
GERDA Scientific / Technical Report: GSTR-07-???

October 31, 2017

# MaGe - the GERDA/MAJORANA Monte Carlo framework

## A user's and developer's guide for GERDA members

Luciano Pandola[a], Markus Knapp[b], Kevin Kröninger[c], Daniel Lenz[c], Jing Liu[c], Xiang Liu[c], Jens Schubert[c], Manuela Jelen[c], Oleksandr Volynets[c]

[a] *INFN Laboratori Nazionali del Gran Sasso, Assergi (AQ), Italy*
[b] *Physikalisches Institut, Universität Tübingen, Germany*
[c] *Max-Planck-Institut für Physik, München, Germany*

Abstract

The Monte Carlo software MAGE , developed and maintained by the MAJORANA and GERDA Monte Carlo groups, is introduced. The focus is on the GERDA and common parts of the software. A user's guide for non-experts explains the installation and macro-based running of MAGE . The developer's guide shows the structure of the C++ code and covers more technical details.

# Contents

# Chapter 1

# Introduction

This note is addressed to new users of the GERDA Monte Carlo software MaGe and those users who would like to become software developers. It is written with the intention to (1) teach non-experts how to install and run MaGe and, for developers, to (2) introduce the guidelines and structure of the C++ code. For non-experts, only a basic knowledge of ROOTand C++ is needed in order to interpret the data created with the Monte Carlo package. Developers are assumed to be familiar with C++ and object-oriented programming.

The MaGe software is developed and used by Majorana and GERDA. This note is focused on the GERDA parts of the software and the parts common to both experiments.

The simulation of physics processes is motivated by several needs. The most important one addressed before the actual start of the experiment are an estimate on the experimental situation to be encountered. For GERDA this includes e.g. the evaluation of the background index and the calculation of the sensitivity. In addition, the simulation can guide the design of detector components. After data becomes available the simulation of the experimental setup has to be verified and can, in a second step, be used to interpret these data. This holds true for GERDA and for the test stands associated with it. It is these three requirements which led to development of the software package MaGe .

The choice of the framework within which the GERDA Monte Carlo simulation is realized is GEANT4 [1]. It is a well established simulation code with a large user community and a broad range of physics applications [2]. The software features the full chain of simulation, from the generation of particles to tracking and read-out. The physics processes implemented in GEANT4 are validated and/or parameterized [3, 4, 5]. As it provides full flexibility due to object-oriented C++-programming it meets the technical requirements for the realization of the needs described earlier.

The MaGe software is based on GEANT4 libraries and was initially written by the Majorana Monte Carlo group. It is now developed and maintained by both, the GERDA and Majorana Monte Carlo groups. As both collaborations aim at a measurement of neutrinoless double beta-decay of $^{76}$Ge, the physics processes encountered in the experiments are the same. With a common Monte Carlo software a duplication of work is minimized.The maintainance of the code through more developers and the physics validation is made easier as more data is and will be available. On the other hand, the MaGe software is flexible enough to provide both collaborations with their particular needs like the geometries, physics processes, etc.

MaGe features a non-expert mode of running via macros maintaining a maximum of flexibility. Geometries, physics processes or the output scheme are individually adjustable without recompiling the code. On the other hand, due to the object-oriented structure, it is easy for developers to add patches such a new test stand geometries, additional physics lists, etc.

Part I of this note is addressed to the non-experts. The installation of MaGe (chapter 2) and the documentation system (chapter 4) are described. Also, the running of simple macros to simulate events or to display the selected geometry are introduced (chapter 5). The generation of events which are needed as input parameters for the simulation is discussed in chapter 6.

Part II is is addressed to MaGe developers. It starts with the basics common to Majorana and GERDA such as compiling the code (chapter 9) and the implemented physics lists (chapter 11). This is followed by the GERDA specific parts like the description of the different I/O schemes available (chapter 12) and the GERDA detector components (chapter 14). The implementation of test stand environments is explained in chapter 15.

As mentioned earlier, MaGe is developed by people in Europe and the U.S. It is therefore important to obey some defined rules when it comes to coding. The three most important ones are

- Each developer is responsible for his or her code. If questions arise or bugs are reported it is their duty to maintain and, if necessary, modify the code.
- Each developer is asked to document his or her changes made to the code. This includes comments in the code itself (header and in-line) and a description in the documentation system.
- There are three parts which are separated and marked with prefixes. These are the common part (MG), the Majorana part (MJ) and the GERDA part (GE). It is understood that only the experiment specific (here: GERDA) code is to be edited independently. For changes to the common part both Monte Carlo groups have to be informed.

# Part I

# User's guide

# Chapter 2

# Installation

## 2.1 Dependencies

MaGe depends upon the following software packages:

1. GEANT4. An installation of CLHEP is required for GEANT4. MaGe works for CLHEP 1.9.x.x and later.
2. ROOT.
3. MGDO. MGDO stands for Majorana Gerda Data Objects. It defines the common data structure shared by MaGe and the PSS (pulse shape simulation) and PSA (pulse shape analysis) packages.
4. Optional accessories: PostgreSQL, GDML, etc.

Fig. 2.1 shows the dependencies graphically.

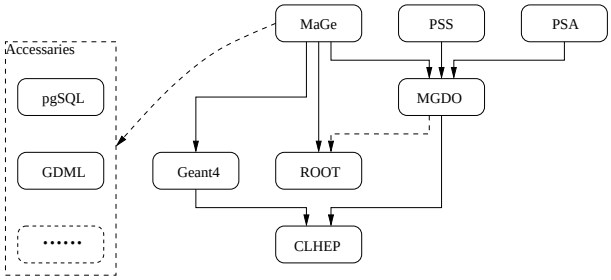


Figure 2.1: The dependencies of the software packages. Dotted lines indicate the optional dependencies.

The dependencies determine the order of the installation, that is,

1. CLHEP
2. GEANT4
3. ROOT
4. MGDO
5. MaGe

## 2.2 Getting the Packages

### 2.2.1 Getting CLHEP

The **C**lass **L**ibrary for **H**igh **E**nergy **P**hysics is required for running GEANT4. Go to the CLHEP web site to obtain the installation package (there are precompiled versions available):

```
http://proj-clhep.web.cern.ch/proj-clhep/DISTRIBUTION
```

download a version newer than 1.9.x.x (2.0.4.5 is recommended by Geant4.9.3p02).

### 2.2.2 Getting GEANT4

Go to the GEANT4 web site:

```
http://geant4.web.cern.ch/geant4/
```

download the newest version of GEANT4. If there are patches for this version, please download them as well. GEANT4 is under development all the time. The newer the version is, the less bugs there are. Move the tar balls of the source codes and the patches (if there are some) to the directory where you want to install GEANT4. Unpack the source codes (taking version 4.8.2 as an example):

```
tar xfvz geant4.8.2.gtar.gz
```

Unpack the patch in the same directory:

```
tar xfvz patch_geant4.8.2.p01.gtar.gz
```

which will add or modify some of the GEANT4 source files.

Download the newest data files on the same web page where you download GEANT4 source codes. Unpack them into a certain directory. Later in the GEANT4 installation process you will be asked the place where you put these data files.

### 2.2.3 Getting ROOT

To install ROOT from source you first have to get the tar file containing the source. This tar file can be found at:

```
http://root.cern.ch/twiki/bin/view/ROOT/Download
```

The files are named root-<version>.source.tar.gz.

1. Get access to the FTP area (substitute any FTP client and appropriate email address below):

   ```
   ftp root.cern.ch
   User: anonymous
   Password: <your-email-address>
   ```

2. Go to the directory, and prepare for binary transfer of files:

   ```
   ftp> cd /root
   ftp> bin
   ```

3. Get the sources tar-ball (substitute the appropriate version number), and exit FTP client:

```
ftp> get root_<version>.source.tar.gz
ftp> bye
```

4. Unpack the distribution:

```
gzip -dc root_<version>.source.tar.gz | tar -xf -
```

An alternative approach is to use the cern public SVN repository to get the latest version. See

```
http://root.cern.ch/drupal/content/subversion-howto
```

1. Get a specific version ($\geq$ 5.14/00), e.g.: version 5.26d:

```
svn co https://root.cern.ch/svn/root/tags/v5-26-00d/ root-52600d
```

2. Alternatively, checkout the head (development version) of the sources:

```
svn co https://root.cern.ch/svn/root/trunk root
```

In both cases you should have a subdirectory called "root" in the directory you ran the above commands in.

### 2.2.4 Introduction to SVN

SVN, known as SubVersioN control system, is a set of software used to implement a version control system which keeps track of all work and all changes in a set of files.
MaGe is a big set of codes used to do Monte Carlo simulations for GERDA and Majorana experiments. Majorana and GERDA use SVN to manage the MaGe source codes and related packages so that physicists from both sides can cooperate smoothly with each other on MaGe development.
This chapter serves as a brief introduction on how to use SVN to help in the development of MaGe .
Only a basic description will be given here, for more detailed information, please use the web.
If you know little about SVN, there is a very nice article about SVN in wikipedia:

```
http://en.wikipedia.org/wiki/Apache_Subversion
```

or (in German)

```
http://de.wikipedia.org/wiki/Apache_Subversion
```

#### Administrators

2004 - Xiang Liu set up the CVS server (xliu@mppmu.mpg.de).
2007 - 15th August. Jing Liu set up the CVS viewer and is now the CVS administrator. If you want to contact him, his e-mail is: jingliu@mppmu.mpg.de.
2009 - August. Jing Liu set up the SVN. The SVN administrator is Oleksandr Volynets: volynets@mppmu.mpg.de.

**Apply for an SVN account**

If you just want to browse the MaGe codes, follow one of these links:

```
http://www.mppmu.mpg.de/~gsvn/cgi-bin/viewvc.cgi
```

```
http://www.mppmu.mpg.de/~gsvn/viewsvn/
```

You will need to know the password. Ask it from your colleagues or email to

```
volynets@mppmu.mpg.de
```

If you want to check out a copy of MaGe and work on it, please fill out the form at

```
http://www.mppmu.mpg.de/~gsvn/cgi-bin/app.html
```

**Some Advice**

The most basic things are done. Now you can play with your copy of the codes. For a pleasant cooperation, please

1. make sure the whole package works well before committing your changes into SVN.
2. write comments for each of your check-ins into SVN. Don't be too lazy to do so, the other users will benefit - and in the end, you will, too.

### 2.2.5 Getting MGDO

MGDO package is located in the SVN repository. To get access to the SVN server, please follow the instructions in chapter 2.2.4.

If you have already setup your SVN account for MaGe , you can easily get a copy of MGDO by typing in your shell prompt:

```
svn co svn://pclg-soft.mppmu.mpg.de/MGDO/trunk MGDO
```

If you do this for the first time, the shell will ask you to enter your SVN password and will store this data to your

```
~/.svn
```

so the next time you don't have to enter the password again.

### 2.2.6 Getting MaGe

You can find the MaGe package in the SVN repository. To get access to the SVN server, please follow the instructions in chapter 2.2.4.

If you have already setup your SVN account properly for MaGe , you can easily get a copy of it.
Go to the directory in which you want to work with MaGe and type there:

```
svn co svn://pclg-soft.mppmu.mpg.de/MaGe/trunk MaGe
```

This will check out MaGe into the assigned directory which means it will copy all the codes in that directory.

## 2.3 Minimum Installation

Here is a guide aiming at helping you to compile MaGe and install the minimum software packages required by the MaGe compilation. The guide should work for any *nix type machine (i.e. Linux, Mac OS X, etc.) and it has been tested successfully on the following platforms:

1. Fedora Core 6 (FC 6) - x86 (Intel) and x86_64 (AMD64) chipsets
2. Ubuntu 9,10 - x86 (Intel) and x86_64 (AMD64) chipsets
3. Arch Linux - x86_64 (AMD64) chipsets
4. Gentoo Linux - x86 (Intel) chipsets
5. Mac OS X (10.4) - PowerPC and x86 chipsets

### 2.3.1 Installation of CLHEP

Unwind the source code tar ball in some relevant directory. Autoconfigure and automake will aready have been run. Determine where the files will be installed. Create a build directory that is NOT in the source code directory tree.

```
cd
/configure --prefix=
   (Note that files will be installed under /usr/local if you do not
    specify a prefix.)
make
   (Build temporary copies of libraries and executables.)
make check
   (Run the tests.)
make install
   (Copy libraries, headers, executables, etc. to relevant
    subdirectories under .)
```

A variety of options can be given to configure. Below is a list of the options that you are likely to find most useful.

```
  --help                  provides a partial list of options
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [default is /usr/local]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [default is the same as PREFIX]
  --disable-shared        build only static libraries
  --disable-static        build only shared libraries
  --enable-exceptions     use the CLHEP/Exceptions package
  --disable-exceptions    DO NOT use the CLHEP/Exceptions package
                          [--disable-exceptions is the default]
```

For more information, see

```
http://proj-clhep.web.cern.ch/proj-clhep/INSTALLATION/newCLHEP-install.html
```

A CLHEP CVS repository has also been set up:

```
http://clhep.cvs.cern.ch/cgi-bin/clhep.cgi/
```

There is also a user's guide/reference guide:

```
http://proj-clhep.web.cern.ch/proj-clhep/manual/UserGuide/
```

### 2.3.2 Installation of GEANT4

Please also read carefully the installation guide on the GEANT4 web site before you continue:

```
http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/InstallationGuide/html/index.htm
```

GEANT4 needs to be informed about the environment within which it will be installed. This can be done by setting a bunch of environment variables manually before the compilation. GEANT4 provides a script named *Configure* to simplify this process. Once you run the script, you will be asked some questions. GEANT4 will set up the environment variables itself according to your answers:

```
cd geant4.8.2
./Configure -build
```

The default answers for most of these questions work fine. However, you need to modify some of them according to the special environment of your computer. In order to make things clear, the whole compiling process printed on the computer screen after you entered the above command is shown here. (The words between < and > are the comments from the authors of this guide, which won't appear on the screen during the real compiling process.)

```
                --- Geant4 Toolkit Build  ---

Would you like to see the instructions? [n] y

This installation shell script will examine your system and ask you questions
to determine how the Geant4 Toolkit should be installed.  If you get stuck on
a question, you may use a ! shell escape to start a subshell or execute a
command.  Many of the questions will have default answers in square brackets;
typing carriage return will set the default.

On AFS it is allowed to specify either absolute or relative
paths (i.e. starting with the ~username construct).

[Type carriage return to continue]

The prompt used in this script allows you to use shell variables and backticks
in your answers. You may use $1, $2, etc...  to refer to the words in the
default answer, as if the default line was a set of arguments given to a
script shell. This means you may also use $* to repeat the whole default line.

Everytime there is a substitution, you will have to confirm.  If there is an
error (e.g. an unmatched backtick), the default answer will remain unchanged
and you will be prompted again.

Running 'Configure -d' will bypass nearly all the questions and
use the computed defaults (or answers saved in a configuration
previously generated).

Type 'Configure -h' for a list of options.
```

```
You may also start interactively and then answer '& -d' at any prompt to turn
on the non-interactive behaviour for the rest of the execution.

[Type carriage return to continue]

Much effort has been spent to ensure that this shell script will run on any
Unix system. If despite that you can't run Configure for some reason, you'll
have to set the proper environment variables by hand and follow the "manual"
installation as specified in the Geant4 Installation Guide.

[Type carriage return to continue]

Definition of G4SYSTEM variable is Linux-g++.
That stands for:

1) OS           : Linux

2) Compiler     : g++

To modify default settings, select number above (e.g. 2)
[Press [Enter] for default settings]

There exists a config.sh file. Shall I use it to set the defaults? [y]
Fetching default answers from your old config.sh file...

I can set things up so that your shell scripts and binaries are more portable,
at what may be a noticable cost in performance.  In particular, if you
ask to be portable, the following happens:

     1) Shell scripts will rely on the PATH variable rather than using
        the paths derived above.
     2) ~username interpretations will be done at run time rather than
        by Configure.

Do you expect to run these scripts and binaries on multiple machines? [n]

< Use the default answer here. The real meaning of the "multiple
machines" above is "multiple architectures". It's enough to compile
Geant4 only for YOUR architecture. >

*OPTIONS FOR GEANT4 INSTALLATION PATHS*
What is the path to the Geant4 source tree?  [/.hb/raidg01/gsoft/geant4.9.3]

< The exact meaning of this question is "Where are your unpacked
Geant4 source codes?" Say you unpack geant4.9.3.tar.gz into
/home/gx336-01/gsoft/usr, then the geant4 source codes should be in
/home/gx336-01/gsoft/usr/geant4.9.3 >
```

```
Where should Geant4 be installed?  [/.hb/raidg01/gsoft/geant4.9.3]

< The exact meaning of this question is "After a successful
compilation of the geant4 libraries, where do you like to copy the
libraries to?" >

Do you want to install all Geant4 headers in one directory? [y]

< The answer here does not effect the compilation of MaGe. However, if
you want to avoid long lists of Geant4 include paths, you can choose
to copy all its header files into one include directory. This is also
convenient for you to browse the source codes of these header
files. Besides, some of the packages, which can enhance the function
of Geant4 (GDML for example), need a yes here. If you want to install
these packages in the future, say yes. >

*GEANT4 LIBRARY BUILD OPTIONS*
Do you want to build shared libraries? [y]

< The static libraries will be included into the executable files of
your Geant4 applications, while the shared libraries won't. The shared
libraries will be used only when you run the executables. So using
shared libraries can dramatically decrease the size of the MaGe
executables. But you should let MaGe know where the shared libs are by
including the libraries directory in an environment variable named
LD_LIBRARY_PATH. If you have no idea what I am talking about, please
say no here. >

Do you want to build static libraries too? [n]

< If you say no to the previous question, you won't be asked this
one. >

Do you want to build global libraries? [y]

< Please say no. If you choose to use the static libraries before and
say yes to this question, the executable files of your Geant4
applications will become very large. >

Do you want to build granular libraries as well? [n]
Do you want to build libraries with debugging information? [n]

< Saying yes here will increase the size of the libraries. But you
might find the DEBUG mode very useful for your code development
later. If you don't care about the size of the libraries, give a yes
here. >

*CHECKS AND OPTIONS FOR GEANT4 PHYSICS DATA FILES*
```

```
Specify the path where the Geant4 data libraries are installed:
[/.hb/raidg01/gsoft/geant4_data]
checking for PhotonEvaporation2.0... yes
checking for RadioactiveDecay3.2... yes
checking for G4EMLOW6.9... yes
checking for G4NDL3.13... yes
checking for G4ABLA3.0... yes
checking for RealSurface1.0... yes

*CHECKS FOR CORRECT MAKE IMPLEMENTATION*
Checking for make... /usr/bin/make
Checking for gmake... /usr/bin/gmake
Checking if make is GNU make... yes
Checking if gmake is GNU make... yes

*CHECKS FOR REQUIRED EXTERNAL PACKAGE CLHEP*
checking for a CLHEP installation... /.hb/raidg01/gsoft

Is this the CLHEP installation you want to use?  [/.hb/raidg01/gsoft]

You can customize paths and library name of you CLHEP installation:

1) CLHEP_INCLUDE_DIR:           /.hb/raidg01/gsoft/include
2) CLHEP_LIB_DIR:               /.hb/raidg01/gsoft/lib
3) CLHEP_LIB:                   CLHEP
To modify default settings, select number above (e.g. 2)
[Press [Enter] for default settings]

*OPTIONS FOR GEANT4 USER INTERFACE MODULES*
Enable building of User Interface (UI) modules? [y]

< Say yes. UI (User Interface) is very useful. >

Enable building of the XAW (X11 Athena Widget set) UI module? [n]

< This enhances the Geant4 UI. If you have XAW installed in your
system, say yes; if you are not sure about that, say no. Without this
you still have a terminal-like UI. So don't worry about this too
much. >

Enable building of the X11-Motif (Xm) UI module? [n]

< This enhances the Geant4 UI. If you have XM Motif installed in your
system, say yes; if you are not sure about that, say no. Without this
you still have a terminal-like UI. So don't worry about this too
much. >

< The visualization drivers that do not depend on external libraries
```

```
are automatically incorporated into Geant4 libraries during the
compilation. The automatically incorporated visualization drivers are:
DAWNFILE, HepRepFile, HepRepXML, RayTracer, VRML1FILE, VRML2FILE and
ATree and GAGTree.

The OpenGL, OpenInventor and RayTracerX drivers are not incorporated
by default. Nor are the DAWN-Network and VRML-Network drivers, because
they require the network setting of the installed machine. In order to
incorporate them, you should say yes to the following questions: >

Enable building of the Qt UI module? [n]

*OPTIONS FOR GEANT4 VISUALIZATION DRIVERS*
Enable building of visualization drivers? [y]

< Say yes. Visualization of the detector geometries is very useful. If
you want to get some basic ideas of the Geant4 Visualization very
quickly, please search on Google "introduction to geant4
visualization". There is a not-that-long presentation which gives you
a big picture about the visualization.>

Enable building of the X11 OpenGL visualization driver? [y]

< In most of the cases, you have OpenGL installed in your system. So
saying yes here should be safe. But if you have some problems with
OpenGL, please consult to your system administrator. >

Enable building of the X11-Motif OpenGL visualization driver? [n]

< Make sure you have the XM Motif installed in your system before you
say yes. If you are not sure about this, say no. >

Enable building of the FukuiRenderer/DAWN visualization driver? [n]

< The DAWNFILE driver is automatically incorporated in Geant4. Saying
no to this question doesn't remove the DAWNFILE driver. Saying yes to
this question means that you want to install the DAWN-Network driver,
which is not necessary if you don't want to use DAWN through the
Internet. >

Enable building of the X11 OpenInventor visualization driver? [n]

< Make sure you have the OpenInventor installed in your system
before you say yes. If you are not sure about this, say no. >

Enable building of the X11 RayTracer visualization driver? [y]

< The compilation of the RAYTRACERX driver only depends on the Xlib,
```

```
which is available on most of the Linux machines nowadays. So saying
yes here should be OK. If you have some compilation problem caused by
this answer later, say no and try again. >

Enable building of the VRML visualization driver? [n]

< The VRML1FILE, VRML2FILE drivers are automatically incorporated in
Geant4. Saying no to this question doesn't remove these
drivers. Saying yes here means that you want to install the
VRML-Network driver, which is not necessary if you don't want to use
VRML through the Internet. >

You have selected to build one or more drivers that require OpenGL.
Specify the correct path (OGLHOME) where OpenGL is installed on your system.
It was found in /usr/X11R6. Press [Enter] to set this path or type the correct one.

< Now that Geant4 has already found the OpenGL, why not just use this
one. >

You can set '-' (without quotation) to CANCEL the OpenGL flag at all:
[/usr/X11R6]

*OPTIONS FOR GEANT4 OPTIONAL EXTENSION MODULES*
Enable the Geometry Description Markup Language (GDML) module? [y]

< The GDML is built-in visualization utility, but it can be also used to create
the geometry from external files. It might be useful if you need to create
a very simple setup. Advantage of using it as a visualizer is that it can modularize
the output. It means that you will be able to see all parts of the detector
as separate files, which is much faster than other drivers. Say yes here if you
did not understand a word from here. >

Enable build of the g3tog4 utility module? [n] < You mostly don't need g3tog4 >
Enable internal zlib compression for HepRep visualization? [y] < Say yes if you want to use

End of configuration phase.

Stripping down executable paths...

Creating configuration setup file...

WARNING: the generated configuration file
can be edited if necessary!
You can introduce any change to the configuration file
/.hb/raidg01/gsoft/geant4.9.3/.config/bin/Linux-g++/config.sh before the final installation
To do so, use a shell escape now (e.g. !vi /.hb/raidg01/gsoft/geant4.9.3/.config/bin/Linux-

Press [Enter] to start installation or use a shell escape to edit config.sh:
```

```
Now starting Geant4 libraries build...

On this machine the G4SYSTEM=Linux-g++
On this machine the G4INSTALL=/.hb/raidg01/gsoft/geant4.9.3
On this machine the G4INCLUDE=/.hb/raidg01/gsoft/geant4.9.3/include
On this machine the G4TMP=/.hb/raidg01/gsoft/geant4.9.3/tmp
On this machine the G4LIB=/.hb/raidg01/gsoft/geant4.9.3/lib
On this machine the G4LEVELGAMMADATA=/.hb/raidg01/gsoft/geant4_data/PhotonEvaporation2.0
On this machine the G4RADIOACTIVEDATA=/.hb/raidg01/gsoft/geant4_data/RadioactiveDecay3.2
On this machine the G4LEDATA=/.hb/raidg01/gsoft/geant4_data/G4EMLOW6.9
On this machine the G4NEUTRONHPDATA=/.hb/raidg01/gsoft/geant4_data/G4NDL3.13
On this machine the G4ABLADATA=/.hb/raidg01/gsoft/geant4_data/G4ABLA3.0
On this machine the CLHEP_BASE_DIR=/.hb/raidg01/gsoft
On this machine the CLHEP_INCLUDE_DIR=/.hb/raidg01/gsoft/include
On this machine the CLHEP_LIB_DIR=/.hb/raidg01/gsoft/lib
On this machine the CLHEP_LIB=CLHEP
On this machine G4UI_NONE is not set - following UIs will be built:
On this machine G4VIS_NONE is not set - following drivers will be built:
On this machine the G4VIS_BUILD_DAWN_DRIVER=1
On this machine the G4VIS_BUILD_OPENGLX_DRIVER=1
On this machine the G4VIS_BUILD_RAYTRACERX_DRIVER=1
On this machine the G4VIS_USE_DAWN=1
On this machine the G4VIS_USE_OPENGLX=1
On this machine the G4VIS_USE_RAYTRACERX=1
On this machine the OGLHOME=/usr/X11R6
On this machine the XMFLAGS=
On this machine the XMLIBS=
On this machine the XMFLAGS=
On this machine the XAWFLAGS=
On this machine the XAWLIBS=
On this machine the G4LIB_BUILD_GDML=1
On this machine the XERCESCROOT=/.hb/raidg01/gsoft
On this machine the G4LIB_BUILD_ZLIB=1
On this machine the G4LIB_USE_ZLIB=1
On this machine the G4LIB_BUILD_SHARED=1

Starting build...

************************************************************
 Installation Geant4 version : geant4-09-03
 Copyright (C) 1994-2009 Geant4 Collaboration
************************************************************

Making dependency for file src/G4ios.cc ...

<... Here is a very long compiling process. Be patient! ...>
```

```
Weeding out paths and files ...
Making libname.map starter file ...
Making libname.map ...
  Reading library name map file...
  Reading dependency files...
  Checking for circular dependencies...
  Reordering according to dependencies...
  Writing new library map file...
Libraries installation completed !

#####################################################
# Your Geant4 installation seems to be successful!
# To be sure please have a look into the log file:
# /.hb/raidg01/gsoft/geant4.9.3/.config/bin/Linux-g++/g4make.log
#####################################################

< Finally, it's ended! >
```

If everything works fine for you, you can go ahead to install GEANT4 now:

```
./Configure -install
```

which will copy the compiled libraries and header files to the installation directory you specified before.

Once libraries have been installed, the user's environment must be correctly set up for the usage of the GEANT4 toolkit. The Configure script provides a way to check the existing installation and provide the correct configuration for the user's environment. Configuration scripts env[.sh/.csh] can be generated, and should be sourced by the final user in order to configure his/her environment according to the performed installation. To generate the configuration scripts, the user should run Configure placed in the installation area, as follows:

```
./Configure
```

This will generate the shell script env.csh (env.sh for bash) to be sourced or integrated into the shell login script (.tcshrc or .bashrc). The shell script will be generated in the same directory where you run the "./Configure" command. You can customize it to specify for example the proper working directory through the variable $G4WORKDIR (Please refer to the later chapters for details).

### 2.3.3 Installation of ROOT

Don't forget to source the shell script env.csh (env.sh for bash) generated by GEANT4 Configure script before you go on.

Please read the installation instruction on ROOT web site carefully. An example is given below:

```
cd /remote/gx336-01/usr/root-5.14 (where the source codes are)
export ROOTSYS=/remote/gx336-01/usr/root-5.14
./configure --disable-cern
make
make install
```

Before you use ROOT, please don't forget to add the following codes into your .bashrc

```
export PATH=$ROOTSYS/bin:$PATH
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
```

Or if you use tcsh, add the following to your .tcshrc

```
setenv PATH $ROOTSYS/bin:$PATH
setenv LD_LIBRARY_PATH $ROOTSYS/lib:$LD_LIBRARY_PATH
```

### 2.3.4 Installation of MGDO

#### Configuring

MGDO is configured by going to the MGDO directory and running the configure command:

```
cd $MGDO
./configure
```

Please set your shell environment as mentioned in the previous sections before compiling MGDO. This way, MGDO configuration system can find all the libraries that it needs automatically. Otherwise you have to pass these information to the configuration script by extra configuration options. To get a list of these options, please type in your shell prompt:

```
./configure -h
```

#### Compiling

After a successful configuration, you can then compile MGDO by typing in your shell prompt:

```
make
```

A successful compilation will create a subdirectory named "lib" under the MGDO directory. You will find symbol links to all the library files under this subdirectory.

#### Setting the Environment

In order to let the MaGe configuration system know where MGDO is installed, you have to set a new shell environment variable named $MGDODIR

```
export MGDODIR=/path/to/MGDO
```

in your ".bashrc" if you use bash, or

```
setenv MGDODIR /path/to/MGDO
```

in your ".tcshrc" if you use tcsh.

If you are going to use the shared libraries of MGDO, you have to add the path of the MGDO shared libraries to $LD_LIBRARY_PATH:

```
export LD_LIBRARY_PATH=/path/to/MGDO/lib:$LD_LIBRARY_PATH
```

if you use bash, or

```
setenv LD_LIBRARY_PATH /path/to/MGDO/lib:$LD_LIBRARY_PATH
```

if you use tcsh.

### 2.3.5 Installation of MaGe

#### Configuring

A new configure/make environment has been created to facilitate the installation process for MaGe :

Go into the MaGe base directory and run the command

```
cd $MaGe
./configure [--disable-database]
```

The –disable-database flag should be passed to the configure script if you are a GERDA user. This flag disables support for the PostgreSQL database used by users of MaGe from the Majorana group. If the flag is not passed to configure, and you do not have PostgreSQL on your system, then the configure script will fail.

The configure script attempts to determine what type of system you are using and automatically generates files important to the installation. The script will fail if any of the required software packages are not installed correctly and will output a message giving a suggested course of action. Once configure has finished successfully, MaGe is ready to be compiled.

It is important to note that if any settings have been changed (i.e. changes to relevant environment variables, changes in installations of the above packages, etc.) then the configure script needs to be rerun and the installation should be made clean (see 'Cleaning up', 2.3.5).

If you are a GERDA user working on one of the machines of the Max-Planck-Institute in Munich you should check whether you sourced the file *~gsoft/setup.sh* since this file sets you a valid environment. Just type

```
source ~gsoft/setup.sh
```

before compiling MaGe, or add it to your .bashrc file.
In case you are a c-shell user, type

```
source ~gsoft/setup.csh
```

or add it to your .cshrc file.

#### Compiling

After a successful configure, one should run

```
make
```

from the MaGe base directory.

This will compile and install binaries into $G4WORKDIR/bin/$G4SYSTEM.

#### Cleaning up

If the installation needs to be rerun, type from the MaGe base directory

```
make clean
```

This will remove temporary installation files and binaries for a fresh compile.

## 2.4 Accesseries of MaGe

### 2.4.1 Overview

MaGe can be enhanced by compiling against some extra packages.

- PostgreSQL. It is used by the Majorana Monte Carlo group. To enable it in MaGe, simply **not pass** "–disable-database" option to the configuration process.
- GDML (**G**eometry **D**escription **M**arkup **L**anguage). GDML is an XML-based language which describes the geometries of detectors in an application-independent format. More information about releases and getting started on GDML are available at:

```
http://gdml.web.cern.ch/GDML/
```

### 2.4.2 Installing PostgreSQL

*This section will undergo updates soon.*

If you are a Majorana user of MaGe , then you should install PostgreSQL (pgsql) to be able to run with all the Majorana geometries. If you do not wish to run a local pgsql server on your machine and only interface with the one located at pdsf, then it is sufficient to install the developer version of pgsql on your machine. You can do this by using your favorite package manager (e.g. apt-get, fink, yum, etc.) and this is probably the easiest method since it will set up your environment for you.

If you wish to run a local pgsql server, you must install the full postgresql package (not just the developer tools). Again, it is simplest to install this package with your package manager. Most, if not all, of these distributions come with methods that will automatically start your server of pgsql at boot time. Please check the documentation that comes with your distribution.

If you did not use a package manager, ensure that the following environment variables are set after a successful pgsql installation:

```
setenv PGSQL_BASE_DIR /path/to/pgsql
setenv PGSQL_LIB_DIR $PGSQL_BASE_DIR/lib
```

Setting these environment variables is unnecessary and not recommended if the installation was performed with a package manager.

If you are running on a laptop, it is likely that you will want to have a local version of the database installed. This way you can run the code even when you are on a plane. To do this, do the following:

1. Start the server: you have to either start the server manually every time or setup the server to start at boot. There should be methods for both in your distribution. Please consult your distribution documentation.
2. Get the majorana database by doing (replace [username] with your own username) the following (you may have to sudo some of these commands):

```
createuser [username] (answer yes to all questions)
createdb majorana
pg_dump -h128.3.11.122 -p5432 -Uakbarm majorana > mjdbtemp.txt
sed -e 's/akbarm/[username]/' mjdbtemp.txt > mjdb.txt
psql majorana < mjdb.txt
rm mjdb.txt mjdbtemp.txt
```

Test it out by doing

```
psql majorana
```

and at the pgsql prompt type

```
psqlprompt> \dt;
```

This should list tables. Quit with the command:

```
psqlprompt> \q;
```

3. Change MJDBProperties.txt so that

```
databaseURL=127.0.0.1
userName=[username]
databaseName=majorana
port=5432
```

MaGe should now be able to access your database. Test it by running in databaseApp (located in $G4WORKDIR/bin/$G4SYSTEM).

### Installation of XERCES-C++

GDML used to be an external package, but now it is integrated to the Geant4 toolkit. GDML depends on XERCES-C++, an XML parser written in a subset of C++. It makes it easy to give your application the ability to read and write XML data. You can find information about XERCES-C++ and the installation of it at:

```
http://xerces.apache.org/xerces-c/
```

The easiest way to install XERCES-C++ is shown below:

```
tar xfvz xerces-c-3.0.1.tar.gz
cd xerces-c-3.0.1
./configure --prefix=/path/to/install
make
make install
export XERCESCROOT=/path/to/install
```

# Chapter 3

# Getting Started

## 3.1 The Gerda README: Getting to know Macros

MaGe is a simulation package based on GEANT4. It can be run using macro files (see chapter 5).

A lot of different macro files exist for various purposes ranging from drawing a test stand setup to simulating the background that is to be expected in the final setup of the GERDA experiment.

A README file was created to support new users in getting comfortable with the usage of the GERDA macros.

The GERDA README file can be found in the GERDA CVS repository under

```
/MaGe/macros/Gerda/README
```

An introduction gives a short overview on visualisation drivers used for different purposes, e.g. for presentations or for applications which require interactive features. The macros used in the MaGe simulations for the GERDA experiment are listed alphabetically and their purpose, the geometrical setup and the expected output are described in a few sentences.

# Chapter 4

# Documentation

*tba*

# Chapter 5

# Macro commands

*all*

The MaGe package is a single executable program that is configured and run by using macro files. The usage is simple: MaGe is run by typing at the command prompt

```
/path/to/MaGeExecutable/MaGe /path/to/macros/SampleMacro.mac
```

where /path/to/MaGeExecutable/ is the directory of the MaGe executable and /path/to/macros/ is the directory of the sample macro. Of course this assumes you already have a macro containing your required configurations for MaGe .
You can also run a macro file in the MaGe interactive mode. This provides the possibility to put the commands from the macro in one by one. This is very helpful for understanding the way the commands work and for debugging a macro script. Furthermore, you can implement additional commands inbetween the macro commands (e.g. if you want to change a command, but don't want to rewrite the macro). You start the interactive mode by typing:

```
/path/to/MaGeExecutable/MaGe
```

Whenever MaGe shows "PreInit>" or "Idle>", you may add new commands. There are commands which can only be given after the "PreInit>"-prompt and some which require the "Idle>"-prompt.

This chapter is all about understanding and creating macro files so you can get MaGe to do what you want it to do.

This chapter attempts to cover many of the macro commands one might use while running MaGe . Many are also covered in other chapters, with varying levels of detail. The appendix **??** contains a list of all messenger commands for inclusion into macro files that have been created for use in MaGe , along with explanations. The complete list of messenger commands native to GEANT4 that one can also use is beyond the scope of this manual.

## 5.1 The Structure of a Macro File

A macro file contains all the important messenger commands that MaGe needs in order to run a simulation. Some of these parameters have "default" values and don't necessarily need to be set. The bare minimum requires you to specify a geometry, initialize the run manager, set up a primary generator of particles, and press "go." This would be a boring run, and you will in all likelihood want to specify much more than this, including how the simulated data is processed and written to files. The structure of a macro file is divided into two parts, as follows:

- Set up any special manager commands (/MG/manager/)
- Specify what sort of physical processes / lists you want (/MG/processes/)
- Define the geometry you want to use (/MG/geometry/)
- Specify the event action commands, such as how the data are processed and written (/MG/eventaction/)
- Specify what type of primary event generator you want to use (/MG/generator/)

At this point, you've told MaGe all about the geometry setup, the physics you're interested in, how you want the data managed and written, and what generator to use. Now you need to initialize the GEANT4 kernel with the messenger command

```
/run/initialize
```

This tells MaGe /GEANT4 that it's time to build the geometry and calculate cross sections for the physics processes you've told MaGe to care about.
You will notice that the prompt changes from "PreInit>" to "Idle>" with this initialization.
The structure of the macro file continues:

- Specify which particles you want to use, and their attributes
- Set up any visualization scheme you want to use

Now it's time to start the run with the command

```
/run/beamOn n
```

where "n" is the number of events you want to run.

This list isn't inclusive of all the types of commands you can give by macro, but it includes all the most important ones. More information about the commands, the choice of values and the default values can be found by going into interactive MaGe mode (as indicated in the introduction of this chapter) and typing:

```
PreInit> cd /MG/
PreInit> ls
```

You can follow the directory tree to the last leaves, the macro commands. Information about these can be obtained by:

```
PreInit> help [macrocommand]
```

The next few sections will go into the commands in more detail.

## 5.2 MGManager Messengers

The manager messengers allow you to do two basic things. You can set the verbosity of the log you want from the output:

```
/MG/manager/mjlog [fatal] [error] [warning] [routine] [trace] [debugging]
```

"Fatal" is the least verbose, while "debugging" is the most. It is worth noting that these logs are specific to MaGe . You can also fiddle with native GEANT4verbosity messengers like

```
/run/verbose j
/event/verbose k
/tracking/verbose l
```

where increasing integers of j, k, and l lead to increasingly detailed output about the run, event, and tracking. The default is 0 for silent, with maximum output for run and event at 2 and maximum output for tracking at 5. Be warned, setting tracking above 2 is like drinking from a firehose, so think about how many events you really want to run.

You can also specify what sort of random number generator seed you want. Your best bet is probably

```
/MG/manager/seedWithDevRandom
```

which reads a seed from /dev/random. The other options are

```
/MG/manager/heprandomseed [integer]
/MG/manager/useInternalSeed [integer]
```

## 5.3 MGProcesses Messenger

The MGProcesses Messenger can set the simulation "realm" to double-beta decay, dark matter or cosmic rays.
The difference between the realms lies in the secondary production cutoff. For the first two realms the distances specified in the code are keyed to germanium, and they provide a double-beta production cutoff of 100 keV and a dark matter production cutoff of 1 keV. In the cosmic rays realm, the cuts are different in the volumes belonging to to the "SensitiveRegion" and the other volumes. In particular, the cuts for the volumes in the sensitive region are the same than for the double-beta realm, while they are more relaxed everywhere else, in order to save CPU time (avoiding the precise tracking of particles in the uninteresting parts of the setups).

These cutoffs specify the energy below which no secondaries will be created (with a few exceptions, such as if the secondary particle would annihilate, such as positrons, or if the particle is relatively close to a volume boundary).

The goal of establishing these different realms is to increase the speed of the simulation when possible. Since the interesting energy range of double-beta decays is much greater than that of dark matter interactions, the simulation should not have to spend so much time tracking secondary particles down to the lower energy

If the "SensitiveRegion" is not defined or no logical volume is registered in it, the cut-per-region approach has no effect in the simulation. The cuts are the same everywhere and correspond to about 140 keV (gammas) and 7.5 MeV (e-) in germanium. The code to register one logical volume in the "Sensitive Region" is

```
G4Region* sensitiveRegion = new G4Region(``SensitiveRegion'');
sensitiveRegion->AddRootLogicalVolume(myLogicalVolume);
```

When a volume is registered to the region, all its daughters are included too. Don't forget to `delete sensitiveRegion;` in the destructor of the geometry class.
The MGProcesses Messenger is used as such:

```
/MG/processes/realm [BBdecay] [DarkMatter] [CosmicRays]
```

The default realm is "BBdecay".

Geant4 has the capability to generate and track optical photons, e.g. originated by scintillation or by Cerenkov emission. In the default MaGe Physics List optical photons are not generated. They can be switched on with the messenger command

```
/MG/processes/optical [true] [false]
```

that has to be given before the `/run/initialize`. In this case, be sure that the geometry definition contains the optical properties of materials (refraction index, absorption length, scintillation yield) and of surfaces (reflection index, etc.).
Geant4 provides two alternative sets of Electromagnetic Processes, called Standard (Std) and Low Energy (LowE). The LowEnergy processes include atomic effects (as fluorescence) and can track gammas, electrons and positrons down to 250 eV; the drawback of LowE processes is that they require a longer computing time. By default, the Electromagnetic processes in the MaGe Physics List are the LowE ones. One can switch to the Standard processes (quicker and less precise) with the messenger command

```
/MG/processes/lowenergy [true] [false]
```

It must be given before the `/run/initialize`.

## 5.4 MGGeometry Messengers

The MGGeometry Messengers are used to specify which detector is to be simulated. The number of geometries in use is growing, and many of them have their own bevy of messenger commands. This chapter won't go into all of them, but they can be found in Appendix **??**The main messenger commands are:

```
/MG/geometry/detector [clover] [cloverinNaIbarrel] [solidBlock] [GerdaArray]
                      [idealCoax] [MunichTestStand] [MJ57Banger] [MPIK_LArGe]
                      [GS_LArGe] [UWLArGe] [SLACBD] [cloverInShield]
                      [MJRDCrystalColumn] [MJRDCryostat] [MJ17A]
                      [MJRDBasicShield] [MCNPTest] [CERN_NA55] [WIPPn]
                      [LLNL8x5] [GeometryFile] [Corrado]
/MG/geometry/addMaterial
/MG/geometry/database
/MG/geometry/dumpG4materials
/MG/geometry/geometryFileName
/MG/geometry/setDetectorOffset
/MG/geometry/setWorldHalfLen
/MG/geometry/WorldMaterial
/MG/geometry/EventBias/
```

```
/MG/geometry/detector
```

is used to select which geometry has to be simulated: the geometry name has to be chosen among the registered ones. There is *not* a default geometry provided. The user hence *must* issue the `/MG/geometry/detector` command before starting the run initialization. If it is not done, the MaGe executable will crash.

```
/MG/geometry/database [true] [false]
```

allows the user to choose if the material table has to be read from the Majorana database (=true) (default) or from the local class materials/MGGerdaLocalMaterialTable (=false).

```
/MG/geometry/dumpG4materials
```

can be used to print on screen all the defined materials. It works properly only after run initialization (namely, in the "Idle>" state of GEANT4).

```
/MG/geometry/geometryFileName [filename]
```

will set the file name for a file-based geometry.

```
/MG/geometry/setDetectorOffset
```

allows you to place the constructed detector at a specific position in the WorldVolume. The default is the origin.

```
/MG/geometry/setWorldHalfLen
```

allows you to set the size of the WorldVolume. The default is a 50 m × 50 m × 50 m cube.

```
/MG/geometry/WorldMaterial
```

determines the material used to construct the WorldVolume. The default is air.

```
/MG/EventBias/
```

is a messenger directory with commands allowing the use of event biasing in runs. For example, one may choose to use importance sampling, as well as the algorithm to use.

### 5.4.1 Definition of user-specific materials

The command

```
/MG/geometry/addNewMaterialfilename filename
```

is used to read a material definition from an external file. It works only if the database is switched off.
The material definition should be as follows:
*First line*: material name, density (g/cm$^3$), number of elemental components
The name of the material cannot contain blank spaces and must be different from the materials already defined in the material table.
The following lines (one for each component) contain:
Element name, Element symbol, Z (double), A(double) and mass fraction.
The information written in the file is used only if the element table does not already contain an element with the same name. An example of a definition file is the following:

```
TelluriumOxide 5.9 2
Oxygen O 8.0 16.000 0.201
Tellurium Te 52.0 127.60 0.799
```

### 5.4.2 Debugging of geometries

A command is available to check for overlapping volumes within the geometry.

```
Idle> /MG/geometry/CheckOverlaps
```

The program will report on the screen the overlaps between physical volumes with their mother volume and/or with other physical volumes at the same hierarchy level. No information is displayed for those volumes that are correctly placed. Additional verbosity from the CheckOverlaps tool, can be obtained with the command

```
Idle> /MG/geometry/OverlapVerbosity true/false
```

(default is false).

### 5.4.3 GDML-macros

The commands used for GDML are:

```
(reading commands, must be executed before initialization:)
/MG/geometry/GDML/sourceFile [sourcefileName]

(writing commands:)
/MG/geometry/GDML/schemaPath /path/to/gdml/schema (optional)
/MG/geometry/GDML/outputName [outputName.gdml] (optional)
/MG/geometry/GDML/outputFormat [true][false] (optional)
/MG/geometry/GDML/modularizeLevels [1][2][3]... (optional)
/MG/geometry/GDML/write (only work after initialization)
```

You can have a look at the complete set of GDML options in the interactive MaGe mode by typing:

```
MaGe
PreInit> cd /MG/geometry/GDML/
PreInit> ls
```

All the commands are displayed now. For more detailed information, type:

```
PreInit> help [macrocommand]
```

Where [macrocommand] is any of the commands displayed.

```
PreInit> /MG/geometry/GDML/sourceFile [sourcefileName]
```

The first command specify the position of the GDML-file.

```
Idle> /MG/geometry/GDML/schemaPath /path/to/gdml/schema
```

The second command sets your own GDML schema definition file path. /path/-to/gdml/schema is the path to your schema definition. The default value is set to:

```
http://service-spi.web.cern.ch/service-spi/app/releases/GDML/GDML_3_0_0/schema/gdml.xsd.xs
```

```
Idle> /MG/geometry/GDML/outputName [outputName.gdml]
```

Where [outputName] is the name of the GDML output file. By default, it is the same as the detector name that you specified using the command:

```
Idle> /MG/geometry/detector [Name]
```

```
Idle> /MG/geometry/GDML/outputFormat [true][false]
```

Set the format of the GDML output file. Candidates are "[true][false]".
true- The names of volumes, materials, etc. in the GDML output file will be concatenated with their logical address in hexadecimal format (save if you have duplicated names).
false - The names will correspond exactly to the ones you have in GEANT4.
By default, true is used.

```
Idle> /MG/geometry/GDML/modularizeLevels [1][2][3]...
```

Specifies the geometry levels that you want to modularize.

```
Idle> /MG/geometry/GDML/write
```

Dumps a Geant4 geometry to a corresponding GDML file. Be careful: this command can *only* be used after the "Idle>"-prompt.

### Example Macros

Have a look at the MaGe macro-directory to find a well commented example for GDML-macros:

```
cd /Path/to/MaGe/macros/GDML
```

The "GDML" subdirectory contains the macro files "g42gdml.mac" and "gdml2g4.mac".
"g42gdml.mac" is an example macro to show how to dump a GEANT4 geometry into a GDML file. You run it using the method described in the beginning of chapter 5, in this case:

```
$ \$ $MaGe g42gdml.mac
```

You want to test what each of the commands in a macro does? Use the interactive mode as described above.
"g42gdml.mac" is well commented inside the script itself. Nevertheless, some commands and the GDML options will be explained here.
The macro runs with the detector "Corrado" (as described above). The initialization can be accelerated by not using hadronic processes which corresponds to setting the following variable to "true":

```
PreInit> /MG/processes/useNoHadPhysics true
```

Of course, you have to run the initialization process:

```
PreInit> /run/initialize
```

With the initialization, the prompt in MaGe changes from "PreInit>" to‘ "Idle>". The tree structure of the geometry can be obtained by:

```
Idle> /vis/open ATree
Idle> /vis/ASCIITree/verbose 1
```

This gets the names of the physical and logical volumes which are dumped and printed out by:

```
Idle> /vis/drawTree
```

Now you have the tree structure of your geometry printed out. You may e.g. specify the levels you want to be modularized (see later).

```
Idle> /MG/geometry/GDML/schemaPath /path/to/gdml/schema
```

This sets your own GDML schema definition file path. The default schema introduced above is also used in case of "g42gdml.mac".

```
Idle> /MG/geometry/GDML/outputName somename.gdml
```

Where "somename" is the name of the GDML output file. The default name is "Corrado.gdml".

```
Idle> /MG/geometry/GDML/outputFormat 2
```

The names of volumes, materials etc. will be the names used in GEANT4.

```
Idle> /MG/geometry/GDML/modularizeVolumes Crystal_log Spider_log ...
```

"Crystal_log" and "Spider_log" are the logical volumes contained in "Corrado.gdml" that are modularized according to "g42gdml.mac". They are daughter volumes of the "Corrado"-volume.

```
Idle> /MG/geometry/GDML/modularizeLevels [1][2][3][4]...
```

Specifies the geometry levels that you want to modularize. If you only use "2", only the second level of the tree will be used, not the first one and not the third one.
With:

```
Idle> /MG/geometry/GDML/write
```

the Geant4 geometry is written to a corresponding mother GDML file, "Corrado.gdml" and two modularized daughter GDML files, "Crystal_log.gdml" and "Spider_log.gdml".
"gdml2g4.mac" shows exemplarily how to use a GDML file in a GEANT4 simulation.

You have to type

```
$ \$ $MaGe
```

to go into the MaGe interactive mode. The command

```
PreInit> /MG/geometry/GDML/sourceFile Crystal_log.gdml
```

specifies "Crystal_log.gdml" as the GDML file that is used. You can replace this file with any GDML-file. If you want MaGe to use the whole geometry, just insert the mother file of "Crystal_log.gdml", in this case "Corrado.gdml".
Initialization is run without hadronic processes:

```
PreInit> /MG/processes/useNoHadPhysics true
PreInit> run initialize
```

The geometry is drawn with the visualization driver **O**pen**GL** **S**tored **X**:

```
Idle> /vis/open OGLSX
Idle> /vis/viewer/set/viewpointVector 0 1 1
Idle> /vis/drawVolume
```

The generator used for simulating particles is the **G**eneral **P**article **S**ource. It is described more detailled in section 6.10. In this case it shoots electrons with an energy of 100 MeV onto the crystal.

```
Idle> /MG/generator/select SPS
Idle> /gps/particle e-
Idle> /gps/energy 100 MeV
Idle> /gps/position 1 1 1
Idle> /gps/direction 0 1 1
```

```
Idle> /vis/scene/add/trajectories
Idle> /run/beamOn 10
```

Now you should see a detector setup with particle trajectories.

"g42gdml.mac" gives out three GDML-files: "Corrado.gdml", "Crystal_log.gdml" and "Spider_log.gdml". These can be displayed running the ROOT-macro "drawGDML.C":

```
$ \$ $ROOT
ROOT[].x drawGDML.C(''filename.gdml'')
```

"filename.gdml" is any of the three files produced by "g42gdml.mac".

## 5.5 MGOutput Messengers

In order to retrieve the relevant information from the simulation, the user should also specify which output scheme has to be used. Output classes can give either a ROOT file or an ASCII file.
The commands to define the output schema and to retrieve the name are:

```
/MG/eventaction/rootschema name
/MG/eventaction/getrootschema
```

A number of alternative output schemes are presently available in MaGe: LANLCloverNoPS, LANLCloverSteps, LANLCloverInNaIBarrel, solidBlock, GerdaArray, GerdaArrayWithTrajectory, LArGeNoPS, G4Steps, MCEvent, GerdaTestStand, GerdaTestStandSimple, GerdaArrayOptical, GerdaArrayCalibration, MPIK_LArGe, GerdaTeststandCoincidence, SLACBD, Shielding, GSS, MCNPTest, CERN_NA55, GerdaTeststandSiegfried, GerdaTeststandSiegfriedCoincidence, NeutronYield and DetectorEfficiency.
This must be done with care, as the output class can be tied to the detector choice, and there is so far no internal checking to make sure the output class and detector choice.
The name of the output file is also set via messenger, using the command

```
/MG/eventaction/rootfilename filename
```

The same command works for ROOT and ASCII output file.
MaGe correctly runs (e.g. for testing purposes) even if the output class is not specified.

```
/MG/eventaction/reportingfrequency
```

determines how often the output tells you which event it is processing.

```
/MG/eventaction/writeOutFileDuringRun
/MG/eventaction/writeOutFrequency
```

allow for writing to the file during the run. The default frequency that it writes is the same as the reporting frequency.

## 5.6 MGGenerator Messengers

A simulation is only useful if you actually fire particles at things. Using the generator macros is how you set this up. The only generator messenger you need to issue before `/run/initialize` is

```
/MG/generator/select [PNNLiso] [RDMiso] [TUNLFEL] [G4gun] [decay0] [cosmicrays]
                     [musun] [SPS] [neutronsGS] [wangneutrons] [AmBe]
                     [MuonsFromFile] [sources4a] [GSS]
```

which selects the generator engine you will use. There is much more information about these generators in Chapter 6

The other basic messenger commands:

```
/MG/generator/name
```

returns the name of the generator.

```
/MG/generator/position
```

This command will set the initial particle position (for point distributions only) for all the generators.

```
/MG/generator/confine [noconfined] [volume] [volumelist]
/MG/generator/volume [noconfined] [volume] [volumelist]
/MG/generator/volumelist
/MG/generator/volumelistfrom
/MG/generator/volumelistto
```

allow you to confine primary vertices to within physical volumes in your geometry. For example, a generator that samples a

```
/MG/generator/gsspositionsfile
/MG/generator/gsseventnumber
/MG/generator/gssoffset
```

volume position

## 5.7 Cone Cut for secondary gammes

Tracks of any gammas (primary and secondary) can be killed if the momentum vector of the photon does not point towards some position within a certain angle.
This can be used to run the simulation more efficient if the volume in which the gammas are generated is far away from crystal array.

Example:

```
/MG/output/killGammasOutsideCone true
/MG/output/GammaConeCut_ArrayCenter 0 0 19.5 cm
/MG/output/GammaConeCut_StartCutRadius 80 cm
/MG/output/GammaConeCut_MaxAllowedOpeningAngle 45 deg
```

When the Gamma-Cone-Cut feature is enabled, all gammas not pointing to the position *GammaConeCut_ArrayCenter* within the angle given by *GammaConeCut_MaxAllowedOpeningAngle* will be killed, if the primary vertex of the photon is outside a spherical range *GammaConeCut_StartCutRadius* of this position.

# Chapter 6

# Generators

*Luciano*

## 6.1 Choosing a generator

Presently eleven alternative generators have been implemented in MaGe. The actual generator to be used has to be set via the macro command

```
/MG/generator/select [PNNLiso] [RDMiso] [TUNLFEL] [G4gun] [decay0]
  [cosmicrays] [musun] [SPS] [neutronsGS] [wangneutrons] [sources4a]
  [AmBe] [MuonsFromFile] [GSS]
```

G4Gun is the default G4ParticleGun provided by Geant4.

## 6.2 AmBe Generator

The americium beryllium (AmBe) generator is a point source of neutrons and gammas. The AmBe neutron spectrum used comes from Figure 5 of Marsh, Thomas, and Burke, "High resolution measurements of neturon energy spectra from Am-Be and Am-B neutron sources", Nucl. Inst. and Meth. A 366 (1995) 340-348. Gamma energies are taken from the $^{12}$C level structure, at from http://www.nndc.bnl.gov/nudat2/getdataset.jsp?nucleus=12C. Choice of gammas depending on neutron energy is taken from Figure 3 of Geiger et al., Nucl. Inst. and Meth. 131 (1975) 315.

## 6.3 PNNL Radioactive Decay Chain Generator

### 6.3.1 General description

The PNNL radioactive decay chain generator consists of a set of G4 classes for sampling charged particles (beta or alpha) and associated gammas arising from the decay of any radioisotopes in a user-specified radioactive decay chain. Information returned by the generator is currently limited to the charged particle type (beta+/- or alpha), charged particle energy, number of "cascade" gammas, and a list of gamma energies. Currently the code is limited to linear decay chains, i.e. multiple branch points in a decay chain yielding distinct nuclear species are not (easily or naturally) supported. (If only gammas are of interest, however, a multiple-branch kludge is possible if the branches merge in a common daughter.) The generator code requires as input the gamma cascade branching fractions for each radioisotope in the decay chain (one

Table 6.1: Summary table of the generators available in MaGe

| Events to be generated | Generator |
|---|---|
| Double beta decay ($2\nu$ or $0\nu$) | decay0 |
| Radioactive contaminations | G4gun<br>PNNLiso<br>RDMiso<br>decay0<br>SPS |
| Cosmic-ray muons | cosmicrays<br>musun<br>MuonsFromFile |
| Cosmogenic neutrons | neutronsGS (for LNGS)<br>wangneutrons |
| Neutrons from fission and ($\alpha$,n) | neutronsGS (for LNGS)<br>Sources4a |
| Neutrons (and $\gamma$-rays) from AmBe | AmBe |
| TUNFEL beam<br>Surface Contamination | TUNFEL<br>GSS |

ascii-format file per radioisotope), and a master "decay chain" ascii-format file specifying the number of radioisotopes in the chain, the decay half-lives for each isotope and the names of each isotope's cascade file. The cascade branching fractions can be calculated using the tool CrazyInput, written by J.E. Ellis and A. Valsan of PNNL. As currently implemented, setup of the generator requires running CrazyInput once for each radioisotope in the decay chain of interest. The ascii-format files output by CrazyInput are then collected in a single directory and read by the generator upon initialization. Terminology used in this paragraph is explained in more detail below.

The main object/data structure in the generator is a DecayChain object, representing e.g. the Thorium decay chain (consisting of the radionuclides Th-232, Ra-228, Ac-228, ..., Pb-208). The DecayChain's primary responsibilities are to serve as a container for radioisotopes, embodied in Radioisotope objects, to calculate the relative decay probability for each member isotope of the chain based upon the grow-in age of the source, and to sample appropriately from each Radioisotope object's decay. Each Radioisotope object can decay by either beta(+/-) or alpha emission, accompanied by the emission of one or more "cascade" gammas from the daughter nucleus. A cascade is defined as a single decay scheme of a single radioisotope: For example, beta decay to a given excited state in the daughter nucleus, followed by emission of N gammas of energy E1, E2, ..., EN. (Currently no time information is associated with the cascade gammas, which are assumed emitted in coincidence.) In general, a single radioisotope can decay via multiple cascade schemes. Within the generator framework, each Radioisotope object thus contains one or more Cascade objects, each of which contains all of the information necessary for setting up and sampling from e.g. a beta decay energy distribution specific to that cascade decay scheme. Each Cascade object also contains a list of coincident gammas associated with the cascade. Each Radioisotope object stores a list of branching fractions for the set of cascade it contains, and its primary responsibility is to sample from the set of cascades associated with it when interrogated by the DecayChain object for the current event. In turn, a

particular Cascade object "knows" how to sample from the beta distribution associated with it. (Mono-energetic alphas can also be associated with a cascade scheme.) The logical structure of the generator is thus a nesting of data structures: a DecayChain object contains one or more Radioisotope objects, which in turn contain one or more Cascade objects. These objects are described in the classes MGGeneratorPNNLDecayChain, MGGeneratorPNNLRadioisotope, and MGGeneratorPNNLCascade, respectively.

The three main tasks of the PNNL generator are as follows: (1) Upon initialization, read a set of user-supplied files defining the decay chain and specifying the set of cascade schemes appropriate to each radioisotope in the chain. (2) Determine the relative activities (via standard Bateman equation calculations) of each radioisotope in the decay chain, given a user-supplied list of the half-lives of each isotope in the chain and the grow-in time, T, appropriate for the material. It is assumed that T=0 corresponds to a material consisting entirely of the isotope at the head of the chain. (3) For event sampling, select the next radioisotope to decay (based upon relative activities of each isotope in the chain), and for that isotope, select a given cascade decay scheme (based on cascade branching fractions determined by the CrazyInput tool). The energy of the decay beta (or the single energy of an alpha) is then sampled from an appropriate distribution and the list of corresponding decay gammas (if any) retrieved from the appropriate Cascade object. The generator returns a CascadeEvent object to the calling routine (e.g. the GeneratePrimaries member function of the PrimaryGeneratorAction class) for the current event, and all information (e.g. particle types and energies) relevant to the radioactive decay can be extracted from this simple data structure in preparation for adding particles to the tracking stack.

### 6.3.2 Using the generator

#### Setup

Setting up the generator currently requires (1) creating a set of ascii-format files (one for each radioisotope in the decay chain of interest) using the CrazyInput cascade branching-fraction calculation tool; (2) collecting these files in a convenient sub-directory; and (3) creating a master "decay chain file" to describe to the generator the number of isotopes in the chain, the half-life of each isotope, and the files listing the cascade branching fractions for each isotope.

An example cascade branching-fraction file for Tl-208 is displayed below.

```
208.0    81.0    10
0.49499E+00     1.79626    1    1    2
 0.58319   2.61453
0.23966E+00     1.28556    1    1    3
 0.51077   0.58319   2.61453
0.12545E+00     1.51889    1    1    2
 0.86056   2.61453
0.96121E-01     1.51889    1    1    3
 0.27736   0.58319   2.61453
0.18924E-01     1.03304    1    1    3
 0.76313   0.58319   2.61453
0.11496E-01     1.03304    1    1    4
 0.25261   0.51077   0.58319   2.61453
0.39991E-02     1.07420    1    1    4
 0.21140   0.51077   0.58319   2.61453
0.37982E-02     1.28556    1    1    2
 1.09390   2.61453
0.31466E-02     1.28556    1    1    3
 0.23336   0.86056   2.61453
```

```
0.24109E-02     1.28556    1    1    0
```

The first line of the file lists A, Z, and the number of cascades contained in the file. Each cascade consists of two lines of input. The first line lists, in order, the branching fraction for the cascade, the endpoint energy for the beta decay spectrum (or the alpha-particle energy) in MeV, the charged particle type (0 for alpha, 1 for beta, 2 for positron), the angular momentum change, "delta(L)" (in units of hbar), suffered by the nucleus in the decay, and the number of gammas associated with the de-excitation cascade emitted by the daughter nucleus. The second line of the cascade description lists the energies of the cascade gammas (MeV). In the example file, 10 cascade decay-schemes for Tl-208 are listed. Reading from the second and third lines of the file, the first decay scheme is beta decay with an endpoint energy of 1.79626 MeV and delta(L) of 1. The beta is accompanied by 2 cascade gammas of energy 0.58319 and 2.61453 MeV, respectively.

In general, for a decay chain consisting of N radioisotopes, each radioisotope must be described by a cascade branching fraction file in the format shown above. In addition to these files, an ascii-format "decay chain file" must be supplied to inform the generator of the general decay chain structure. An example file for the Th-232 decay chain is exhibited below.

```
10
 1  Th-232    1.41e10   y  dat/cascades_232Th.dat
 2  Ra-228    5.75      y  dat/cascades_228Ra.dat
 3  Ac-228    6.15      h  dat/cascades_228Ac.dat
 4  Th-228    1.910     y  dat/cascades_228Th.dat
 5  Ra-224    3.64      d  dat/cascades_224Ra.dat
 6  Rn-220   55.0       s  dat/cascades_220Rn.dat
 7  Po-216    0.15      s  dat/cascades_216Po.dat
 8  Pb-212   10.64      h  dat/cascades_212Pb.dat
 9  Bi-212   60.6       m  dat/cascades_212Bi.dat
10  Tl-208    3.05      m  dat/cascades_208Tl.dat
```

The first line in the decay chain file lists the number of radioisotopes in the chain (10 in this example). Each radioisotope is described in a single line. The first entry in a radioisotope line is an (arbitrary) integer index which is not used by the code but may be useful to the user. The second entry is a string isotope label, used only in diagnostic messages output by the generator. The third and fourth entries specify the decay half-life and the units of the entry: (y,d,h,m,s) for (year, day, hour, minute, and second) respectively. The final entry lists the name and path of the cascade branching fraction file for the radioisotope, relative to the directory in which the G4 code is to be run. (This "main" directory will contain e.g. a macro file containing G4 messenger commands, as well as the master decay chain file.)

In summary, note that the user must (currently) construct the master decay chain file by hand. The cascade branching-fraction files, one for each radioisotope, are generated by running the CrazyInput tool. Once these files are available and the decay chain file has been constructed, generator setup for the particular decay chain of interest is complete.

### Initializing the generator

Once the required set of input files has been prepared, initialization of the generator is controlled by two messenger commands in a G4 macro file, as demonstrated in the example which follows:

```
/generator/PNNL/init Th232_DecayChain.dat
/generator/PNNL/setSourceAge 5.0
```

In this example, the command "/generator/PNNL/init Th232_DecayChain.dat" informs the primary event generator code that the PNNL radioactive decay chain generator will be used in the G4 run, and initializes the generator with the name of the decay chain control file. Typically this control file will reside in the same directory as the G4 macro file controlling the batch run. The command "/generator/PNNL/setSourceAge 5.0" sets the source grow-in age to 5.0 years in this example. Recall that the relative activities of each radioisotope in the decay chain are calculated using the Bateman equations, with the assumption that time T=0 corresponds to a source consisting solely of the material at the head of the decay chain (Th-232 in this example.) If the setSourceAge messenger command is not issued, the default value for the grow-in age is 0. In this case, only radioactive decays from the isotope at the head of the chain are sampled by the generator.

In summary, the general messenger command structure is as follows:

```
/generator/PNNL/init [master decay chain filename]
/generator/PNNL/setSourceAge [source grow-in age in years]
```

## 6.4 Decay0 Generator

DECAY0 is a primary generator developed by Vladimir I. Tretyak. It can generate two-neutrinos double beta decay events, and neutrinoless double beta decay according to a large number of theoretical models and background events from the most common $\gamma/\beta$ sources (low-probability transitions are included); it can also produce generic $\beta$ spectra and mono-energetic particles. The most recent version of DECAY0 accounts for the angular correlation between $\gamma$-rays emitted in $^{60}$Co and in the two-neutrino double beta decay of $^{76}$Ge to the $0^+_1$ excited state of $^{76}$Se (1122 keV).
DECAY0 accounts for the kinematic of the final state (i.e. momenta of all the particles) and for the time, which is relevant for chains or delayed de-excitation (e.g. $^{137}$Cs+$^{137m}$Ba). It is a `Fortran` standalone program and the MG generator class (MGDecay0Interface.hh) reads from the ASCII files dumped by it: the user must take care that the file contains enough events for the Geant4 simulation (otherwise the program will give an exception).
DECAY0 is not a part of MAGE , but the `Fortran` source code for the most recent version is included in the CVS repository, in the directory `GETools`. It must be compiled and linked with the CERN Fortran libraries.
The DECAY0 generator is instantiated with the macro command:

```
/MG/generator/select decay0
```

and the file to be read is set with the command

```
/MG/generator/decay0/filename myfile.txt
```

The file name must be set before the `/run/beamOn`.

## 6.5 Cosmic rays generator

It is used to generate high-energy muons with the characteristic spectrum of cosmic rays in underground sites. The primary position is sampled from a disk of radius $r$ (default = 8 m) placed at the height $h$ (default = 8.1 m). Both these values can be changed via messenger.
The energy spectrum is sampled using the analytical parametrization of Lipari and Stanev [6],

Table 6.2: Example table of some dangerous muons for MaGe . The header must not be included into the file.

| pos(x) | pos(y) | pos(z) | dir(x) | dir(y) | dir(z) | energy |
|---|---|---|---|---|---|---|
| 1199.75 | 69.6281 | 810 | -0.836506 | -0.0649792 | -0.544091 | 370888 |
| 200.717 | 541.989 | 810 | -0.301985 | -0.469251 | -0.829825 | 914548 |
| 305.095 | -292.536 | 810 | -0.355895 | 0.34182 | -0.869769 | 1.31997e+06 |
| -1476 | -727.268 | 810 | 0.812514 | 0.390033 | -0.433238 | 570688 |
| -474.305 | -731.501 | 810 | 0.421681 | 0.602593 | -0.677545 | 213939 |

which applies to rock depths between 1 km w.e. and 10 km w.e. The spectral index of the muon spectrum can be 3.7 (standard), 2.7 (Prompt sources, e.g. charm decay) or 2.0 (Exotic sources). The defaults are 3400 m w.e. (Gran Sasso depth) and 3.7 respectively.
The angular spectrum is sampled according to the measurements performed at Gran Sasso by the MACRO experiment. The MACRO data are loaded from an ASCII file in the directory `$MGGENERATORDATA`. If the file is not present, the distribution is sampled according to $\cos^{-1}\theta$, as described in Ref. [6]. In the latter case, a warning message is printed on the screen.

The cosmic ray generator is instantiated with the macro command:

```
/MG/generator/select cosmicrays
```

The commands to change the parameters of the generator are:

```
/MG/generator/cosmicray/radius radius [unit]
/MG/generator/cosmicray/height radius [unit]
/MG/generator/cosmicray/depth  depth [unit]
/MG/generator/cosmicray/index index
```

for radius, height, depth (with unit) and spectral index respectively.

## 6.6 The muons from file generator

This generator was developed, to allow the user to select muons from a file, instead of using the cosmic rays generator, that was introduced in the previous chapter 6.5. It reads the starting position, the direction and the energy of the muon from a file with a structure like:

```
pos(x) pos(y) pos(z)  dir(x) dir(y) dir(z)  energy
```

with the three dimensions of the starting position *pos(x,y,z)*, the three elements of the direction vector *dir(x,y,z)* and the *energy* of the muon. It was mainly used to simulate a set of muons, that were identified as dangerous, i.e. during their simulation, an energy deposition around 2 MeV in the germanium crystals was registered. An example of a small file is given in tabular **??**.

The muons from file generator is initiated with the macro command:

```
/MG/generator/select MuonsFromFile
```

The commands to change the parameters of the generator are:

```
/MG/generator/MuonsFromFile/filename file
```

to change the name of the file, from which the muons are read in. The files have to be stored into the MGGENERATORDATA folder.

## 6.7 MUSUN Generator

MUSUN is a primary generator developed by Vitaly Kudryavtsev and described in [7]. It is `Fortran`-based and it is able to track high-energy muons through large thicknesses of rock, to obtain the muon spectrum (energy and direction) inside an underground Laboratory. It can also take into account the depth profile of the specific Laboratory, if available, in order to reproduce the possible asymmetries in the azimuth distribution. The MaGe generator is istantiated with the command:

```
/MG/generator/select musun
```

It reads the event kinematic parameters (energy and direction) of the primary muon from an ASCII file, whose name (and path, if it is not in the current directory) has to be provided using the macro command

```
/MG/generator/musun/filename myfile.txt
```

The primary position is sampled from a disk of radius $r$ (default = 8 m) placed at the height $h$ (default = 8.1 m). Both these values can be changed via messenger, using the same commands described in the previous section for the CosmicRay generator, i.e.

```
/MG/generator/cosmicray/radius radius [unit]
/MG/generator/cosmicray/height height [unit]
```

## 6.8 NeutronsGS generator

This generator produces neutrons induced by natural radioactivity and interactions of cosmic-ray muons in the rock. Energy is sampled according to the spectrum at Gran Sasso reported in Ref. [8]. The generator is instantiated with the command

```
/MG/generator/select neutronsGS
```

The data are read from ASCII files, that are looked for in the directory `$MGGENERATORDATA`. It is possible to generate either the low-energy component from fission and ($\alpha$,n) from the rock or the high-energy component induced by muon interactions in the rock. The latter is the default. The spectrum to be generated can be selected via messenger:

```
/MG/generator/neutrons/fission [true] [false]
```

If the value `true` is assigned, the low-energy component produced by natural radioactivity in the rock is considered. The fission/($\alpha$,n) neutrons are considered to be isotropic, while the angular distribution of the high-energy component is sampled according to [8]. The position of the primary tracks is sampled from the upper surface, the lower surface or the lateral surface of a cylinder; it can be set with the command:

```
/MG/generator/neutrons/direction [roof] [floor] [walls]
```

The dimensions of the cylinder are 8.0 m radius and 8.0 m height. The defaults can be changed with the commands

```
/MG/generator/neutrons/radius radius [unit]
/MG/generator/neutrons/height height [unit]
```

## 6.9 Sources-4A Generator

SOURCES-4A is a `Fortran`-based generator of neutrons produced in a given material by fission and by ($\alpha$,n) reactions on light elements. It has been developed by the LANL; a large number of cross section tables for various elements has been recently added in the context of the ILIAS EU work. The code calculates the energy spectrum of the emitted neutrons; the angular distribution is assumed to be isotropic. The explicit tracking of the neutron (out of its source, for instance) is not done by SOURCES-4A but has to be performed using MAGE and GEANT4.
The output from SOURCES-4A is the absolute energy distribution of neutrons; the total production rate is expressed in neutrons $cm^{-3}$ $s^{-1}$. The name of the output file from SOURCES-4A containing the absolute energy spectrum is `tape7`. The MAGE generator is istantiated with the command

```
/MG/generator/select sources4a
```

In the default case, the data file to be read is called `tape7` and it is looked for in the current directory. The file name can be alternatively specified (with its path) using the command

```
/MG/generator/sources4a/filename myfile.txt
```

The generator also works if the normalized energy spectrum (called `tape8` in the default case) is used.
The only limitation of the generator is that the lower egde of the lowest-energy bin must be zero. The generator does not take care of the position sampling of the neutron; in most case, it will be appropriate to use the uniform position sampling in a given volume, which is described in Sect. 6.11.

## 6.10 General Particle Source

*Note: The GPS has many, many options, not all of which are covered in this manual. For a complete list, look at the* GEANT4 *source code. Once you give that up, look at the much more accessible GPS user's manual at http://reat.space.qinetiq.com/gps/. It explains everything and also contains many example macros.*

The GEANT4 General Particle Source (GPS) was designed to supplant the G4Gun as a general source for any particles you might want to use. It also allows for position sampling, angular and energy distributions, and multiple sources. To use the GPS, the messenger command is

```
/MG/generator/select SPS
```

The GPS can sample from several types of planes(circle, annulus, ellipsoid, square, rectangle) and surfaces/volumes(sphere, ellipsoid, cylinder, parallepiped). Messenger commands set the sizes, positions, and attributes of the original positions. Some examples are

```
/gps/position 1.0 2.0 2.5 cm
/gps/pos/type Plane
/gps/pos/shape annulus
```

```
/gps/pos/radius 5.0 cm
/gps/pos/inner_radius 3.0 cm
```

The angular and energy distributions can also be customized. The choices for angular distributions are isotropic, cosine-law, or user-defined. You can also choose parallel beam or a diverging accelerator beam. For energy distributions, you can choose from mono-energetic, linear, exponential, power-law, gaussian, bremsstrahlung, black body, and cosmic diffuse gamma ray. Some example messenger commands utilizing these(taken from Example 13 in the GPS user's manual, http://reat.space.qinetiq.com/gps/examples/examples.htm):

```
/gps/ang/type cos
/gps/ene/type Exp
/gps/ene/min 2. MeV
/gps/ene/max 10. MeV
/gps/ene/ezero 6.
```

The GPS uses the GEANT4 native radioacative decay module (RDM). To set up an ion and set limits on it's decay,

```
/gps/ion <Atomic Number> <Mass Number> <Ionic Charge> <Excitation Energy>
/grdm/nucleusLimits <Amin> <Amax> <Zmin> <Zmax>
```

To set up a radioactive ion, say $^{238}$U, and let it decay to $^{222}$Rn, the messenger commands would be

```
/gps/particle ion
/gps/ion 92 238 0 0
/gps/energy 1e-20 eV
/grdm/nucleusLimits 222 238 86 92
```

## 6.11 Position sampling

Only a few of the generators described above take care of sampling the initial position of the primary particle. Notably, they are: TUNFEL, MUSUN, CosmicRays, NeutronsGS, and SPS/GPS(General Particle Source). For all the others, the source is considered to be a fixed point. The default is the origin of the coordinate system, but it can be set (for any generator) with the macro command

```
/MG/generator/position coordinates [unit]
```

The command has no effect on the TUNFEL, MUSUN, CosmicRays and NeutronsGS generators, that internally overwrite the position of the primary particle. For the G4gun generator only, the command

```
/gun/position coordinates [unit]
```

can also be used. The General Particle Source (GPS) has all of the features of the G4gun, but also much more in the way of position sampling and angular/energy biasing.

### 6.11.1 Uniform sampling

For some applications (e.g. simulation of internal contamination) it is useful to sample the position of the primary event uniformly within a given physical volume or a given skin surface.

The MaGe package includes an algorithm for this purpose. The confinement level (noconfined, volume, surface) is set with the command:

```
/MG/generator/confine [noconfined] [volume] [surface] [volumelist]
```

The default is `noconfined` (e.g. primary generated in a fixed position). If the confinement level is set to `volume`, the physical volume has to be set with the command

```
/MG/generator/volume volumename
```

where `volumename` is the name of an existing physical volume. This command has to be called after /run/initialize. If the volume name is not unique (e.g. replicas exist), the copy numbered "0" will be taken into account. The surface confinement level has not been implemented in this context, but an alternate method exists and is covered in Section 6.11.2.

#### Confinement in geometrical volumes or surfaces

It is also possible to sample the position of the primary event uniformly in a geometrical volume (e.g. a spherical shell, or a cylinder) or a geometrical surface (e.g. a disk, or a sphere). Although this functionality is already provided by the General Particle Source generator (see sect. 6.10), the General Particle Source treats also direction sampling and it is not possible to use the tools described in Sect. 6.12, as the sampling of the direction of the primary particle within a given angle from a target position. The position sampling over a geometrical volume/surface, as provided by MaGe , can be coupled with the tools for direction sampling.

The confinement level is set with the command

```
/MG/generator/confine [noconfined] [geometricalvolume] [geometricalsurface]
```

The shape for the geometrical volume/surface can be set with the commands

```
/MG/generator/geomSampling/volume/name [Sphere] [Cylinder]
/MG/generator/geomSampling/surface/name [Sphere] [Cylinder] [Disk]
```

The geometrical volume/surface defined in this way may include one or more MaGe physical volumes. The center of the target volume/surface is given as

```
/MG/generator/geomSampling/volume/center x y z [cm]
/MG/generator/geomSampling/surface/center x y z [cm]
```

The default position is the origin of the reference frame. Dimensions of volumes are given using the messenger:

```
/MG/generator/geomSampling/volume/innerSphereRadius x [cm]
/MG/generator/geomSampling/volume/outerSphereRadius x [cm]

/MG/generator/geomSampling/surface/sphereRadius x [cm]

/MG/generator/geomSampling/surface/innerDiskRadius x [cm]
/MG/generator/geomSampling/surface/outerDiskRadius x [cm]

/MG/generator/geomSampling/surface/innerCylinderRadius x [cm]
/MG/generator/geomSampling/surface/outerCylinderRadius x [cm]
/MG/generator/geomSampling/surface/cylinderHeight x [cm]

/MG/generator/geomSampling/volume/innerCylinderRadius x [cm]
/MG/generator/geomSampling/volume/outerCylinderRadius x [cm]
/MG/generator/geomSampling/volume/cylinderHeight x [cm]
```

Default values for the inner radii is 0; all other dimensions (outer radius, and possibly heights) must be specified before the `/run/beamOn`.

### 6.11.2 Surface sampling

GEANT4 cannot effectively sample complex surfaces(in particular, boolean solids composed of unions of elementary solids). The Generic Surface Sampler (GSS) was created to get around this. Randomly sampled points from surfaces in MaGe can be acquired by using the GSS. These points are stored as x, y, and z coordinates in a ROOTtree (GSSTree), within a ROOTfile. The actual physics simulation is then run, using this file as a source for primary vertices corresponding to the surface of interest. Note that this involves running MaGe twice, first to get the position points, and then for the physics. The GSS can sample one or many different volumes within a geometry at once. The macro commands to use the GSS and to name the GSS ROOTfile are

```
/MG/eventaction/rootschema GSS
/MG/eventaction/rootfilename GSSfilename.root
```

The names of the physical volumes that are to be sampled are given to the GSS by messenger commands. For example, the following messenger commands would add three volumes; two crystals in the MajoranAreference design and the outer cryostat.

```
/MG/io/gss/addVolume InnerCryostatPhysical
/MG/io/gss/addVolume Crystal1CrystalColumn2
/MG/io/gss/addVolume Crystal0CrystalColumn13
```

For a given run of the GSS, all of the sampled volumes will have the same surface density of sampled points. Alternatively, you can of course sample individual volumes separately.

The algorithm for GSS requires the radius and center (centre) of a sphere that completely envelops all of the volumes to sample. There are two methods to do this. Messenger commands can set these values directly:

```
/MG/generator/gss/origin x y z
/MG/generator/gss/boundingR R
```

where x, y, z, and R must of course have recognized units of length. If all of the volumes to be sampled are contained within some kind of holding container (like an array of crystals in a cryostat), or if there is only one volume to be sampled, then there is an easier way to handle the origin and radius. The messenger command

```
/MG/generator/gss/boundvol EncompassingPhysicalVolume
```

will ensure that the radius and origin are properly set. Note, "EncompassingPhysicalVolume" is the name of the physical volume that contains all of the volumes to be sampled, and it need not be sampled itself.

The algorithm also requires the maximum number of possible intersections that a line would make with the sampled volumes. For example, Figure 6.1 shows three semi-coax germanium crystals in an enclosure. For sampling the three crystals, the maximum number of intersections a line would make with a sampled surface is 12. If we were to also sample the enclosure, the number would be 14. To set the number of intersections, the messenger command is

```
/MG/io/gss/setMaxIntersections aNumber
```

MaGe will throw a warning if you pick a number that's too low. Some quick words about the efficiency of the GSS: setting the maximum number of intersections too high will cause the GSS

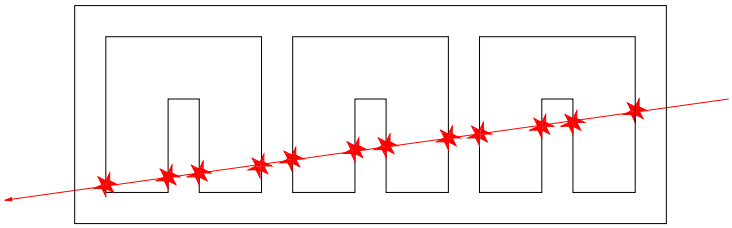

Figure 6.1: Cartoon showing intersections of a geantino with sampled surfaces for the GSS

to unnecessarily throw away more events, requiring a larger number of initial events to achieve the desired number of points. For a given number of initial events, the number of achieved random points will vary depending on the geometry of the sampled volumes. Sampling a sphere or pseudo-spherical assembly would be quite efficient, whereas sampling a long, thin pole would not be.

Once the desired surface points have been gathered by the GSS into a ROOTfile, it is a simple matter to use them for a physics simulation. To read the positions from a file, the messenger command is

```
/MG/generator/gsspositionsfile GSSfilename.root
```

The GSS ROOTfiles will have some number of position points. There is an option to start at a point that is not the first one,

```
/MG/generator/gsseventnumber aNumber
```

but the default is to start at the beginning of the file. If you add more than one file, they will be chained together.

## 6.12 Direction sampling

All generators but G4GUN sample the direction of primary particles, isotropically or according to specific distributions.

If G4GUN is used, primary particles are shot along a given direction, that can be set with the macro command

```
/gun/direction dx dy dz
```

It is not necessary that the components $(dx, dy, dz)$ are normalized. For instance, the command to generate particles directed in the negative direction of the z-axis is /gun/direction 0 0 $-1$.

Alternatively, it is possible to generate particles with direction sampled isotropically from a cone with given direction and given opening angle. The command to activate this option, which works only for the G4GUN generator, is

```
/MG/generator/g4gun/cone_on true [true] [false]
```

By default, the `cone_on` option is set to `false` (i.e. pencil beam). The direction of the cone axis can be specified with the command

```
/MG/generator/g4gun/coneDirection dx dy dz
```

The default direction is along the positive z-axis. Similarly, the opening angle of the cone is set with the command

```
/MG/generator/g4gun/thetaDelta angle [rad/deg]
```

The default angle is 180 degrees. In this case, the direction of the primary particles is sampled isotropically in all the space, irrespectively of the specified direction.

In some applications, in order to save CPU time and optimize MaGe performaces, it is useful to generate primary particles only towards the detector region. MaGe has the capability of sampling the direction of the primary particles only in the direction of a given point. This option is activated with the macro command

```
/MG/generator/g4gun/centric_effect_on [true] [false]
```

and the target coordinates are set using the command

```
/MG/generator/g4gun/detector_center_position coordinates [unit]
```

Anyway, often one is not interest in a dimensionless target point, but rather in a target volume surrounding that point. This is necessary for instance to take into account for the extension of the real volumes and/or for changing in the particles direction along its propagation. There are two ways of specifying the extension of the target region around target point:

1. the target volume is a box (default). The coordinates of the box are given with the command /MG/generator/g4gun/detector_center_position, while its dimensions along the x-, y- and z-axis are set with

   ```
   /MG/generator/g4gun/detector_position_smear dimension [unit]
   ```

2. the direction is sampled within a cone pointing towards the target point. This option is activated with the command

   ```
   /MG/generator/g4gun/centric_effect_cone [true] [false]
   ```

   while the opening angle of the cone is specified by the macro command

   ```
   /MG/generator/g4gun/opening_angle angle [unit]
   ```

   In this case, the direction of the cone axis depends on the initial position of the track, and it is internally re-calculated event-by-event.

## 6.13 Energy sampling

All generators but G4Gun sample the energy of primary particles (e.g. cosmic ray muons, DecayO, etc.) according to their specific spectra.

If G4Gun is used, primary particles are shot with a fixed kinetic energy, that can be set with the macro command

```
/gun/energy energy [unit]
```

For instance, the command to generate 1-MeV particles is /gun/energy 1 MeV

Alternatively, it is possible to generate particles with energy spectrum sampled according to an histogram stored on a local file. This can be useful to generate energy spectra that are not provided by GEANT4, as for instance $\beta$ spectra from forbidden transitions (e.g. $^{39}$Ar). In fact GEANT4 samples $\beta$ spectra according to the Fermi function, which is appropriate for allowed $\beta$-transitions only. The command to activate this option, which works only for the G4GUN generator, is

```
/MG/generator/g4gun/spectrum_from_file [true] [false]
```

By default, the `spectrum_from_file` option is set to `false` (i.e. mono-energetic beam).
*Notice* that energy sampling activated with the `spectrum_from_file` can be coupled with the direction sampling described in Sect 6.12 and the position sampling described in Sect. 6.11. For instance, it is possible to generate with G4GUN primary particles emitted with a given energy spectrum (read from file), with direction in a given cone and with initial position distributed in a given volume.
The file name containing the spectrum can be specified with the command

```
/MG/generator/g4gun/spectrum_filename [filename]
```

The information is taken into accouunt only if the flag from `spectrum_from_file` is set to `true` before the `/run/beamOn`. The program looks for the data file first in the current directory, and then in the directory pointed by the environment variable `MGGENERATORDATA`. Notice that MAGE will exit with an error code if one tries to run the `spectrum_from_file` option without having specified the file name and/or the data file is not found. The data file should have two column showing:

```
energy (keV)  pdf of energy distribution (a.u.)
```

The energy spectrum does not need to be normalized, as it is re-normalized internally at the initialization time. The energy provided in the file should be the left edge of each bin. For the last bin, it is assumed that its width is equal to the width of last-but-one bin.

`Warning`: the sampler DOES NOT interpolate inter-bin ranges. It means that the sampling will be uniform within the binning you specify. For example, if you have 1-keV binning in the beginning, and 100-keV binning in the end, the final distribution will have seizable steps-shape.

For instance, the following piece of macro generates $^{39}$Ar $\beta$ decays, with energy spectrum read from the file `ar39.dat` and isotropic angular distribution:

```
/MG/generator/select G4gun
#
# Here set the energy sampling from ar39.dat
#
/MG/generator/g4gun/spectrum_from_file true
/MG/generator/g4gun/spectrum_filename ar39.dat
#
# Get isotopic particles. By default the axis of the cone is
# the z-axis, with 180 deg opening angle (= isotropic)
#
/MG/generator/g4gun/cone_on true
#
```

## 6.14 TUNL Generator

## 6.15 Wang Neutron Generator

# Chapter 7

# Geometries

## 7.1 Common Geometries

### 7.1.1 Geometry From File

To switch on the geometry definition from an external file, the messenger command

```
/MG/geometry/detector GeometryFile
```

has to be given. The name of the file containing the geometry to be read is set using the command

```
/MG/geometry/geometryFileName mygeometry.def
```

If the command `/MG/geometry/geometryFileName` is issued more than once, the file name is overwritten, and the last one is taken into account. If the general geometry has not been set with the command `/MG/geometry/detector GeometryFile`, the file name is anyway accepted but it is not used. If the file name is not given explicitly, the default `geometry.def` is looked for in the current directory.
It is possible to define an arbitrary number of boxes, cylinders and spheres. Volumes can be daughters of the world or of an other volume. In the latter case, it is possible to define "holes" in a given volume.
An example of geometry definition file is the following:

```
1 Crystal 2 1 0 SodiumIodide
0. 0. 2.55
2.55 5.10 0.
0. 0. 0.
2 Hole    2 0 1 Air
0. 0. 0.65
1.43 3.80 0.
0. 0. 0.
```

Each volume is defined in four lines.
(1) The entries for the first line are:

- ordering number (int).
- volume name (string). This is the name of the physical volume defined in the geometry. It can be used, for instance, for the uniform sampling routine. The names of the volumes should be different, though there is no specific control in the program.

- shape of the volume (int). The code is 1 for boxes, 2 for cylinders and 3 for spheres. If the code is different from the values listed above, the volume is ignored.
- sensitive detector flag (int). If the flag is 1, the volume is registered as a sensitive detector for the subsequent analysis.
- mother flag (int). If the flag is set to 0, the volume is placed inside the world volume. If the flag is a non-zero integer $n$, the volume is considered to be a daughter of the volume $n$. The mother volume must be defined *before* the daughter one (namely, the ordering number of the Daugherty must be larger than $n$). It is possible to nest daughter volumes one inside the other.
- material name. The material has to be included in the MaGe database or defined using an external file, as described in Sect. 5.4.1.

(2) The second line contains the three coordinates of the volume center, given in cm (double). Coordinates are always expressed with respect to the mother volume reference system.
(3) The third line contains the three physical dimensions of the volume (double), given in cm. For boxes, the numbers are the sizes along the $x$, $y$ and $z$ axes, respectively. For cylinders, the first parameter is the radius, and the second the height (the third parameter is unused). For spheres, the first parameter is the radius, the other two are unused.
(4) The fourth line contains the three Euler angles (degrees) defining the volume rotation. The angles are always referred to the mother volume reference system.
The volumes defined in the external file are placed in a world volume made of air. The world volume is a cube of 5 m size. Therefore, the dimensions of the volumes in the file cannot exceed 5 m.
The geometry file presented above represents: a cylindrical sodium iodide detector (radius 2.55 cm, height 5.10 cm) with a cylindrical hole (radius 1.43 cm, height 3.8 cm) displaced of 0.65 cm along the $z$-coordinate of the detector. The coordinates of the center of the detector are (0,0,2.55 cm) with respect to the world reference frame. The sketch of the geometry is displayed in Fig. 7.1.

## 7.2 GERDA Geometries

### 7.2.1 GERDA Phase I

```
/MG/geometry/detector GerdaArray
/MG/geometry/detector/geometryfile geometry.dat
/MG/geometry/detector/matrixfile matrix_phase_i.dat
```

The standard GERDA setup (to be specified) including the detector array (with cables and holders), the cooling liquid (argon), the cryogenic tank, the water tank, the lock and the cleanroom. Also included are the PMTs and the plastic scintillator on top of the cleanroom. Parts of the infrastructure, in particular beams between the cleanroom and the water tank, are also simulated. The detectors are described in geometry.dat, the array configuration is described in matrix_phase_xxx.dat

The Phase I setup includes an array of 9 non-true coaxial detectors. They are stacked in three strings of three detectors each.

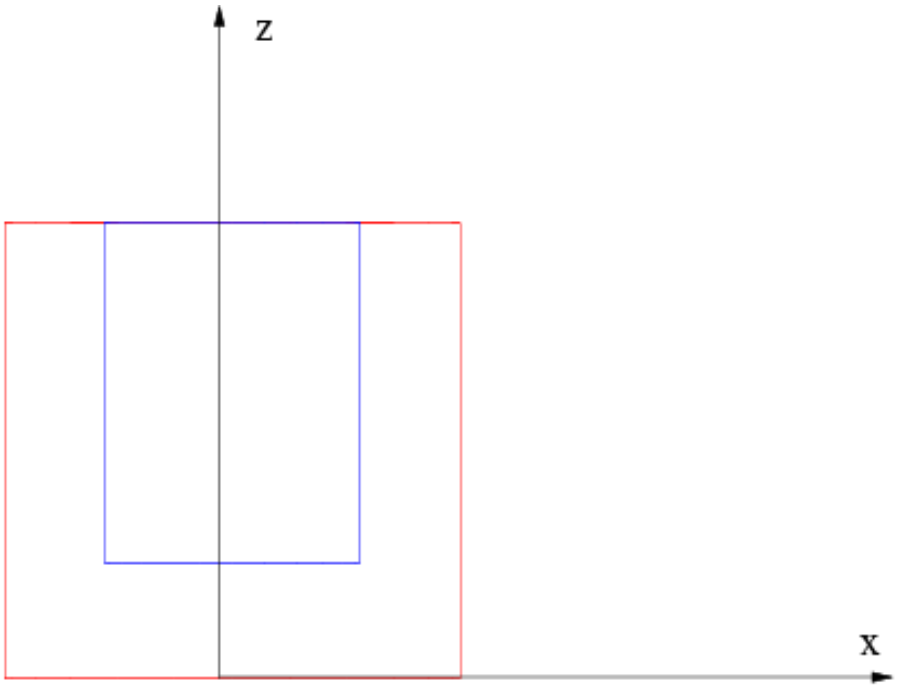


Figure 7.1: Sketch of the setup geometry defined in MaGe with the external data files used as example in the text.

### 7.2.2 GERDA Phase II

```
/MG/geometry/detector GerdaArray
/MG/geometry/detector/geometryfile geometry.dat
/MG/geometry/detector/matrixfile matrix_phase_ii.dat
```

See GERDA Phase I for the general setup. The Phase II setup includes five true coaxial and three non-true coaxial detectors per layer for three layers in total.

### 7.2.3 GERDA Phase II Ideal

```
/MG/geometry/detector GerdaArray
/MG/geometry/detector/geometryfile geometry.dat
/MG/geometry/detector/matrixfile matrix_phase_ii_ideal.dat
```

See GERDA Phase I for the general setup. The Phase II ideal setup includes seven true coaxial detectors per layer for three layers in total.

### 7.2.4 Test Stand Simple

```
/MG/geometry/detector MunichTestStand
/MG/geometry/teststand/teststandtype simple
```

Describes a ReGe detector with a source placed infront of it. The source is surrounded by lead bricks. The distance between the crystal and the source as well the distance between the source and the lead bricks can be set by macro with `/MG/geometry/teststand/crystaltosource` and `/MG/geometry/teststand/sourcetobrick`, respectively.

### 7.2.5 Test Stand LN2

```
/MG/geometry/detector MunichTestStand
/MG/geometry/teststand/teststandtype ln2
```

Simulates an unsegmented DSG p-type detector in the GERDAlinchen test stand, i.e. a cryostat filled with liquid nitrogen. It includes an AEA source inside the cryogenic liquid.

### 7.2.6 Heidelberg Detectors

```
/MG/geometry/detector Bruno
/MG/geometry/detector Corrado
/MG/geometry/detector Dario
```

Simulates the three main detectors used at Heidelberg for material screening. Includes full models of shielding and samples. The samples can be defined by the following command:

```
/MG/geometry/dario/sample [sphere] [box] [tube] [sbox] [liquid] [twobox]
[marinelli] [custom]
```

For different detectors replace the detector name in the command (and all following commands) accordingly, e.g. for Bruno : `/MG/geometry/bruno/sample`.
Description of sample options :
(1) The default geometry option is `sphere`, which creates a sphere of fixed radius 0.1 $\mu$m. This is used only to put the volume `sample` out of way when generating gammas from a single point.
(2) The outer dimensions of geometry `box` can be set by the following self-explanatory commands:

```
/MG/geometry/dario/boxwidth dimension [unit]
/MG/geometry/dario/boxheight dimension [unit]
/MG/geometry/dario/boxthickness dimension [unit]
```

(3) The geometry `tube` produces a cylinder with a possible concentric hole. The dimensions can be specified by the following parameters:

```
/MG/geometry/dario/tubelength dimension [unit]
/MG/geometry/dario/tubeouterradius dimension [unit]
/MG/geometry/dario/tubeinnerradius dimension [unit]
```

(4) The geometry `sbox` is identical to `tube` (the same dimension-parameters apply), but adds a standard cylindrical box used for measurements at MPIK laboratory (lid facing the detector). The position of the container changes according to the position of the sample, but the size is fixed, so it's users responsibility to ensure the `sample` volume fits inside and doesn't overlap with the container volume. The default dimensions of the sample cylinder correspond to the inner dimensions of the standard cylindrical box, so for samples which fill the inner volume completely, these parameters don't need to be specified.
(5) The geometry `liquid` is an extension of the `sbox` geometry. It subtracts a flat volume from the top of the sample, to simulate liquid not filling completely the volume of the standard box. For defining the liquid level, the command `/MG/geometry/dario/boxheight` is used. The value

corresponds to liquid level measured from top (vertical dimension of the missing part).
(6) The geometry `twobox` adds two (empty) standard cylindrical containers in front of the detector.
(7) The geometry `marinelli` (not available for `Bruno` detector) is an extension of the `sbox` geometry. It creates a marinelli shaped sample (a box with a hole), with external dimensions defined by `boxwidth`, `boxheight` and `boxthickness` parameters, and internal hole depth and radius defined by `tubelength` and `tubeouterradius` parameters. The parameter `tubeinnerradius` is used to displace the hole from the center in vertical direction.
(8) The option `custom` selects additional geometry, which can't be described by previous commands, but has to be coded in the detector construction source file.
The following parameters define the sample position with respect to the center of the front detector window :

```
/MG/geometry/dario/samplexpos dimension [unit]
/MG/geometry/dario/sampleypos dimension [unit]
/MG/geometry/dario/samplezpos dimension [unit]
```

The command:

```
/MG/geometry/dario/samplematerial [material name]
```

defines the material of the sample. A material with the same name has to be defined in `MGGerdaLocalMaterialTable.cc` or from external file with the `/MG/geometry/addMaterial` command, as described in Sect. 5.4.1.
Additional commands can be used with detector `Dario` to adjust the parameters of the crystal:

```
/MG/geometry/dario/detzpos dimension [unit]
```

Guidance: Sets the distance in `z` direction between the detector window and crystal surface.

```
/MG/geometry/dario/deadthickness dimension [unit]
```

Guidance: Sets the dead layer thickness.

```
/MG/geometry/dario/detdiameter dimension [unit]
```

Guidance: Sets outer diameter of the crystal (including dead layer).

```
/MG/geometry/dario/detlenght dimension [unit]
```

Guidance: Sets lenght of the crystal (including dead layer).

## 7.3 Majorana Geometries

### 7.3.1 17-A

```
/MG/geometry/detector MJ17A
```

### 7.3.2 57-Banger

```
/MG/geometry/detector MJ57Banger
```

### 7.3.3 CERN NA55 Experiment

```
/MG/geometry/detector CERN_NA55
```

This geometry describes an experiment performed at CERN in which neutrons produced from a 190 GeV muon beam incident on various beam stops were measured. The geometry consists of a (selectable) beam stop and a bounding sphere. For more information on the experiment, please see V. Chazal et al., Nucl. Inst. Meth. Phys. Res. A 490, p 334-343 (2002)

**Options**

1. ```
   /MG/geometry/CERN_NA55/setBeamDumpType [Graphite] [Copper] [Lead]
   ```

   Guidance: Set type of beam dump.

### 7.3.4 Clover Detectors

```
/MG/geometry/detector [clover] [cloverinNaIbarrel] [cloverInShield]
```

The clover geometry models the Canberra clover detector, which consists of four n-type Ge crystals. The "clover" option is the basic clover detector. The "cloverinNaIbarrel" option produces the clover detector in an NaI veto barrel used for TUNL FEL testing. The "cloverInShield" option is based on an experiment performed at LANL, and produces the basic clover detector in a lead shield, with a polyethylene moderator between the shield and detector on one side, as used in an experiment at LANL. The experiment at LANL used an americium beryllium source, and the "cloverInShield" detector includes an option to create a housing for the source. If the housing is used, its position must be specified.

**Clover In Shield Options**

1. ```
   /MG/geometry/cloverInShield/moderatorThickness [dimension with units of length]
   ```

   Guidance: Set thickness of moderator.

2. ```
   /MG/geometry/cloverInShield/shieldThickness [dimension with units of length]
   ```

   Guidance: Set thickness of lead shield.

3. ```
   /MG/geometry/cloverInShield/useAmBeHousing [true] [false]
   ```

   Guidance: Select whether to use housing for the AmBe generator.

4. ```
   /MG/geometry/AmBeHousing/position [x y z units of length]
   ```

   Guidance: Set location of center of housing, which should match source position.

### 7.3.5 Ideal Coaxial Crystal in a Shield

```
/MG/geometry/detector idealCoax
```
```

### 7.3.6 MJ LArGe

```
/MG/geometry/detector LArGe
```

### 7.3.7 LLNL Detector

```
/MG/geometry/detector LLNL8x5
```

### 7.3.8 MEGA cryostat

```
/MG/geometry/detector MJMEGA
```

This geometry models an experiment at the Waste Isolation Pilot Plant (WIPP) in Carlsbad, New Mexico. The geometry consists of an annular copper cryostat, and four identical Ge crystals arranged in two two-crystal stacks. The cryostat is surrounded by a lead shield. This geometry uses the Ideal Coax Crystal class. Dimensions and properties of the crystals should be specified with options for the Ideal Coax Crystal class.

### 7.3.9 MJ Reference Design Detectors

```
/MG/geometry/detector [MJRDBasicShield] [MJRDCryostat] [MJRDCrystalColumn]
```

### 7.3.10 SLAC Beam Dump Experiment

```
/MG/geometry/detector SLACBD
```

This geometry models an electron beam experiment that took place at SLAC. (Please see S. Taniguchi et al., Nucl. Inst. Meth. Phys. Res. A 503, p 595 (2003) and S. Roesler et al., Nucl. Inst. Meth. Phys. Res. A 503, p 606 (2003) for information on the experiment.) The geometry consists of an aluminum beam dump surrounded by a steel and concrete shield. This geometry has been used to model and validate neutron propagation.

**Options**

1. ```
   Command /MG/geometry/SLACBD/setShieldThickness [None] [Thin] [Medium] [Thick]
   ```

   Guidance : Set thickness of concrete shield.

### 7.3.11 Solid Block

```
/MG/geometry/detector solidBlock
```

### 7.3.12 UW LArGe

```
/MG/geometry/detector UWLArGe
```
```

### 7.3.13 WIPPn

```
/MG/geometry/detector WIPPn
```

# Chapter 8

# Analysis tools

*Xiang*

## 8.1 Handling of decay chains

Radioactive decay in GEANT4 is handled by the radioactive decay module(RDM). The RDM will handle an entire decay chain, including proper branching ratios. The chain all happens in one GEANT4 event, so it is up to the user to divvy up the chain into events that a physical detector would actually measure. One way of doing this is to keep track of each particle's global time. The global time of a track in GEANT4 is the time since the track was created(`aTrack→GetGlobalTime()`). Steps of an event that fall within a user-prescribed window of time are then gathered together. The time window would typically correspond to the time resolution of the DAQ. The user's output class would then write to a ROOT object more often than once a GEANT4 event.

A problem arises because of the finite precision of a double. In decimal form, a G4double is only accurate to 15 digits or so. Following the decay of $^{238}$U, the global time of the daughter nucleus, $^{234}$Th would be on the order of 5 $\times 10^9$ years, or $10^{17}$ seconds. Each additional decay only adds more to the global time. If one wishes to gather steps that fall within a typical time window of say, 15 $\mu$s, then one must find steps that differ in time by $\sim$ 1 part in $10^{22}$. This is treated as zero.

The workaround to this involves resetting the GlobalTime of a track that was spawned by radioactive decay to zero if it is too large. Any comparison of global times can then yield the desired sensitivity. A new variable, `fOffsetTime` is kept separate from GlobalTime, and is incremented to keep track of the actual global time.

### 8.1.1 Stacking Actions

A brief comment on stacking actions. When a track is created by a process, it is sent to any UserStackingAction that is in place. In MaGe , the UserStackingAction resides in the output classes that derive from `MGVOutputManager`. The StackingAction can look at the track and determine whether to put it in the "Urgent" stack or the "Waiting" stack. When all tracks in the urgent stack are finished, then all tracks in the waiting stacks are transferred(all at once) to the urgent stack. The urgent stack is then processed as normal by GEANT4. The StackingAction allows for a user-defined `NewStage()` function that is called once all tracks in the urgent stack are finished.

### 8.1.2 TimeWindower Implementation

The UserStackingAction set in MaGe.cc is `MGManagementStackingAction`. It has three functions, `ClassifyNewTrack(const *G4Track)`, `NewStage()`, and `PrepareNewEvent()`. These functions call their corresponding counterparts in whatever output class you use that derives from `MGVOutputManager`. One note of possible confusion, the `ClassifyNewTrack(const *G4Track)` points to output function `StackingAction(const *G4Track)` in the output class. The `StackingAction(const *G4Track)` and the `NewStage()` functions are written in `MGVOutputManager` and probably don't need to be messed with unless you're doing something complicated. The `MGVOutputManager` also defines a `WritePartialEvent(const *G4Event)` and `ResetPartialEvent(const *G4Event)` functions. Each individual output class needs to have its own version of these two functions to properly use the TimeWindower.

The use of time windowing is specified by macro(default is off), as is the actual window of time to use. If the macro specifies time windowing, then the `StackingAction(const *G4Track)` looks at each track as it is created. If the creator process was RadioactiveDecay and the GlobalTime is larger than the time window, then:

- the track's global time is stored
- the track's global time is set to zero
- the track is given the "waiting" classification, so it is sent to the waiting stack

If all tracks in the urgent stack have been processed and there are tracks in the waiting stack, then `NewStage()` is called. The `fOffsetTime` is updated, and `WritePartialEvent(const *G4Event)` and `ResetPartialEvent(const *G4Event)` are called. These functions obviously depends on the output class being used, but the general idea is that the event is written to a ROOT file, and any necessary arrays/variables/etc... are reset. Only after all this are the tracks in the waiting stack sent to the urgent stack. It's worth mentioning that `NewStage()` only happens if there are tracks in the waiting stack. If not, then the output class's `EndOfEventAction()` should handle the end of the actual GEANT4 event.

### 8.1.3 How to use the TimeWindower

First, make sure that the WritePartialEvent() and ResetPartialEvent() functions have been implemented in your output class(look at the `MGOutputG4Steps` class for an example of how to do this). You will also need to add messenger commands to your output messenger class. You can handle everything by messenger in a macro. The macro commands are:

```
/MG/io/G4Steps/useTimeWindow
/MG/io/G4Steps/setTimeWindow
```

Where you would substitute your output class for G4Steps.

# Part II

# Developer's guide

# Chapter 9

# Compiling MaGe

# Chapter 10

# MaGe Code Structure

## 10.1 Introduction

The simulation package is written in C++ and uses object-oriented programming techniques and the STL. The geometry, tracking and physics package is Geant 4 based. A working knowledge of Geant 4 is required to follow the discussions in this section. Waveform simulation is performed with custom software.

## 10.2 Program Structure

The program consists of sets of C++ classes that are grouped into software packages. All the software packages are built into a single executable, `MaGe`. The simulation is setup by selecting detector geometries, physics processes, generators, etc. via text macros based on Geant 4 messengers. The software packages name's, with their corresponding CVS directory names in brackets, are given below:

```
MaGe—
   |– Database (database)
   |– Generators (generators)
   |– Geometry (geometry)
   |– Gerda specific geometry (gerdageometry)
   |– Majorana specific geometry (mjgeometry)
   |– Output (io)
   |– Gerda specific output (gerdaio)
   |– Majorana specific output (mjio)
   |– Management (management)
   |– Materials (materials)
   |– Processes (processes)
   |– Waveform (waveform)
```

Note that not every CVS directory contains a software package, since some CVS directories, ie. `doc`, do not contain parts of the simulation program. Other packages, such as `Geometry`, are spread over several CVS directories to separate code from the Gerda and Majorana. Each package is discussed in detail in subsequent chapters. We will give a brief summary of each package here, as well as how they are connected.

**Database** Interfaces with the PostGreSQL database server. The database is used to store geometry data and calibration constants and also contains all the materials definitions. Currently only used by the Majorana experiment.

**Generators** The different event generators. Currently several event generators are supported and are user-selectable at runtime. See Chapter **??**.

**Geometry** The classes describing the different detector geometries. It is possible to combine different geometries into a single detector, ie. a NaI veto barrel around a clover detector, the Gerda detector geoemetry, etc. See Chapter **??**.

**Output** The classes describing the different output for the different detectors. Both AIDA and Root are supported. It is possible to have more than one output class per detector. Also contains the `MGLogger` class that handles terminal output and error messages. See Chapter **??**

**Management** Management classes, including `MaGe/management/MaGe.cc` that contains the `main()`. See Section 10.3 for more information on the management classes.

**Materials** These classes automatically build requested materials from elements that are in turn build from isotopes. All material, elemental and isotopic data are stored in the database. All new materials have to be added to the database. See Chapter **??**

**Processes** User-selectable Geant 4 physics lists for specific simulations, ie. double-beta decay studies requires different physics process than dark matter studies. See Chapter **??**.

**Waveform** Waveform simulation package. Can be run independently from Geant 4. Not yet implemented. See Chapter **??**.

## 10.3 How Everything is Implemented and Fits Together.

In the `main()` function a collection of "management" classes are instantiated. These classes control specific components of the framework. Classes can communicate with each other via the singleton `MGManager` class. The `MGManager` class is instantiated in the file `MG/management/MaGe.cc`, and is a collection of pointers to all the management classes. By using the `MGManager` class, global variables are avoided.

The management class are listed below in the order they are instantiated:

`MGGeometryDetectorConstruction` Selects detector geometry via Geant 4 messenger and constructs it in Geant 4. Also constructs all required materials from the database, if requested by the selected detector. See Chapter **??**. Accessed via: `MGManager::GetMGGeometryDetectorConstruction()`

`MGProcessesList` Geant 4 physics list. User can select different physics packages via Geant 4 messengers. See Chapter **??**. Accessed via: `MGManager::GetMGProcessesList()`

`MGManagementVisualization` Registers Geant 4 visualization packages. Performs no other actions.

`MGGeneratorPrimary` Selects and manages event generators. See Chapter **??**. Accessed via: `MGManager::GetMGGeneratorPrimary()`

`MGManagementRunAction` Performs pre- and post run actions by executing `BeginOfRunAction()` and `EndOfRunAction()` methods of output classes. Derived from `G4UserRunAction`. Accessed via: `MGManager::GetMGRunAction()`

`MGManagementEventAction` Performs pre- and post event actions by Executing `BeginOfEventAction()` and `EndOfEventAction()` methods of output classes. Selects and contains pointer to Root output class (Chapter **??**). Derived from `G4UserEventAction`. Accessed via: `MGManager::GetMGEventAction()`

`MGManagementSteppingAction` Performs actions during each step of Geant 4 tracking by executing `UserSteppingAction()` method of root output classes. Derived from `G4UserSteppingAction`. Accessed via: `MGManager::GetMGSteppingAction()`

A pointer to the `MGManager` singleton class is accessible via the `MGManager::GetMGManager()` method. The Geant 4 visualization and run manager are also accessible via: `MGManager::GetG4RunManager()` and `MGManager::GetG4VisManager()` methods.

There are two packages that are not accessible via the `MGManager`. The first is the Waveform package that is not implemented yet. It will be accessible via `MGManager` when it is implemented. The other is the database. For historical reasons, it is accessed via it's own singleton control class: `MGDatabase`. See Chapter **??**.

## 10.4 Programming Guidelines and Naming Conventions

MaGe requires the use of good object-oriented programming techniques. We strongly discourage the use of "Fortranized" C++. Our coding conventions are based on modified Root and Taligent guides . Enforcing too rigid programming rules are not feasible, although we require the following reasonable conventions:

- All class names begin with `GE, MG`, or `MG`. `GE` is reserved for classes specific to the Gerda simulation, such as geometries. `MJ` is specfically for Majorana classes. Common classes, such as management, start with `MG`. If in doubt, use `MG`.
- All virtual base class names begin with `MGV, MGV, GEV`.
- All class member variables names begin with `f`
- All local automatic variables names must begin with a lowercase, ie `someLocalRandomNumber`
- All variable, class names, etc. should have descriptive, written out names. Think carefully about naming and avoid ambiguity.
- All unitialized pointers should be set to 0 or `NULL`.
- All terminal output should be done via the `MGLogger` class, if possible. See Section 12.1.
- Code should be indented 3 spaces per nesting level.
- Lines longer than 80 columns should be wrapped with a carriage return.
- The use of a separator between methods, ie.

  ```
  //---------------------------------------------------------------------------//
  ```

  is strongly encouraged.

Some of the older code in the MaGe package do not follow these rules, since they were written before the rules were made explicit. Please do not follow their conventions. If you encounter any cases not covered here, use the Root/Taligent guideline, use you best judgment, or contact Reyco Henning (Majorana) or Xiang Liu (Gerda).

*Remember that you are not coding for yourself! Many others will have to read you code. Write the code the way you would like to read it!*

# Chapter 11

# Physics lists

*Luciano*

## 11.1 Introduction

This section gives a high-level description of the physics list used in the MaGe framework, and documents the MGProcesses Messenger that provides a small amount of user control over that list. All macro commands relative to physics list must be issued in the `Pre_init` phase, namely before the `/run/initialize`.

## 11.2 MGProcesses description

The MaGe framework physics list is an amalamation of two physics lists that are distributed with the GEANT4 simulation package. One part of the list comes from the list included as part of the Underground Physics advanced example. The other part of the list comes from the QGSP_HP hadronic list. Documentation on these lists are available on the GEANT4 website. It manages the processes for Optical Photons (Cherenkov emission, Scintillation, Absorption) supports the cut-per-region approach: this allows to set different production cuts in different regions of the geometry.

The primary class is the MGProcessesList class, but it uses supporting classes MGProcessesMaxTimeCuts, MGProcessesMinEkineCuts, and MGProcessesSpecialCuts.

## 11.3 Details of the physics lists

### 11.3.1 Interactions of hadrons

The hadronic part of the physics list (default) is composed by

1. theory-driven quark-gluon string models (QGSP) for the energetic pions, kaons, and nucleons, up to 100 TeV;
2. LEP parametrized models for inelastic interactions of pions and nucleons below 25 GeV;
3. tabulated cross-section data derived from the ENDF/B-VI database to model capture, fission elastic scattering and inelastic scattering of neutrons from thermal energies up to 20 MeV (HP models).

Alternatively, the physics list developed by M. Bauer (QGSP_GN_HP_BIC_ISO) for the description of muon-induced neutron production can be used. The list is chosen using the macro command

```
/MG/processes/qgsp_hadron_list true
```

The main difference with respect to the default physics list is that Binary cascade models (BIC) are used insted of LEP parametrized models for the description of nucleon and pion interactions below 10 GeV. It is also possible to replace the Binary cascade models with the Bertini (BERT) cascade models for the same energy range. The macro command is

```
/MG/processes/useBertCascade true
```

If the physics list is set to M. Bauer's one , the quark-gluon plasma string model (QGSP) for the description of high-energy interactions can be replaced with chiral-invariant-phase-space models (CHIPS) for the de-excitation for the quark-gluon plasma. The macro command is

```
/MG/processes/useQGSC true
```

As specified above, the latter command works only after

```
/MG/processes/qgsp_hadron_list true
```

For some applications, it may be useful to switch off all the hadronic interactions. It is possible in MaGe with the command

```
/MG/processes/useNoHadPhysics true
```

In this case, all the additional specifiers described avove (e.g. useBertCascade or useQGSC) have no effect.

### 11.3.2 Interactions of electrons, positrons and $\gamma$-rays

The specific low-energy GEANT4 models [3] are used by default for the description of the electromagnetic interactions of $\gamma$-rays, electrons, positrons and ions. The low-energy models include atomic effects (fluorescence, Doppler broadening) and can handle interactions down to 250 eV. The processes registered for $\gamma$-rays are coherent scattering (a.k.a. Rayleigh scattering), Compton scattering, photo-electric effect and pair production. Electromagnetic processes registered for electrons and positrons are ionisation, bremsstrahlung, $e^+$-annihilation and synchrotron radiation. Specific low-energy models are not available for $e^+$ (only for $e^-$); standard GEANT4 models (see below) are used for the positron electromagnetic interactions.
Alternatively, electromagnetic interactions in MaGe can be treated using the so-called Standard models provided by GEANT4. These models are tailored for high-energy applications; they are less precise in the low-energy part, not including atomic effect, but are faster from the point of view of CPU-time. The choice between Low-energy (default) and Standard electromagnetic models is done at run-time by the macro command

```
/MG/processes/lowenergy true/false
```

(true is the default).
Interactions of $\gamma$-rays and $e^{\pm}$ with hadrons are described by theory-driven quark-gluon string models for energetic particles ($> 3.5$ GeV) and by chiral invariant phase space (CHIPS) models for lower energies.

### 11.3.3 Interactions of muons

Hadronic interactions of high-energy muons are simulated using the `G4MuNuclearInteraction` model, replacing the `G4MuonNucleus` model. Although `G4MuNuclearInteraction` is known to be in closer agreement with experimental data, in old versions of GEANT4 the processes had a bug in the desctructor, causing a crash of the program at the exit. While this bug had no effect on the simulation results, it prevented the clean exit from the program.
For those application not requiring the precise description of muon-induced effects (e.g. studies of radioactive background), the `G4MuonNucleus` can be registered to the physics list instead of `G4MuNuclearInteraction`, giving the command

```
/MG/processes/useMuonNucleusProc true
```

The advantage is that one always has clean exit from the MaGe runs.
Standard models only are available in GEANT4 for electromagnetic interactions of muons (ionization, bremsstrahlung and pair production).

### 11.3.4 Interactions of optical photons

GEANT4 is able to handle optical photons and MaGe takes advantage from such capability. While the default MaGe physics list does *not* include interactions of optical photons, these processes can be switched on with the command

```
/MG/processes/optical true/false
```

(false is the default). The models included in MaGe encompass scintillation light emission (possibly with different light yield for electrons, $\alpha$-particles and nuclei), Cherenkov light emission, absorption, boundary processes, Rayleigh scattering and wavelength shifting. In order to produce correct results for optical photons, it is indeed necessary to specify in the geometry definition all the relevant optical properties of interfaces and bulk materials (scintillation yield, refraction index, absorption length, etc.). An example of this can be found in the geometry `GEMPIKLArGe.cc` (`munichteststand` directory).

### 11.3.5 Cuts definition

As a general feature, GEANT4 has not tracking cut, but only production cuts for $\delta$-rays and for soft bremsstrahlung photons. Cuts are expressed in range, and they are internally converted in energy threshold for each material used in the geometry.
In most applications, it is necessary to find a trade-off between accuracy and computing time. For this reason MaGe provides three simulation "realms", with different production cuts. The realms are called DarkMatter, BBDecay and CosmicRays.

- DarkMatter realm is used for high precision simulations, especially related to background studies for dark matter applications: cut-in-range for $\gamma$-rays and $e^{\pm}$ are 5 $\mu$m and 0.5 $\mu$m, respectively, corresponding to 1-keV threshold in metallic germanium.

- BBdecay realm (default) is suitable for background studies related to double-beta decay, i.e. in the MeV region: the cut-in-range is tuned to give a 100-keV threshold for $\delta$-rays in metallic germanium (it is 0.1 mm for $e^{\pm}$ and $\gamma$-rays).

- CosmicRay realm is used for the simulation of extensive electromagnetic showers induced by cosmic ray muons. The cuts are different in the volumes belonging to to the `Sensitive Region` and the other volumes. Cuts for the volumes in the sensitive region are the same than for the double-beta realm, while they are more relaxed everywhere else (5 cm for $\gamma$-rays and 1 cm for $e^{\pm}$), in order to save CPU time and to avoid the precise tracking of particles in the uninteresting parts of the setups). Such a capability is based on the general cut-per-region feature of GEANT4. Sensitive volumes are registered in the geometry definition using the code reported in 5.3.

The simulation realm is selected by macro using the command

```
/MG/processes/realm BBdecay/DarkMatter/CosmicRays
```

(BBdecay is the default). The simulation realm can be changed at run-time, even after the run initialization.

# Chapter 12

# I/O scheme and the ROOT interface

*Xiang*

## 12.1 Terminal output via the MGLogger class.

The MaGe package comes with a terminal output class that handles messages you want to write to the screen. A message is written to the screen only if its severity exceeds that of the current severity setting. The allowed severity levels are: debugging,trace,routine,warning,error, fatal. The severity level is set via a Geant 4 messenger at run-time. For example, if we set the severity to warning :

```
MG/manager/mglog warning
```

then the following line of code will produce terminal output:

```
MGLog(warning) << ``Your Warning here" << endlog;
```

However, if the severity level is set higher at run-time, to error or fatal, then nothing will be written to the screen. Remember to put an endlog at the end of your statement. If you have a statement that is set to fatal, nothing is written to the screen, and the program is stopped immediately.

## 12.2 Common Output Classes

The common output classes are all listed under directory io. As concerning to the GERDA developers, the most important class is MGOutputRoot. This is the base class for Root-format output. New Root-output classes all have to base on this class.

## 12.3 Gerda Array Output

For simulation of the default Gerda Phase-I and Phase-II geometry, following macros are used:

```
/MG/geometry/detector GerdaArray
/MG/eventaction/rootscheme GerdaArray [or GerdaArrayWithTrajectory]
/MG/eventaction/rootfilename put_root_output_filename_here
```

The macro commands are registered in class `MGManagementEventActionMessenger` under directory `management`. All output variables can be found in class `GEOutputGermaniumArray`.

The output variables for the Gerda Array can be roughly divided into 3 classes:

- class A: Primary particles:

  The input to the MaGe simulation i.e. the number of primary verteces and their positions, the number of primary particles at each vertex, the particle types and their momenta, is saved to the ROOT ntuple.

- class B: Trajectories and interactions along each trajectory:

  In GEANT4 the path of each particle (including secondary ones) through the user-defined geometries (including sensitive and non-sensitive materials) is defined as a trajectory. Each interaction is defined as one trajectory point. For GERDA the true Monte Carlo information about each trajectory, including the particle type, the number of interaction points, the types of the interaction, and the position and energy loss at each point, is saved.

- class C: Energy deposit in sensitive volumes:

  The trajectory points in the user-defined sensitive volumes are defined as hits in GEANT4. They represent the energy deposits as measured in the sensitive detectors. Thus the list of hits is a sub-group of the trajectory points. Again, the number of hits, their positions and energy depositions inside the germanium detectors are saved. The overall energy depositions in the detector segments, namely the sum of all hits in each segment, and the energy depositions in water, liquid nitrogen and the scintillator are also saved. The information about which primary particle created which hits is saved to the ROOT ntuple as well.

With the option `GerdaArrayWithTrajectory` variables in all three classes are saved, while with option `GerdaArray` only variables in class C are saved, so as to save disk space.

**Variables in Class A**

`vertex_totnum`: number of vertex (max 10)
`vertex_xpos,_ypos,_zpos`: x-y-z position of the vertex
`vertex_ypos`
`vertex_zpos`
`vertex_time`: the time for the vertex
`vertex_numparticle` : number of primary particles at each vertex

`mc_totnumparticles`: total number of primary particles (max 2000)
`mc_ivertex`: which vertex is each primary particle positioned at
`mc_px,mc_py,mc_pz,mc_pe`: 4-momenta of each particle
`mc_ekin`: kinetic energy of each particle
`mc_id`: what kind of particle

**Variables in Class B**

`hits_totnum`: total number of hits (max 2000)
`hits_tote`: total amount of energy deposited in sensitive volume
`hits_edep`: energy deposited at each hit
`hits_xpos,_ypos,_zpos`: position of each hit
`hits_idseg`: the ID of the segment in which the hit is

`hits_time`: time of creation of each hit
`hits_trackid`: the ID of the trajectory that created the hit
`hits_trackpdg`: the PDG encode of the trajectory

`hits_passivation_totnum,_tote,_edep,etc`: hits in passivation layer
`hits_deadlayer_totnum,_tote,_edep,etc`: hits in dead layer

`seg_totnum`: total number of segments with any amount of energy deposit (max 400)
`seg_id`: which segment it is
`seg_numhits`: the number of hits each segment has
`seg_edep`: total energy deposit in each segment

**Variables in Class C**
`trj_totnum`: total number of trajectories (max 2000)
`trj_id`: trajectory ID, referred by `hits_trackid` etc.
`trj_pid`: parent trajectory ID
`trj_pdgencode`: PDG encode of the trajectory (what kind of particle)
`trj_charge`: particle charge `trj_px,_py,_pz`: initial momentum of the trajectory `trj_npoints`: number of interacting points along each trajectory
`trj_istart,_iend`: pointers to the part of points in the 1D array `trjp_` that belong to each trajectory
`trj_leptonnumber`: lepton- and baryon- number of the trajectory
`trj_baryonnumber`

`trjp_totnum`: total number of trajectory points (max 5000)
`trjp_xpos,_ypos,_zpos`: positions of the interacting points
`trjp_de`: energy deposit at each point
`trjp_steplength`
`trjp_insidege`: whether the interaction happens inside the detector or not
`trjp_processid`: what kind of interaction, see `GEOutputGermaniumArray.hh` for the list of interactions.

## 12.4 Gerda Test Stand Output

**UNDER CONSTRUCTION**

Several Gerda test stand geometries are implemented in MaGe . To simulate the test stand, the macro

```
/MG/geometry/detector MunichTestStand
/MG/geometry/teststand/teststandtype options_test_stand_name
/MG/eventaction/rootscheme options_root_scheme
```

has to be executed first.

Here the `options_test_stand_name` can be chosen from the following list:
`ln2`
`simple`
`GerdaLinchenII`
`coincidence`

For these test stands, three different types of `options_root_scheme` can be chosen:
`GerdaTestStandEnergyOnly`: only variables `hits_tote,seg_ttonum,seg_edep` are saved.
`GerdaTestStandEnergyandHits`: hits and energy deposit in segments are saved.
`GerdaTestStandEnergyHitsTrajectories`: all variables are saved.

Two special test stands requiring coincidence trigger have their own root scheme definition. Only for simulated events with a coincidence trigger, all variables are saved for further analysis.

```
/MG/geometry/teststand/teststandtype siegfried
/MG/eventaction/rootscheme GerdaTeststandSiegfried
```

```
/MG/geometry/teststand/teststandtype siegfriedcoincidence
/MG/eventaction/rootscheme GerdaTeststandSiegfriedCoincidence
```

### 12.4.1 GEOutputCrystal- a new Output Class

A new output class with an easier implementation of segmentation and sensitive volumes has been created.
It is not only created to control the output but it is also a approach to fully control the simulation in a general way.
The class gets as much information as possible directly from the G4Step object. The information is filled into a tree and saved to a ROOT file. This way, the file can be opened in ROOT without loading any extra libs.

#### Usage

For users who only want to use the output provided by the new class, the information they can get from the ROOT tree is described in detail here.
The class provides a runID and an eventID and an output tree:

You can use the features of the class by typing:

```
/MG/eventaction/rootscheme Crystal
/MG/eventaction/Crystal/save [source, hits, tracks, all]
```

The options in "[ ]" can be chosen from the ones listed above. You may use either one of them or a combination. In detail, they imply:

- source - saves only the primary vertex and particles information
- hits - saves the hits and trajectories in the sensitive volumes
- tracks - saves all hits and all trajectories
- all - saves, well, all..

The default option is "hits", if the parameter is omitted.

Users can get the crystal type from the following ROOT branch:

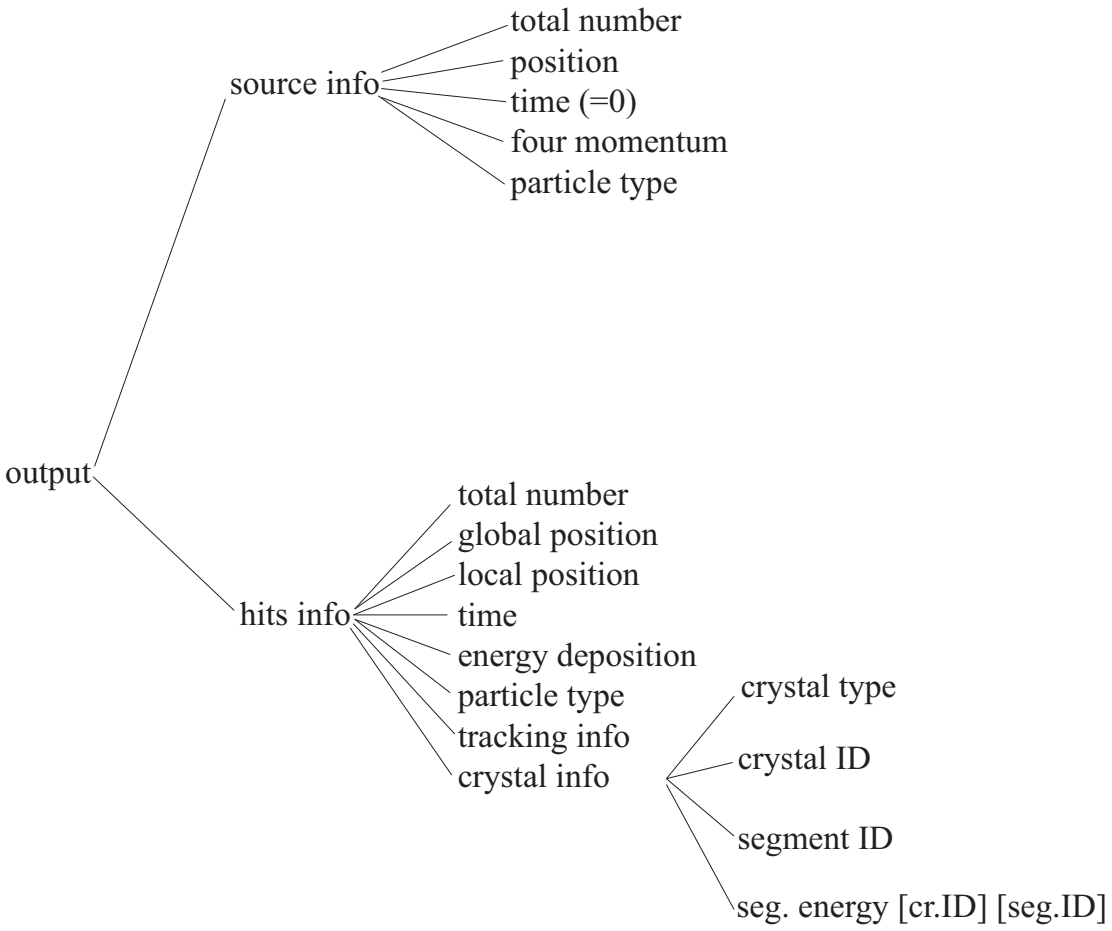


Figure 12.1: ROOT output tree of the class GEOutputCrystal. The segment energy is saved with respect to the crystal ID and the segment ID.

```
hitCrystalType[maxNhits]
```

The values correspond to the hit being in:

- -1 : not a sensitive volume at all
- 0 : a sensitive volume but not a crystal
- 1 : RolandI (unsegmented true coaxial p-type detector)
- 2 : RolandII (6-fold segmented true coaxial p-type detector)
- 3 : RolandIII (18-fold segmented true coaxial p-type detector)
- 4 : Siegfried (18-fold segmented true coaxial n-type detector)

All the segments are arranged according to their local coordinate system. They are numbered from 1 for the first segment, R==0 and Φ==0, counterclockwise and bottom up. The segmentation scheme and the segment numbering of the crystal "Siegfried" are shown in figure 12.4.1. Every event in a crystal is assigned to a detector segment according to the segmentation scheme using its z-, R- and Φ- values.

The segment shown in the ROOT branch is specified by the parameter:

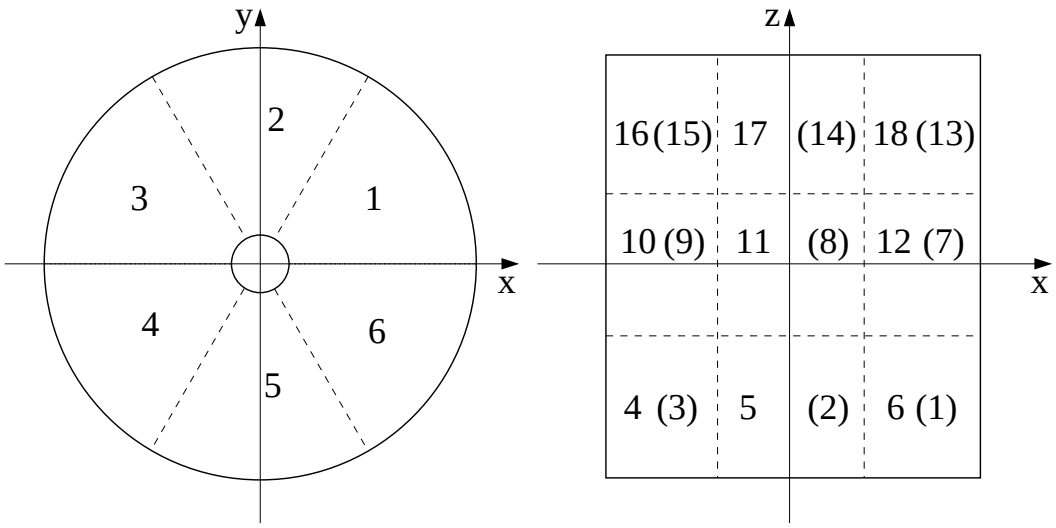


Figure 12.2: Segmentation scheme for "Siegfried"

```
hitSegmentID[maxNhits]
```

The values correspond to:

- -1 : not a sensitive volume at all
- 0 : the whole volume which is sensitive (the whole detector)
- 1 : the first segment
- 2 : the second segment
- ...

The crystal position in the GERDA array is given in the ROOT branch:

```
hitCrystalID[maxNhits]
```

The values correspond to:

- -1 : not a sensitive volume at all
- 0 : sensitive volume but not a crystal
- 1 : the first crystal
- 2 : the second crystal
- ...

The energy deposited in a segment of a crystal is saved in the ROOT branch:

```
segmentEnergy[crystalID][segmentID]
```

Only positive crystal IDs can be used here, of course (negative values are not sensitive volumes). For crystal ID = 0, only segment ID = 0 is available (a volume with crystal ID = 0 is not a crystal). Crystal ID = n and segment ID = 0 shows you the energy deposited in the complete $n^{th}$ crystal.

### Geometry developers' guide

The implementation of segmentation and sensitive volumes is the pat that interests a developer. How this is done is explained here in a few steps.
The definition of a logical volume and a physical volume in MaGe works as follows:
The geometry of a crystal is defined. It becomes a logical volume with attributes in the geometry construction class, e.g. in "GEMunichTestStandDB":

```
fCrystalLogical
= new G4LogicalVolume(CrystalTubs,          //the name of the created geometry
                      enrichedGermanium,    //the material the crystal consists of
                      "CrystalLogical");    //its name
```

This logical volume is inserted in the physical volume that is constructed like this:

```
fCrystalPhysical
  = new G4PVPlacement(0,                                //no rotation
                      G4ThreeVector(0,0,0),             //translation position
                      fCrystalLogical,                  // pointer to its logical vol.
                      "CrystalPhysical",                //its name
                      fCrystalMotherVolumeLogical,      //its mother logical volume
                      false,                            //no boolean operations
                      0);                               //its Copy Number
```

The physical volume name of a crystal defines its type. The class recognizes this and creates a segmentation scheme according to the type. You don't have to define each segmentation in your geometry construction class.
You can take a simple G4Tubs object and name the physical volume "siegfried". The segmentation scheme predefined as "siegfried"- scheme is implemented on the crystal. This means that hits inside this crystal have a segment ID according to their positions and the "siegfried" segmentation scheme.
A sensitive volume will be automatically recognized by the class from a physical volume. You can change a usual physical volume to a sensitive one very easily: You only have to rename it as: "PhysicalVolume" $\longrightarrow$ "sensitive_PhysicalVolume".
The hits in a sensitive volume are assigned to the segment of a detector using their local coordinates with the origin at the geometric center of the crystal: $x_i$, $y_i$, $z_i$, $T_i$, $p_{x,i}$, $p_{y,i}$, $p_{z,i}$ and $E_{k,i}$ for the $i^{th}$ hit. The local coordinate system is also given in $R_i$, $\Phi_i$ and $z_i$ with $z_i$==0 at half the height of the crystal.
The GERDA array consists of more than one detector. The position of each crystal is defined by its "CopyNo.".
Detectors in LAr are characterized by their logical volumes inside the mother volume "LAr" that are numbered top to bottom: 0, 1, 2... Hits inside these volumes will have a crystal ID (equals the CopyNo.+1): 1, 2, 3...

The definition of a physical volume in reality might look like this:

```
fCrystalPhysical
  = new G4PVPlacement(0,
                      G4ThreeVector(0,0,0),
                      fCrystalLogical,
                      "sensitive_siegfried",
                      fCrystalMotherVolumeLogical,
                      false,
                      1);
```

The class GEOutputCrystal extracts the following information from this physical volume:

- It is a sensitive volume with the segmentation scheme of the crystal siegfried (the **crystal type** is defined).
- The copy number of the crystal is 1, its **crystal ID** is 2 according to the scheme mentioned above.

If a hit is in segment 15 of "Siegfried", for example, this means:

- hitCrystalType = 4
- hitCrystalID = 2
- hitSegmentID = 15

## 12.5 Majorana Output

# Chapter 13

# Material definitions

*tba*

# Chapter 14

# Detector components

*all*

## 14.1 MGGeometry Classes

The following is a list of classes in the MGGeometry framework and a description of their uses.

MGGeometryDetectorConstruction

```
This class is the primary class that builds the MG materials as
well as instantiates any detector chosen in the
/MG/geometry/detector messenger described above. It creates a
world volume and places the detector as a single object inside
it. Eventually it will first place a shield inside the world
volume, and then a detector inside the shield, but for now we
don't have any shields.
```

MGGeometryDetector

```
This is the base class for all MGGeometry detector classes.
When instantiated, it must be called with the serial number of
the desired detector. It allocates a pointer to the outermost
detector and sets its name. All other functionality must be built
                            within each individual kind of detector class.
```

MGGeometryDetectorMessenger

```
The object that is used to issue commands to the MGGeometry
framework.
```

MJGeometry800gCrystal MJGeometryCloverDetector MJGeometryCloverInNaIBarrel MJGeometrySolidBlock

```
These are subclasses of the MGGeometryDetector class. These
classes may contain active detector subparts (e.g.,
the MJGeometryCloverDetector), but as far as the simulation is
concerned, these classes correspond to the ``entire'' detector
detector. (``Entire'' is in quotes because what makes up the
``entire'' detector is somewhat subjective. A dewar, for instance,
may or may not be thought of as part of the detector.)
```

MJGeometryCloverCrystal MJGeometryNaIBarrel

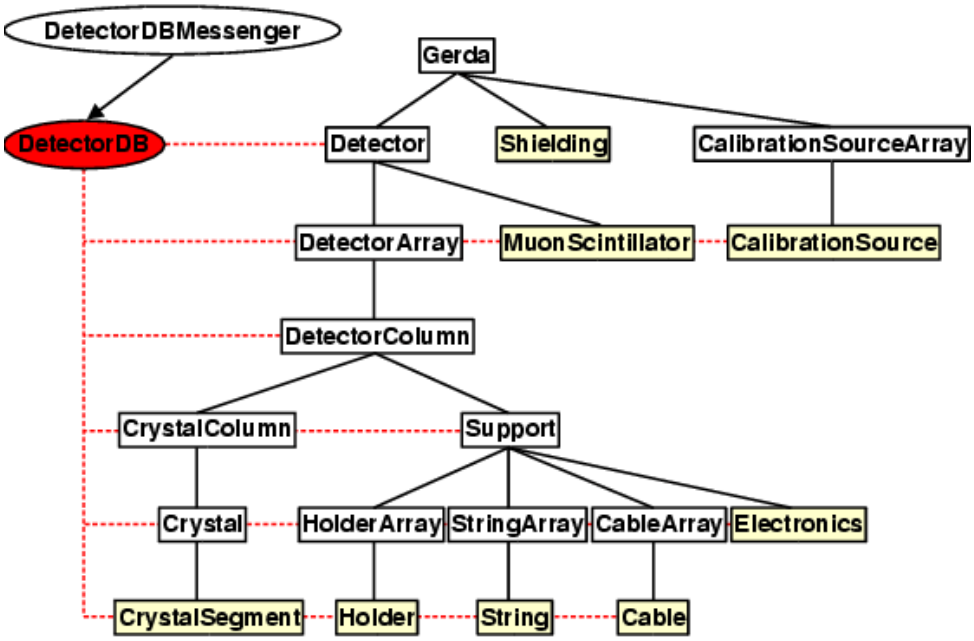


Figure 14.1: Class structure for GerdaArray

```
These classes are subparts of detectors. For examples, the
MJGeometryCloverDetector contains pointers to four
MJGeometryCloverCrystals, and each crystal is treated as a
separate object. When the four crystals are instantiated as
part of the MJGeometryCloverDetector, however, the entire
clover detector is handled as as object incorporating the
crystals.
```

## 14.2 GERDA Geometry

The GERDA geometry can be chosen by setting the command

```
/MG/geometry/detector [detector]
```

in the corresponding macro. For the standard GERDA geometry, the detector has to be "GerdaArray". Also other geometries exist that are related to the GERDA experiment, i.e. they are either modified geometries or teststands.

The geometry is organized in a group of classes: the following figure gives an overview of the structure. The class names in the figure do not include the prefix GEGeometry since it is common to all classes displayed. The class "GEGeometryGerda" inherits from "MGGeometryDetector", i.e. it is the basic class which gets instantiated from the MaGe framework. Since the real geometry of the experiment is still under investigation and not fixed yet, the Monte Carlo is kept as flexible as possible. For that reason a database class, "GEGeometryDetectorDB" is introduced which governs all the parameters for the geometry. Those parameters can be adjusted by commands, i.e. there exists a messenger class which controls the database. As can be seen

in the diagram, the database is connected to all classes in the geometry structure.

The first three branches “Detector”, “Shielding” and ”CalibrationSourceArray” are rough groupings. In order to focus on particular parts of the geometry each of those branches can be switched off. The calibration sources for example are not needed for the study of internal background of the crystals and the support. The detector itself consists of an array of crystals which is made out of bare crystals (or crystal segments) and a support structure that includes holders, strings, cables and electronics. This allows easy replacements of parts of the geometry for test reasons or changes in the setup. Auxiliary detectors are also installed, namely the muon plastic scintillator on top of the vessel and photomultipliers inside the water tank (not yet in the Monte Carlo). The shaded classes are those in which physical volumes are created. Also, most of the calculations for positioning and orientation are done via class functions. In order to add new elements to the simulation new classes are added. Existing examples are the scintillator and the germanium detectors. The solids and logical volumes should be introduced in the database class whereas the physical volume should be instanciated in the corresponding class. All used parameters are held flexible by adding the corresponding commands to the database messenger class. This has to include a flag that controls if the new element is to be included in the run or not. By following these instructions, the changes can be submitted to the CVS repository and are made available for other users

The following setups are currently available. They are defined by the settings in the corresponding macros, i.e. the deviation from the default values of the geometry parameters. Phase I: Unsegmented crystals. So far, this setup is only used in the study of cosmic muons. Phase II: 7 strings with 3 crystals. Each crystal is segmented into 6 phi- and 3 z-segments (standard). Simple Test Stand: A standard crystal within an aluminum cryostat which is placed in front of a capsulated source and lead bricks. The comparison of Monte Carlo and real data provided a first validation of the new MaGe code development.

## 14.3 Germanium detectors

*Kevin*

## 14.4 Muon veto

Table 14.1: Overview over the different photomultiplier distributions for the Cherenkov veto for GERDA.

| distributionnumber | # of pmts in pillbox | # of pmts in water | comment |
|---|---|---|---|
| 0 | 6 | 66 | standard distribution |
| 1 | 4 | 70 | alternative distribution |
| -1 | 0 | 0 | no Cherenkov veto |

The GERDA muon veto consists out of two systems. The plastic scintillator panels on top of the penthouse and the Cherenkov veto in the water tank surrounding the cryostat. In MaGe the plastic panels are hard coded in the geometry structure, while the Cherenkov veto has some commands to activate it or to select a different placement of the photomultipliers in the water

tank.
The photomultipliers in the GERDA geometry are currently constructed out of stainless steel housings and a PET-foil covering the upper part of the PMTs. If a photon hits the PET foil, a hit is scored and written into the output file. To account for the efficiency of the photomultiplier, it is recommendened to kill four of five photons in the MGManagmentStackingAction.cc, to save simulation time. Otherwise the efficiency has to be considered in the analysis of the root files. Currently there are two different distributions, the user can select (see table 14.1), or one can simply deactivate the Cherenkvo veto.

The Cherenkov veto is initiated with the macro command:

```
/MG/geometry/cherenkov distributionnumber
```

Of course, the optical processes have to be activated with

```
/MG/processes/optical true
```

and the optical output has to be selected:

```
/MG/eventaction/rootschema GerdaArrayOptical
```

Otherwise no Cherenkov effect takes part, and no photons are registered with the photomultipliers. The data is than accumulated in the root output file in the branches $ph\_\star$ and $PMT\_\star$ for the Cherenkov veto and in the branch $\star\_$ *scint* $\_\star$ for the plastic panels.

## 14.5 Electronics and calibration sources

*Daniel*

## 14.6 Infrastructure

*Jens*

# Chapter 15

# Test stands

*all*

## 15.1 Existing test stands

## 15.2 How to add a test stand to MaGe

# Bibliography


[1] S. Agostinelli *et al.* [GEANT4 Collaboration], "GEANT4: A simulation toolkit," Nucl. Instrum. Meth. A **506** (2003) 250.

[2] J. Allison *et al.*, "Geant4 developments and applications," IEEE Trans. Nucl. Sci. **53** (2006) 270.

[3] Physics Reference Manual, available at http://geant4.web.cern.ch/geant4.

[4] K. Amako *et al.*, "Comparison Of Geant4 Electromagnetic Physics Models Against The Nist Reference Data," IEEE Trans. Nucl. Sci. **52** (2005) 910.

[5] K. Amako *et al.* [GEANT4 Collaboration], "Geant4 and its validation," Nucl. Phys. Proc. Suppl. **150** (2006) 44.

[6] P. Lipari and T. Stanev, Phys. Rev. D **44** (1991) 3543.

[7] Astropart. Phys. **7** (1997) 357.

[8] H. Wulandari *et al.*, hep-ex/0401032 v1