolean solids). You should then use the same format as for the :SOLID tag, but adding as last word the material name

Example: Instead of

```
:SOLID HALL BOX  5000. 5000. 20000.

:VOLU HALL HALL Air
```

use

```
:VOLU HALL BOX  5000. 5000. 20000. Air
```

Placement of a volume

All the possible ways to place a volume in Geant4 are supported: a single placement, a parameterised one, a division, a replica and an assembly.

Single placement
:PLACE

1. Volume name
2. Copy number
3. Parent volume name
4. Name of rotation matrix
5. X position
6. Y position
7. Z position

Example:

```
:VOLU yoke :TUBS  Iron   3  620. 820. 1270. \\
:PLACE  yoke  1   expHall  R00       0.0     0.0     370.
```

Parameterisation

The parameterisations supported are the placement of several copies of a volume along a line, in a circle and in a bidimensional grid (other types of parameterisation may be added at user request).

:PLACE_PARAM

1. Volume name

2. Copy number
3. Parent volume name
4. Parameterisation type
5. Name of rotation matrix
6. Number of copies
7. Step (separation between copies)
8. Offset
9. Extra arguments (optional, depend on parameterisation type)

There are three types of linear parameterisation, along the three axis X,Y,Z (types: **LINEAR_X, LINEAR_Y, LINEAR_Z**) and a general one (type **LINEAR**) for which you have to add as extra arguments the axis direction **DIR_X DIR_Y DIR_Z**. The offset for linear parameterisations represents the distance from the centre of the first copy to the point (0,0,0) along the line.

In the case of circle parameterisation, the circle is around the Z axis by default. If you want a circle around another axis you can provide as extra arguments the axis and, optionally, the position of the first copy.

There are three types of square parameterisation, in the planes XY,XZ,YZ (types: **SQUARE_XY, SQUARE_XZ, SQUARE_YZ**) and a general one (type **SQUARE**). For this bidimensional parameterisations you have to provide two copy numbers, two steps and, optionally, two offsets. For the general case, **SQUARE** the offset is not needed, but you have to add as extra arguments the two axis, that do not have to be orthogonal, and, optionally, the position of the first copy. In the case of this paramerisation type, you have to provide two number of copies, one for each axis.

Example:

```
:PLACE_PARAM mytube 1 subworld2 LINEAR_X RM0 5 20. 0.

:PLACE_PARAM mytube 1 subworld1 LINEAR RM0 5 20
0. 1. 1. 1. -50. 0. 0.

:PLACE_PARAM mybox 0 mother CIRCLE RM0 30 0.209 1 150

:PLACE_PARAM mybox 1 subworld2 SQUARE_XZ RM0 5 5 20. 20.

:PLACE_PARAM mybox 1 subworld1 SQUARE RM0 5 8
20. 10. 0. 1. 1. 0. 1. 0.
```

Be aware that putting offset = 0 means that the first copy is placed at (0,0,0). This may be not what you want if, for example, you are filling a box with an square of small boxes using an square parameterisation: offset 0 will mean that all the copies are placed in the positive-positive quarter of the mother box.

Division

There are several ways to define a division in Geant4, by giving:

- the number of divisions (so that the width of each division will be automatically calculated)
- the division width (so that the number of divisions will be automatically calculated to fill as much of the mother as possible)
- both the number of divisions and the division width (this is especially designed for the case where the copies do not fully fill the mother)

To each of these types correspond a different tag

:DIV_WIDTH

1. Volume name
2. Parent volume name
3. Material name
4. Axis of division
5. Division width
6. Offset (not mandatory)

:DIV_NDIV

1. Volume name
2. Parent volume name
3. Material name
4. Axis of division
5. Number of divisions
6. Offset (not mandatory)

:DIV_NDIV_WIDTH

1. Volume name

2. Parent volume name
3. Material name
4. Axis of division
5. Number of divisions
6. Division width
7. Offset (not mandatory)

Example:

```
:DIV_WIDTH mybox mother AIR Z 10.

:DIV_NDIV_WIDTH mytube mother copper PHI 12 10.*deg
```

Replica
To define a replica the following tag must be used:
:REPL

1. Volume name
2. Parent volume name
3. Axis of division
4. Number of divisions
5. Division width
6. Offset (not mandatory)

Example:

```
:REPL crystal Block X 10 5.*cm
```

Remember, that different to the divisions, where the solid type and dimensions are calculated automatically be Geant4, in the case of replicas the volume name used must be the name of a previously defined volume. This solid is not really used for navigation but should have the correct type and dimensions for visualisation.

Assembly volumes
Assembly volumes are sets of logical volumes that are combined together, so that they act as if there were in a real mother, but without creation of the mother.

To define assembly volumes you have to define the relative rotations and positions of all the logical volumes
:VOLU_ASSEMBLY

1. Volume name
2. Number of logical volumes
3. Axis of division
4. Number of divisions
5. Division width
6. Offset (not mandatory)

One line per logical volume with

1. Logical volume name
2. Rotation matrix name
3. position X
4. position Y
5. position Z

Then to place the assembly volume you can use:
:PLACE_ASSEMBLY

1. Volume name
2. Copy number
3. Parent volume name
4. Name of rotation matrix
5. X position
6. Y position
7. Z position

Example:

```
:SOLID Crystal BOX 10 10 10
:SOLID Crystal2 BOX 5 5 5
:VOLU_ASSEMBLY CrystalSet 3
Crystal RM0 0. 0. 0.
Crystal RM1 0. 0. 20.
Crystal2 RM0 0. 20 -10

:PLACE_ASSEMBLY CrystalSet  1  expHall  R00   100.    0.    0.
```

Rotation matrix

A rotation matrix is interpreted as the rotation that should be applied to the object in the reference system of its mother. It can be defined in three ways:

a) By giving the three rotation angles around the X, Y and Z axis (in this order of rotations)

b) By giving the polar and azimuthal angles of the X, Y and Z axis after the rotation is applied

c) By giving the nine matrix elements of the rotation matrix: XX, XY, XZ, YX, YY, YZ, ZX, ZY, ZZ

The tag for the three cases is the same. The parser will know which case is meant by the number of parameters.

a) :ROTM

1. Name
2. Angle of rotation around global X axis
3. Angle of rotation around global X axis
4. Angle of rotation around global X axis

Example:

```
:ROTM   R0 0. 0. 0.
```

b) :ROTM

1. Name
2. Polar angle for axis X
3. Azimuthal angle for axis X
4. Polar angle for axis Y
5. Azimuthal angle for axis Y
6. Polar angle for axis Z
7. Azimuthal angle for axis Z

Example:

```
:ROTM   R0 90. 0.  90. 90.  0. 0.
```

c) :ROTM

1. Name
2. 9 parameters defining the rotation matrix

Example:

```
:ROTM   RO 1. 0. 0.  0. 1. 0.  0. 0. 1.
```

Visibility

:VIS

1. Volume name
2. ON or TRUE, OFF or FALSE

Example:

```
:VIS yoke OFF
```

By default the visibility of all volumes is set to ON

Colour and transparency

To define the colour of a volume
:COLOUR/:COLOR

1. Volume name
2. Red colour proportion
3. Green colour proportion
4. Blue colour proportion
5. Transparency

The four parameters can take a value between 0 and 1. The transparency parameter is not mandatory.
Example:

```
:COLOUR NDC_chamber 0.2 0.4 0.1
```

By default, the three colour proportions will be set to -1.

Check overlaps

Geant4 offers the possibility to check if a volume overlaps with other volumes. By default it is not set, but you can activate it with the commands
:CHECK_OVERLAPS

1. Volume name
2. ON or TRUE, OFF or FALSE

Example:

```
:CHECK\_OVERLAPS NDC_chamber 0.2 0.4 0.1
```

By default, the three colour proportions will be set to -1.

Use of '*' in names

In the case of the **:VIS**, **:COLOUR** and **:CHECK_OVERLAPS** tags, you may use '*' to define the volume name. This '*' will be replaced by 'any name'. For example **Crys*** means all the volume names starting by 'Crys', ***** means all volume names.

Use of parameters

You can also define a parameter for later use in any tag.
:P

1. parameter name
2. parameter value

You can then use the parameter as: '$' + parameter_name
Example:

```
:P InnerR 12.
:VOLU yoke :TUBS  Iron  3  $InnerR 820. 1270.
```

Units

Any value in a tag has a default unit that depends on the dimension of the value (automatically known by the parser). The default units are the following:

- length: *mm*
- angle: *degrees*
- density: *g/cm3*

- atomic mass: *g/mole*

The user can override the default value of a unit by indicating the unit of each value. This can be done adding at the end of the value the unit name (see CLHEP file **Units/SystemOfUnits.h**) preceded by a '*' character; e.g. 3*mm, 1.4*rad,...

Arithmetic Expressions

For any value you want to define in a tag you can use the most common mathematical expressions instead of directly writing the figures, e.g.

3-sin(8.2/3.5) , (3+4)*(7-log(3))

You can also use parameters in the expressions, e.g.

7.2*$RADIUS$−X_LENGTH/1.5

If you use a regular expression, remember that there can only be a unit in the whole expression, and it must be at the end.

The regular expressions used include (their meaning is evident): **+, -, *, /, sin, cos ,tan ,asin ,acos ,atan ,atan2 ,sinh ,cosh ,tanh ,sqrt ,exp ,log ,log10 ,pow)**.

Including other files

You can next several files by using the **#include'** directive any where in your geometry files.

Example:

```
#include mygeom2.txt
```

Combining C++ and ASCII files

If you want to define part of your geometry with C++ and another part with ASCII files, you should follow the following instructions.

Write a C++ class inheriting from **G4VuserDetectorConstruction** and in the **Construct()** method build the geometry from a set of ASCII files

```
G4tgbVolumeMgr* volmgr = G4tgbVolumeMgr::GetInstance();

volmgr->AddTextFile(‘‘mifile1.txt’’);
volmgr->AddTextFile(‘‘mifile2.txt’’);

G4VPhysicalVolume* physiSubWorld =
  volmgr->ReadAndConstructDetector();
```

You can then use the returned **G4VPhysicalVolume** as the world volume or get its logical volume and place it inside any other logical volume.

You can also use the materials and volumes of the ASCII geometry in your C++ geometry retrieving them by name. To retrieve the pointer of a material:

```
GmGeometryUtils* geomUtils = GmGeometryUtils::GetInstance();
G4Material* mate =
  geomUtils->GetMaterial("Air",exists=true);
```

To retrieve a logical volume

```
GmGeometryUtils* geomUtils = GmGeometryUtils::GetInstance();
// one volume
G4LogicalVolume* world_logic =
 (geomUtils->GetLogicalVolumes("world",true))[0];

// several volumes with same name
std::vector<G4LogicalVolume*> crystal_logic =
  (geomUtils->GetLogicalVolumes("crystal",exists=true));
```

To retrieve the physical volumes

```
GmGeometryUtils* geomUtils = GmGeometryUtils::GetInstance();
std::vector<G4VPhysicalVolume*> crystal_phys =
  (geomUtils->GetPhysicalVolumes("crystal",exists=true));
```

3.1.2 Dumping your Geant4 geometry in text file format

If you have already a Geant4 geometry written with C++ code or any other method you can get a geometry text file by simply adding a line in your code

```
G4tgbGeometryDumper::GetInstance()
  ->DumpGeometry(theFileName);
```

This line should be executed once your geometry has been built, for example in a **BeginOfRunAction** method, or at the end of the **Construct** method in your detector constructor class.

You can do this in GAMOS by simply adding in your command file the user action named **GmGeomTextDumperUA** :

/gamos/userAction GmGeomTextDumperUA

The name of the output file can be set by setting the parameter (before the user action)

/gamos/setParam GmGeomTextDumperAction:OutputName FILE_NAME

3.1.3 Adding new tags to your input text file

You may want to add new tags to your text file and give them any meaning you like. For example you may use your text file to define your G4Region's and assigning different production cuts to each region, as it will be illustrated in the following lines.

The first step is to define a class inheriting from **G4tgrLineProcessor** (see for example **GamosCore/GamosGeometry/include/ GmGeomTextLineProcessor.hh**). You should then define a method

```
virtual G4bool GmGeomTextLineProcessor::
  ProcessLine( const std::vector<G4String>& wl )}
```

This is the method that will be invoked each time a line in your file is read, passing to it the line as a vector of strings. In this method you should first call the default line processor to process the standard tags defined through this chapter.

```
  G4bool iret = G4tgrLineProcessor::ProcessLine( wl );
```

iret will be set to 1 if the tag is found, else you should process the tag yourself. For example

```
  if( !iret ) {
    //------------------------------ parameter number
    if( wl0 == ":REGION" ) {
      GmRegionCutsMgr::GetInstance()->AddRegionData( wl );
      iret = 1;
    }
  }
```

The second step is to define a class inheriting from **G4tgbDetectorBuilder** (see **GamosCore/GamosGeometry/include/ GmGeomTextDetectorBuilder.hh**). You should then define a method

```
virtual const G4tgrVolume*
 GmGeomTextDetectorBuilder::ReadDetector()
```

to set as line processor the one you have created, and to trigger the reading of the file

```
//------------------- construct g4 geometry
GmGeomTextLineProcessor* tlproc =
   new GmGeomTextLineProcessor;
G4tgrFileReader* tfr = G4tgrFileReader::GetInstance();
tfr->SetLineProcessor( tlproc );
```

```
tfr->ReadFiles();
//---------- find top G4tgrVolume
G4tgrVolumeMgr* tgrVolmgr = G4tgrVolumeMgr::GetInstance();
const G4tgrVolume* tgrVoltop = tgrVolmgr->GetTopVolume();
return tgrVoltop;
```

You should also define another method

```
virtual G4VPhysicalVolume* GmGeomTextDetectorBuilder::
    ConstructDetector( const G4tgrVolume* tgrVoltop )
```

You should first call the default detector builder to build the Geant4 geometry using the standard tags defined through this chapter.

```
G4VPhysicalVolume* topPV =
  G4tgbDetectorBuilder::ConstructDetector( tgrVoltop );
```

After that you can add your code that will process your new tags

```
//--- Create regions
GmRegionCutsMgr::GetInstance()->BuildRegions();
```

Finally, in your detector construction class, inheriting from **G4VUserDetectorConstruction**, you have to set up your detector builder

```
//---- Construct your detector builder
GmGeomTextDetectorBuilder* gtb =
  new GmGeomTextDetectorBuilder;

//---- Inform G4tgbVolumeMgr to use your detector builder,
//     instead of the default one
G4tgbVolumeMgr* volmgr = G4tgbVolumeMgr::GetInstance();
volmgr->SetDetectorBuilder( gtb );

//----- Trigger the detector construction
G4VPhysicalVolume* physiWorld =
  volmgr->ReadAndConstructDetector();

return physiWorld;
```

3.1.4 Parallel geometries

You can define a parallel geometry by including a second file for **GmGeometryFromText**, with the command

/gamos/setParam GmGeometryFromText:FileNameParallel FILE_NAME FILE_NUMB ER

where **FILE_NAME** is the name of a file similar to the one that describe your mass geometry (you can indeed interchange them). **FILE_NUMBER** is the number you associate to the parallel geometry, as you may have several parallel geometries at the same time. The only difference between a parallle geometry file and a mass geometry file is that in the case of parallel geometry, the volume at the top of the hierarchy (world volume) should not appear in the file, as Geant4 creates it automatically copying the mass world volume. This means that you should place your geometry in the same world as the volumes of the mass geometry.

The parallel geometry will not be seen by Geant4 unless a process is instatntiated to take care of it. To do it you can create a **G4ParallelWorldScoringProcess** with the following command

/gamos/physics/useParallelScoringProcess

When it is a parallel geometry volume boundary the one that limits the step, the process that defined the step is called **parallelWorldProcess_N**, where **N** is the number you gave to the parallel geometry that is acting.

The **G4ParallelWorldScoringProcess** process takes care of changing the touchable so that it points to the parallel geometry, therefore if an scorer acts on a step, the **G4PreStepPoint** and **G4PostStetPoint** will return a touchable corresponding to a volume of the parallel geometry in case the track navigates in it, else a touchable of the mass geometry. Nevertheless, the **G4VPhysicalVolume** is not cahnged and it will always point to the mass geometry volumes. In this way the user can access at the same time the volume of the parallel geometry and the volume of the mass geometry (see an example in the **Histograms and scorers** tutorial). The user must be aware that is the scorer mechanism that makes these changes, therefore the user actions will not see the parallel geometry. Please ask for this functionality in case you think you need it.

Simulating materials and interactions in parallel geometries

In any case, Geant4 can navigate in the parallle geometries, but the materials are never taken into account. This means that a track never interacts on a parallel geometry volume. We have developed in GAMOS an utility that allows to have interactions in both geometries at the same time, that is to have real overlapping geometries. This maybe useful for example to simulate the real geometry of brachytherapy seeds or ionisation chambers inside a phantom. This utility is based on making a copy of the parallel geometry in the mass geometry. When a particle is going to enter the parallel geometry volume its position is shifted to the border of the copy of it in the mass geometry and when a particle exits the parallel volume copy its position is shifted back to the border of the parallel geometry. To activate this utility you just have to use the command:

/gamos/geometry/copyParallelToMassGeom VOL_NAME_1 VOL_NAME_2 ... VOL_NAME_N DISP_X DISP_Y DISP_Z
where **VOL_NAME_1 VOL_NAME_2 ... VOL_NAME_N** is the list of volumes in a parallel geometry that will be copied and **DISP_X DISP_Y DISP_Z** are the values of the displacement vector.

The user should check that the copy of the parallel geometry volumes in the mass geometry are inside the user-defeind world volume, and also that they do not overlap with any of the preexisting mass geometry volumes. GAMOS will check that these two conditions are satisfied, but only a warning message will be sent. We also recommend taht the copy is placed far from the rest of the mass geometry voules. If this is not done, it may happen taht some particles navigating in the mass geometry will enter the copy, what is a non-physical situation. Alternatively, the user can take care of killing the particles that approach the copy, for example by using the user action **GmKillAllUA** with the corresponding filters.

3.2 Building simple geometries

There are several examples of geometries for common medical devices, for example the ones you find in the PET directory and the one to build simple voxelised phantoms. They have been designed to be used for describing different devices by simply changing the configuration data. For more details, please see the corresponding sections of this manual.

3.3 Building your geometry with C++ code

You can build your geometry by writing your C++ class inheriting from G4VUserDetectorConstruction (see example in [7]). After that you have to transform it into a GAMOS plug-in. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmGeometryFactory.

3.4 Reading DICOM files

GAMOS is able to read the patient data resulting from a CT, PET, SPECT, NMR, etc. in DICOM format. To convert the DICOM format to a format readable by GAMOS adding the material and density information you may use the Geant4 example **examples/extended/medical/DICOM**. In this example you can also find the description of the DICOM format readable by Geant4 and GAMOS.

To use this utility you should define as your geometry the class
/gamos/geometry GmReadPhantomG4Geometry

You should have then a file where your phantom is described, whose name is set with the parameter

/gamos/setParam GmReadPhantomGeometry:Phantom:FileName MY_FILENAME

and another file where you describe the rest of your geometry (at least the world volume where the phantom is placed). The name of this file is set with the parameter

/gamos/setParam GmReadPhantomGeometry:FileName MY_FILENAME

In a phantom file, the voxels of the same material may have a different density. GAMOS allows you to group densities in intervals. You have to set true the parameter

/gamos/setParam GmReadPhantomGeometry:RecalculateMaterialDensities 1

and choose the interval width with the parameter

/gamos/setParam GmReadPhantomGeometry:Phantom:DensityStep DENSITY_INTERVAL

so that that the voxels of each material will be grouped in density intervals of **DENSITY_INTERVAL** and a new material will be created for each group of voxels.

The navigation in the voxels is done using the Geant4 algorithm, **G4RegularNavigation**, that is the optimal one for regular geometries (see [8]). The user may select if when a track navigates through contiguous voxels with the same material the frontier between will be skipped or not, with the parameter

/gamos/setParam GmReadPhantomGeometry:Phantom:SkipEqualMaterials VALUE

that by default takes a value of 1.

For testing purposes, other navigation algorithms may be selected, namely voxel navigation with 1-dimensional optimization (that occupies similar memory as **RegularNavigation** but is very slow)

/gamos/setParam GmReadPhantomGeometry:Phantom:RegularStructureID 0

or 3-dimensional optimization (that occupies a lot of memory and it is almost as fast as **G4RegularNavigation**)

/gamos/setParam GmReadPhantomGeometry:Phantom:OptimAxis kUndefined

GAMOS is also able to read the DOSXYZnrc format for DICOM files that is commonly used in EGSnrc. Simply use

/gamos/geometry GmReadPhantomEGSGeometry

and the rest of parameters are the same as for the Geant4 DICOM files.

By default the phantom is placed in the world volume. If you want to place it into another physical volume, you can set the parameter

/gamos/setParam GmReadPhantomGeometry:MotherName PHYSVOL_NAME

3.4.1 Simple phantom geometries

The user may build simple regular phantom geometries without the need of writing a DICOM file by using

/gamos/geometry GmSimplePhantomGeometry

The number of voxels is defined with the parameter

/gamos/setParam GmSimplePhantomGeometry:NVoxels NVOXEL_X NVOXEL_Y NVOXEL_Z

The minimum and maximum extensions in the three axes are defined with the parameter

/gamos/setParam GmSimplePhantomGeometry:PhantomDims MIN_X MAX_X MIN_Y MAX_Y MIN_Z MAX_Z

Then you can divide the phantom in different regions along the Z axis with the parameter

/gamos/setParam GmSimplePhantomGeometry:MaterialZVoxels NZ_1 NZ_2 ...

where **NZ_i** is the number of voxels along Z of the i region.

Then you can assign the material and material densities of each Z region with the parameters

/gamos/setParam GmSimplePhantomGeometry:MaterialNames MATERIAL_1 MATERIAL_2 ...

/gamos/setParam GmSimplePhantomGeometry:MaterialDensities DENSITY_1 DENSITY_2 ....

3.5 Movements

Thanks to the functionality of Geant4, it is possible to displace or rotate a volume during a run. To do this in GAMOS you have first to select the volume you want to move and if you want to move it after a certain number of events or after a certain time is elapsed (the time will be checked after each event, therefore the time interval will only be approximated). You also have to choose how much you want to move it and the axis (the axis of displacement or the axis around which happens the rotation). You have then to set the interval of events or time between movements. After you can define an offset so that the first interval does not start at 0. Finally you can choose that your movement is done forever (i.e. until the number of events in the run are exhausted) or it is only done n times.

The commands to tell GAMOS to make a movement are:

/gamos/movement/moveEachTime

if you want that the movement happens after a certain interval of time, and

/gamos/movement/moveEachNEvents

if you want that the movement happens after a certain number of events.

In both cases the command has to be followed by these arguments:

'displace/rotate' volume_name value axis_x axis_y axis_z time_interval/nevents_interval (offset=0) (number_of_intervals=infinite)
where the first word has to be either *displace* or *rotate*, **volume_name** must be one of the Geant4 volumes of your geometry, **value** is the amount by which you want to displace or rotate (default Geant4 units are assumed, i.e. 'mm' and 'rad'; you can change them in the usual way, e.g., '*cm', '*deg'), **axis_x axis_y axis_z** are the three coordinates of the axes, **time/nevents_interval** is the interval after which the movements will happen, **(offset=0)** is an optional argument (0 if not set) to change the value for the first movement, and **(number_of_intervals=infinite)** is an optional argument (infinite by default) to set the number of times the movement will happen.

If you want several movements in your run, you can use these commands as many times as you want. Then after each event GAMOS will check which of the movements must be done. If you want for example to move the same volume with different types of movements one after the other, you can use the 'number_of_intervals' and 'offset' arguments to do it.

The movements are managed in GAMOS by a user event action, called **GmMovementEventAction**, that checks at the beginning of event if the movement must be done. Therefore you cannot forget to activate this action with the GAMOS command:

/gamos/userAction GmMovementEventAction
If you forget it, the above commands will not exist and you will get a Geant4 exception.

There is an example of GAMOS movements that you can run in the directory **MY_GAMOS_DIR/examples/test**. Just run

gamos testMovement.in
and you will see an OpenGL view of a moving box.

3.6 Geometry utilities

There is a set of geometry utilities that are meant to help the user that is writing some C++ code to for example debug the geometry, get a touchable or a volume by name, etc. They are all in the **GmGeometryUtils** class, which is a singleton. To use them in your C++ code you can do it like in the following example:

```
GmGeometryUtils::GetInstance()->DumpG4LVList();
```

We list here the available methods, with an explanation of their functionality:

- **void DumpSummary(std::ostream& out = G4cout)**: Dumps a summary of the geometry, i.e. number of solids, logical volumes, physical volumes, touchables and materials

- **void DumpG4LVList(std::ostream& out = G4cout)**: Dumps list of logical volumes
- **void DumpG4LVTree(std::ostream& out = G4cout)**: Dumps the hierarchy of logical volumes
- **void DumpG4PVLVTree(std::ostream& out = G4cout)**: Dumps in the following order:
  1. a logical volume with details
  2. list of physical volumes that are daughters of this logical volume with details
  3. list of logical volumes daughters of this logical volume and for each go to 1
- **void DumpMaterialList(std::ostream& out = G4cout)**: Dumps list of materials
- **void DumpSolid(G4VSolid* sol, size_t leafDepth, std::ostream& out = G4cout)**: Dumps a solid with its attributes
- **void DumpLV(G4LogicalVolume* lv, size_t leafDepth, std::ostream& out = G4cout)**: Dumps a logical volume with its attributes
- **void DumpPV(G4VPhysicalVolume* pv, size_t leafDepth, std::ostream& out = G4cout)**: Dumps a physical volume with its attributes
- **G4LogicalVolume* GetTopLV()**: Gets a pointer to the logical volume on top of the geometry hierarchy
- **G4VPhysicalVolume* GetTopPV()**: Gets a pointer to the physical volume on top of the geometry hierarchy
- **G4Material* GetMaterial(const G4String& name, bool exists)**: Gets the material with the given name
- **std::vector<G4LogicalVolume*> GetLogicalVolumes(const G4String& name, bool exists)**: Gets the list of logical volumes with the given name
- **std::vector<G4VPhysicalVolume*> GetPhysicalVolumes(const G4String& name, bool exists)**: Gets the list of physical volumes with the given name
- **std::vector<GmTouchable*> GetTouchables(const G4String& name, bool exists)**: Gets the list of touchables with the given name.
- **std::set¡G4String¿ GetAllSDTypes()**: Gets all distinct sensitive detector types

3.7 Magnetic field

You can set a constant magnetic field with the command

/gamos/magneticField/setField FIELD_X FIELD_Y FIELD_Z + where **FIELD_X FIELD_Y FIELD_Z** are the field values along the three axes. Remember than in Geant4 internal units 1 Tesla is equal to 0.001; therefore if you do not use any unit it will be understood as 1, that is 1000 Teslas.

Chapter 4

Generator

You can use the GAMOS generator selecting the time, energy, position and momentum distribution, using the generator that reads the primary particles from a text file or a binary file or in the standard Geant4 way, by writing your C++ class inheriting from
G4VUserPrimaryGeneratorAction.

4.1 Using GAMOS generator

4.1.1 Introduction

The GAMOS generator provides several time, energy, position and direction distributions that the user may combine to his/her will. The user can select to generate as primary particles one or several single particles together with one or several isotopes and set any of the available time, energy, position or direction distributions for each one of the single particles or isotopes.

We describe below the commands to select the single particles, the commands to select the isotopes and then the commands for selecting the time, energy, position and direction distributions.

The first command you have to use is the one that tells GAMOS that you want to use the GAMOS generator:

/gamos/generator GmGenerator

This command, apart from instantiating the GAMOS generator manager, also instantiates the messenger. Therefore, if you forget to write this command first, the other generator commands described below will not exist and Geant4 will throw an exception.

After this command you have to add one or several particle sources (there is no default primary particle source, therefore if you do not choose one, GAMOS will throw an exception). You may combine several particle sources in each event by repeatedly using the commands described below. Each type of particle source has default distributions of time, energy, position and direction, that you may change with the commands described below.

An important point not to forget is that when you define a particle source you have to give it a name. This name serves to distinguish it when you want later to change its time, energy, position or direction distribution.

4.1.2 Single particle source

To add a single particle source you have to use the command

/gamos/generator/addSingleParticleSource SOURCE_NAME PARTICLE_NAME ENERGY

SOURCE_NAME is the name of this source, that you have to use if you want later to change its time, energy, position or direction distribution.
PARTICLE_NAME can be any of the Geant4 particles [1].
ENERGY is the initial energy of the particle.

If you don't change any property the particle will be generated at time 0., position (0,0,0) and random direction.

4.1.3 Isotope source

GAMOS implements an isotope generator, that simulates the activity of different isotopes that decay in one or several photons, electrons or positrons. First a file is read with the description of the isotope decays in a format as the one that can be found at
MY_GAMOS_DIR/src/GamosCore/GamosGenerator/test/isotopes.dat, part of which we reproduce here

```
:ISOTOPE Na22
       215.5   0.905    e+
      1275.0   0.9995   gamma

:ISOTOPE F18
       249.8   0.967    gamma
```

For each isotope there must be a first line starting by **:ISOTOPE** and followed by the isotope name. Then there is a line for each of the possible isotope decays with three columns describing the decay particle energy, the probability of the decay and the particle type. This file contains the most common isotopes in medical physics. If you want to use another isotope you can add it following the format described above.

If you selected this generator, each event will be generated with one of several primary particles in the decay list of each of the selected isotopes. The presence of each decay particle will occur following its corresponding probability.

[1] For a list of the names of the available particles use the Geant4 command **/run/particle/dumpList** or see Appendix

To choose an isotope as particle source you have to use the command:

/gamos/generator/addIsotopeSource SOURCE_NAME ISOTOPE_NAME ACTIVITY

SOURCE_NAME is the name of this source, that you have to use if you want later to change its time, energy, position or direction distribution.
ISOTOPE_NAME is one of the isotopes read from the input file,
ACTIVITY is the activity you want to set for that isotope[2].

You may choose several isotopes by repeatedly writing this command.

This command triggers the reading of the file named **"isotopes.dat"** if it has not been read yet. You may change the name of this file with the command

/gamos/setParam Generator:Isotope:FileName MY_FILENAME

To learn how to change the directory list where GAMOS looks for this file, please read the section *"Managing the input data files"*.

If you don't change any property the particle will be generated with a time distribution of type "Decay"(see below), energy distribution of type "constant isotope decay" (see below), position at (0,0,0) and random direction.

Time distributions

- Constant time

  /gamos/generator/timeDist SOURCE_NAME GmGenerDistTimeConstant TIME

  All the primary particles will be generated at time 0. If you want to set it at a different time, you can add the extra parameter **TIME**.

- Decay time

  /gamos/generator/timeDist SOURCE_NAME GmGenerDistTimeDecay ACTIVITY

  The primary particles will be generated with a time following a typical decay distribution, with the activity of the particle source. To be concrete the time is sampled with a Poisson distribution that is obtained as follows:

  rnd_poiss = -(1.0/ (activity/second)) * log(RandFlat::shoot());

  the activity is given by the parameter **ACTIVITY**.

Energy distributions

- Constant energy

[2]The available units in Geant4 are becquerel and curie

/gamos/generator/energyDist SOURCE_NAME GmGenerDistEnergyConstant ENERGY

All the primary particles will be generated with energy given by the parameter **ENERGY**.

- Constant decay energy

  /gamos/generator/energyDist SOURCE_NAME GmGenerDistEnergyConstantIsotopeDecay

  All the primary particles will be generated with energy given by the energy of the isotope decay selected by the isotope source, as read from the file **"isotopes.dat"**.

- Random flat energy

  /gamos/generator/energyDist SOURCE_NAME GmGenerDistEnergyRandomFlat MIN_ENERGY MAX_ENERGY

  The primary particles will be generated with an energy given by a random distribution between the minimum and maximum energy.

- Beta decay energy

  /gamos/generator/energyDist SOURCE_NAME GmGenerDistEnergyBetaDecay

  The energy will be sampled following the energy distribution of the decay of the isotope [3]. The energy distribution will be read from a file called **"EnergyDist."+source_particle_name+".dat"**. The data is taken from the **LBNL/LUND Table of Radioactive Isotopes, http://ie.lbl.gov/toi.html; goto "LBNL/LUND Table of Radioactive Isotopes", then "Nuclide search" and save the table "Beta Spectrum"**. There are several examples at **MY_GAMOS_DIR/src/data/EnergyDist.XXX.dat**.

- Gaussian

  /gamos/generator/energyDist SOURCE_NAME GmGenerDistGaussian MEAN SIGMA

  Primary particles will be generated with an energy given by the gaussian distribution of mean **MEAN** and sigma **SIGMA**.

Position distributions

- Position at a point

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionPoint POS_X POS_Y POS_Z

[3] This distribution can not be used for single particle sources

All primary particles are generated at the same position. If no extra argument is given the point is (0.,0.,0.).

- Position in a Geant4 volume

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionInG4Volumes LV_NAME1 LV_NAME2 ...

  The position is distributed randomly inside one or several volumes of the Geant4 geometry. The user must add as extra parameters the list of volume names. The volumes can be physical volumes or touchables[4].

  This distribution can only be used if the volumes are G4Box, G4Orb, G4Sphere, G4Ellipsoid, G4Tubs or G4Cons [5]. If you want to use it for any other volume shape, you have to use the distribution

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionInG4VolumesGeneral LV_NAME1 LV_NAME2 ...

  This distributions works with any solid shape, including boolean solids, but it is in general quite slower than the previous one (it creates a random position in the whole world volume and then looks if it is in one of the selected volumes, which can be quite slow if the volume dimensions are small).

- Position in a user defined volume

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionInUserVolumes POS_X POS_Y POS_Z ANG_X ANG_Y ANG_Z SOLID_TYPE SOLID_DIMENSIONS

  The particles are randomly distributed inside a volume defined by the user (it does not need to be a real volume in the geometry). The user must provide the definition of the volume as extra parameters. **SOLID_TYPE** can be **Orb, Sphere, Ellipsoid, Tubs, Box**. **SOLID_DIMENSIONS** are the solid dimensions. For the order and meaning of the solid dimensions, please look at the corresponding Geant4 solid.

  By default the volume is placed at position (0.,0.,0). If you want it placed at a different position you have to add three optional extra parameters **POS_X POS_Y POS_Z**

  By default the volume is not rotated. If you want it rotated you have to add three optional extra parameters (and always add the three

[4] For understanding the notation to identify touchables in GAMOS, see *Identifying touchables* section.

[5] Other volume types may be added at user request

positions, even if they are (0.,0.,0.)) **ANG_X ANG_Y ANG_Z**. Those angles are interpreted rotating the volume first around the X axis, then around the Y axis and finally around the Z axis.

- Position in a Geant4 volume surface

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionInG4Surfaces LV_NAME1 LV_NAME2 ...

  The position is distributed randomly in the surface of one or several volumes of the Geant4 geometry. The user must add a number of extra parameters with the list of volume names. The volumes can be physical volumes or touchables[6].

  This distribution can only be used if the volumes are G4Box, G4Orb, G4Sphere, G4Tubs or G4Cons[7].

- Position in a user defined volume surface

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionInUserSurfaces POS_X POS_Y POS_Z ANG_X ANG_Y ANG_Z SOLID_TYPE SOLID_DIMENSIONS

  The particles are randomly distributed in a volume surface of a volume defined by the user. The user must provide the definition of the volume as extra parameters. **SOLID_TYPE** can be **Box, Orb, Sphere, Tubs, Cons**. **SOLID_DIMENSIONS** are the solid dimensions. For the order and meaning of the solid dimensions, please look at the corresponding Geant4 solid.

  By default the volume is placed at position (0.,0.,0). If you want it placed at a different position you have to add three optional extra parameters **POS_X POS_Y POS_Z**

  By default the volume is not rotated. If you want it rotated you have to add three optional extra parameters (and always add the three positions, even if they are (0.,0.,0.)) **ANG_X ANG_Y ANG_Z**. Those angles are interpreted rotating the volume first around the X axis, then around the Y axis and finally around the Z axis.

- Adding an extra volume

  If you have defined a distribution of type **GmGenerDistPositionInG4Volumes**, **GmGenerDistPositionInUserVolumes**, **GmGenerDistPositionInG4Surfaces** or **GmGenerDistPositionInUserSurfaces** you can add more volumes with the following user command

[6] For understanding the notation to identify touchables in GAMOS, see *Identifying touchables* section.

[7] Other volume types may be added at user request

/gamos/generator/GmPositionVolumesAndSurfaces/addVolumeOrSurface SOURCE_NAME LV_NAME1 LV_NAME2

for the Geant4 volumes or

/gamos/generator/GmPositionVolumesAndSurfaces/addVolumeOrSurface SOURCE_NAME POS_X POS_Y POS_Z ANG_X ANG_Y ANG_Z SOLID_TYPE SOLID_DIMENSIONS

for the user-defined volumes. The primary particles will be distributed equally in the volumes proportionally to their volume or surface.

- Position in steps along a line

/gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionLineSteps POS_X POS_Y POS_Z DIR_X DIR_Y DIR_Z STEP

The position is distributed uniformly along a line starting at position **POS_X POS_Y POS_Z** and with direction given by the three director cosines **DIR_X DIR_Y DIR_Z**. Each event is generated in a different point along the line, starting at the initial position, and in steps given by **STEP**.

- Position in a square

/gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionSquare WIDTH POS_X POS_Y POS_Z DIR_X DIR_Y DIR_Z

The position is randomly distributed in a 2D square in the XY plane of width **WIDTH** at position (0.,0.,0.). If you want it placed at a different position you have to add three optional extra parameters **POS_X POS_Y POS_Z**. By default the square is not rotated. If you want it rotated you have to add three optional extra parameters (and always add the three positions, even if they are (0.,0.,0.)) **DIR_X DIR_Y DIR_Z**. Those are the director cosines of the Z axis of the square (the axis perpendicular to the 2D surface).

- Position in a rectangle

/gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionRectanglee WIDTH_X WIDTH_Y POS_X POS_Y POS_Z DIR_X DIR_Y DIR_Z

The position is randomly distributed in a 2D rectangle in the XY plane of widths **WIDTH_X** in X and **WIDTH_Y** in Y at position (0.,0.,0.). If you want it placed at a different position you have to add three optional extra parameters **POS_X POS_Y POS_Z**. By default the rectangle is not rotated. If you want it rotated you have to add three optional extra parameters (and always add the three positions,

even if they are (0.,0.,0.)) **DIR_X DIR_Y DIR_Z**. Those are the director cosines of the Z axis of the rectangle (the axis perpendicular to the 2D surface).

- Position in a disc

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionDisc RADIUS POS_X POS_Y POS_Z DIR_X DIR_Y DIR_Z

  The position is randomly distributed in a disc in the XY plane of radius **RADIUS** at position (0.,0.,0.). If you want it placed at a different position you have to add three optional extra parameters **POS_X POS_Y POS_Z**. By default the cylinder is not rotated. If you want it rotated you have to add three optional extra parameters (and always add the three positions, even if they are (0.,0.,0.)) **DIR_X DIR_Y DIR_Z**. Those are the director cosines of the Z axis of the cylinder (the axis perpendicular to the 2D surface).

- Position in a disc with gaussian distribution

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionDiscGaussian SIGMA POS_X POS_Y POS_Z DIR_X DIR_Y DIR_Z

  The position is distributed in a disc in the XY plane with the radius in a gaussian distribution of sigma **SIGMA** and random in phi, at position (0.,0.,0.). If you want it placed at a different position you have to add three optional extra parameters **POS_X POS_Y POS_Z**. By default the cylinder is not rotated. If you want it rotated you have to add three optional extra parameters (and always add the three positions, even if they are (0.,0.,0.)) **DIR_X DIR_Y DIR_Z**. Those are the director cosines of the Z axis of the cylinder (the axis perpendicular to the 2D surface).

- Position in the voxels of a phantom

  /gamos/generator/positionDist SOURCE_NAME GmGenerDistPositionVoxelPhantomMaterials MATERIAL1 MATERIAL2 ...

  The position is randomly distributed in the voxels of a phantom with material equal to one of the materials in the list of parameters. There must be at least one volume defined by a parameterisation of type **G4PhantomParameterisation**.

Direction distributions

- Random distribution

/gamos/generator/directionDist SOURCE_NAME GmGenerDistDirectionRand

The primary particles will be generated in a random distribution so that each solid angle receives the same number of particles.

- Constant distribution

  /gamos/generator/directionDist SOURCE_NAME GmGenerDistDirectionConst

  The primary particles will be generated all in the same direction, given by the extra parameters **DIR_X DIR_Y DIR_Z**.

- Cone distribution

  /gamos/generator/directionDist SOURCE_NAME GmGenerDistDirectionCone

  The primary particles will be generated in a random distribution around a cone, given by the extra parameters **DIR_X DIR_Y DIR_Z OPENING_ANGLE**, so that each solid angle receives the same number of particles.

Position and direction distributions

It is also possible to create distributions where several of the four variables (time, energy, position and direction) are generated at the same time, so that they are related. You have nevertheless to keep in mind that the order of calling will be time, position, energy and direction distributions.

In case you need a different order, for example if the position is determined by the value of the direction, you can calculate both the direction and the position in the *GeneratePosition* method, return only the position and keep the direction in a class data so that it can be returned in the *GenerateDirection* method.

- Position in volume surface, pointing towards centre

  GmGenerDistPositionDirectionInVolumeSurface SOLID_TYPE SOLID_DIMENSIONS POS_X POS_Y POS_Z ANG_X ANG_Y ANG_Z

  The position is distributed in the surface of a volume defined by the user. The user must provide the definition of the volume as extra parameters. **SOLID_TYPE** can be of type **Box**. **SOLID_DIMENSIONS** are the solid dimensions.

  By default the volume is placed at position (0.,0.,0). If you want it placed at a different position you have to add three optional extra parameters **POS_X POS_Y POS_Z**

By default the volume is not rotated. If you want it rotated you have to add three optional extra parameters (and always add the three positions, even if they are (0.,0.,0.)) **ANG_X ANG_Y ANG_Z**. Those angles are interpreted as rotating the volume first around the X axis, then around the Y axis and finally around the Z axis.

The direction is taken as the line that goes from the position to the point (0.,0.,0.).

As this distribution is of position and direction types at the same time, you have to use the two commands

/gamos/generator/positionDist

/gamos/generator/directionDist

You can see editing the file

GamosCore/GamosGenerator/src/GmGenerDistPositionDirectionInVolumeSurface.cc

that internally the method *GenerateDirection* knows the position by interrogating the source.

Creating your own distribution

If you want to use a time, energy, position or direction distribution that is not foreseen in GAMOS, you can easily create your own one. Let's see as example the creation of a time distribution randomly distributed between two values.

You have to create your class inheriting from the class **GmVGenerDistTime** (see for example the class **GmGenerDistTimeConstant**). You have then to implement the method

```
virtual G4double GenerateTime(const GmParticleSource* source);
```

that will return the time value for each event. If you want the user to be able to input some parameters of your class from the command line, like the minimum and maximum time, you have to implement the method

```
virtual void SetParams(const std::vector<G4String>& params);
```

This method will be called automatically passing the extra parameters in the command line selecting your distribution.

Last, you have to transform your distribution into a plug-in. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmGenerDistTimeFactory.

Then you can choose that any of the particle sources use your new distribution in any GAMOS job by adding the command line

/gamos/generator/directionDist SOURCE_NAME MyTimeDist

where **MyTimeDist** is the name you chose when defining your plug-in (which can be different from the name of the class itself).

4.2 Building your generator with C++

You can build your generator by writing your C++ class inheriting from G4VUserPrimaryGeneratorAction (see example in [9]). After that you have to transform it into a GAMOS plug-in. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmGeneratorFactory.

4.3 Reading your generator particles from a text file

You can also define your event primary particles with a text file. The format of the input file is the following:

The first line of each primary event should start with a line that contains as first word **EVENT:** and as second word the number of primary particles in the event. After this each line corresponds to a particle, with a first word **P:** followed by the following variables: particle ID (PDG encoding), position x, y, z (in mm), momentum x, y, z (in MeV), time (in ns).

To select this generator you have to use the command

/gamos/generator GmGeneratorFromFileText

By default the file to be read is called "generator.txt", you can see an example at MY_GAMOS_DIR/data/generator.txt. You can change its name with the command

/gamos/setParam GmGeneratorFromFileText:FileName MY_FILENAME

To change the search path please read the section *"Managing the input data files"*.

4.4 Reading your generator particles from a binary file

You can also define your event primary particles with a binary file. The format of the input file is the following:

The first word is an integer containing the number of particles. Then follows a record for each particle of the type **GenerFileData**

```
struct GenerFileData
{
  int partID;
  float posx, posy, posz, momx, momy, momz, time;
};
```

To select this generator you have to use the command

/gamos/generator GmGeneratorFromFileBin

By default the file to be read is called "generator.bin". You can change its name with the command

/gamos/setParam GmGeneratorFromFileBin:FileName MY_FILENAME

To change the search path please read the section *"Managing the input data files"*.

4.4.1 Event generator histograms

These are histograms of event generator particles that may serve to check that you have actually generated what you meant.

The names of all these histograms start with **GmGenerHistosUA:** and they are all dumped into the file **gener.root/csv**. The following histograms are produced:

- Kinetic energy of primary particles (" Primary Generator kinEnergy")
- X position (" Position X")
- Y position (" Position Y")
- Z position (" Position Z")
- Theta angle (" Angle theta")
- Phi angle (" Angle phi")
- Difference between consecutive event times (taken as time of the first primary particle) (" Time between source decays (ns)")

The user can control the minimum, maximum and number of steps of these histograms, with the following parameters:

/gamos/setParam gener:hEMin
/gamos/setParam gener:hEMax
/gamos/setParam gener:hENbins
/gamos/setParam gener:hPosMin
/gamos/setParam gener:hPosMax
/gamos/setParam gener:hPosNbins
/gamos/setParam gener:hAngleMin
/gamos/setParam gener:hAngleMax
/gamos/setParam gener:hAngleNbins
/gamos/setParam gener:hTimeMin
/gamos/setParam gener:hTimeMax
/gamos/setParam gener:hTimeNbins

To activate this user action use the command:

/gamos/userAction GmGenerHistosUA

Chapter 5

Physics

Yo can build your physics list using one of the Geant4 physics lists or, following the standard Geant4 way, by writing your C++ class inheriting from
G4VUserPhysicsList or using anyone of the Geant4 physics lists.

5.1 GAMOS electromagnetic physics list

The GAMOS electromagnetic physics list is based on the hadron-therapy Geant4 advanced example. Only photons, electrons, positrons and optical photons are defined. This physics list lets the user choose among the standard, low energy or Penelope models.

The following physics models are available:

- **photons**
  - **photon-standard**: standard electromagnetic processes (no low energy)
  - **photon-epdl**: low energy Evaluated Particle Data Library
  - **photon-penelope**: processes a' la Penelope [6]
- **electrons**
  - **electron-standard**: standard electromagnetic processes (no low energy)
  - **electron-eedl**: low energy Evaluated Particle Data Library
  - **electron-penelope**: processes a' la Penelope [6]
- **positrons**
  - **positron-standard**: standard electromagnetic processes (no low energy)

  - **positron-penelope**: processes a' la Penelope [6]

- **optical photons**
  - **opticalphoton**: the scintillation process is activated for all the particles and the **G4OpAbsorption**, **G4OpRayleigh** and **G4OpBoundaryProcess** processes are activated for optical photons

To tell your job to use one of the possible combinations of physics models just described, you have first to select the GAMOS electromagnetic physics list, using the command:

/gamos/physicsList GmEMPhysics

and then you can select one of the physics models for each particle with the command

/gamos/GmPhysics/addPhysics PHYSICS_MODEL_NAME

where **PHYSICS_MODEL_NAME** is one of the above names. If you do not select anyone for a given particle, the first one in each list is taken as default.

For details on the physics implemented in each of these physics models, please read the Geant4 physics manual [10].

5.2 GAMOS hadronic physics list

The GAMOS hadronic physics list is based on the hadron-therapy Geant4 advanced example. It includes the previously defined electromagnetic physics list and adds several options for the physics models for protons and ions.

The following physics models are available (plus all the models described in the previous section):

- **photons**
  - **photon-nuclear**: include photo-nuclear reactions (not included by default)

- **electrons**
  - **electron-nuclear**: include electro-nuclear reactions for electrons (not included by default)

- **positrons**
  - **positron-nuclear**: include electro-nuclear reactions for positrons (not included by default)

- **muons**

  - **muon-standard**: standard electromagnetic processes

- **protons**
  - **proton-precompound**: precompound evaporation model
  - **proton-precompoundFermi**: precompound evaporation plus Fermi break-up models
  - **proton-precompoundGEM**: precompound GEM evaporation model
  - **proton-precompoundGEMFermi**: precompound GEM evaporation plus Fermi break-up models
  - **proton-precompund-binary**: binary cascade model with the default precompound

- **ions**
  - **ion-LowE**: low energy processes, with ICRU49 as stopping power parameterisation
  - **ion-standard**: standard electromagnetic processes
  - **ion-LowE-ziegler1977**: low energy processes, with Ziegler 1977 as stopping parameterisation
  - **ion-LowE-ziegler1985**: low energy processes, with Ziegler 1985 as stopping parameterisation
  - **ion-LowE-ziegler2000**: low energy processes, with SRIM2000 as stopping parameterisation
  - **ion-inelastic-binary-cascade**: ion binary cascade model

To tell your job to use one of the possible combinations of physics models just described, you have first to select the GAMOS electromagnetic physics list, using the command:

/gamos/physicsList GmHadronicPhysics

and then you can select one of the physics models for each particle. If you do not select anyone for a given particle, the first one in each list is taken as default.
and then you can select one of the physics models for each particle with the command

/gamos/GmPhysics/addPhysics PHYSICS_MODEL_NAME

where **PHYSICS_MODEL_NAME** is one of the above names or the names of the GAMOS electromagnetic physics list. If you do not select anyone, the first one in each list is taken as default.

For details on the physics implemented in each of these physics models, please read the Geant4 physics manual [10].

5.3 Building your physics list with C++ code

To build your physics list, first write it in the usual Geant4 way, that is, inheriting from G4VUserPhysicsList (see example in [10]). After that you have to transform it into a plug-in. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmPhysicsFactory.

5.4 Other physics lists

You can use easily in GAMOS any of the Geant4 physics lists [12]. All you have to do to use a Geant4 physics list is to add in the src/module.cc file (in any of the ones in GAMOS code or create your own one) two lines like:

```
#include "QGSP.hh"
DEFINE_GAMOS_PHYSICS(QGSP);
```

Compile it and then you can use it in your command file with the line **/gamos/physicsList QGSP**

There is also in GAMOS a **GmDummyPhysicsList** that defines all the particles, but only the process **G4Transportation**.

5.5 Production cuts

Several physics processes, namely bremsstrahlung, ionization and e+e- pair production from muons have very high cross sections at low energies. It is therefore necessary to implement a production cut so that all particles below it are not generated, but their energy is accounted as energy deposited. Geant4 uses production cuts in range, instead of in energy as used previously by GEANT3 and most Monte Carlo codes. A cut of for example 1. mm for photons means that no photon will be produced if the expected range in the current material is less than 1. mm.

If you use the GAMOS electromagnetic physics list, the default production cut value is 0.1 mm for all processes in all materials. The Geant4 command **/run/particle/setCut 'value' 'unit'** set the cuts for all process to the desired value. But do not forget to use the command **run/initialize** after setting the cuts, if you want that your change is effective. The Geant4 command **/run/particle/dumpCutValues** dumps the list of materials and for each one the list of cuts for each particle.

5.5.1 Production cuts by region

A region in Geant4 is a set of G4LogicalVolume's that share common properties. You can define a region in the text file where you defined your geometry, by using the tag

:REGION REGION_NAME LOGICAL_VOLUME_NAME(s)
where REGION_NAME is the name that identifies the region and LOGICAL_VOLUME_NAME(s) is the list of G4LogicalVolume's that belong to the region. For example:

```
:REGION myRegion Crystal Wall
```

Alternatively you can define a new region in your user script through a user command:

/gamos/geometry/createRegion REGION_NAME LOGICAL_VOLUME_NAME

Do not forget that in Geant4 when a logical volume belongs to a region automatically all its daughters belong to the same region, unless there is another region explicitly defined for some of the daughters.

Also do not forget that regions in Geant4 have to be set in a hierarchical way: if you place a volume A in the world and inside it you place a volume B, you cannot create a new region for B unless you have explicitly created a region for A.

Once you have defined a region, you may set a cut for the particles that traverse that region with the tag

```
:CUTS REGION_NAME  gamma_CUT e-_CUT e+_CUT
```

where REGION_NAME is the name of a previously defined region, gamma_CUT e-_CUT e+_CUT are the cuts for gamma, electrons and positrons (the cut for positron is optional; if not set it will take the one for electrons).

Alternatively you can set the cuts in your user script through a user command:

/gamos/physics/setCuts REGION_NAME gamma_CUT e-_CUT e+_CUT

5.5.2 Energy cuts to range cuts conversion

If you want to know the traslation from an energy cut value to a range cut value for a given particle in a given material, you can do with the following instructions. First you have to instantiate the user action

/gamos/userAction GmCutsEnergy2RangeUA

and then you can use the command

/gamos/physics/ECuts2RangeCuts MATERIAL_NAME CUT_VALUE PARTICLE_NAME

You may use in the material name an '*' if you want to name several materials at the same time. For the particle name only the following names have a meaning: **gamma, e-, e+, e*, ***. This will produce a table with the conversion for each material and particle similar to the following one:

```
GmCutsEnergy2RangeUA:  MATERIAL:  G4_AIR  PART: gamma
  ENERGY CUT: 0.1 (MeV)  = RANGE CUT: 286588
GmCutsEnergy2RangeUA:  MATERIAL:  G4_AIR  PART: e-
  ENERGY CUT: 0.1 (MeV)  = RANGE CUT: 129.155
GmCutsEnergy2RangeUA:  MATERIAL:  G4_AIR  PART: e+
  ENERGY CUT: 0.1 (MeV)  = RANGE CUT: 131.938
GmCutsEnergy2RangeUA:  MATERIAL:  G4_WATER  PART: gamma
  ENERGY CUT: 0.1 (MeV)  = RANGE CUT: 334.152
GmCutsEnergy2RangeUA:  MATERIAL:  G4_WATER  PART: e-
  ENERGY CUT: 0.1 (MeV)  = RANGE CUT: 0.134781
GmCutsEnergy2RangeUA:  MATERIAL:  G4_WATER  PART: e+
  ENERGY CUT: 0.1 (MeV)  = RANGE CUT: 0.137686
```

5.6 User limits

User limits are the way Geant4 gives the user to limit the tracking of a particle. There are five types of user limits:

- Limit the step size
- Limit the track length
- Limit the time of flight
- Stop the particle when the kinetic energy is below a limit (and deposit its energy locally)
- Stop the particle when the expected range is below a limit (and deposit its energy locally)

Different user limits can be applied in Geant4 for different logical volumes, although there is the limitation that all particles must have the same user limits in the same logical volume.

In GAMOS a user can set the user limits through simple user commands and user limits can be set independently for different particle types in the same or different logical volumes. The set of commands to set user limits is

- **/gamos/physics/userLimits/setUserLimits USER_LIMITS_NAME LOGICAL_VOLUME_NAME PARTICLE_NAME MAX_STEP MAX_TRK_LENGTH MAX_TOF MIN_KIN_E MIN_RANGE**

  where USER_LIMITS_NAME is the name of the user limits (every user limits must have a name, so that new logical volumes and particles can be added with user commands later), MAX_STEP MAX_TRK_LENGTH MAX_TOF MIN_KIN_E and MIN_RANGE are the values of the five user limit types described above. If you want to set only one user limit type you can use one of the following commands:

- **/gamos/physics/userLimits/setMaxStep USER_LIMITS_NAME LOGICAL_VOLUME_NAME PARTICLE_NAME MAX_STEP**
- **/gamos/physics/userLimits/setMaxTrkLen USER_LIMITS_NAME LOGICAL_VOLUME_NAME PARTICLE_NAME MAX_TRK_LENGTH**
- **/gamos/physics/userLimits/setMaxTOF USER_LIMITS_NAME LOGICAL_VOLUME_NAME PARTICLE_NAME MAX_TOF**
- **/gamos/physics/userLimits/setMinEKin USER_LIMITS_NAME LOGICAL_VOLUME_NAME PARTICLE_NAME MIN_KIN_E**
- **/gamos/physics/userLimits/setMinRange USER_LIMITS_NAME LOGICAL_VOLUME_NAME PARTICLE_NAME MIN_RANGE**

There is another command in GAMOS that serves to set the minimum range user limit using a distance value, but internally it is applied as a minimum kinetic energy limit. This permits to use range values but avoids the lengthy process of converting the kinetic energy at each step into range. For this you can use the command:

- **/gamos/physics/userLimits/setMinEKinByRange USER_LIMITS_NAME LOGICAL_VOLUME_NAME PARTICLE_NAME MIN_RANGE**

Once a user limit is set, you may apply it to a new pair of logical volume and particle type with the following command:

- **/gamos/physics/userLimits/addLVAndParticle USER_LIMITS_NAME LOGICAL_VOLUME_NAME PARTICLE_NAME**

  for the list of particle names in GAMOS, see section **Particle names**.

5.7 Automatic optimisation of cuts

The production cuts and user limits are powerful methods to tune your simulation so that you can save a lot of CPU time by not tracking the particles that are not going to contribute to your results. Neverthelss the tuning of cuts is usually a long and difficult task. We have developed in GAMOS a method to help the user to obtain the optimal value of the production cuts and user limits for her/his application in a single job. We describe here the basic idea of the method and then in the corresponding sections the details of its implementation for different cases.

To use this utility you have to define in a clear way which are the results you don't want to change when cuts change, for example

- Number of particles reaching a region

- Dose distribution
- Number/Energy/spatial distribution of hits
- Shower shape in a volume
- .....

For each track that contributes to your result GAMOS stores all its history: for the track itself and each of its ancestors stores energy, range, region, process and particle type. In the case of production cuts this information is stored at particle creation. In the case of user limits this information is stored at each step (for parents only the steps before the creation of the interesting track).

At the end of run you can can get for each region/process/particle a list of all the ranges or energies of all the particles created. Then you can easily know if you apply a cut how many particles are below it.

This is valid if you are only interested in counting how many particles you lose. For other cases another approach should be used. For example if you want to check how your dose distribution changes, you can build a set of filters, each one with a set of cut values. These filters do not really cut the particles but only serve to tag a particle if it would have been killed by the set of cuts in the filter. Therefore, for each track that contributes to your results, GAMOS can check if it (or any of its ancestors) would have been killed by each of these sets of cut values. If you build your results N times, each one using only those tracks that pass one filter, you can compare each result to see how it changes with each set of cuts.

5.7.1 Automatic determination of production cuts for an accelerator simulation

The method used in GAMOS to determine the best production cuts is based on what we can call an 'inverse reasoning'. We count each particle that reaches a given Z plane (corresponding to the phantom surface) and we calculate first the range of the particle in the region where it is created. Then we can know that if we put a range cut in that region smaller than the calculated range, that particle would not reach our target plane. We also compute the range of the mother particle in the region where it was created and the same consecutively for all the ancestors. We know then that if we set in any of the regions where each of the ancestor particles are created a cut smaller than the corresponding range, we would stop the chain of particles and therefore we would have no particle in the target plane. After running a big number of tracks we can know for each particle type and for each region which is the biggest range we can put if we do not want to lose any particle. Indeed we may allow to lose a small amount of particles if this speeds up

our simulation. To know easily which is the biggest cut you can use to lose less than a given percentage of particles, GAMOS provides a set of plots (one per each particle type and per each region) and a simple script to get automatically the cut values.

One warning is due here: as mentioned above when a track reaches the target, its range fills a histogram, but also the range of all the ancestors of this track. It may happen then that when you set a certain cut and the abovementioned script gives you how many tracks would be killed, more than one killed track correspond to the same track reaching the target (i.e., with a cut you kill the track that reaches the target and the parent track). Therefore you might have an overcounting of the number of tracks killed by a cut. To avoid this the total number of tracks (the last lines of output) is not computed as the sum of tracks in the region. This number uses a histogram that contains only one entry per track reaching the target, the one corresponding to the track with the smallest range. If you want to set a different cut for each region and are worried for this double counting, you may have a look at the histogram named "trackInfos per Track in target", that plots per each track reaching the target how many track informations are kept in the histograms. Another useful histogram for this case may be the 2D histogram "trackInfo Region vs trackInfo Region", that plots all the region number of all the pairs of track informations that correspond to the same track reaching the target (you can get a list of which region number corresponds to which region at the end of the standard output file).

Although as mentioned above, the production cuts are only necessary for ionization and bremsstrahlung processes, this method allows to extend the production cuts to other processes. To do this we provide the abovementioned numbers and plots separately for each process so that the cuts can be automatically set in an easy way. If you want to apply the same cuts to all processes you can use the GAMOS command

/gamos/GMphysics/applyCutsToAllProcesses

that will instantiate an object of type G4EmProcessOptions and invoke the method SetApplyCuts(true).

To use this utility in GAMOS it is only needed to add this command in your script: **/gamos/userAction GmProdCutsStudyUA PETCutsStudyFilter**

that will use as target condition that a track enters a sensitive detector

/gamos/userAction GmProdCutsStudyUA RTCutsStudyFilter

or that will use as target condition that a track reaches a plane perpendicualr to the Z axis defined with the parameters

/gamos/setParam RTCutsStudyFilter:PlaneZ ZPOS
/gamos/setParam RTCutsStudyFilter:PlaneXDim XDIM
/gamos/setParam RTCutsStudyFilter:PlaneYDim YDIM

These commands will produce at the end of run a table and a histogram

file with the needed information. The table will contain the minimum range that can be applied for each region/particle/process not to lose any track reaching the target, and it will look like this

```
 %%%%% PRODUCTION CUTS STUDY RESULTS
GmProdCutsStudyUA: REGION= DefaultRegionForTheWorld PARTICLE=
 gamma PROCESS= ALL MIN RANGE= 353161.38
GmProdCutsStudyUA: REGION= DefaultRegionForTheWorld PARTICLE=
 gamma PROCESS= eBrem MIN RANGE= 353161.38
```

To get the cuts values for not losing a given percentage of particles in the target plane you can execute the ROOT script that can be found at GamosCore/GamosPhysics/GamosCuts/getProdCutsEffect.C :

```
 root -b -p -q .x getProdCutsEffect.C++\(\"prodcuts.root\",percentage\)
```

and look at the last lines of output, those that contain the word 'FINAL', like the following ones

```
PARTICLE= e+ FINAL= 17 / 19
PARTICLE= e- FINAL= 72 / 34185
PARTICLE= gamma FINAL= 72 / 34184
```

5.7.2 Automatic determination of production cuts for a dose in a phantom simulation

The use of production cuts in a dose computation may introduce a bias when a particle is killed (and then its energy is deposited locally) and it has enough energy to reach the next phantom voxel, or enough energy to create a particle that reaches the next phantom voxel (this happens mainly for electrons creating gammas, which have a much higher range).

To calculate automatically the best production cut, that is the one that gives the smallest CPU while biasing the dose computation a minimal amount, GAMOS uses an inverse reasoning. For a given set of cuts for electron and gamma it does not apply them but tags the particles that would have been killed by them. It also tags the voxel in which the particle is produced and then it computes all the dose deposited by the tagged particle or any of its children in a voxel that is not the same as the tagged voxel.

To use this utility in GAMOS you just have to associate to your dose scorer a filter of type GmProdCutOutsideVoxelFilter, passing to it as arguments the gamma cut and the electron cut, like in the following example

/gamos/scoring/addFilter2Scorer ProdCutFilter GmProdCutOutsideVoxelFilter PDDscorerPC10.1. 10.*mm 1.*mm

You should add another scorer without filter to get the total dose. After running your job with as many scorer-filter combinations as you like, you can look at the total dose deposited by each scorer and compare it with the total dose. It may happen that the dose lost with certain cuts, despite being a small proportion of the total dose, is distributed in a different manner than the total dose, introducing some bias in some region that you consider not acceptable. To check in detail the dose produced with a certain filter you can add a scorer printer of type RTPSPDoseHistos, that will produce several histograms of the dose (PDD, X & Y profiles, dose, dose-volume):

/gamos/scoring/addPrinter2Scorer PDDhistoPC10.1. RTPSPDoseHistos PDDscorerPC10.1.

The name of the printer will be passed to the name of the file containing the histograms.

5.7.3 Automatic determination if user limits for an accelerator simulation

The method used in GAMOS to determine the minimum range user limits is similar to the one used to determine the best production cuts. The main difference is that when a track reaches the target we do not have to look at the range it had when created, but at the range it had in every step. This is because even if we want the minimum step, the track may have crossed several regions and the smallest range may not correspond to the last step. What we do nevertheless is only consider the last step when there are a set of contiguous steps in the same region. Also for the ancestor tracks we have to store the information of each step, starting of course with the one when the track that reached the target (or its n-th ancestor if we are looking at the (n+1)-th ancestor) was created.

The same warning as for the production cuts should be mentioned here, but in this case it is more than a mere warning: when a track reaches the target, we accumulate one-track information of the last step in each region, for each of the ancestor tracks. Therefore it is very likely that there are more than one track information per track reaching the target, and therefore there will be overcounting of the number of tracks killed by a cut. As for the production cuts you should keep an eye on this and in any case use only the statistics.

To use this utility in GAMOS it is only needed to add the command in your script:

/gamos/userAction GmMinRangeLimitsStudyUA RTCutsStudyFilter

what will produce at the end of run a table and a histogram file with the needed information.

```
root -b -p -q .x getMinRangeCutsEffect.C++\(\"prodcuts.root\",percentage\) |& tee out
```

and look at the last lines of output, those that contain the word 'FINAL', like the following ones

```
PARTICLE= e+ FINAL= 17 / 19
PARTICLE= e- FINAL= 72 / 34185
PARTICLE= gamma FINAL= 72 / 34184
```

5.7.4 Automatic determination for a dose in phantom simulation

The method used in GAMOS to determine the minimum range user limits is similar to the one used to determine the best production cut.

To use this utility in GAMOS you just have to associate to your dose scorer a filter of type GmProdCutOutsideVoxelFilter, passing to it as arguments the gamma cut and the electron cut, like in the following example

/gamos/scoring/addFilter2Scorer ProdCutFilter GmMinRangeCutOutsideVoxelFilter PDDscorerPC10.1. 10.*mm 1.*mm

After running your job with as many scorer-filter combinations as desired, you can look at the total dose deposited and compare it with the dose with very small cuts. It may happen that the dose with certain cuts, despite being a small number, is distributed in a different manner than with very small cuts, introducing some bias that you consider not acceptable. To check in detail the dose produced with a certain filter you can add a scorer printer of type RTPSPDoseHistos, that will produce several histograms of the dose (PDD, X & Y profiles, dose, dose-volume):

/gamos/scoring/addPrinter2Scorer PDDhistoPC10.1. RTPSPDoseHistos PDDscorerPC10.1.

The name of the printer will be passed to the name of the file containing the histograms.

5.7.5 Range rejection

The range rejection technique consists on killing a particle at creation and depositing all its energy locally if it is not going to leave the current volume. To do this in practical terms, the particle is killed if the range is smaller than the distance to the volume boundary (although it would have a chance to exit the volume, this chance is considered negligible).

5.7.6 Automatic determination for an accelerator simulation

You can analyze which would be the effect of applying this technique by using the same user action as for the production cuts:

/gamos/userAction GmProdCutsStudyUA RTCutsStudyFilter

It will produce a table with the number of tracks that would be killed by the range rejection and would not reach the target (the tracks themselves

or any of their children). As for the production cuts, they are printed by region, by particle and by creator process type. To get a closer inside on this technique several plots are produced (one per each region, each particle and each creator process) representing the logarithm of the difference safety-range vs the logarithm of the range. To distinguish the cases where the range is bigger than the safety (no range rejection) those cases are plotted in the bins -15 to -5, while the cases where the range is smaller than the safety occupy the bins -5 to 5 (if there is a case, not likely for a radiotherapy simulation) where the log10(fabs(safety-range)) is smaller than -5 (of course before the -10 substraction), it is set to -5 and if is bigger than 5 it is set to 5.

Chapter 6

User Actions

Geant4 user actions are the way the user can interact with a job at the beginning/end of each run, beginning/end of each event, beginning/end of each track or at each step. The user can write a class, inheriting from one of the Geant4 user action abstract classes, and Geant4 will take care of calling the user code.

The GAMOS user actions classes provide all the functionality of the Geant4 classes, and also allow the user to define several user actions of the same type in the same job and to define a class that inherits from several user action types at the same time. Moreover, as they are plug-in's, the user can activate them by means of a user command.

User actions can be associated to filters or classifiers, as explained below.

Several functionalities are already implemented in GAMOS as user actions (like the example histograms, the event classifiers, ...) and you may use them as examples for creating your own one.

See the Geant4 user manual for a more detailed explanation of the user actions.

6.1 Adding a filter

One or several filters can be added to a user action by simply adding their names in the user command where an action is selected. See section on *Filters* to get a list of the available filters in GAMOS and how to add parameters to a filter.

If the filter you are using is a step filter and your user action is a stepping user action, the method SteppingAction will only be called if all the filters accept the step. If the filter you are using is a track filter it affects the callings to the **PreUserTrackingAction** and **PostUserTrackingAction** for tracking actions and **ClassifyNewTrack** for stacking actions.

For example

/gamos/userAction GmTrackHistosUA GmGammaFilter

will only produce histograms for tracks whose particle is a gamma.

6.2 Adding an indexer (= classifier)

One or several classifiers can be added to a user action by simply adding their names in the user command where an action is selected. See section on *Classifiers* to get a list of the available classifiers in GAMOS and how to add parameters to a classifier.

It is up to the concrete user action to use the classifiers or not.

6.3 Creating your GAMOS user action

To create your class you have to inherit from one or several of the GAMOS user actions: **GmUserRunAction, GmUserEventAction, GmUserTrackingAction, GmUserSteppingAction, GmUserStackingAction**.

Then you implement the same methods as for the Geant4 user actions:

- **GmUserRunAction**:
  - virtual void BeginOfRunAction(const G4Run* aRun);
  - virtual void EndOfRunAction(const G4Run* aRun);
- **GmUserEventAction**:
  - virtual void BeginOfEventAction(const G4Event* anEvent);
  - virtual void EndOfEventAction(const G4Event* anEvent);
- **GmUserTrackingAction**:
  - virtual void PreUserTrackingAction(const G4Track* aTrack);
  - virtual void PostUserTrackingAction(const G4Track* aTrack);
- **GmUserSteppingAction**:
  - virtual void UserSteppingAction(const G4Step* aStep);
- **GmUserStackingAction**:
  - virtual G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track* aTrack, G4ClassificationOfNewTrack oldClassificationx) = 0;
  - virtual void NewStage();
  - virtual void PrepareNewEvent();

Finally you have to transform your class into a plug-in. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmUserActionFactory.

If you define twice the same user action in your command file, you will get a warning message, but it will be executed twice.

Chapter 7

Sensitive Detector and Hits

7.1 Attaching a sensitive detector to a volume

The sensitive detector class in Geant4 has the task of creating hits (deposits of energy) each time a track traverses a sensitive volume and loses some energy. You can write your own sensitive detector class, inheriting from G4VSensitiveDetector, that produces your own hits, and attach it to any volume in your geometry. However GAMOS provides some utilities to make this easier and without the need of C++ programming or a detailed knowledge of how the sensitive detector and hits work in Geant4.

GAMOS provides several predefined sensitive detectors, that you can find in **GamosCore/GamosSD**:

- **GmSDSimple**. It is a general-purpose class, that produces hits with the position at the centre of the detector. The identification of each detector unit is done as explained in the sub-chapter *Identifying each sensitive detector copy.*
- **GmSDSimpleExactPos**. It is similar to **GmSDSimple** but the hits position is the centroid of the energy depositions of the different tracks that produced it (weighted by their energy).
- **GmSDOpticalPhoton**. This class inherits from **GmSDSimple**, but only produces hits if the process that defined the step is **"OpAbsorption"**

There are other classes that serve to make a virtual segmentation, when you have a big sensitive volume that you want to segment in different pieces, although you have not segmented it in your geometry. The classes currently implemented are

- **GmSDVirtSegmentedSphereThetaPhi**. It divides a sphere into cubes of equal size in R-phi and R-theta

- **GmSDVirtSegmentedSphereRThetaPhi**. It divides a sphere into cubes of equal size in R, R-phi and R-theta

To attach a GAMOS sensitive detector to a logical volume in your geometry, you have to use the command

/gamos/SD/assocSD2LogVol SD_CLASS SD_TYPE LOGICAL_VOLUME_NAME

The SD_CLASS has to be one of the sensitive detector types described above (or any other that you create). The SD_TYPE serves to differentiate your different sensitive detectors, so that you can later apply different properties to them (e.g. different energy resolutions) [1]. The LOGICAL_VOLUME_NAME is the name of the G4LogicalVolume in your geometry that you want to make sensitive. You may repeat this command with different logical volumes.

7.2 Building your sensitive detector with C++ code

To build a new sensitive detector you can do it the usual Geant4 way, that is, inheriting from G4VSensitiveDetector (see example in [10]). After that you have to transform it into a plug-in. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmSensDetFactory.

You may also choose to inherit your sensitive detector from the GAMOS class **GmVSD**, so that you can profit easily from its extra functionality. Namely, it will take care of applying the energy and time resolutions by detector type, accumulating the energy of different energy depositions if they happen in the same detector unit, taking into account the measuring time and the dead time and invoking the digitization and reconstruction of hits at the end of the event. There are at least two methods that you have to define in your class:

```
virtual long int GetDetUnitID( G4Step* aStep );
```

Serves to define the logic you want to apply to give a different number to each detector unit (for example to each individual crystal)

```
virtual void CalculateAndSetPosition( GmHit* hit,
    G4Step* aStep ) = 0;
```

Serves to define the way you want to define the position of the hit.

[1] You may see the PET application for an illustration of this

7.3 Hits

If you have activated any of the GAMOS sensitive detectors, each time a track deposits some energy in any copy of the selected logical volume (you can indeed select several volumes by repeating the command) a GmHit will be created. If the energy deposition happens in the same volume copy (a real one, or a virtual one in case of a virtually segmented sensitive detector) than a previous one in the same event, a new hit is not created, but the existing hit is updated, adding to it the new energy deposition.

The GmHit stores the following variables:

- **long int theDetUnitID;** Identification of the touchable
- **G4int theEventID;** Event number
- **G4double theEnergy;** Total energy
- **G4double theTimeMin;** Minimum time of energy depositions[2]
- **G4double theTimeMax;** Maximum time of energy depositions
- **G4ThreeVector thePosition;** Position (it is defined in the Sensitive detector class; it can be the centre of gravity of the energy depositions, the centre of the volume, ...)
- **std::set<G4int> theTrackIDs;** The list of track numbers
- **std::set<G4int> theOriginalTrackIDs;** The list of original track numbers (a track is called original if it is a gamma, electron or positron and it is a primary particle, or if it is a gamma created in an original positron annihilation)
- **std::vector<GmEDepo*> theEDepos;** The list of energy depositions. A **GmEDepo** contains the energy and position of each step.
- **G4String theSDType;** The type of sensitive detector

7.4 Detector effects

7.4.1 Energy and time resolutions

If you use one of the GAMOS sensitive detectors or you inherit your own one from **GmVSD** you can smear automatically the energy and time of the hits for each detector type with a gaussian given by the value of the parameters

/gamos/setParam SD:EnergyResol:*SDTYPE*

[2] This is the time considered when you call the method *GetTime()*. You may also invoke explicitly *GetTimeMin()* or *GetTimeMax()*

/gamos/setParam SD:TimeResol:*SDTYPE*[3]

If you have a resolution function that is not gaussian you may implement it by creating a new sensitive detector class inheriting from **GmVSD** (or **GmSDSimple** if you don't want to change the logic to define the detector unit IDs) and overwrite the methods:

```
virtual G4double SmearEnergy( G4double energy, G4double enerResol );
virtual G4double SmearTimeMin( G4double time, G4double timeResol );
```

7.4.2 Detector measuring time

A detector has a finite time resolution, so that it is not able to distinguish hits that come from different events when their time is close.

This effect is simulated in GAMOS with the help of the GmHitsEventMgr class. This class accumulates the hits of several events and for each event builds up a list of good hits, i.e. those that have a time after the event time minus the measuring time[4]. You can define the value of the measuring time for each detector type (*SDTYPE*) with the parameter

/gamos/setParam SD:MeasuringTime:*SDTYPE*

that takes a default value of 10 ns.

7.4.3 Detector dead time

A detector takes a finite time to transform an energy deposit into an electronic signal, and during that time it is *dead* and cannot account for any other energy deposition.

This effect is simulated in GAMOS also with the help of the GmHitsEventMgr class. It holds a list of the dead sensitive detectors, i.e. those that have produced a hit in a time prior than the current time minus the dead time. You can define the value of the dead time for each detector type with the parameter

/gamos/setParam SD:DeadTime:*SDTYPE*

that takes a default value of 100 ns.

The dead time affects by default all detectors in a block. This means that if a detector is dead it considers that all detectors that are placed in the same mother are also dead (the usual behaviour for example in a PET detector, where all crystals in a block share the readout). You can tell GAMOS to consider dead only the crystal itself by setting the parameter

/gamos/setParam SD:DeadTimeType:*SDTYPE* **byCrystal**

that by default takes the value **byBlock**

You also have the option to define your detector as paralizable (default) or non-paralizable by setting the parameter

[3] The time smeared is *theTimeMin*

[4] The event time is computed as the time of creation of the first particle in the event

/gamos/setParam SD:DeadTimeParalizable:*SDTYPE* **TRUE/FALSE**

In a non-paralizable detector, an event happening during the dead time since the previous event is simply lost, while in a paralizable detector, an event happening during the dead time since the previous one will not just be missed, but will restart the dead time.

7.5 Hits digitization and reconstruction

7.5.1 Hits digitization

The conversion of the hits into digital signals is very dependent on the detector. Therefore GAMOS just provides a general class, **GmVDigitizer**. The user may inherit her/his own digitizer from it and implement the two methods:

```
virtual std::vector<GmDigit*> DigitizeHits(const
    std::vector<GmHit*>&) = 0;

virtual void ClearDigits();
```

These two methods will be called automatically. The first one at the end of each event, to convert the hits in digits, and the second one at the beginning of each event, to clear the digits of the previous event.

You can then convert your digitizer into a GAMOS plug-in. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmDigitizerFactory. After this, you select it with the command **/gamos/digitizer MY_DIGITIZER**

7.5.2 Hits and digits reconstruction

The digital signals are usually treated so that they become "reconstructed hits", which contain sensible variables, like energy, time, ... This conversion is also very dependent on the detector, and therefore GAMOS just provides a general class, **GmVRecHitBuilderFromDigits**. There is also another class **GmVRecHitBuilderFromHits**, which serves in case the user wants to transform the hits into reconstructed hits directly.

The user may inherit her/his own reconstructed hit builder from it and implement the two methods:

```
virtual std::vector<GmRecHit*> ReconstructDigits(const
    std::vector<GmDigit*>&) = 0;

virtual void ClearDigits();
```

in the case of **GmVRecHitBuilderFromDigits**, or

```
virtual std::vector<GmRecHit*> ReconstructHits(const
    std::vector<GmHit*>&) = 0;

virtual void ClearDigits();
```

in the case of **GmVRecHitBuilderFromHits**.

These two methods will be called automatically. The first one at the end of each event, to convert the digits or hits into reconstructed hits, and the second one at the beginning of each event, to clear the reconstructed hits of the previous event.

You can then convert your reconstructed hit builder into a GAMOS plug-in. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmRecHitBuilderFactory. After this, you select it with the command **/gamos/recHitBuilder MY_RECHITBUILDER**

7.5.3 Examples of reconstructed hit builders

There are a few simple reconstructed hit builders implemented in GAMOS that serve to merge hits that are close to each other. The energy of the reconstructed hit is the sum of the hit energies, while the position is the weighted sum of the hit positions (weighted by the energy of each hit).

This can be useful, for example, for recovering the total energy of a photon when it has suffered a Compton scattering near the photoelectric interaction (the user has always the freedom to choose how near the other hits are); or for clustering together all the energy depositions of the particle shower produced by the electron following a photoelectric interaction.

The merging of this is always started by the hit that has bigger energy in each detector type (only hits in the same detector type are merged). Then the hits are looked one by one to check if they are close. Each time a hit is added, the centre is recalculated.

The reconstructed hit builders implemented are:

- **GmRecHitBuilderByDistance**. Two hits are merged if they are separated by a distance closer than the parameter

  /gamos/setParam SD:GmRecHitBuilderByDistance:HitsDistInRecHit

  that by default takes a value of 10 mm.

- **GmRecHitBuilderByBlock**. Two hits are merged if they are in sensitive detectors that belong to the same block, i.e. a volume whose parent volume is the same. To check if two hits belong to the same block, the detector unit ID (i.e. the identification of the touchable where the hits is located) , is used. Usually the detector unit ID is

built from the volume copy numbers of the volume ancestors (e.g. volume_copy_number + 100 * parent_volume_copy_number + 100*100*grandparent_volume_copy_number), and therefore two detector unit IDs are considered to belong to the same block if their division by a number given by the parameter

/gamos/setParam SD:GmRecHitBuilderByBlock:NShift

gives the same results. This parameter takes a default value of 100.

- **GmRecHitBuilder1to1**. Two hits are never merged, so that a reconstructed hit is built for each hit.

If you want any of these reconstructed hit builders to be active you can select one of them with the command

/gamos/recHitBuilder *RHITBUILDER_NAME*

where *RHITBUILDER_NAME* is one of the classes described above.

7.6 Identifying each sensitive detector copy

To identify each detector unit individually you have to give a different detector unit ID to each copy of your sensitive detectors (each G4Touchable in Geant4 terminology). GAMOS does this automatically for you. If you use the **GmSDSimple** class, it will give each SD copy a detector unit ID that will be the

copy number of physical volume + 100 * copy number of parent physical volume + 100*100* copy number of grandparent physical volume

You can change the number of ancestor particles to build the detector unit ID with the parameter

/gamos/setParam SD:DetUnitID:NAncestors

that takes a default value of 3.

You can also change the value of the "shift" that multiplies each ancestor copy number using the parameter

/gamos/setParam SD:DetUnitID:NShift

that takes a default value of 100.

7.7 Storing and retrieving hits

GAMOS provides yuo with the option of storing in a file the hits produced during a run, and reading them back in another run. Creating hits from track in the sensitive detector or reading them from a file will produce the same *in-memory* representation, therefore you can apply any of the above described effects and produce any histogram in the same way independently on how the hits are produced.

This can serve you for example for doing a study on how much your results change with different energy resolutions: you produce the hits (with zero resolution) and store them, and in another run you can read applying a certain energy resolution; this way you spare the time to recreate them (what usually is several orders of magnitude slower than reading them).

To use this utility you just have to activate the user action

/gamos/userAction GmHitsWriteUA

The name of the file can be controlled with the parameter

/gamos/setParam Output:hits:FileNameOut MY_FILENAME

If this parameter is not found the name will be **hits.out**

You can read back the hits by activating the user action

/gamos/userAction GmHitsReadUA

The name of the file can be controlled with the parameter

/gamos/setParam Output:hits:FileNameIn MY_FILENAME

If this parameter is not found the name will be **hits.out**

As commented above, you can use all the other options in your script and just read the hits instead of generating them. But probably you do not want that new hits are created when you are reading them from a file; in this case, you should also activate the user action

/gamos/userAction GmKillAllUA

7.7.1 File format

The file to be written can be a text file or a binary file. The format depends on the value of the parameter

/gamos/setParam SD:GmHitsWriteUA:BinFile TRUE/FALSE

The text file contains a line for each hit with the following information:

- Event ID
- Sensitive detector type
- Detector unit ID
- Energy (MeV)
- Minimum time (ns)
- Maximum time (ns)
- Position X (mm)
- Position Y (mm)
- Position Z (mm)
- Number of original tracks One line per original track with track ID

- Number of tracks One line per track with track ID

The binary file contains the same information in the following format

- Event ID : float
- Sensitive detector type : char[10] (only first 10 characters are stored, if type has less than 10 characters blank spaces will be added at the end)
- Detector unit ID: unsigned long long
- Energy (MeV) : float
- Minimum time (ns) : unsigned long long
- Maximum time (ns) : unsigned long long
- Position X (mm) : float
- Position Y (mm) : float
- Position Z (mm) : float
- Number of original tracks : unsigned int One line per original track with track ID : unsigned int
- Number of tracks : unsigned int One line per track with track ID : unsigned int

7.8 Hits histograms

There are two user actions that provide a number of hits statistics. **GmHitsHistosUA** provides statistics about simulated hits, while **GmRecHitsHistosUA** provides statistics about reconstructed hits.

Chapter 8

Scoring

Geant4 provides several classes to score different quantities in the selected volumes. GAMOS provides all the Geant4 functionality through user commands and also some extra functionality that we describe in this section.

The first thing you should do to use GAMOS scoring is to create a multifunctional detector [15] and associate it with a list of logical volumes, with the user command

/gamos/scoring/createMFDetector MFD_NAME LOGICAL_VOLUME_NAME(s)

where **MFD_NAME** is the detector name that will be used later. and **LOGICAL_VOLUME_NAME(s)** is a list of logical volumes that you associate to this detector.

Then you should add to the detector one of the GAMOS scorers, or your own ones, with the user command

/gamos/scoring/addScorer2MFD SCORER_NAME SCORER_CLASS MFD_NAME SCORER_PARAMETERS

SCORER_NAME is a name you give to the scorer to be used later, **SCORER_CLASS** is one of the available scorer classes, **MFD_NAME** is one of the multifunctional detectors created above and **SCORER_PARAMETERS** are the extra parameters a scorer may need (see below for the description of the scorers). You may repeat this command to associate several scorers to the same detector.

To each of the defined scorers you can add a filter, to select for which track conditions the scoring will be done:

/gamos/scoring/addFilter2Scorer FILTER_NAME/CLASS SCORER_NAME

FILTER_NAME/CLASS is the name of a **GmVFilter** class or the name you gave to a filter built from a filter class by using the command **/gamos/filter** (see section on *Filters*) and **SCORER_NAME** is one of the scorers defined above.

You may repeat this command to associate several filters to the same scorer. See section on *Filters* for a description of the available filters and

how to create your own one.

Finally you may select the format of the scoring results by associating one of the available scorer printers for each scorer,

/gamos/scoring/addPrinter2Scorer PRINTER_NAME/CLASS SCORER_NAME

PRINTER_NAME/CLASS is the name of a **GmVPSPrinter** class or the name you gave to a printer built from a printer class by using the command **/gamos/printer** (see section on *Scorer printers* below) and **SCORER_NAME** is one of the scorers defined above.

If no printer is attached to a scorer, it will use the printer type **GmG4PSPrinterDefault**.

By default a different count is scored for each of the copies of the selected volumes with different copy number. This is managed by the scorer classifier **GmScorerClassifierBy1Ancestor**. The user can attach different classifiers to the different scorers so that the counts are done in different ways:

/gamos/scoring/assignClassifier2Scorer CLASSIFIER_NAME/CLASS SCORER_NAME

CLASSIFIER_NAME/CLASS is the name of a **GmVClassifier** class or the name you gave to a classifier built from a classifier class by using the command **/gamos/classifier** (see section on *Classifiers*) and **SCORER_NAME** is one of the scorers defined above.

The scoring is done by default taking into account the track weight, except for the scorers when it is explicitly mentioned (see scorers description). If you do not want to take weights into account you can switch them off with the command

/gamos/scoring/useTrackWeight SCORER_NAME FALSE

SCORER_NAME is one of the scorers defined above.

The quantities scored are given per event by default. If you want the score without dividing by the number of events, use the command

/gamos/scoring/printByEvent PRINTER_NAME FALSE

PRINTER_NAME is one of the printers defined above.

The error in the scored quantity per voxel is also calculated by default, using the following formula:

$$\sqrt{(SumW2 * nEvents - SumW * SumW)/(nEvents - 1)/nEvents}$$

;

where **nEvents** is the total number of events in the run, **SumW** is the sum of the scored quantity value times its weight (i.e. the scored quantity itself) and **SumW2** is the sum of squares of scored quantity value × weight. When the scoring is done per event, this sum of squares is done summing first all the values × weight belonging to all the particles of the same event and then

squaring this quantity. In this way the correlations between particles of the same event is properly taken into account. If the scoring is not by event, the error calculation uses the same formula, but the sum of squares is not done summing the contribution of the particles of the same event, but squaring each contribution individually.

Calculating the errors makes it necessary to store the square of the weights, increasing substantially the memory usage and CPU time. If you want to deactivate this option for a scorer, use the command

/gamos/scoring/scoreErrors SCORER_NAME FALSE

SCORER_NAME is one of the scorers defined above. You can substitute **FALSE** by TRUE if you want to activate back the error calculation.

As mentioned above the errors that are calculated taking into account the number of events. You have to be careful then if you set to off the option of scoring by event and keep on the option of calculating the errors. In the default GAMOS scorer printers, the errors are printed are relative, i.e. the error divided by the value, so no caution is necessary, but be careful if you define a printer yourself.

8.1 Scorer classes

All the available scorers in Geant4 are also available in GAMOS. The classes have been slightly changed to provide the extra functionality.

The scorers can be classified in the following types:

- **Track length scorers**
  - **GmG4PSTrackLength** The track length is defined by the sum of step lengths of the particles inside the cell (i.e., the volume where the scoring happens). A particle weight is not applied by default. There are two extra parameters, that are FALSE by default and can be set TRUE or FALSE: to multiply by the kinetic energy and to divide by the velocity. If the energy track flux is required then you should set them to TRUE FALSE. Alternatively to measure the flux per unit velocity then you should set them to FALSE TRUE. Finally to measure the flux energy per unit velocity then you should set them to TRUE TRUE.
  - **GmG4PSPassageTrackLength** The passage track length is the same as the track length in **GmG4PSTrackLength**, except that only tracks which pass through the volume are taken into account. This means that newly-generated or stopped tracks inside the cell are excluded from the calculation. A particle weight is not applied by default.
- **Deposited energy scorers**

- **GmG4PSEnergyDeposit** This scorer stores a sum of particles' energy deposits at each step in the cell.
- **GmG4PSDoseDeposit** In some cases, dose is a more convenient way to evaluate the effect of energy deposit in a cell than simple deposited energy. The dose deposit is defined by the sum of energy deposits at each step in a cell divided by the mass of the cell. The mass is calculated from the density and volume of the cell taken from the methods of G4VSolid and G4LogicalVolume.

- **Current and flux scorers**

  There are two different definitions of a particle's flow for a given geometry. One is a current and the other is a flux. In our scorers, the current is simply defined as the number of particles (with the particle's weight) passing through a certain surface or volume, while the flux takes the particle's injection angle to the geometry into account. The current and flux are usually defined at a surface, but volume current and volume flux are also provided.

  - **GmG4PSFlatSurfaceCurrent** Flat surface current is a surface based scorer. The present implementation is limited to scoring only at the -Z surface of a G4Box solid. The quantity is defined by the number of tracks that reach the surface. The user must choose a direction of the particle to be scored (as extra argument in **/gamos/scoring/addScorer2MFD**). The choices are IN, OUT or INOUT. Here, IN scores incoming particles to the cell, while OUT scores only outgoing particles from the cell. INOUT scores both directions. The current is normalized for a unit area if an extra second parameter is set to TRUE.
  - **GmG4PSCylinderSurfaceCurrent** Cylinder surface current is a surface based scorer, and similar to the **GmG4PSFlatSurfaceCurrent**. The only difference is that the surface is defined at the inner surface of a G4Tubs solid.
  - **GmG4PSSphereSurfaceCurrent** Sphere surface current is a surface based scorer, and similar to the **GmG4PSFlatSurfaceCurrent**. The only difference is that the surface is defined at the inner surface of a G4Sphere solid.
  - **GmG4PSPassageCellCurrent** Passage current is a volume-based scorer. The current is defined by the number of tracks that pass through the volume.
  - **GmG4PSFlatSurfaceFlux** Flat surface flux is a surface based flux scorer. The surface flux is defined by the number of tracks that reach the surface. The expression of surface flux is given by the sum of W/cos(t)/A, where W, t and A represent particle weight, injection angle of particle with respect to the surface

normal, and area of the surface. The user must enter one of the particle directions, as in **GmG4PSFlatSurfaceCurrent**.

- **GmG4PSCylinderSurfaceFlux** Cylinder surface flux is a surface based flux scorer, and similar to the **GmG4PSFlatSurfaceFlux**. The only difference is that the surface is defined at the inner surface of a G4Tubs solid.
- **GmG4PSSphereSurfaceFlux** Sphere surface flux is a surface based flux scorer, and similar to the **GmG4PSFlatSurfaceFlux**. The only difference is that the surface is defined at the inner surface of a G4Sphere solid.
- **GmG4PSCellFlux** Cell flux is a volume based flux scorer. The cell flux is defined by a track length (L) of the particle inside a volume divided by the volume (V) of this cell. The track length is calculated by a sum of the step lengths in the cell. The expression for cell flux is given by the sum of (W*L)/V, where W is a particle weight, and is multiplied by the track length at each step.
- **GmG4PSPassageCellFlux** Passage cell flux is a volume based scorer similar to G4PSCellFlux. The only difference is that tracks which pass through a cell are taken into account. It means that tracks generated or stopped inside the volume are excluded from the calculation.

- **In/Out behaviour** For the following scorers **GmG4PSCylinderSurfaceCurrent, GmG4PSCylinderSurfaceFlux, GmG4PSFlatSurfaceCurrent, GmG4PSFlatSurfaceFlux, GmG4PSSphereSurfaceCurrent, GmG4PSSphereSurfaceFlux, GmG4PSTrackCounter** you can make the scoring only for tracks that are entering, only for tracks that are exiting or both for tracks that are entering or exiting (default behaviour). To select among these three options you can add an extra parameter when defining the scorer that can be **In**, **Out** or **InOut**

- **Other scorers**

  - **GmG4PSMinKinEAtGeneration** This scorer records the minimum kinetic energy of secondary particles at their production point in the volume in an event. This primitive scorer does not integrate the quantity, but records the minimum quantity.
  - **GmG4PSNofSecondary** This class scores the number of secondary particles generated in the volume. A particle weight is not applied by default. The user can choose if the scoring is done for all types of particles (default) or only for a set of particles, by naming them as extra parameters.
  - **GmG4PSNofStep** This class scores the number of steps in the cell. A particle weight is not applied by default. If an extra

parameter is set to TRUE those steps with step length zero will not be taken into account.

- **GmG4PSCellCharge** This class scores the total charge of particles which have stopped or have been created in the volume, i.e. the tracks that enter count as +1 and the tracks that exit count as -1.
- **GmG4PSTrackCounter** This class scores the number of tracks in a cell.

- **for Event Biasing**

  Scoring for event biasing is a very specific use case whereby particle weights and fluxes through importance cells are required. The goals of the scoring technique are to:

  - appraise particle quantities related to special regions or surfaces,
  - be applicable to all "cells" (physical volumes or replicas) of a given geometry,
  - be customizable.

  A number of scorers have been created for this specific application:

  - **GmG4PSNofCollision** This scorer records the number of collisions that occur within a scored volume/cell.
  - **GmG4PSPopulation** This scores the number of tracks within in a given cell per event. A particle weight is not applied by default.
  - **GmG4PSTermination** This scores the number of tracks that are terminated in a given cell per event. A particle weight is not applied by default.

8.2 Filter classes

See section on *Filters and classifiers*.

8.3 Scorer printers

A scorer printers serves to select the format of the output of a scorer. These classes are unique to GAMOS, as Geant4 does not provide this functionality. As mentioned above, several printers can be associated to the same scorer. The general use printer scorers currently in GAMOS are the following:

- **GmPSPrinterDefault** Prints in the standard output the summary of scoring in the following format:

```
MultiFunctionalDet: MFD_NAME
PrimitiveScorer: SCORER_NAME
Number of entries= 5
 copy no.: 0  = 2.6625344e-18 +-(REL) 0.031622777 Gy
 copy no.: 1  = 8.6617421e-17 +-(REL) 0.011622713 Gy
 copy no.: 2  = 2.1034987e-17 +-(REL) 0.021166675 Gy
 copy no.: 6  = 5.9651155e-17 +-(REL) 0.01418326 Gy
 copy no.: 7  = 8.2850179e-17 +-(REL) 0.013747866 Gy
```

where **MFD_NAME** is the name of multifunctional detector and **SCORER_NAME** is the name of the scorer. The columns after **copy no.:** have the following meaning:

Index, scorer value, scorer error (relative, i.e. error/value), unit name.

Other scorer printers are provided for specific applications. See corresponding sections in this guide.

8.4 Classifier classes

See section on *Filters and classifiers*.

Chapter 9

Analysis (extracting information)

9.1 Using histograms

GAMOS supports several data analysis formats. The format is selected at run time by the user, so that the same C++ code can be used to write any format. In this GAMOS version there are two formats implemented, ROOT and CSV (Comma Separated Value). For the ROOT format, we refer you to the ROOT documentation [5]. The CSV format is explained below.

You can choose which format to use with the command

/gamos/analysis/fileFormat FORMAT

where **FORMAT** can be **ROOT, root, CSV, csv**

9.1.1 Histogram files common name

If you are running a job and you want to identify all your histogram files with a characteristic suffix, you may do it by defining the parameter

/gamos/setParam GmAnalysisMgr:FileNameSuffix SUFFIX

The name **SUFFIX** will be added at the end of all histogram names, before the file type (.root or .csv).

9.1.2 Histograms in CSV format

The CSV (Comma Separated Value) format is a simple text file where the values are separated by commas. The utility of this format is that it can be easily read by any analysis package (Excel, Origin, Matlab, ..) and converted to its own format.

The information written in GAMOS is the following:

- **Histograms 1D:** The first word is "HISTO1D", then the following info is dumped: his_name,number_of_bins,Xaxis_minimum,

Xaxis_maximum,bin_contents,number_of_entries,mean,RMS. The bin_contents is the list of entries in each bin. It has indeed number_of_binsX+2 numbers, as the first one is the underflow (entries below axis_minimum) and the last one is the overflow (entries above axis maximum).

- **Histograms 2D:** The first word is "HISTO2D", then the following info is dumped: his_name,number_of_binsX,Xaxis_minimum, Xaxis_maximum,number_of_binsY,Yaxis_minimum,Yaxis_maximum, bin_contents,number_of_entries,mean,RMS. The bin_contents is the bi-dimensional list of entries in each bin. It has indeed (number_of_binsX+2)*(number_of_binsY+
numbers, as the first row/column is the underflow (entries below axis_minimum) and the last row/column is the overflow (entries above axis maximum).

You can see an example of 1D histogram here:

```
"1D","example",10,0,1000,0,3,3,1,3,5,6,12,15,16,2,66,0,460.2,353.81
```

9.1.3 Using a common histogram class

Several histogram classes inherit from a class named **GmVHistoBuilder** to facilitate the creation of histograms and to provide a common interface for its use. These classes should invoke the method **SetHistoNames** passing to it a histogram name and a file name. The histogram name will be the prefix that all histograms will have on their name. The file name will be the prefix of the histogram file name.

As the histogramming files in GAMOS are user actions, they can be used together with filters and classifiers (see section on *Filters and classifiers*). The names of the filtes used will be added to the histogram name prefix, separated by a ':', and to the file name prefix, separated by a '_'. In a similar way the name of the classifier will be added to the file name and the name of each classifier index will be added to the histogram name prefix. For example, the class **GmStepHistosUA** invokes the method **SetHistoNames** passing to it as arguments **"step"** and **"GmStepHistsoUA"**, therefore the command

/gamos/userAction GmStepHistosUA GmSecondaryFilter GmClassifierByParticle

will produce a file named **step_GmSecondaryFilter_GmClassifierByParticle.root** and the histograms will have as prefix

GmStepHistosUA:GmSecondaryFilter:gamma
GmStepHistosUA:GmSecondaryFilter:e-
...

The **GmVHistoBuilder** class takes also care of building a base histogram number, multiple of 1,000,000, guaranteeing that it is not repeated if several histogram classes are used. The histogram classes may define their histograms by using the number as a base, provided that they add histogram numbers smaller than 1,000,000.

Another utility of the **GmVHistoBuilder** class is that it provides a set of parameters to define the number of bins, the minimum and the maximum limits of different histogram types (position, angle, energy, ...). The default values of these parameters can be overriden in the command script by using the **/gamos/setParam**

command with parameter names equal to the file name plus the type of parameter.
In the example above, the commands to change the histogram definitions would be

/gamos/setParam step_GmSecondaryFilter_GmClassifierByParticle:hENbins NBINS

/gamos/setParam step_GmSecondaryFilter_GmClassifierByParticle:hEMin MIN

/gamos/setParam step_GmSecondaryFilter_GmClassifierByParticle:hEMax MAX

The following parameters are defined (in parenthesis their default value)

- Energy type histograms:
  - hENbins (100)
  - hEMin (0.)
  - hEMax (10.*MeV)
- Position type histograms:
  - hPosNbins (100)
  - hPosMin (-200*mm)
  - hPosMax (200*mm)
- Angle type histograms:
  - hAngleNbins (100)
  - hAngleMin (0.)
  - hAngleMax (180.)
- Time type histograms:
  - hTimeNbins (100)
  - hTimeMin (0.)
  - hTimeMax (1.*ms)
- Number of steps type histograms:
  - hNStepNbins (100)
  - hNStepMin (0)
  - hNStepMax (100)
- Number of secondary particles type histograms:
  - hNSecoNbins (100)
  - hNSecoMin (0)
  - hNSecoMax (100)

9.2 User action utilities

These are a set of utilities that can be instantiated through user commands. As for any user action, filters can be assigned to them to select for which type of tracks they will be activated.

9.2.1 Counting the number of tracks and events

This utility prints a line every N events with the event number, the number of tracks in this event and the accumulated number of tracks in all events:

```
%%% EVENT 0 NTRACKS 4 TOTAL NTRACKS 4
%%% EVENT 1000 NTRACKS 6 TOTAL NTRACKS 4663
%%% EVENT 2000 NTRACKS 4 TOTAL NTRACKS 9440
```

Its main use is to inform the user of the progress of the job in interactive running. To activate this utility use the command:

/gamos/userAction GmCountTracksUA

The user can control the interval of events as well as the first event to start printing with the parameters:

/gamos/setParam GmCountTracksUA:EachNEvent NEV (10)

/gamos/setParam GmCountTracksUA:FirstEvent NEV (0)

This utility distinguishes for the ionisation and bremsstrahlung processes those cases when a secondary particle is emitted and those when the step is limited to assure a correct energy loss and multiple scattering but no secondary particle is emitted (it adds *_NoSeco* at the end of the process name).

9.2.2 Counting the processes

This utility prints four tables:

- At the beginning of run all the active processes for each particle type:

```
PROC_LIST e+ : Transportation
PROC_LIST e+ : annihil
PROC_LIST e+ : eBrem
PROC_LIST e+ : eIoni
PROC_LIST e+ : msc
PROC_LIST e- : LowEnBrem
PROC_LIST e- : LowEnergyIoni
PROC_LIST e- : Transportation
PROC_LIST e- : msc
...
```

- At the end of run how many times a process determined the step for each particle type:

```
PROC_COUNT e+ : Transportation = 31
PROC_COUNT e+ : annihil = 999
PROC_COUNT e+ : eBrem = 19
PROC_COUNT e+ : eIoni = 1439
PROC_COUNT e+ : msc = 788
PROC_COUNT e- : LowEnBrem = 46
PROC_COUNT e- : LowEnergyIoni = 2362
PROC_COUNT e- : Transportation = 18
PROC_COUNT e- : msc = 1036
PROC_COUNT gamma : LowEnCompton = 684
```

```
PROC_COUNT gamma : LowEnPhotoElec = 549
PROC_COUNT gamma : LowEnRayleigh = 60
PROC_COUNT gamma : Transportation = 5963
```

- At the end of run how many times a process was the creator of a particle for each particle type:

```
PROC_CREATOR_COUNT e+ : Primary = 1000
PROC_CREATOR_COUNT e- : LowEnCompton = 541
PROC_CREATOR_COUNT e- : LowEnPhotoElec = 549
PROC_CREATOR_COUNT e- : LowEnergyIoni = 140
PROC_CREATOR_COUNT e- : eIoni = 255
PROC_CREATOR_COUNT gamma : LowEnBrem = 33
PROC_CREATOR_COUNT gamma : LowEnPhotoElec = 235
PROC_CREATOR_COUNT gamma : annihil = 1998
PROC_CREATOR_COUNT gamma : eBrem = 19
```

- At the end of run how many particles of each type were created:

```
PART_LIST: e+ = 1000
PART_LIST: e- = 1485
PART_LIST: gamma = 2285
```

To activate this utility use the command:
/gamos/userAction GmCountProcessesUA

9.2.3 Counting the number of tracks in a volume

This utility prints a table with the number of tracks and steps in each of the geometry logical volumes.

```
COUNT_PARTICLES: arm  #steps= 178 #tracks= 4
COUNT_PARTICLES: block  #steps= 0 #tracks= 0
COUNT_PARTICLES: body  #steps= 248 #tracks= 4
COUNT_PARTICLES: crystal  #steps= 253 #tracks= 5
COUNT_PARTICLES: head  #steps= 58 #tracks= 3
COUNT_PARTICLES: leg  #steps= 289 #tracks= 2
COUNT_PARTICLES: mother  #steps= 406 #tracks= 12
COUNT_PARTICLES: ring  #steps= 38 #tracks= 1
```

Also a histogram file **countTracks.root/csv** is created with the plots of kinetic energy of the tracks in each volume. The minimum and maximum of the histogram X axis are defined by the parameters

/gamos/setParam GmCountTracksInVolumeUA:Emin ENER
/gamos/setParam GmCountTracksInVolumeUA:Emax ENER

the default value of **Emin** is **1.E-12*MeV** and of **Emax** is **1.E2*MeV**.

You ca also choose the number of steps with the parameter

/gamos/setParam GmCountTracksInVolumeUA:NSteps NSTEPS (100)

and also whether the axis of energies is logarithmic

/gamos/setParam GmCountTracksInVolumeUA:OptLogE LOG (1)

To activate this utility use the command:

/gamos/userAction GmCountTracksInVolumeUA

9.2.4 Killing all tracks

The action

/gamos/userAction GmKillAllUA

This action serves to kill all particles at the stacking action G4ClassificationOfNewTrack method, i.e. before they start being tracked. You may use it in combination with one or several filters to kill only the particles that are accepted by them. For example,

/gamos/userAction GmKillAllUA GmPrimaryFilter

will only kill the primary particles.

9.2.5 Histograms of track information

This is a set of histograms produced for each track. They are created by writing the command

/gamos/userAction GmTrackHistosUA

This class inherits from **GmVHistoBuilder** and invokes the **SetHistoNames** method with parameters **"track"** and **GmTrackHistosUA** .

The following histograms are produced (in parenthesis their name without prefix, and the parameter type)

- Energy of tracks at creation(" E initial")("E")
- Total energy lost (" E lost")("E")
- Total energy deposited (" E deposited")("E")
- Number of steps (" N steps")("NStep")
- Track length (" Track length")("Pos")
- Deviation in position, i.e. distance from the track end point to the line formed by the original position and the original momentum direction (" Deviation position")("Pos")
- Deviation in angle, i.e. angle between the original momentum direction and the final momentum direction (" Deviation angle")("Angle")
- Number of secondary tracks created (" N secondaries")("NSeco")

9.2.6 Histograms of step information

This is a set of histograms produced for each track step. They are created by writing the command

/gamos/userAction GmStepHistosUA

This class inherits from **GmVHistoBuilder** and invokes the **SetHistoNames** method with parameters **"step"** and **GmStepHistosUA** .

The following histograms are produced (in parenthesis their name without prefix, and the parameter type)

- Energy (" Energy")("E")
- Energy lost (" E lost")("E")
- Energy deposited (" E deposited")("E")

- Step length (" Step length")("Pos")
- Number of secondary tracks created (" N secondaries")("NSeco")
- Energy of secondary tracks created (" E secondaries")("E")
- X coordinate of postion (" Position X")("Pos")
- Y coordinate of position (" Position Y")("Pos")
- Z coordinate of position (" Position Z")("Pos")
- 2D radius coordinate of position (" Position R2")("Pos")
- 3D radius coordinate of position (" Position R")("Pos")
- phi coordinate of position (" Position phi")("Angle")
- theta coordinate of position (" Position theta")("Angle")
- Change in position (" Position difference")("Pos")
- Change in angle (" Angle difference")("Angle")
- Change in time (" Time difference")("Time")

9.2.7 Histograms of secondary track information

This is a set of histograms produced for each secondary track created at each track step. They are created by writing the command

/gamos/userAction GmTrackSecondaryHistosUA

This class inherits from **GmVHistoBuilder** and invokes the **SetHistoNames** method with parameters **"secondary"** and **GmTrackSecondaryHistosUA** .

The following histograms are produced (in parenthesis their name without prefix, and the parameter type)

- Energy of secondary tracks created (" E secondary")("E")
- Energy of primary track at interaction (" E primary")("E")
- Fraction of energy taken by secondary tracks created (" E secondary/E primary")("E")
- Angle between primary track pre step direction and secondary track direction (" Angle primary_pre-secondary")("Angle")
- Angle between primary track post step direction and secondary track direction (" Angle primary_post-secondary")("Angle")
- Change of angle of primary track when secondary track is created (" Angle change of primary")("Angle")

9.2.8 Event classification by interaction types

There is a utility in GAMOS that helps you in counting and classifying the tracks by the type of interactions they have suffered. You just have to create at each step a new **GmTrajPoint** and at the end of track pass this list to a **GmVSimuEventClassifier** that will return the classification.

You can see an example at **GamosCore/GamosAnalysis/src/GmHistosGammaAtSD.cc**, that we explain here in detail:

This class counts the type of interaction of the photons in the sensitive detectors of your geometry.

The first thing it does, at the *PreUserTrackingAction* is checking if the current track is an 'original' gamma. To do this it gets the help of the **GmCheckOriginalGamma** class, that classifies the gammas as

- 0: not an 'original' gamma
- 1: it is a primary particle, created at the beginning of the event
- 2: is is created at the annihilation of the positron (it is assumed that the positron is a primary particle)

At each step, the *UserSteppingAction* method checks that it is inside a volume declared as sensitive detector. In this case, it adds a new **GmTrajPoint** for this track, with all the information of the track at this moment (plus it adds at the beginning another point with the vertex information).

At the end of track, if it is an original gamma, it asks the class **GmClassifierByInteraction** to classify it based on the type and number of interactions. The method *Classify()* of this class returns an integer with the meaning: *100*100* Number of LowEnPhotoElec interactions + 100 * Number of LowEnCompton interactions + Number of LowEnRayleigh interactions.*

See PET section for more details on the concrete class **GmHistosGammaAtSD**

9.2.9 Table of tracks and steps

You may get a table of the number of tracks and steps by instantiating the user action

/gamos/userAction GmCountTracksAndStepsUA

It will produce a table with the number of tracks and steps in the whole run. You may get more details by using it with filters and classifiers. For example the command

/gamos/userAction GmCountTracksAndStepsUA GmClassifierByParticle

will produce a table similar to this one

```
COUNT_TRACKS_AND_STEPS: GmClassifierByParticle
COUNT_TRACKS: gamma = 100
COUNT_TRACKS: e- = 8
COUNT_TRACKS: e+ = 1
COUNT_TRACKS: ALL = 109
COUNT_NSTEPS: gamma = 399
COUNT_NSTEPS: e- = 29
```

```
COUNT_NSTEPS: e+ = 4
COUNT_STEPS: ALL = 432
```

9.2.10 Detailed report of where CPU time is spent

You may get a detailed report of where the CPU time is spent by instantiating the user action

/gamos/userAction GmTimeStudyUA CLASSIFIER_1 CLASSIFIER_2 ...

By selecting different classifiers you can get a report of the time spent by each particle, in each logical volume, in each energy bin, etc. (see section on *Classifiers*).

The table will have a format similar to the following one:

```
%%%%% TIMING RESULTS for timer GmTimeStudyUA
_GmClassifierByParticle_GmClassifierByKineticEnergy
e+/0.0001-0.001:   User=0  Real=0  Sys=0
e+/0.001-0.01:   User=0  Real=0  Sys=0
e+/0.01-0.1:   User=0  Real=0.02  Sys=0
e+/0.1-1:   User=0.28  Real=0.26  Sys=0.01
e+/1-10:   User=0.18  Real=0.18  Sys=0.02
e-/0.0001-0.001:   User=0.94  Real=1.12  Sys=0.12
e-/0.001-0.01:   User=17.13  Real=19.2  Sys=1.85
e-/0.01-0.1:   User=31.67  Real=35.96  Sys=3.31
e-/0.1-1:   User=54.81  Real=58.4  Sys=4.61
e-/1-10:   User=209.36  Real=226.51  Sys=15.73
e-/1e-05-0.0001:   User=0.06  Real=0.09  Sys=0
e-/1e-06-1e-05:   User=0.03  Real=0.02  Sys=0
e-/1e-07-1e-06:   User=0  Real=0  Sys=0
e-/1e-08-1e-07:   User=0  Real=0  Sys=0
gamma/0.001-0.01:   User=8.49  Real=9.29  Sys=0.79
gamma/0.01-0.1:   User=14.55  Real=15.64  Sys=0.92
gamma/0.1-1:   User=33  Real=35.47  Sys=2.15
gamma/1-10:   User=8.99  Real=9.35  Sys=0.49
```

that can be obtained with the command

/gamos/userAction GmTimeStudyUA GmClassifierByParticle GmClassifierByKineticEnergy

The time in each category is the time counted at each step. Exactly it is the counting from the beginning to the end of the method G4SteppingManager::Stepping(). To do this without modifying the Geant4 class, the class GmTimeStudyMgr inherits from G4VSteppingVerbose and it is set as the stepping verbose class, substituting the class G4SteppingVerbose class, that is the one that controls the verbosity of the command /tracking/verbose . This means that this command will have no effect and if you want to use it you should do with the parameter

/gamos/setParam GmTimeStudyUA:G4VerboseLevel VERB

You may observe that the time summmed over all the categories is smaller than the run time given by the command /run/verbose 1. This is because the time is only the time spent at the method mentioned above,

which does not take into account the initialisation and termination times of each track, event and run.

9.3 Creating your own histograms

When you write your histogram you just have to take care of creating and filling it. The class GmAnalysisMgr will take care of automatically writing it in the file format you chose.

To use GmAnalysisMgr you have to instantiate it in your histogram class, passing to it the name of your file (you may use the same name for several of your histogram classes or different ones):

GmAnalysisMgr* myAnaMgr = GmAnalysisMgr::GetInstance("MY_FILENAME")

There are four types of histograms currently supported by GAMOS: 1-dimensional, 2-dimensional, profile 1-dimensional and profile 2-dimensional [1]. To create your histogram and register it to GAMOS you have to create it with a line like:

myAnaMgr->CreateHisto1D(HNAM,NBINS,MAXBIN, MINBIN,HIS_NUMBER);

myAnaMgr->CreateHisto2D(HNAM,NBINSX,MAXBINX, MINBINX,NBINS,MAXBINY,MINBINY,HIS_NUMBER);

myAnaMgr->CreateHistoProfile1D(HNAM,NBINS,MAXBIN ,MINBIN,HIS_NUMBER);

myAnaMgr->CreateHistoProfile2D(HNAM,NBINSX, MAXBINX,MINBINX,NBINS,MAXBINY,MINBINY, HIS_NUMBER);

The last argument is the histogram number, that can be later used to retrieve this histogram from any method in any class. If you don't set it, GAMOS will assign it automatically starting from 1.

Once a histogram is registered you can get a pointer to it by asking GmAnalysisMgr for a histogram by its number or its name:

myAnaMgr->GetHisto1(HIS_NUMBER)->Fill(value)
myAnaMgr->GetHisto2(HIS_NUMBER)->Fill(value)
myAnaMgr->GetHistoProfile1(HIS_NUMBER)->Fill(value)
myAnaMgr->GetHistoProfile2(HIS_NUMBER)->Fill(value)
or similarly if you want to retrieve by the histogram name.

[1] A profile histogram sets the value of a bin as the average of all entries in that bin, and the error as the RMS of these entries

Chapter 10

Filters and classifiers

10.1 Filters

A filter is a class that receives an G4Step or a G4Track and accepts it or not depending on some given criteria. A GAMOS filter has therefore two main methods

- **AcceptStep(const G4Step*): receives a G4Step pointer and decides to return true or false**
- **AcceptTrack(const G4Track*): receives a G4Track pointer and decides to return true or false**

A filter may implement the two methods or only one of them. If the AcceptStep method is not implemented by a filter, the AcceptStep method from the base class is invoked and it calls the AcceptTrack method passing to it the G4Track corresponding to the G4Step. If the AcceptTrack method is not implemented by a filter, the method from the base class returs true.

Filters can act on user actions or scorers. If one or several filters are set to act on a user action, the PreUserTrackingAction, PostUserTrackingAction and ClassifyNewTrack methods will only be invoked if the AccepTrack method of every filter returns true, and UserSteppingAction method will only be invoked if the AccepStep method of every filter returns true. For details on filters acting on scorers see the section on *Scorers*. The use of user actions and scorers together with filters is a powerful means to obtain very detailed information on the simulation through simple user commands. See the tutorial on *Histograms and scorers* for more details on this.

Several filters need some extra parameters (see list of filters below) that control their behaviour. To use these filters they have to be declared first with the following command

/gamos/filter FILTER_NAME FILTER_CLASS PARAMETER_1 PARAMETER_2 ...
where FILTER_NAME is the new name you want to give to a filter to attach it to a user action of a scorer, FILTER_CLASS is the name of the filter class, and PARAMETER_1 PARAMETER_2 ... are the values of

the parameters that the filter needs. Those filters that do not need any parameter, can be assigned directly to a user action or a scorer without giving them a new name: the name will be the one of the filter class.

To attach one or more filters to a user action you put them after the user action name in the command line that selects it

/gamos/userAction USER_ACTION FILTER_NAME/CLASS
or you can use filters to act on a scorer with the command

/gamos/scoring/addFilter2Scorer FILTER_NAME/CLASS SCORER_NAME

Filter are plug-in's, so that a user an create her/his own filter and select it with a user command. To learn how to do this, see the instructions in the section *Creating your plug-in*, **using the GmFilterFactory, or the** *Histograms and scorers* **tutorial.**

10.1.1 Simple filters

The list of available filters can be obtained by typing SealPluginDump on your terminal window and looking for the word Filter . The list of simple filters in the current GAMOS version is the following:

- **GmGammaFilter: accepts a track if the particle is a gamma**
- **GmElectronFilter: accepts a track if the particle is an electron**
- **GmPositronFilter: accepts a track if the particle is a positron**
- **GmElectronOrPositronFilter: accepts a track if the particle is an electron or positron**
- **GmEMParticleFilter: accepts a track if the particle is of electromagnetic type (gamma, electron or positron)**
- **GmParticleFilter: accepts a track if the particle is in the list of particles given as extra arguments (see the list of particle names in the section** *Using particle names*.
- **GmChargedFilter: accepts a track if the particle is charged.**
- **GmNeutralFilter: accepts a track if the particle is neutral**
- **GmPrimaryFilter: accepts a track if it is a primary (it does not come from another track)**
- **GmSecondaryFilter: accepts a track if it is a secondary (it comes from another track)**
- **GmKineticEnergyFilter: accepts a track if its kinetic energy is between the two values given as extra arguments. For steps it considers the energy at the beginning, that is the G4PreStepPoint energy**
- **GmPostKineticEnergyFilter: accepts a track if its kinetic energy is between the two values given as extra arguments. For steps it considers the energy at the end, that is the G4PostStepPoint energy**

- **GmVertexKineticEnergyFilter: accepts a track if its vertex kinetic energy (the energy at creation) is between the two values given as extra arguments**
- **GmDepositedEnergyFilter: accepts a step if the deposited energy is between the values given by the two extra parameters. It does not implement the AcceptTrack method**
- **GmInitialRangeFilter: accepts a track if its range at creation is between the two values given as extra arguments**
- **GmRangeFilter: accepts a track if the range is between the values given by the two extra parameters**
- **GmStepNumberFilter: accepts a step if the step number is between the values given by the two extra parameters. It does not implement the AcceptTrack method**
- **GmNumberOfSecondaries: accepts a step if the number of secondaries created is between the values given by the two extra parameters. It does not implement the AcceptTrack method**
- **GmProcessNameFilter: accepts a step if the process name that defined it is in the list given as extra arguments. It does not implement the AcceptTrack method**
- **GmParticleProcessFilter: accepts a step if the particle and the process that defined the step are in the list given as extra arguments. The arguments must be provided as a list of pairs particle name - process name. It does not implement the AcceptTrack method**
- **GmCreatorProcessFilter: accepts a step if the process that defined it is in the list given as extra arguments. It does not implement the AcceptTrack method**
- **GmFilterFromClassifier: accepts a step if the classifier given as first parameter returns a value equal to the second parameter. It does not implement the AcceptTrack method**

10.1.2 Filters of filters

There are another set of filters that receive as argument one or several filters and act on them. The list of composed filters in the current GAMOS version is the following:

- **GmORFilter: returns true if one of the filters returns true**
- **GmXORFilter: returns true if one and only one of the filters returns true**
- **GmANDFilter: returns true if every filter returns true**
- **GmHistoryFilter: returns true if all the filters in one of the previous step, or the beginning of track have returned true (i.e. it does not check again the current step if it was accepted in a previous one or begin of track)**

- **GmHistoryAllFilter: returns true if all the filters in all previous steps, and the beginning of track have returned true (i.e. it does not check again the current step if it was rejected in a previous one or begin of track)**
- **GmHistoryAncestorsFilter: behaves similarly as the GmHistoryFilter but also returns true if the condition is fulfilled by any step or track of the ancestors of the current track**
- **GmHistoryAncestorsAllFilter: behaves similarly as the GmHistoryAllFilter but also returns true if the condition is fulfilled by any step or track of the ancestors of the current track**
- **GmOnSecondaryFilter: makes the list of filters act of the secondary tracks created in the step; returns true if one of the secondary tracks created accepts all the filters. It does not implement the AcceptTrack method**
- **GmOnSecondaryAllFilter: makes the list of filters act of the secondary tracks created in the step; returns true if all secondary tracks created accept all the filters. It does not implement the AcceptTrack method**
- **GmInverseFilter: returns the opposite that the filter it receives as only argument**

10.1.3 Volume filters

These are a set of filters that accept tracks under one of the following conditions

- **In: particle is in a volume**
- **Enter: particle is entering a volume. AcceptTrack method returns always false**
- **Exit: particle is exiting a volume. AcceptTrack method returns always false, except in the case where one of the selected volumes is the world and track is exiting it**
- **Traverse: particle traverses a volume, it does neither enter nor exits it**
- **Start: particle is starting in a volume. AcceptTrack method may only return true if it is the first step**
- **End: particle is ending in a volume. AcceptTrack method may only return true if the track is ending**

The volume names are given as extra arguments to the filter. The types of volume are the following ones

- **LogicalVolume: a G4LogicalVolume object**
- **PhysicalVolume: a G4VPhysicalVolume object. The volume name and copy number are set separated with a ':' character, e.g.** *volA:1* **(see section on** *Identifying touchables***)**

- **Touchable: a G4VTouchable object. The volume name and copy number are set separated with a ':' character, the ancestors are separated by a '/' character, e.g.** *volB:3/volA:1* **(see section on** *Identifying touchables*)
- **Region: a G4Region object**
- **LogicalVolumeChildren: a G4LogicalVolume object or any of the G4LogicalVolume children of it**
- **PhysicalVolumeChildren: a G4VPhysicalVolume object or any of the G4VPhysicalVolume children of it**
- **RegionChildren: a G4VTouchable object or any of the G4VTouchable children of it**
- **TouchableChildren: a G4Region object or any of the G4Region children of it**

The name of these filters is constructed combining the geometry condition and the volume type, e.g. GmInPhysicalVolumeFilter, GmTraverseTouchableFilter, GmEndRegionFilter, GmTraverseLogicalVolumeChildrenFilter, GmExitRegionChildrenFilter.

There is a special case when parallel worlds are used: the scorers see the parallel world volumes, but the user actions do not (please ask for this new feature if you need it). This means that you cannot filter on a parallel world volume if you are using the filter for a user action, while you can if you use it for a scorer. And it also means that you cannot filter on a mass world whose position coincides with the one of a parallel world in the case of scorers, because the scorer filter will see the parallel world volume instead of the mass one. If you need to filter on mass volumes for a scorer a few volume filters are implemented, namely GmInMassLogicalVolumeFilter, GmInMassPhysicalVolumeFilter and GmInMassRegionFilter.

10.2 Classifiers

A classifier is a class that contains a method that receives a G4Step and returns a different index (an integer) depending on some given criteria. In other words it classifies the step and returns the index of its classification. These classes are unique to GAMOS, as Geant4 does not provide this functionality.

Classifiers can act on user actions or scorers. If one or several classifiers are set to act on a user action, it is up to the concrete user action to determine which use it makes of them, or to ignore them. The most common use of classifiers by user actions is to produce a different histogram set or table for each classification index. For details on classifiers acting on scorers see the section on Scorers.

The use of user actions and scorers together with classifiers is a powerful means to obtain very detailed information on the simulation through simple user commands. See the tutorial in section *Histograms and scorers* **for more details on this.**

Several classifiers need some extra parameters (see list of classifiers below) that control their behaviour. To use these classifiers they have to be declared first with the following command

/gamos/classifier CLASSIFIER_NAME CLASSIFIER_CLASS PARAMETER_1 PARAMETER_2 ...
where CLASSIFIER_NAME is the new name you want to give to a classifier to attach it to a user action or to a scorer, CLASSIFIER_CLASS is the name of the classifier class, and PARAMETER_1 PARAMETER_2 ... are the values of the parameters that the classifier needs. Those classifiers that do not need any parameter, can be assigned directly to a user action or a scorer without giving them a new name: the name will be the one of the classifier class.

To attach one classifier to a user action you put its name after the user action name in the command line that selects it

/gamos/userAction USER_ACTION CLASSIFIER_NAME/CLASS
or you can use a classifier to act on a scorer with the command

/gamos/scoring/assignClassifier2Scorer CLASSIFIER_NAME/CLASS SCORER_NAME

If you want to use more than one classifier for a user action or a scorer, you have to use the classifer GmCompoundClassifier (see below).

Each classifier has a method, GetIndexName(G4int index), that returns a different name for each index value. If a classifier does not implement this method, the one in the base class returns the index number converted to a string. This method is used by user actions and scorers to add the index number to the name of the histograms, tables or scores.

Classifiers are plug-in's, so that a user an create her/his own classifier and select it with a user command. To learn how to do this, see the instructions in the section *Creating your plug-in*, using the GmClassifierFactory, or the *Histograms and scorers* tutorial.

The following classifiers are currently implemented in GAMOS:

- **GmPSClassifierBy1Ancestor: It assigns a different index to different copy numbers of a volume. It has an extra argument that sets the level of the ancestor; if it is $N = 0$, it will use the copy numbers of the volume itself, if it is $N > 0$, it will look for the copy numbers of the N-th ancestor**

- **GmPSClassifierByAncestors: It assigns a different index to different copy numbers of the sensitive volume. It has two extra arguments that set the number of ancestor levels (NAncestor) and the maximum number of copies in a level (NShift). The index is built as**

$$\sum_{N=0}^{NAncestor-1} NShift^N * copyNumber\_of\_Nancestor$$

- **GmClassifierByLogicalVolume: It assigns a different index to different logical volumes**

- **GmClassifierByPhysicalVolume: It assigns a different index to different physical volumes**

- **GmClassifierByRegion: It assigns a different index to different regions**
- **GmClassifierByKineticEnergy: The user must define a minimum, a maximum and a width of the energy intervals. It creates kinetic energy intervals with these values and assigns a different index to different intervals**
- **GmClassifierByProcessNames: It assigns a different index to different process names that define the G4Step**
- **GmClassifierByParticleProcess: It assigns a different index to different particle-process name pairs that define the G4Step. This means that the ionisation for electrons and positrons will produce a different index, despite the process being called the same for both particles**
- **GmClassifierByParticle: It assigns a different index to different particle types**
- **GmClassifierByPrimaryParticle: It assigns a different index following the particle type for the primary that originated the current particle, or the primary particle itself**
- **GmCompoundClassifier: This classifier receives a list of classifiers and builds an index as Classifier_1*NShift+Classifier_2*NShift*NShift+... Where NShift is defined by the parameter /gamos/setParam GmCompoundClassifier:NShift NSHIFT , that by default takes a value of 100. Be aware that the classifier index is stored as a 32-bit integer, so be careful that the index is not too big (bigger than 2^{32}).**

Filters are plug-in's, so that a user can create her/his own filter and select it with a user command. To learn how to do this, see the instructions in the section *Creating your plug-in*, **using the GmClassifierFactory.**

10.2.1 Passing parameter values to a classifier

Some of the classifiers need a parameter, like the GmClassifierByKineticEnergy, that needs the minimum, the maximum and the interval width of energies. To pass parameter values to a classifier, so that it can later be used in a scorer or a user action, you have to use the command

/gamos/classifier CLASSIFIER_NAME CLASSIFIER_CLASS PARAMETER_1 PARAMETER_2 ...

where CLASSIFIER_NAME is the new name you want to give to the classifier with the list of parameters, so that you can use this name to assign your classifier to a scorer or a user action. CLASSIFIER_CLASS is one of the above-mentioned classifier classes, or one that you create your own.

Chapter 11

Managing the verbosity

11.1 GAMOS verbosity managers

While GAMOS is running you can control the amount of information you get on the screen with the GAMOS verbosity management. There are six levels of verbosity (each level includes the verbosity of the previous levels):

- **Silent (= -1): no output is printed (only when there is an exception and the job stops, you will get the details of why it happened)**
- **Error (= 0): only error messages are printed**
- **Warning (= 1): only error and warning messages are printed**
- **Info (= 2): you get some detailed information of what is happening. Mainly messages at each run and each event.**
- **Debug (= 3): you get a quite detailed information of what is happening. Mainly messages at each track and each step**
- **Test (= 4): this level is only meant for testing your code the first time you write it**

On top of this, the verbosity in GAMOS is classified in different types, so that you can set different verbosity options for different parts of the code

- **Generation: controls the verbosity of the GAMOS generator.**
- **Physics: controls the verbosity of the physics classes.**
- **Sensitive detector: controls the verbosity of the sensitive detectors, hits, digits and reconstructed hits.**
- **User action: controls the verbosity of the base classes for user action management.**
- **Scoring: controls the verbosity of the scoring classes.**
- **Analysis: controls the verbosity of the analysis classes.**

- **PET: controls the verbosity of the classes in the PET package.**
- **RT: controls the verbosity of the classes in the RadioTherapy package.**

You can set the verbosity of each of the GAMOS verbosity types with a simple command on your command input file:

/gamos/verbosity GmGenerVerbosity VERB
/gamos/verbosity GmPhysicsVerbosity VERB
/gamos/verbosity GmSDVerbosity VERB
/gamos/verbosity GmUAVerbosity VERB
/gamos/verbosity GmAnaVerbosity VERB
/gamos/verbosity GmScoringVerbosity VERB
/gamos/verbosity PETVerbosity VERB
/gamos/verbosity RTVerbosity VERB

where VERB can be any of the six values described above (in non-capital letters). Instead of the names, you may use the numbers that appear besides them.

By default the values of all GAMOS verbosities are warning

11.2 Using a GAMOS verbosity manager in your code

If you write some new code, for example a new generator distribution, you may use one of the GAMOS verbosity managers following the instructions below.

Each of the GAMOS verbosity managers instantiates an object of the type GmVerbosity. If you want that your code is only printed when the corresponding verbosity level is set, you have to write this verbosity with the value of the level in parenthesis. For example, if you write one new generator position distribution and you want that a message is printed when somebody chooses it in the input file, you can write a message like the following one in the constructor of your class:

```
G4cout << GenerVerb(infoVerb)
  << ‘‘MyPositionGeneratorDistribution created’’ << G4endl;
```

This message will only be printed if the generator verbosity is set to info or a level above (debug or test).

If you want for example that your distribution prints a message with the calculated position at each event, you may do

```
G4cout << GenerVerb(debugVerb)
  << ‘‘MyPositionGeneratorDistribution position = ’’
  << position << G4endl;
```

This message will only be printed if the generator verbosity is set to debug or a level above (test).

As you may have deduced the rules for using each of the GAMOS verbosity managers are that the name of the verbosity is the same as

the name of the verbosity manager simplified: no Gm at the beginning and Verb instead of Verbosity (e.g. from GmAnaVerbosity, you use AnaVerb). And the name of the level in C++ code is the same as the one in the input command file adding Verb (e.g. for warning, you use warningVerb).

11.3 Creating your own verbosity manager

You can create your own verbosity manager for the code you use taking as example one of the GAMOS verbosity managers (for example the class GmGenerVerbostityMgr in the package GamosCore/GamosGenerator).

First create a class inheriting from GmVerbosityMgr and fill it as follows:

- **In the include file (i.e. the one with suffix *.hh*) of this class define an object of type GmVerbosity as extern**

```
extern GmVerbosity MyVerb;
```

- **In the method void SetFilterLevel(int fl) call the same method of your GmVerbosity object**

```
MyVerb.SetFilterLevel( fl );
```

- **In the method void GetFilterLevel(int fl) call the same method of your GmVerbosity object**

```
MyVerb.GetFilterLevel( fl );
```

Finally you have to transform your class into a plug-in:

```
DEFINE_GAMOS_VERBOSITY(MyVerbosityMgr);
```

11.4 Controlling the Geant4 verbosity by event and track

If you want to print the detailed step information provided by Geant4 for a given interval of events or tracks, but you do not want that it is printed for all, you can use the user action

/gamos/userAction GmTrackingVerboseUA

You have to define the minimum and maximum events for which you want the verbosity on, and you can also set it ON only each N events:

/gamos/setParam GmTrackingVerboseUA:EventMin
/gamos/setParam GmTrackingVerboseUA:EventMax
/gamos/setParam GmTrackingVerboseUA:EventStep

If these parameters are not set, the verbosity will be ON for all events.

You can also define for which tracks interval in the selected events the verbosity will be ON, and set it ON only each N tracks:

/gamos/setParam GmTrackingVerboseUA:TrackMin
/gamos/setParam GmTrackingVerboseUA:TrackMax

/gamos/setParam GmTrackingVerboseUA:TrackStep
If these parameters are not set, the verbosity will be ON for all tracks.

Finally you may select the Geant4 verbosity level (by default 1) with the parameter

/gamos/setParam GmTrackingVerboseUA:VerboseLevel

Chapter 12

PET application

The PET example contains two directories. The first one, PetGeometry, contains an utility to build a simple PET ring detector by just defining a few parameters. The second one contains the PET event classifier and a couple of histogram classes.

12.1 PET geometry

The directory PET/PETGeometry contains an utility to build a simple PET ring detector by just defining the following parameters:

- **c_block Number of crystals per block**
- **Nblocks Number of blocks of crystals per ring**
- **Nrings Number of rings of blocks**
- **c_transaxial Crystal size, trans-axial**
- **c_axial Crystal size, axial**
- **c_radial Crystal size, radial**
- **diameter Detector ring diameter**

There are several examples of simplified commercial PET detectors in the files with suffix .dat in that directory. To use this utility you just have to choose as your geometry the PETGeometry one:

/gamos/geometry PETGeometry

By default it reads the filename PETGeometry.dat. If you want to change it, you can do it with the parameter

/gamos/setParam PET:Geometry:FileName MY_FILENAME

There are two types of sources available, NEMA1994 and DOLL. You can select them with the parameter

/gamos/setParam PET:Geometry:Source MY_SOURCE

NOTE: This module is just thought for simple PET geometries. If you want to do more complicated geometries, we recommend you to describe tehm with a text file (see section Building your geometry with a text file).

12.2 PET analysis

12.2.1 PET event classification

The class PETEventClassifierUA in the directory PET/PETAnalysis classifies the events as PET by looking at the reconstructed hits. It is a GAMOS user action, so you can activate it with the command

/gamos/userAction PETEventClassifierUA

First it counts how many reconstructed hits have 511 keV within a precision given by the two parameters

/gamos/setParam PET:EvtClass:511EPrecMin ENERGY_MIN

/gamos/setParam PET:EvtClass:511EPrecMax ENERGY_MAX

where ENERGY_MIN is the minimum energy, that by default takes a value of 0.7*511 keV, and ENERGY_MAX is the maximum energy, that by default takes a value of 1.3*511 keV.

Only hits whose relative time difference is less than the value given by the parameter

/gamos/setParam PET:EvtClass:CoincidenceTime COINCIDENCE_TIME

will be taken into account to make a pair (it is assumed that one of the two started the trigger and the other must be in the coincidence time open at that moment).

To recover hits when one of several Compton interactions have occurred you may switch the merging of hits that are close into one. You may set the distance to merge hits with the parameter

/gamos/setParam PET:EvtClass:ComptonRecHitDist DIST

DIST takes by default a value of 0, that is no Compton hits merging will be done. In this case you may select as position of the combined hits the one of the biggest energy, or the second biggest, or the n-th biggest, where the order is given by the parameter

/gamos/setParam PET:EvtClass:SelectPosOrder ORDER

ORDER takes by default a value of 1, that is, the position is that of the hit with biggest energy.

If two 511-keV hits are finally found, the event is classified as a good PET event. Then the sub-classification code enters in the game:

- **a)More than 2 511-keV hits: If more than two hits are found, the two that are closer to 511 keV will be taken and the event will receive a subclassification type of $3- > 2$.**
- **b)Random coincidence: It is checked that each of the two 511-keV hits is built only from tracks from the same 'original' gamma[1], and that the two hits come from the same event**
- **c)Scattered: The event is classified as scattered if any of the 511-keV gammas has suffered a Compton interaction in the list of volumes defined by the parameter**

  /gamos/setParam PETCountScatteringUA:VolumeNames VOLUME_1 VOLUME_2 ...

[1] 'original' gammas are gammas that are primary particles or that are directly created by the annihilation of positron that is a primary particle

and this interaction is of one of the process types defined by the parameter

/gamos/setParam PETCountScatteringUA:ProcessNames PROCESS_1 PROCESS_2 ...

and it has lost in the volumes more energy than the parameter

/gamos/setParam PETCountScatteringUA:EnergyMin ENER_MIN

- **d) Check PET line distance: a line joining the position of the two reconstructed hits is built and the distance of closest approach (DCA) to the origin of the positron is calculated; the events are classified as 'near' of 'far' if the DCA is smaller or bigger than the parameter**

  /gamos/setParam PET:EvtClass:LineDistToVtx DISTANCE

  where DISTANCE has a default value of 4*mm.

The ClassifyPET method returns an integer with several digits containing the event classification:

- **0 if it is not PET, 1 if it is PET and PET line is close to the event vertex, 2 if it is PET and PET line is far from event vertex**
- **10*1 if the search for 511-keV reconstructed hits found more than 2**
- **100*1 if the event is a random coincidence event**
- **1000*1 if the event is a scattered event**

At the end of the run a table is printed with the number of events in each of the combinations of the sub-classification types.

12.2.2 PET histograms: event classification

These histograms are related to the event classification explained above. They are produced if the event classification user action is selected. The name of all these histograms starts with "PETEvtClass: " and all are written in the file pet.root/csv.

The following histograms are written:

- **Classification index of event ("PET classification"). This index is the one described in the precedent section.**
- **Number of 511-keV reconstructed hits before cleaning if there are more than two and searching for Compton hits, i.e., hits produced merging two crystals (see above) ("N 511 recHits initial")**
- **The energy of the 511-keV reconstructed hits ("PETEvtClass: Extra PET RecHit energy (keV)")**
- **The energy of the 511-keV reconstructed hits that are rejected because there are more than two ("PETEvtClass: Extra PET RecHit energy (keV)")**

- **Distance of closest approach between vertex and the line joining the two 511-keV reconstructed hits (=DCA) ("PET dist line - vertex (mm)")**

There are also a histogram of the DCA and the reconstructed hit energies for each of the sixteen subclassification types (combinations of the 4 subindices) and also for the all the events that are far from vertex, all the events where there were more than two hits, all the events with random coincidences and all the scttered events.

12.2.3 PET output for reconstruction

If the event is classified as a PET one, it is dumped in a binary file, given by the name

/gamos/setParam Output:pet:FileNameOut MY_FILENAME

that takes the name pet.out by default. The variables written are given by the structure

```
struct PetOutput
{
  char name[8];
  float xVtx,yVtx,zVtx,x1,y1,z1,x2,y2,z2;
}
```

where name is PET, x/y/zVtx are the coordinates of the event vertex, x/y/z1 are the coordinates of the first reconstructed hit, x/y/z2 are the coordinates of the second vertex.

The same data that is written to the file can be written in the standard output if the parameter

/gamos/setParam Output:pet:DumpToCout

is set to true. The positions in the standard output (not in the file) will be written in cylindrical coordinates by default. If you want them in cartesian coordinates you should set to true the parameter

/gamos/setParam Output:pet:DumpCartesian

12.2.4 PET histograms: positrons

These histograms are related to the original positron and the two secondary gammas created at its annihilation. They are produced if the user action PETHistosPositron is activated with the command /gamos/userAction PETHistosPositron. The name of all these histograms starts with "PET-Positron" and all are written in the file pet.root/csv.

The histograms are the following:

- **Positron energy at creation ("e+ initial energy (keV)")**
- **Positron range ("e+ range (mm)")**
- **Positron energy at annihilation ("e+ energy at annihilation (keV)")**
- **Positron energy at creation vs range ("e+ initial energy (keV) vs range (mm)")**

- **Positron energy at annihilation vs secondary gammas energy ("e+e-gammas vs e+ energy (keV)")**
- **Positron energy at annihilation vs sum of secondary gammas energy - 1.022 MeV - positron energy at annihilation ("total gamma energy vs e+ energy (keV)"). This is the energy deposited locally at annihilation.**

12.2.5 PET histograms: distance between two gammas

These histograms are related to the distance between the line joining two gammas originated at the positron annihilation and the vertex position, at the moment of the gamma creation and when they hit the sensitive detector. Also other histograms about these two gammas are provided for further information. They are produced if the user action PETHistosGammaDist is activated with the command /gamos/userAction PETHistosGammaDist. The name of all these histograms starts with "PETGammaDist: " and all are written in the file pet.root/csv.

The histograms are the following:

- **Angle between the direction of the two gammas when they are created ("angle between gammas at vertex (deg)")**
- **Angle between the direction of the two gammas when they enter the sensitive detector ("angle between gammas at entering SD (deg)")**
- **Distance of closest approach between vertex and the line joining the vertices of the two gammas ("DCA orig from Gamma vertex (mm)")**
- **Distance of closest approach between vertex and the line joining the points where the gammas enter the sensitive detectors ("DCA orig from Gamma entering SD (mm)")**
- **Z position of vertex of gamma if it reaches the sensitive detector ("Orig Pos Z if Gamma reaches SD (mm)")**
- **R position of vertex of gamma if it reaches the sensitive detector ("Orig Pos R if Gamma reaches SD (mm)")**

12.2.6 Histograms of gammas at sensitive detectors

We describe here the GmHistosGammaAtSD, an utility that prints information about the interaction of 'original' gammas in the sensitive detector. It is a user action and therefore it can be activated with the command

/gamos/userAction GmHistosGammaAtSD

At the end of run a table like the following one is printed:

```
$$$$$$$$$$ Classification of Gamma Interactions in SD $$$$$$$$
$$GC: nEvents       : 1000000
$$GC: n gamma in SD : 516170 : 25.8085 %
```

```
$$GC:  n PE           : 321588 : 62.30273 %
$$GC:    PE 0 COMP      : 157614 : 49.011157 %
$$GC:    PE 1 COMP      : 111304 : 34.610744 %
$$GC:    PE >1 COMP     : 52670 : 16.378099 %
$$GC:  n COMP         : 222590 : 43.12339 %
$$GC:    1 COMP         : 159193 : 71.518487 %
$$GC:    >1 COMP        : 63397 : 28.481513 %
$$GC: nGamma_COMP/event: 0.585931
```

- **nEvents : total number of events.**
- **n gamma in SD : number of 'original' gammas reaching one sensitive detector.**
- **n PE : number of 'original' gammas with photoelectric interaction in SD**
- **PE 0 COMP : number of 'original' gammas with photoelectric interaction and no Compton interactions in SD (percentage relates to "n PE").**
- **PE 1 COMP : number of 'original' gammas with photoelectric interaction and one Compton interaction in SD (percentage relates to "n PE").**
- **PE >1 COMP : number of 'original' gammas with photoelectric interaction and more than one Compton interaction in SD (percentage relates to "n PE").**
- **n COMP : number of 'original' gammas with Compton interactions in SD.**
- **1 COMP : number of 'original' gammas with only one Compton interaction in SD (percentage relates to "n COMP").**
- **>1 COMP : number of 'original' gammas with more than one Compton interactions in SD (percentage relates to "n COMP").**
- **nCOMP/event : number of Compton interactions in SD per 'original' gamma.**

This class also makes several histograms, all of them have the words Gamma At SD: included in their names. And all are written to the file gammaSD.root/csv. The following histograms are defined:

- **Event Type / 100, i.e. no Rayleigh counting ("Event Type")**
- **Number of photoelectric interactions per 'original' gamma ("N PhotoElec")**
- **Number of Compton interactions per 'original' gamma ("N Compton")**
- **Number of Rayleigh interactions per 'original' gamma ("N Rayleigh")**
- **Number of photoelectric interactions vs Number of Compton+Rayleigh interactions per 'original' gamma ("N PhotoElec vs Compton+Rayleigh")**

- **Energy of gamma upon entering SD("Energy at entering SD (keV)")**
- **Energy lost in the photoelectric interactions ("Energy lost PhotoElec (keV)");**
- **Difference in position from the gamma entering SD to the point where a photoelectric interaction happened ("Diff pos when PhotoElec (mm)")**
- **Difference in direction from the gamma entering SD to the point where a photoelectric interaction happened ("Diff dir when PhotoElec (mm)")**
- **Difference in kinetic energy from the gamma entering SD to the point where a photoelectric interaction happened ("Diff energy when PhotoElec (mm)")**
- **Energy lost in the Compton interactions ("Energy lost Compton (keV)");**
- **Angle variation in the Compton interactions ("Angle variation Compton (mrad)")**
- **Energy lost in the Rayleigh interactions ("Energy lost Rayleigh (eV)")**
- **Angle variation in the Rayleigh interactions ("Angle variation Rayleigh (mrad)")**

All these histograms are repeated for the different types of events (a prefix in the name marks the event type the histogram refers to):

- **"ALL: ": every event**
- **"No PE: " events where no photoelectric interaction occurred**
- **"Only PE: " events where only photoelectric interaction occurred**
- **"PE + 1 Compt: " events where one photoelectric interaction plus only one Compton interaction occurred**
- **"PE + >1 Compt: " events where one photoelectric interaction plus more than one Compton interaction occurred**

Chapter 13

Radiotherapy application

The Radiotherapy example contains some utilities for the simulation of a teletherapy linear accelerator and dose calculations in the patient.

13.1 Using phase spaces

A very common utility in teletherapy simulation is the writing of a phase space, i.e. the set of particles that reach a certain plane in Z, and the later starting of the next simulation from this set of particles.

13.1.1 Writing phase spaces

The writing of phase space can be done automatically in GAMOS by selecting the user action

/gamos/userAction RTPhaseSpaceUA

When a particle crosses any of the planes defined by the parameter

/gamos/setParam RTPhaseSpaceUA:ZStops Z_1 Z_2 Z_3 ...

its information is stored in a file whose name is given by the parameter

/gamos/setParam RTPhaseSpaceUA:FileName MY_FILENAME

plus a suffix .IAEAphsp. The default value of this parameter is test. If there are several Z planes, the Z value of each plane will be added to the name, with a "_" in front, so that each phase space will go to a different file. If there is only one Z defined, the Z value may be written in the file name if the parameter

/gamos/setParam RTPhaseSpaceUA:ZStopInFileName TRUE/FALSE

is set to TRUE.

If the plane crossed is the one with maximum Z, the particle may be stopped, if the parameter

/gamos/setParam RTPhaseSpaceUA:KillAfterLastZStop TRUE/FALSE

is set to TRUE.

The format of the phase space file is the one defined by the IAEA[17], generated using the official C files from IAEA. First there is a header file, that will have the same name, but with the suffix .IAEAheader. It

is generated by the IAEA code and you can find the description of it at [17].

The variables stored for each particle are the following

- X coordinate (cm)
- Y coordinate (cm)
- X direction cosine
- Y direction cosine
- Kinetic energy (MeV)
- Particle statistical weight
- Type of the particle (1 = gamma, 2 = electron, 3 = positron, 4 = neutron, 5 = proton)
- Z direction cosine * particle charge
- Is new history = 0
- Extra integer = 0
- Extra float = 0

The Z value may optionally be stored if the parameter
/gamos/setParam RTPhaseSpaceUA:StoreZ TRUE/FALSE
is set to TRUE.

The header file may be written each N events, so that if the job ends abnormally the first N events may be recovered in the phase space. The number of events is controlled with the parameter
/gamos/setParam RTPhaseSpaceUA:NEventsToSave
N_EVENTS
By default this parameter takes a value of -1 and no header is saved until the end of the run.

As the Z plane is probably not a physical plane in the geometry, particle steps will start before the plane and end after the plane. Therefore, the position and energy are rescaled as if the particle would have stopped at the Z plane (by a simple linear interpolation).

13.1.2 Phase space histograms

When a phase space is generated, a set of histograms is produced to give you some information about the particles in the phase space. The names of all these histograms start with PhaseSpace: and they are all dumped into the file rt.root/csv. For each of the following particle types a different set of histograms are produced, containing the particle type in the histogram name: "gamma", "e-", "e+", plus a set of histograms for all particles, named "ALL". Also histograms for "neutron", "proton", will be produce if the parameter
/gamos/setParam RTPhaseSpaceHistos:Hadrons TRUE/FALSE
is set to TRUE.

A full set of histograms is produced for each Z plane, with the Z value on their names.

The following histograms are produced:

- **Position X (cm) ("X at Zstop")**
- **Position Y (cm) ("Y at Zstop")**
- **Position X vs Y (cm) ("XY at Zstop")**
- **Radial position in the XY plane (cm) ("R at Zstop")**
- **Theta angle of direction (degrees) ("Direction Theta at Zstop")**
- **Phi angle of direction (degrees) "Direction Phi at Zstop")**
- **Energy (MeV) ("Energy at Zstop")**
- **X direction cosine (" Vx at Zstop")**
- **Y direction cosine (" Vy at Zstop")**
- **Z direction cosine (" Vz at Zstop")**
- **Radial position in the XY plane (cm) vs theta angle of direction (degrees) ("R vs Direction Theta at Zstop");**
- **Radial position in the XY plane (cm) vs energy (MeV) ("R vs Energy at Zstop");**
- **Theta angle of direction (degrees) vs energy (MeV) ("Direction Theta vs Energy at Zstop");**

The user can choose not to produce any of these histograms by setting the parameter

/gamos/setParam RTPhaseSpaceUA:Histos TRUE/FALSE

The name of the histogram file can be controlled with the parameter

/gamos/setParam RTPhaseSpaceHistos:FileName MY_FILENAME

that by default is "phaseSpace".

Several parameters serve to control the number of histogram bins:

/gamos/setParam RTPhaseSpaceHistos:Nbins NBINS

that takes by default a value of 100; and the maximum (absolute) value for histograms of position

/gamos/setParam RTPhaseSpaceHistos:HisRMax MAX

that takes by default a value of 100., histograms of angle

/gamos/setParam RTPhaseSpaceHistos:HisAngMax MAX

that takes by default a value of 180., and histograms of energy

/gamos/setParam RTPhaseSpaceHistos:HisEMax MAX

that takes by default a value of 10.*MeV.

13.1.3 Reading phase spaces

To use the generated phase space you have to define as your primary generator

/gamos/generator RTGeneratorPhaseSpace

You can change the filename (by default *test*) with the parameter

/gamos/setParam RTGeneratorPhaseSpace:FileName MY_FILENAME

The particles in the phase space can be translated or rotated by using the parameters

/gamos/setParam RTGeneratorPhaseSpace:InitialDisplacement POS_X POS_Y POS_Z

/gamos/setParam RTGeneratorPhaseSpace:InitialRotAngles ANG_X ANG_Y ANG_Z

the position of the particles is first changed by the initial displacement, then the momentum vector is rotated around the X axis by ANG_X, around the Y axis by ANG_Y and around the Z axis by ANG_Z; and finally the position vector is rotated around the X axis by ANG_X, around the Y axis by ANG_Y and around the Z axis by ANG_Z.

If the number of phase space particles is not enough to calculate the dose in the phantom with enough precision you may reuse the particles several times (that is use the same particle in several consecutive events) by setting the parameter

/gamos/setParam RTGeneratorPhaseSpace:MaxNReuse N

to a value bigger than 1. In fact if this parameter is not explicitly set to 1, GAMOS calculates it automatically by dividing the number of events asked for by the number of events in the input phase space.

Alternatively you may recycle the full phase space several times (i.e. when all particles in the phase space are read, the file is closed and restarted again). The number of times a phase space is recycled is controlled by the parameter

/gamos/setParam RTGeneratorPhaseSpace:MaxNRecycle N

which must take a value bigger than 1. If no reusing is explicitly set, the number of reuses will be automatically calculated as mentioned above and the number of recyclings will be set to 1, so that no recycling is done.

Caution should be taken when recycling phase spaces, as the error correlations due to the reusing of the same particles is not taken account. Therefore you should first consider reusing instead of recycling.

If you think your phase space has X/Y symmetry you may reduce the correlations due to reusing the same particle several times by mirroring your particles each time they are reused, by setting the parameter

/gamos/setParam RTGeneratorPhaseSpace:MirrorWhenReuse OPT

where OPT can be X or Y or XY. If it is X it means that when a particle is used for n-th time, with even n, the X original value of position and momentum will be changed sign, while when it is for an n-th time, with n odd, it will not be changed. If it is Y, the same but mirroring in Y. And if it XY the second time a particle is used the X values will be changed sign, the third time the Y values will be changed sign, the fourth time both X and Y values will be changed sign, the fifth time there will be no change, and the cycle restarts.

When reading a phase space file you may skip the first N events by using the parameter

/gamos/setParam RTGeneratorPhaseSpace:NEventsSkip N

You can produce the same histograms that were produced when writing the phase space file by setting the parameter

/gamos/setParam RTGeneratorPhaseSpace:Histos TRUE/FALSE

to TRUE.

When you use a phase space as input you may use as number of events the real number of events run in your job (which would equal be to the number of particles in the phase space multiplied by the number of times each particle is reused), or you may alternatively use as number of events the original number of events that were used to generate the phase space (see section on *Scoring dose in phantom*. To use the first the first one you can invoke the standard Geant4 method G4RunManager::GetRunManager()-¿GetNumberOfEvent(), to use the second one, you may invoke the GAMOS method GmNumberOfEvent::GetNumberOfEvent().

13.1.4 Adding extra information to a phase space

The IAEA format allows the storing of two long integers and two floats in the phase space file as extra information. In GAMOS we have extended this functionality by putting the 32+32 bits of the two integers in a continuous stream, that the user can divide in groups of bits of the desired length to store several informations. The user can even add more sets of 32 bits by changing the the following line at source/RadioTherapy/include/iaeaRecord.hh :

```
#define NUM_EXTRA_LONG  2
```

and recompiling (*cd MY_GAMOS_DIR/source/RadioTherapy; make*).

To store some information in this format the user has to instantiate one of the following user actions

- **RTExtraInfoProviderCrossings: fills a bit if the track has crossed the corresponding region before reaching the phase space**
- **RTExtraInfoProviderInteractions: fills a bit if the track has interacted in the corresponding region before reaching the phase space**
- **RTExtraInfoProviderOrigin: fills a bit if the track has been created in the corresponding region before reaching the phase space**

The user may select how many bits each information must occupy by using the GAMOS parameter :

/gamos/setParam EXTRA_INFO_NAME:NBits NBITS
where EXTRA_INFO_NAME is the name of the extra information class (see above). GAMOS will check that the index to be filled by a class is not bigger than the number of bits reserved for it. And it will also check that the total number of bits is not bigger than the available quantity (32*NUM_EXTRA_LONG).

The order of declaration of the extra information user actions sets the order of bits occupancy. At the end of the job each of these extra information user actions fills a file explainning the information contained in each bit. By default this file is called RTExtraInfoProvider.summ, but the user may change the name of it with the parameter /gamos/setParam RTExtraInfoProviderLong:FileName FILE_NAME

An example of a file is the following one:

```
RTExtraInfoProviderOrigin INDEX/REGION 0 DefaultRegionForTheWorld
RTExtraInfoProviderOrigin INDEX/REGION 1 targetReg
RTExtraInfoProviderOrigin INDEX/REGION 2 collimatorReg
RTExtraInfoProviderOrigin INDEX/REGION 3 filterReg
RTExtraInfoProviderOrigin INDEX/REGION 4 monitorReg
RTExtraInfoProviderOrigin INDEX/REGION 5 shieldingReg
RTExtraInfoProviderOrigin INDEX/REGION 6 jawsReg
```

The float extra information providers fill one of the two available words in the order they have been defined.

The user can add more float words by changing the the following line at source/RadioTherapy/include/iaeaRecord.hh :

```
#define NUM_EXTRA_FLOAT  2
```

To store some information as float the user has to instantiate one of the following user actions

- **RTExtraInfoProviderZOrigin: stores the Z position of the origin of the track**
- **RTExtraInfoProviderZLast: stores the Z position of the last interaction before reaching the phase space**

At the end of the job each of these extra information user actions fills a file explaining the information contained in each bit. By default this file is called RTExtraInfoProvider.summ, but the user may change the name of it with the parameter

/gamos/setParam RTExtraInfoProviderFloat:FileName FILE_NAME

13.1.5 Reusing a particle at a phase space without filling the phase space file

It is common in radiotherapy treatment simulations that in a first job a phase space file is created and in a second job the particles therein are read and reused several times to calculate the dose in the patient. GAMOS provides the possibility of reusing particles in a phase space file without having to divide the jobs in two. To use this utility the user has to instantiate the user action

/gamos/userAction RTReuseAtZPlaneUA

and then use the command

/gamos/RT/ReuseAtZPlane

and before that, define the z value of the plane where particles will be replicated, with the parameter

/gamos/setParam RTReuseAtZPlane:ZReusePlane Z_POS

and the number of times particles will be reused

/gamos/setParam RTReuseAtZPlane:NReuse NREUSE

The user may independently create the phase space file or not at the same Z position or at others.

13.2 Optimisation of a linac simulation

To optimise your simulation in GEANT4 you may tune the physics parameters (production cuts, special cuts, multiple scattering options, ...) as well as try some variance reduction techniques.

Please read the sections on automatic optimisation of production cuts and user limits for accelerator and dose calculation simulations. See also the web page http://fismed.ciemat.es/GAMOS/RToptim to get a list of the best electromagnetic physics parameters that can optimise your simulation. See also sections on bremsstrahlung splitting.

13.2.1 Bremsstrahlung splitting

Bremsstrahlung splitting is a non-biased variance reduction technique that may reduce the CPU time of your accelerator simulation by a big factor. It basically consists on splitting the secondary particles (in radiotherapy mainly gammas generated by bremsstrahlung) N times, so that each time a bremsstrahlung gamma is created, other N-1 gammas are created at the same position, with weight 1/N, re-sampling the energy and/or angle distribution. As most of the particles that reach a patient in a radiotherapy accelerator are bremsstrahlung gammas, by using this technique we can spare the time spent simulating the original electron (usually even close to 50% of the total time can be saved) and we can reduce the time spent tracking gammas that have small possibility of reaching the patient.

There are three splitting techniques implemented in GAMOS.

Uniform bremsstrahlung splitting

This technique is the simplest one. When a gamma suffers a bremsstrahlung interaction, the resulting gamma is split N times producing N equal particles, each of weight 1/N. To apply it the following command should be used

/gamos/physics/varianceReduction/splitting uniform

and before that, the splitting number should be set with the parameter

/gamos/setParam GmBremsSplittingProcess:NSplit NSPLIT

that by default takes a value of 10. Another parameter controls how many times a particle will be split

/gamos/setParam GmBremsSplittingProcess:SplitLevel SPLIT_LEVEL

that by default takes a value of 1, i.e. particles already split will not be split again.

Z-plane directional bremsstrahlung splitting

In this technique the user must define first a plane perpendicular to the Z axis at a given Z position and limit its dimensions in X and Y. This can be done with the parameters

/gamos/setParam GmBSZPlaneDirChecker:ZPos ZPOS
/gamos/setParam GmBSZPlaneDirChecker:XDim XDIM
/gamos/setParam GmBSZPlaneDirChecker:YDim YDIM

When a bremmstrahlung interaction occurs, it will be checked if the direction of each of the split gamma points towards the user-defined square. If it does, the gamma is kept, if it does not, Russian roulette is played with a probability 1/N so that if the gamma survives its weight will be set back to 1.

It is recommended that the square is placed close to the phantom's upper plane and has dimensions a few centimeteres wider than the phantom.

To apply this technique the following command should be used

/gamos/physics/varianceReduction/splitting zPlaneDir

This bremsstrahlung splitting technique is more efficient than the uniform bremsstrahlung splitting, because only a few gammas that are not aimed to the region of interest are tracked, but it has the inconvenient that tracks with different weights (1/N and 1) reach the patient, spoiling the efficiency gain. To take profit of the definition of a region of interest while keeping the same weight for all the particles that reach the phantom, a third splitting technique has been developed, that is described below.

Equal weight directional bremsstrahlumg splitting

The main ideas of this splitting technique are first that gammas are split at each interaction and Russian roulette is played with those that are not directed towards a square perpendicular to Z as in the technique above. But this technique makes that all gammas that reach the user-defined square have the same weight, 1/N. To achieve this, gammas are split at each interaction, not only if it is bremsstrahlung, if their weight is 1, and they are not split if the weight is 1/N. This technique is based on the Directional Bremsstrahlung Splitting [18]

To apply this technique the following command should be used

/gamos/physics/varianceReduction/splitting equalWeight

Several paramters control the different options. The same parameters as for the other techniques are used to set the splitting number and the definition of the square of interest.

The implementation of this technique in this GAMOS version is a preliminary one. Efficiency improvements of about 40 have been reported, but it is currently being improved and we expect to at least duplicate the efficiency gain.

13.2.2 Killing particles at big X/Y

This utility serves to kill the particles that would probably not reach your detector because they have too big X and Y positions (it is assumed that your detector is described along Z and that your initial particles move in this direction in the positive sense).

It makes a list of the volumes that are placed directly inside the world volume and computes the minimum and maximum extension in X and Y (using the method *G4VSolid::GetExtent()*). This rectangular area extends from the minimum Z value of this volume until the minimum Z

value of the next volume, so that all tracks that are outside it will be killed. This utility is activated by selecting the user action

/gamos/userAction RTZRLimitsAutoUA

You may argue that too many particles (or too few) are killed with this automatic definition of limits. You may simply change the dimensions of the volumes placed in the world (adding container volumes made of air will not change your physics), or you can define the limits yourself by selecting the user action

/gamos/userAction RTZRLimitsUA

Then you have to write a file named rtzrlimits.lis with the list of values you want to use. The format of that file should be the following: a set of lines with three numbers representing the Z value of the plane and then the X and Y limits. The planes will be extended in Z until the previously defined Z (for the minimum Z defined the world negative Z limit will be used).

If you want to change the name of the limits file, you can do it with the command (remember to define a parameter always before selecting the user action)

/gamos/setParam RTZRLimitsUA:FileName MY_FILENAME

13.3 Scoring dose in phantom

To use a phantom geometry you can use the GAMOS utilities to read phantom geometries in EGS or GEANT4 formats or build simple phantoms with a few user commands. The scoring of the dose in the phantom volumes can be done using the scorer GmG4PSDoseDeposit and selecting as detector the voxels, that are named patient. For example you can use the following commands:

/gamos/scoring/createMFDetector DoseDet patient

/gamos/scoring/addScorer2MFD DoseScorer GmG4PSDoseDeposit DoseDet

This is all you need to get on the standard output the dose by event deposited in each voxel. As explained in the section on *Reading phase spaces*, remember that if phase space are used the number of events is the original number of events that were used to generate the phase space.

In fact, GAMOS gets the ratio of particles written in the phase space per original event transported through the accelerator and multiplies this ratio by the number of phase space particles used. This allows to get the best approximation to the correct number of events when the phase space is not used fully, or when particles are reused (it is indeed not the exact number if for example you use only the 10 first particles in the phase space: the number of events that generated those 10 particles may be bigger or smaller than the number of events that generated the following 10 particles, due to statistical fluctuations).

13.3.1 Saving scores in file

You can also store the dose in each voxel in a file, which allows you to calculate the average dose from several jobs (see dose analysis section).

The first file is a text file where the dose and dose error in each voxel are written. To obtain it you just have to add the scorer printer GmPSPrinter3ddose to your scorer, for example

/gamos/scoring/addPrinter2Scorer PSPrinter3ddose GmPSPrinter3ddose DoseScorer

The output file is named by default 3ddose.out. You may change it with the parameter

/gamos/setParam Output:3ddose:FileNameOut MY_FILENAME

The format is the same as the 3ddose format used in DOSXYZnrc, except that the first line contains the number of events:

- **Number of voxels in the X, Y and Z directions (e.g. n_x, n_y, n_z)**
- **Array of voxel boundaries (cm) in the X direction (n_x+1 values)**
- **Array of voxel boundaries (cm) in the Y direction (n_y+1 values)**
- **Array of voxel boundaries (cm) in the Z direction (n_z+1 values)**
- **Array of dose values (Gy/cm^3) (n_x n_y n_z values)**
- **Array of dose relative error values (n_x n_y n_z values)**

The second available file is a binary file where the dose and dose squared in each voxel are written. The binary format allows for a faster writing and reading and the fact of writing the dose squared instead of the error serves to calculate in a proper way the error correlations when the doses from several jobs are added. To obtain it you just have to add the scorer printer GmPSPrinterSqdose to your scorer, for example

/gamos/scoring/addPrinter2Scorer PSPrinterSqdose GmPSPrinter-Sqdose DoseScorer

The output file is named by default sqdose.out. You may change it with the parameter

/gamos/setParam Output:sqdose:FileNameOut MY_FILENAME

All variables are of type *float* and the format is the following:

- **Number of events (original number of events used to generate a phase space file if phase space file is used as primary generator)**
- **Number of voxels in the X, Y and Z directions (e.g. n_x, n_y, n_z)**
- **Array of voxel boundaries (cm) in the X direction (n_x+1 values)**
- **Array of voxel boundaries (cm) in the Y direction (n_y+1 values)**
- **Array of voxel boundaries (cm) in the Z direction (n_z+1 values)**
- **Array of dose values (Gy/cm^3) (n_x n_y n_z values)**
- **Array of dose squared values ((Gy/cm^3)^2) (n_x n_y n_z values)**

13.3.2 Saving scores in histograms

If you want to store the scores in the phantom in a histogram file, you can use the scorer printer RTPSPDoseHistos

/gamos/scoring/addPrinter2Scorer PSPrinterHistos RTPSPDoseHistos DoseScorer

By default the number of bins in the histograms will be equal to the number of voxels in the phantom (nVoxel) and the histogram limits are -nVoxel/2 +to nVoxel/2. The user can redefine the number of bins as well as the minimum and maximum values of the histograms independently for each of the three dimensions, with the following parameters:

/gamos/setParam RTPSPDoseHistos:Xmin
/gamos/setParam RTPSPDoseHistos:Xmax
/gamos/setParam RTPSPDoseHistos:XNbins

/gamos/setParam RTPSPDoseHistos:Ymin
/gamos/setParam RTPSPDoseHistos:Ymax
/gamos/setParam RTPSPDoseHistos:YNbins

/gamos/setParam RTPSPDoseHistos:Zmin
/gamos/setParam RTPSPDoseHistos:Zmax
/gamos/setParam RTPSPDoseHistos:ZNbins

The histograms limits also define the limits of the voxels that will be used to fill the histograms: before filling the histograms with the dose of a given voxel it is checked that the centre of the voxel is inside the limits.

The names of all these histograms start with RTPSPDoseHistos: and they are all dumped into the file dose.root/csv.

The following histograms are produced:

- **Dose in X direction ("Dose in X")**
- **Dose in Y direction ("Dose in Y")**
- **Dose in Z direction ("Dose in Z")**
- **Dose in XY direction ("Dose in XY")**
- **Dose in XZ direction ("Dose in XZ")**
- **Dose in YZ direction ("Dose in YZ")**
- **Dose of each voxel ("Dose")**
- **Integrated dose of each voxel, i.e. all the voxels that have dose equal to or greater that a given bin fill that bin ("Dose-volume")**

13.4 Analysis utilities

This is a set of utilities that may serve in the analysis of phase space and dose files, for example to sum phase space or dose files from different jobs, to get basic information of the file contents, to fill histograms out of the file or to compare files from two jobs.

All these utilities are under the directory *MY_GAMOS_DIR/analysis*. They are compiled by default with the rest of GAMOS code and then

they are available as executables as mentioned in the following subsections.

13.4.1 Summing phase space files

This utility serves to sum phase space files corresponding to different jobs with the same setup. To use it you have to write a file containing the list of phase space header files, one file per line, for example

```
ps.iaea.20000.IAEAheader
ps.iaea.20001.IAEAheader
ps.iaea.20002.IAEAheader
```

Then you just have to run the executable
sumphsp INPUT_FILE_LIST_NAME OUTPUT_FILE_NAME
where INPUT_FILE_LIST_NAME is the name of the file containing the list of files to add and OUTPUT_FILE_NAME is the name of the output file that will contain the sum of all the files (two files indeed as usual for IAEA phase space files: OUTPUT_FILE_NAME.IAEAheader and OUTPUT_FILE_NAME.IAEAphsp).

When running you will see on the screen something similar to this:

```
Opening phase space contained in     ps.iaea.20000.IAEAheader of type IAEA
 PARTICLES 225437 NPART_TOT 225437 NPARTORIG_TOT 5000000 RATIO 0.0450874 +- 0.000101661 RATIO_TOT 0.0450874
Opening phase space contained in     /scratch/arce/gamos/prod/ps.iaea.20001_25.IAEAheader of type IAEA
 PARTICLES 224635 NPART_TOT 450072 NPARTORIG_TOT 10000000 RATIO 0.044927 +- 0.000101455 RATIO_TOT 0.0450072
Opening phase space contained in     /scratch/arce/gamos/prod/ps.iaea.20002_25.IAEAheader of type IAEA
 PARTICLES 224813 NPART_TOT 674885 NPARTORIG_TOT 15000000 RATIO 0.0449626 +- 0.000101501 RATIO_TOT 0.0449923
 * * * * N Particles   = 674885
 * * * * N Photons    = 673804
 * * * * N Electrons = 1057
 * * * * N Positrons = 24
 * * * * N Original Histories =  1.5e+07
```

For each phase space file after the name of the file comes a line with the file statistics: number of particles, accumulated number of particles of all files, accumulated number of original histories, ratio of particles/original histories +- ratio error, accumulated ratio if particles/original histories. And at the end the statistics on the total number of particles, photons, electrons and positrons, and the total number of original histories in all summed files.

13.4.2 Comparing number of particles per event in two phase space files

You can obtain information about the particle content of a phase space file and compare if it is statistically equal to the one from another phase space file.

If you type
analysePS PHASESPACE_FILE_NAME
you will get on the screen this information in an output similar to this

```
READING ps.iaea.20000.IAEAheader
ORIGINAL HISTORIES= 5000000
```

```
PARTICLES PER EVENT= 0.0450874 +- 9.70776e-05
GAMMAS PER EVENT= 0.0450136 +- 9.69947e-05
ELECTRONS PER EVENT 7.28e-05 +- 3.8159e-06
POSITRONS PER EVENT= 1e-06 +- 4.47214e-07
```

If you want to compare two files you just have to add a second file name

analysePS PHASESPACE_FILE_NAME_1 PHASESPACE_FILE_NAME_2

and you will get the contents of the first file, the contents of the second file and the ratio of particles per event of the first file divided by the second one with the statistical errors included:

```
 READING ps.iaea.20000.IAEAheader
ORIGINAL HISTORIES= 5000000
PARTICLES PER EVENT= 0.0450874 +- 9.70776e-05
GAMMAS PER EVENT= 0.0450136 +- 9.69947e-05
ELECTRONS PER EVENT 7.28e-05 +- 3.8159e-06
POSITRONS PER EVENT= 1e-06 +- 4.47214e-07
 READING ps.iaea.20001.IAEAheader
EVENTS= 5000000
PARTICLES PER EVENT= 0.044927 +- 9.68973e-05
GAMMAS PER EVENT= 0.0448562 +- 9.68176e-05
ELECTRONS PER EVENT 6.92e-05 +- 3.72034e-06
POSITRONS PER EVENT= 1.6e-06 +- 5.65686e-07

RATIO PARTICLES PER EVENT= 1.00357 +- 0.00305842
RATIO GAMMAS PER EVENT= 1.00351 +- 0.00306059
RATIO ELECTRONS PER EVENT= 1.05202 +- 0.0789916
RATIO POSITRONS PER EVENT= 0.625 +- 0.356305
```

13.4.3 Making histograms out of a phase space file

You can make histograms out of the particles contained in a phase space file by running gamos with the input script that you can find in RadioTherapy/analysis/phaseSpace/analysePS/rt.analysePS.in. If you edit it and change the name of the input phase space file, at /gamos/setParam RTGeneratorPhaseSpace:FileName , and run

gamos rt.analysePS.in

You will get a file named phaseSpace.root, that contains the same histograms that you get when you run the job to write the phase space file (see section on Phase space histograms).

13.4.4 Merging '3ddose' files

This utility serves to merge dose files in the 3ddose format corresponding to different jobs with the same setup, and averages the dose in them. To use it you have to write a file containing the list of 3ddose files, one file per line, for example

```
3ddose.water.20000.out
3ddose.water.20001.out
3ddose.water.20002.out
```

Then you just have to run the executable

merge3ddose INPUT_FILE_LIST_NAME OUTPUT_FILE_NAME

where INPUT_FILE_LIST_NAME is the name of the file containing the list of files to merge and OUTPUT_FILE_NAME is the name of the output file that will contain the merging of all the files.

When merging files it will be checked that they correspond to the same phantom by checking the number of voxels and voxel limits.

13.4.5 Merging 'sqdose' files

This utility serves to merge dose files in the sqdose format corresponding to different jobs with the same setup, and averages the in them. To use it you have to write a file containing the list of sqdose files, one file per line, for example

```
sqddose.water.20000.out
sqdose.water.20001.out
sqdose.water.20002.out
```

Then you just have to run the executable

mergeSqdose INPUT_FILE_LIST_NAME OUTPUT_FILE_NAME

where INPUT_FILE_LIST_NAME is the name of the file containing the list of files to merge and OUTPUT_FILE_NAME is the name of the output file that will contain the merging of all the files.

When merging files it will be checked that they correspond to the same phantom by checking the number of voxels and voxel limits.

13.4.6 Making histograms out of a 'sqdose' file

You can make histograms out of dose information contained in a sqdose file by running

analyseSqdose SQDOSE_FILE_NAME

You will get a file named dose_analyseSqdose.root, that contains the same histograms that you get when you run the job to write the dose file using the scorer printer RTPSPDoseHistos.

Chapter 14

Appendix

14.1 Converting a Geant4 example into a GAMOS example

Converting a Geant4 example into a GAMOS examples usually requires only adding a few lines to transform the different simulation components into plug-in's. In examples/N02 you can see the official Geant4 example novice/N02 transformed into a GAMOS example. We will use it to illustrate the procedure to follow.

The first thing to do is substituting the Geant4 GNUmakefile with a GAMOS GNUmakefile. You can use the file in this example for any other example you want to transform, just substitute the line

name := exampleN02

with the name of your example (indeed you can use any name you want, it will be automatically detected by GAMOS).

Then you have to add a file, that we called src/module.cc where all the plug-in's are created. The first line of this file (after the corresponding 'includes'), must be

DEFINE_SEAL_MODULE

For the detector construction it is only needed to add a line

DEFINE_GAMOS_GEOMETRY(ExN02DetectorConstruction);

For the physics, in a similar way, we add

DEFINE_GAMOS_PHYSICS(ExN02PhysicsList);

The primary generator requires some more changes. As you can see, ExN02PrimaryGeneratorAction constructor receives as argument the detector construction class, so that it can then ask it for the dimensions of the world. This is not needed in GAMOS because all volumes are available in any class through the singleton class GmGeometryUtils. Therefore we have deleted the detector construction argument in the constructor, and then we have substituted the line

```
G4double position = -0.5*(myDetector->GetWorldFullLength());
```

with these

```
G4Box* worldBox = (G4Box*)(GmGeometryUtils::GetInstance()->
   GetLogicalVolumes("World")[0]->GetSolid());
G4double position = -0.5*worldBox->GetXHalfLength();
```

After this in the src/module.cc a line has to be included
DEFINE_GAMOS_GENERATOR(ExN02PrimaryGeneratorAction);
For the user actions, we have first to transform them into GAMOS user actions, which requires simply to edit the .hh classes and make them inherit from GmUserXXXAction, instead of G4UserXXXAction. Then we have to add the following lines in module.cc
DEFINE_GAMOS_USER_ACTION(ExN02RunAction);
DEFINE_GAMOS_USER_ACTION(ExN02EventAction);
DEFINE_GAMOS_USER_ACTION(ExN02SteppingAction);
It is not needed to convert the sensitive detector into a plug-in, that could be called in the user command file, because it is explicitly called in the detector construction class. Indeed, if you want to do it, you should delete the lines that instantiate it there and then you can write
DEFINE_GAMOS_SENS_DET(ExN02TrackerSD);
The main class, exampleN02.cc, is not needed anymore. We use the GAMOS main and a macro file, that we may call exampleN02.mac, with the user commands that select the example components, like the following one:

```
/gamos/geometry ExN02DetectorConstruction
/gamos/physicsList ExN02PhysicsList
/gamos/generator ExN02PrimaryGeneratorAction

/gamos/userAction ExN02RunAction
/gamos/userAction ExN02EventAction
/gamos/userAction ExN02SteppingAction

/run/initialize

/run/beamOn 10
```

You can then run *gamos exampleN02.mac* **and you will get the same results as if you run the original Geant4 example.**

14.2 Creating your plug-in

There are several "factories" in GAMOS that take care of the different plug-in types. To transform your class into a plug-in you have to follow the instructions in this section, using the relevant "factory" and class as indicated in the relevant section of this guide (e.g. GmPhysicsFactory and G4VUserPhysicsList for a geometry plug-in, GmVerbosityFactory and GmVerbosityMgr for a verbosity plug-in).

To write a new plug-in of any type follow these steps

1. **Create your class or use one of the existing Geant4 classes. This class should inherit from the corresponding Geant4 or GAMOS class:**

   - **G4VUserDetectorConstruction: geometry**
   - **G4VUserPhysicsList: physics list**
   - **G4VUserPrimaryGeneratorAction: primary generator**
   - **GmVGenerDistPosition: primary particles position distribution**
   - **GmVGenerDistDirection: primary particles direction distribution**
   - **GmVGenerDistEnergy: primary particles energy distribution**
   - **GmVGenerDistTime: primary particles time distribution**
   - **GmVUserAction: user action**
   - **G4VSensitiveDetector: sensitive detector**
   - **GmVDigitizer: hits digitizer**
   - **GmVRecHitBuilder: reconstructed hits builder**
   - **GmVPrimitiveScorer: scorer**
   - **GmVPSPrinter: scorer printer**
   - **GmVFilter: filter**
   - **GmVClassifier: classifier**
   - **GmVVerbosityMgr: verbosity**

   If you are creating a new class you can use as example one of the classes in the directory examples/PlugInTemplates, that are nearly-empty classes that contain the necessary methods that you have to implement for each plug-in type.

2. **Include the SEAL module definition:**

   ```
   #include "PluginManager/ModuleDef.h"

   DEFINE_SEAL_MODULE ();
   ```

   Then you have to include the relevant "factory", and define your plug-in

   ```
   #include "GmCore/GmFactories/include/GmXxxFactory.hh"

   DEFINE_SEAL_PLUGIN(GmXxxFactory,MY_CLASS,"MY_PLUGIN_NAME");
   ```

   Alternatively, if you do not mind that the plug-in name has the same name as your class, you can use a short notation, instead of DEFINE_SEAL_PLUGIN

```
DEFINE_XXX(MY_CLASS);
```

where XXX is the type of object you are defining (the name of the factory, without "Factory", e.g. from GmGenerDistEnergyFactory, the "Gm" substituted by "GAMOS", and the separation of words with "_"). For example GAMOS_GEOMETRY, GAMOS_USER_ACTION, GAMOS_GENER_DIST_POSITION, ... (beware the capitals).

You can add these lines in your class or create a new file with these lines only (see as example the files called module.cc in almost all the GAMOS code directories). Remember in any case that you cannot have two definitions of "DEFINE_SEAL_MODULE ()" in the same directory.

3. **Once this is done, you can select your geometry with the command:**

   /gamos/xxx MY_PLUGIN_NAME

   For example, if you have written

   DEFINE_SEAL_PLUGIN (GmGeometryFactory, MyGeometry, "MyGeom");

   you can then use

   /gamos/geometry MyGeom

   to select your geometry

   Or if you have written

   DEFINE_GAMOS_GEOMETRY(MyGeometry);

   you can tell your job to select your geometry with the command:

   /gamos/geometry MyGeometry

NOTE: If you are creating a plug-in in a new directory that you have created, you have to be sure to have the "plugin" option in the GNUmakefile, as explained in the section Compiling your new code.

14.3 Managing the input data files

To run an example you will probably need some input data, like for example the file describing the geometry, the list of isotopes, etc.

You can set the name of your file in your script, but you do not need to tell the path where to look for it. An environmental variable, called GAMOS_SEARCH_PATH contains the list of directories where GAMOS will look for your file. The directories are separated by a colon, and their order in this variable reflects the order in which they will be searched. This variable is set up when you configure GAMOS, and takes a default value of

.:MY_GAMOS_DIR/data

You may change this value at your will, but remember not to reset the GAMOS configuration after that.

14.4 Checking the usage of parameters

Many of the GAMOS classes or your own classes have a different behaviour depending on the value of some parameters. You can see many examples of this throughout this guide.

You have to remember always to set a parameter before you invoke any code that may use it (we recommend you to set all the parameters at the beginning of your command file). The parameters are read usually in the class constructors; therefore, if you set a parameter after the class has been constructed, it will take no effect and the default value will be used.

To guarantee that you have done it this way, you will get at the end of a GAMOS job the information on the usage of parameters. You will always get a list of the parameters that have not been used in the code. This is probably an indication that you have mistyped a parameter name. This list appears at the end of your output and looks similar to this one:

```
 %%%%% PARAMETERS NOT USED (DEFINED IN SCRIPT BUT NOT USED BY C++ CODE)
%%%    MAYBE YOU HAVE MISSPELLED THEM?
PARAMETER: GmGeometryText:FileNam
```

Other lists are available at user request to get a more detailed information. You may get them anywhere in your simulation by using the command

/gamos/printParametersUsage LEVEL

If LEVEL takes a value >= 0 you will get the same list as above. If LEVEL takes a value >= 1 you will get a list of parameters that are using the default value (you may then check if this list contains one of the parameters whose values you thought you have changed). This list looks similar to this one:

```
%%%%% PARAMETERS USING DEFAULT VALUE (DEFINED IN C++ CODE BUT VALUE NOT DEFINED IN SCRIPT)
PARAMETER: Generator:Isotope:FileName
PARAMETER: GmCountTracksUA:FirstEvent
```

If LEVEL takes a value >= 2 you will get a list of how many times each parameter has been used. This list looks similar to this one:

```
%%%%% NUMBER OF TIMES EACH PARAMETER IS USED IN C++ CODE
PARAMETER GmCountTracksUA:EachNEvent TIMES USED TIMES= 1
PARAMETER GmGeometryFromText:FileName TIMES USED TIMES= 1
```

14.5 Using a parameter in your C++ code

We describe in this section how to create and use a new parameter if you are creating a new C++ class.

GAMOS provides an utility that allows you to change the value of a parameter in the input file, together with the line commands, and use

it in any of your classes (even in several of them). There are four types of parameters: numbers, string, list of numbers, list of strings.

To change the value of a parameter (of any of the four types) in your input file, you have to use the command

/gamos/setParam MY_PARAM_NAME MY_PARAM_VALUE(s)
Then you can use this parameter in your class with a line like this:

```
G4double value  = GmParameterMgr::GetInstance()->
 GetNumericValue("MY_PARAM_NAME",DEFAULT_VALUE);
```

if it is a number, or

```
G4String value  = GmParameterMgr::GetInstance()->
  GetStringValue("MY_PARAM_NAME",DEFAULT_VALUE);
```

if it is a string, or

```
std::vector<G4double> vdefault;
std::vector<G4double> values  = GmParameterMgr::GetInstance()->
  GetVNumericValue("MY_PARAM_NAME",vdefault);
```

if it is a list of numbers, or

```
std::vector<G4String> vdefault;
std::vector<G4String> values  = GmParameterMgr::GetInstance()->
  GetVStringValue("MY_PARAM_NAME",vdefault);
```

if it is a list of strings.

14.5.1 Log files

While running GAMOS the output that appear on your screen as standard output is also written to a log file, called *gamos.log*, **while the standard error messages are written to another log file***gamos_error.log*. **You may select another name for the standard output log file by using the Geant4 command:**

/gamos/log/setCoutFile FILE_NAME
and you can also change the name of the standard error log file by using the Geant4 command

/gamos/log/setCerrFile FILE_NAME

You may choose to give the tow log files the same name if you want them to appear together in the same order they appear on your screen.

It is also possible to suppress the writing of any log file by using the Geant4 file:

/gamos/log/writeFiles 0

Take into account that the above commands start to take effect when they are read, this means that you may have part of the output written in a file (the one that appeared before the command) and the rest in another file (the one that appeared after the command), or, in a similar way, write part of the output only using the third command in the middle of your input file.

14.6 Identifying touchables

As explained in several points through this guide, you can use the concept of touchable available in GAMOS. We will explain first in a few lines the concept of touchable in Geant4 and GAMOS:

In Geant4 there are several geometrical objects [7]:

- G4VSolid A solid is a geometrical object that has a shape and specific values for each of that shape dimension.
- G4LogicalVolume A logical volume contains the volume's full properties. It includes the geometrical properties of the solid, and adds physical characteristics: the material of the volume, whether it contains any sensitive detector elements, the magnetic field, etc.
- G4VPhysicalVolume A physical volume is a volume placed already in another volume.
- G4VTouchable A touchable is each copy of a volume. To understand the difference with a physical volume, we put an example: If you place a volume A in 5 places inside volume B, and place volume B in 12 places, you will have 5 + 12 physical volumes, each one with a distinct position and rotation matrix. But you will have $5X12 = 60$ individual copies, that is 60 touchables.

To save memory usage the G4VTouchable are instantiated in Geant4 when a track traverses the corresponding volume in space, and they are immediately deleted when the track leaves. In GAMOS you have the possibility of accessing any individual touchable whenever you like. When you need it you can ask GAMOS to create a GmTouchable, which will have the same characteristics as the corresponding G4VTouchable that would be created when a track reaches it.

To identify the touchable you want to use, you have to use the following notation:

For example the name CRYSTAL identifies all individual crystals of your detector that have name "CRYSTAL", while BLOCK#2/CRYSTAL#1 refers only to the crystal(s) with copy number 1 in block(s) with copy number 2. RING#3/BLOCK/CRYSTAL#1 refers to all the crystal(s) with copy number 1 in all the block(s) whatever copy number they have in the ring(s) that have copy number 3.

If you are writing a new plug-in you can have easy access to the list of touchables with a given name with the line

```
GmGeometryUtils::GetInstance()->GetTouchables( touch_name, itExists )
```

If the touchables do not exist in your geometry you will get a warning in case itExists is false and an exception if itExists is true.

There is a limitation on the use of touchables: they cannot be used for assembly volumes, as Geant4 creates internally the physical volumes.

14.7 Using wildcards to get volume names

In many commands described in this guide, you give the name of a logical volume, physical volume or touchable so that GAMOS finds the corresponding Geant4 object. In case you want to apply your command to several volumes with similar names, you can use an asterisk that would mean 'any character'. For example if you have the volumes named CRYSTAL_BGO_1, CRYSTAL_BGO_2, CRYSTAL_LUYAP_1 and CRYSTAL_LUYAP_2

/gamos/SD/assocSD2LogVol GmSDSimple Calor CRYSTAL*
will associate a sensitive detector to the four volumes, while
/gamos/SD/assocSD2LogVol GmSDSimple Calor CRYSTAL_BGO_*

will associate a sensitive detector to the two volumes CRYSTAL_BGO_1 and CRYSTAL_BGO_2, and
/gamos/SD/assocSD2LogVol GmSDSimple Calor CRYSTAL_*1
will associate a sensitive detector to the two volumes CRYSTAL_BGO_1 and CRYSTAL_LUYAP_1.

14.8 Using particle names

Each particle type in Geant4 is identified by a unique name. No particle is created by Geant4 unless the user does it explicitly. At any time in your command file you can ask for a list of created particles with the command /run/particle/dumpList. If you do this after instantiating the GAMOS electromagnetic physics list, you will get the following list:

```
anti_nu_e, chargedgeantino, e+, e-
gamma, geantino, nu_e, opticalphoton
```

If you use the GAMOS hadronic physics list, you will get the following list:

```
B+, B-, B0, Bs0
D+, D-, D0, Ds+
Ds-, GenericIon, He3, J/psi
N(1440)+, N(1440)0, N(1520)+, N(1520)0
N(1535)+, N(1535)0, N(1650)+, N(1650)0
N(1675)+, N(1675)0, N(1680)+, N(1680)0
N(1700)+, N(1700)0, N(1710)+, N(1710)0
N(1720)+, N(1720)0, N(1900)+, N(1900)0
N(1990)+, N(1990)0, N(2090)+, N(2090)0
N(2190)+, N(2190)0, N(2220)+, N(2220)0
N(2250)+, N(2250)0, a0(1450)+, a0(1450)-
a0(1450)0, a0(980)+, a0(980)-, a0(980)0
a1(1260)+, a1(1260)-, a1(1260)0, a2(1320)+
a2(1320)-, a2(1320)0, alpha, anti_B0
anti_Bs0, anti_D0, anti_N(1440)+, anti_N(1440)0
anti_N(1520)+, anti_N(1520)0, anti_N(1535)+, anti_N(1535)0
```

```
anti_N(1650)+, anti_N(1650)0, anti_N(1675)+, anti_N(1675)0
anti_N(1680)+, anti_N(1680)0, anti_N(1700)+, anti_N(1700)0
anti_N(1710)+, anti_N(1710)0, anti_N(1720)+, anti_N(1720)0
anti_N(1900)+, anti_N(1900)0, anti_N(1990)+, anti_N(1990)0
anti_N(2090)+, anti_N(2090)0, anti_N(2190)+, anti_N(2190)0
anti_N(2220)+, anti_N(2220)0, anti_N(2250)+, anti_N(2250)0
anti_b_quark, anti_c_quark, anti_d_quark, anti_dd1_diquark
anti_delta(1600)+, anti_delta(1600)++, anti_delta(1600)-, anti_delta(1600)0
anti_delta(1620)+, anti_delta(1620)++, anti_delta(1620)-, anti_delta(1620)0
anti_delta(1700)+, anti_delta(1700)++, anti_delta(1700)-, anti_delta(1700)0
anti_delta(1900)+, anti_delta(1900)++, anti_delta(1900)-, anti_delta(1900)0
anti_delta(1905)+, anti_delta(1905)++, anti_delta(1905)-, anti_delta(1905)0
anti_delta(1910)+, anti_delta(1910)++, anti_delta(1910)-, anti_delta(1910)0
anti_delta(1920)+, anti_delta(1920)++, anti_delta(1920)-, anti_delta(1920)0
anti_delta(1930)+, anti_delta(1930)++, anti_delta(1930)-, anti_delta(1930)0
anti_delta(1950)+, anti_delta(1950)++, anti_delta(1950)-, anti_delta(1950)0
anti_delta+, anti_delta++, anti_delta-, anti_delta0
anti_k(1460)0, anti_k0_star(1430)0, anti_k1(1270)0, anti_k1(1400)0
anti_k2(1770)0, anti_k2_star(1430)0, anti_k2_star(1980)0, anti_k3_star(1780)0
anti_k_star(1410)0, anti_k_star(1680)0, anti_k_star0, anti_kaon0
anti_lambda, anti_lambda(1405), anti_lambda(1520), anti_lambda(1600)
anti_lambda(1670), anti_lambda(1690), anti_lambda(1800), anti_lambda(1810)
anti_lambda(1820), anti_lambda(1830), anti_lambda(1890), anti_lambda(2100)
anti_lambda(2110), anti_lambda_c+, anti_neutron, anti_nu_e
anti_nu_mu, anti_nu_tau, anti_omega-, anti_omega_c0
anti_proton, anti_s_quark, anti_sd0_diquark, anti_sd1_diquark
anti_sigma(1385)+, anti_sigma(1385)-, anti_sigma(1385)0, anti_sigma(1660)+
anti_sigma(1660)-, anti_sigma(1660)0, anti_sigma(1670)+, anti_sigma(1670)-
anti_sigma(1670)0, anti_sigma(1750)+, anti_sigma(1750)-, anti_sigma(1750)0
anti_sigma(1775)+, anti_sigma(1775)-, anti_sigma(1775)0, anti_sigma(1915)+
anti_sigma(1915)-, anti_sigma(1915)0, anti_sigma(1940)+, anti_sigma(1940)-
anti_sigma(1940)0, anti_sigma(2030)+, anti_sigma(2030)-, anti_sigma(2030)0
anti_sigma+, anti_sigma-, anti_sigma0, anti_sigma_c+
anti_sigma_c++, anti_sigma_c0, anti_ss1_diquark, anti_su0_diquark
anti_su1_diquark, anti_t_quark, anti_u_quark, anti_ud0_diquark
anti_ud1_diquark, anti_uu1_diquark, anti_xi(1530)-, anti_xi(1530)0
anti_xi(1690)-, anti_xi(1690)0, anti_xi(1820)-, anti_xi(1820)0
anti_xi(1950)-, anti_xi(1950)0, anti_xi(2030)-, anti_xi(2030)0
anti_xi-, anti_xi0, anti_xi_c+, anti_xi_c0
b1(1235)+, b1(1235)-, b1(1235)0, b_quark
c_quark, chargedgeantino, d_quark, dd1_diquark
delta(1600)+, delta(1600)++, delta(1600)-, delta(1600)0
delta(1620)+, delta(1620)++, delta(1620)-, delta(1620)0
delta(1700)+, delta(1700)++, delta(1700)-, delta(1700)0
delta(1900)+, delta(1900)++, delta(1900)-, delta(1900)0
delta(1905)+, delta(1905)++, delta(1905)-, delta(1905)0
delta(1910)+, delta(1910)++, delta(1910)-, delta(1910)0
delta(1920)+, delta(1920)++, delta(1920)-, delta(1920)0
delta(1930)+, delta(1930)++, delta(1930)-, delta(1930)0
```

```
delta(1950)+, delta(1950)++, delta(1950)-, delta(1950)0
delta+, delta++, delta-, delta0
deuteron, e+, e-, eta
eta(1295), eta(1405), eta(1475), eta2(1645)
eta2(1870), eta_prime, f0(1370), f0(1500)
f0(1710), f0(600), f0(980), f1(1285)
f1(1420), f2(1270), f2(1810), f2(2010)
f2_prime(1525), gamma, geantino, gluon
h1(1170), h1(1380), k(1460)+, k(1460)-
k(1460)0, k0_star(1430)+, k0_star(1430)-, k0_star(1430)0
k1(1270)+, k1(1270)-, k1(1270)0, k1(1400)+
k1(1400)-, k1(1400)0, k2(1770)+, k2(1770)-
k2(1770)0, k2_star(1430)+, k2_star(1430)-, k2_star(1430)0
k2_star(1980)+, k2_star(1980)-, k2_star(1980)0, k3_star(1780)+
k3_star(1780)-, k3_star(1780)0, k_star(1410)+, k_star(1410)-
k_star(1410)0, k_star(1680)+, k_star(1680)-, k_star(1680)0
k_star+, k_star-, k_star0, kaon+
kaon-, kaon0, kaon0L, kaon0S
lambda, lambda(1405), lambda(1520), lambda(1600)
lambda(1670), lambda(1690), lambda(1800), lambda(1810)
lambda(1820), lambda(1830), lambda(1890), lambda(2100)
lambda(2110), lambda_c+, mu+, mu-
neutron, nu_e, nu_mu, nu_tau
omega, omega(1420), omega(1650), omega-
omega3(1670), omega_c0, opticalphoton, phi
phi(1680), phi3(1850), pi(1300)+, pi(1300)-
pi(1300)0, pi+, pi-, pi0
pi2(1670)+, pi2(1670)-, pi2(1670)0, proton
rho(1450)+, rho(1450)-, rho(1450)0, rho(1700)+
rho(1700)-, rho(1700)0, rho+, rho-
rho0, rho3(1690)+, rho3(1690)-, rho3(1690)0
s_quark, sd0_diquark, sd1_diquark, sigma(1385)+
sigma(1385)-, sigma(1385)0, sigma(1660)+, sigma(1660)-
sigma(1660)0, sigma(1670)+, sigma(1670)-, sigma(1670)0
sigma(1750)+, sigma(1750)-, sigma(1750)0, sigma(1775)+
sigma(1775)-, sigma(1775)0, sigma(1915)+, sigma(1915)-
sigma(1915)0, sigma(1940)+, sigma(1940)-, sigma(1940)0
sigma(2030)+, sigma(2030)-, sigma(2030)0, sigma+
sigma-, sigma0, sigma_c+, sigma_c++
sigma_c0, ss1_diquark, su0_diquark, su1_diquark
t_quark, tau+, tau-, triton
u_quark, ud0_diquark, ud1_diquark, uu1_diquark
xi(1530)-, xi(1530)0, xi(1690)-, xi(1690)0
xi(1820)-, xi(1820)0, xi(1950)-, xi(1950)0
xi(2030)-, xi(2030)0, xi-, xi0
xi_c+, xi_c0,
```

These are the names that should be used in the commands that need a particle name. If you want to use hadrons, GAMOS also provides the possibility of grouping them, so that with a single name you can identify

the whole group. The groups defined are the following:

- **lightMeson: Mesons that only contain up and down quarks**

  ```
  pi+, pi-, pi0, eta, eta_prime, kaon+, kaon-, kaon0, kaon0L,
  kaon0S, a0(*), a1(*), a2(*), k(*), k1(*), k2(*), k_star(*),
  k0_star(*), k2_star(*), k3_star(*), anti_k(*), anti_k0(*),
  anti_k1(*), anti_k2(*), anti_k_star(*), anti_k2_star(*),
  anti_k3_star(*), b1(*), f0(*), f1(*), f2(*), f2_prime(*),
  h1(*), eta(*), eta2(*), phi(*), phi3(*), pi(*), pi2(*),
  rho(*), rho3(*)
  ```

- **charmMeson: Mesons that contain a charm quark**

  ```
  D+, D-, D0, anit_D0, Ds+, Ds-, J/psi
  ```

- **bottomMeson: Mesons that contain a bottom quark**

  ```
  B+, B-, B0, anti_B0, Bs0, anti_Bs0
  ```

- **meson All mesons**
- **lightBaryon: Baryons that only contain up and down quarks**

  ```
  proton, anti_proton, neutron, anti_neutron, N(*), anti_N(*),
  delta(*), anti_delta(*)
  ```

- **strangeBaryon: Baryons that contain a strange quark**

  ```
  lambda, anti_lambda, sigma0, anti_sigma0, sigma+,
  anti_sigma+, sigma-, anti_sigma-, xi0, anto_xi0, xi-,
  anti_xi-, omega-, anti_omega-, lambda(*), anti_lambda(*),
  sigma(*), anti_sigma(*), xi(*), anti_xi(*), omega(*),
  omega3(*)
  ```

- **charmBaryon: Baryons that contain a charm quark**

  ```
  lambda_c+, anti_lambda_c+, sigma_c0, anti_sigma_c0,
  sigma_c+, anti_sigma_c+ sigma_c++, anti_sigma_c++, xi_c+,
  anti_xi_c+, xi_c0, anti_xi_c0, omeca_c0, anti_omega_c0
  ```

- **baryon: All baryons**
- **ion: ions**

  ```
  GenericIon, alpha, He3, deuteron, triton
  ```

- **ALL: All particles**

Bibliography

[1] http://www.cern.ch/geant4

[2] http://geant4.web.cern.ch/geant4/support/userdocuments.shtml

[3] http://seal.cern.ch

[4] http://proj-clhep.web.cern.ch/proj-clhep

[5] http://root.cern.ch/

[6] Penelope - A Code System for Monte Carlo Simulation of Electron and Photon Transport, Workshop Proceedings Issy-les-Moulineaux, France, 5–7 November 2001, AEN-NEA

[7] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch02s02.html

[8] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch04.html#sect.Geom.Navig

[9] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch02s06.html

[10] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/PhysicsReferenceManual/html/ch02s05.html

[11] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/PhysicsReferenceManual/html/PhysicsReferenceManual.html

[12] http://www.slac.stanford.edu/comp/physics/geant4/slac_physics_lists/G4_Physics_Lists.html , http://geant4.web.cern.ch/geant4/physics_lists

[13] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/apas08.html#sect.G4MatrDb.NISTCmp

[14] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/Detector/geomSolids.html

[15] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch04s04.html#sect.Hits.G4Multi

[16] http://www.irs.inms.nrc.ca/BEAM/beamhome.html

[17] http://www-nds.iaea.org/phsp/phsp.htmlx

[18] Kawrakow I., Rogers D. W. O., Walters B. R. B. Large efficiency improvements in BEAMnrc using directional bremsstrahlung splitting. Medical physics 2004;31(10):2883-98.