

# **G4beamline User's Guide**

## **2.14**

**Tom Roberts**  
**Muons, Inc.**  
[tjrob@muonsinc.com](mailto:tjrob@muonsinc.com)

**November 2012**

<http://g4beamline.muonsinc.com>

**Copyright © 2004 – 2012 by Tom Roberts**  
**All rights reserved.**

**See license and distribution terms in section 1.2.**

# G4beamline User's Guide

## Contents

1	Introduction .....	5
1.1	Quick Start Guide .....	6
1.2	License and Distribution .....	10
2	Basic Concepts .....	11
2.1	Particle tracking in simulations .....	11
2.2	Units .....	12
2.3	Geometry .....	12
2.4	Coordinates .....	13
2.5	Rotations .....	16
2.6	Materials .....	17
2.7	Electromagnetic Fields .....	18
2.8	Visualization .....	18
2.9	Obtaining Results – Virtual Detectors and other NTuples .....	24
2.10	Random Number Generator .....	27
2.11	Tuning the Beamline .....	27
2.12	The Geant4 User Interface .....	29
2.13	Event Time Limit .....	29
2.14	Wall Clock Limit .....	29
2.15	Event and Track Numbering (EventID, TrackID) .....	30
2.16	Signals .....	30
3	Important Values that Affect the Validity and Accuracy of Simulations .....	31
3.1	Physics List .....	31
3.2	Tracking Accuracy Parameters .....	31
3.3	Electromagnetic Field Map Tolerance .....	33
3.4	Secondary Creation Threshold in Physics Processes .....	33
3.5	Track Cuts .....	33
3.6	Stepping and Hard Edges .....	33
3.7	Miscellaneous .....	34
4	Input File Description .....	35
4.1	Expressions .....	36
4.2	Element Names .....	36
4.3	Parameters .....	37
4.4	Pillbox Geometry and Dimensions .....	39
4.5	Absorber geometry and Dimensions .....	40
4.6	G4beamline Commands by Type .....	40
5	G4beamline Commands (Alphabetical) .....	44
6	Examples .....	96
6.1	Example1 – Simple Tracking and Virtualdetectors .....	97
6.2	ExampleN02.g4bl – The Geant4 ExampleN02 .....	97
6.3	FieldLines.g4bl .....	98

6.4	Study2Cooling.g4bl .....	98
6.5	MultipleScattering.g4bl .....	100
6.6	TungstenTarget.g4bl .....	100
6.7	MICE_StageVI.g4bl .....	101
6.8	Idealized_g-2.g4bl .....	108
6.9	SpaceCharge.g4bl .....	109
6.10	SampleMovie.g4bl .....	111
6.11	Movie.in .....	111
6.12	triplet.sh – tune a quad triplet for point-to-point focus .....	112
6.13	emittancematch.sh – attempt to match a quad triplet into a solenoid .....	113
7	Tips and Techniques .....	114
7.1	Getting Help on Using G4beamline .....	114
7.2	Reporting Bugs in G4beamline .....	114
7.3	Requesting New Features in G4beamline .....	114
7.4	Getting Help on Individual G4beamline Commands .....	114
7.5	In what Directory should I Work? .....	115
7.6	Files Common to Multiple Simulations .....	115
7.7	Basic Execution in a Command-Line Environment .....	115
7.8	Basic Execution in a GUI Environment .....	116
7.9	Putting Shielding into a Simulation .....	117
7.10	How to Debug a Simulation .....	118
7.11	Geant4 Commands .....	119
7.12	Obtaining Plots and Histograms .....	119
7.13	Obtaining Pictures of the System and Events .....	120
7.14	Warning and Error messages – Which Ones can be ignored .....	121
7.15	Secondary Tracks and Particles .....	122
7.16	Finding Example Input Files using the XXX command .....	123
7.17	Parameterizing the Input File .....	123
7.18	Setting Fields of Magnets .....	124
7.19	Tuning Bending Magnets .....	125
7.20	Setting the Phase of RF Cavities (pillbox) .....	127
7.21	Tuning the maxGradient of RF Cavities (pillbox) .....	127
7.22	Multiple Jobs in Parallel .....	128
7.23	Performing a Scan of Values of Some Parameter .....	129
7.24	Visually Scanning Events via the Command Line .....	129
7.25	Optimizing the Value of Some Parameter(s) .....	130
7.26	Using Two or More Reference Particles .....	131
7.27	Fitting to Plots and Histograms in HistoRoot .....	131
7.28	Interfacing to Other Programs .....	132
7.29	EventID and TrackID, and Encoding Information in them .....	132
7.30	Examining Outlier Events .....	132
7.31	Increasing the Number of Events Displayed Visually .....	134
7.32	Building G4beamline, Adding Your Own Code .....	134
7.33	Displaying Magnetic Field Lines .....	136
7.34	Setting the Phase of RF Cavities (rfdevice) .....	137
7.35	Tuning the maxGradient of RF Cavities (rfdevice) .....	138

8	Advanced Topics .....	140
8.1	Writing Scripts Using G4beamline .....	140
8.2	Violating the Rules on Geometrical Intersections .....	141
8.3	Making a Movie .....	141
8.4	User Code for the usertrackfilter Command .....	146
8.5	Multiple Instances of G4beamline using MPI .....	147
8.6	G4beamline Helper Programs and Scripts .....	152
8.7	Setting up an rfdevice .....	154
9	File Formats .....	166
9.1	BLTrackFile (generated by <i>virtualdetector</i> , read by <i>beam</i> ) .....	166
9.2	Trace File .....	166
9.3	FOR009.DAT .....	166
9.4	BLFieldMap .....	167
9.5	Window Files .....	170
9.6	Root Files .....	170
10	Acknowledgments .....	171
	Appendix 1 – README.txt .....	172
	Appendix 2 – README-Linux.txt .....	175
	Appendix 3 – README-Windows.txt .....	178
	Appendix 4 – README-MacOSX.txt .....	179
	Appendix 5 – Annotated Output from Example1.g4bl .....	181
	Appendix 6 – Particle IDs .....	185
	Appendix 7 – Error Messages .....	194
	References .....	196

# 1 Introduction

G4beamline is a particle tracking and simulation program based on the Geant4 toolkit [1] that is specifically designed to easily simulate beamlines and other systems. It is flexible enough to simulate complex beamlines like the MICE muon beam, the Neutrino Factory Study 2 SFOFO muon-cooling channel, complex helical cooling channels, and many others. Because of its simple and straightforward method of specifying the system to be simulated, it is also well suited for quickly answering questions about particle interactions and tracking (e.g. “On average, how much energy does a 150 MeV proton lose in a 1 mm Al window?”, “How large does the multiple-scattered beam grow 20 meters downstream of the window?”). As a Cosmic Ray “muon beam” is included, the notion of “beamline” can be rather more general than usual.

The primary advantage of using G4beamline is that its description of the simulation is commensurate with the complexity of the system being simulated, instead of being a significantly more complicated C++ program. Most users need not face the challenges of learning C++ programming and the details of the Geant4 toolkit – to use G4beamline there is no need to: a) know C++, b) learn the many aspects of the Geant4 toolkit API, c) face the non-trivial challenges of installing the Geant4 toolkit and all its required libraries, and d) learn how to solve any problems that arise while linking a complicated and rather large program. All of that is done during the production of the G4beamline distribution. Users with special needs can download and install the source distribution, and learn how to build the program, which will permit them to add their own C++ code and custom commands to G4beamline – this can be *much* simpler than the direct use of Geant4 and its libraries.

The basic structure of a G4beamline simulation is to first define beamline elements (magnets, beam pipes, windows, RF cavities, etc.), including their geometry, materials, fields, etc., and then to place them into the world, usually along the beam direction. As bending magnets can be modeled, the “beam direction” can change – see “Centerline Coordinates” below; it remains simple to place elements along the nominal beam centerline. It should be noted that a G4beamline simulation is closer to specifying a real beamline than it is to the abstractions and approximations used in most accelerator-physics codes. All descriptions and configurations are contained in a single ASCII input file, which also provides values for various program parameters, specification of the initial beam, etc.

The tracking of particles through the simulated system is as accurate and realistic as the Geant4 toolkit implements. The input file selects from any of the Geant4 physics lists, and can set values for the various Geant4 tracking-accuracy parameters. This permits users to make trade-offs between CPU time and simulation accuracy. Similarly, G4beamline permits the specification of magnetic map parameters, permitting a trade-off between memory usage (and the CPU time to generate the map) and simulation accuracy.

While G4beamline can make it rather simple to specify a simulation, it cannot substitute for knowledge and experience about the problem domain or about particle-tracking simulations in general. Like all computer programs, G4beamline is prone to “garbage in, garbage out”, especially when used by unskilled users. It is **strongly** suggested that you use visualization to verify the geometry of your simulation and that a handful of particles are tracked properly through it. Whenever possible you should arrange to track through a simple geometry that you can compare to independent results, to make sure that what you think is happening actually does occur in the simulation.

This document does not discuss compiling and building the G4beamline program. That is an advanced topic complicated by the complexity of linking a Geant4 executable with several external libraries and diverse visualization drivers. Most users will not need to do that, and can just use the distribution as is. The README-\*.txt and BUILD.txt files in the distribution and the appendices provide considerable detail about how to build the program.

## 1.1 Quick Start Guide

This section gives a quick overview of how to install and run G4beamline. It cannot discuss all the subtleties involved in performing simulations. New users should note in particular section 3 on important values that affect the accuracy, and section 7 on tips and techniques.

### 1.1.1 Installation

**NOTE:** The G4beamline distributions no longer include the *HistoRoot* program. It is available at <http://historoot.muonsinc.com>.

G4beamline requires the Geant4 data needed by the physics list used in the simulation. These data are placed into a directory named *Geant4Data* that is normally located in the user's HOME directory. The first time you run G4beamline, a Java program is run to assist you in downloading these data. If you did not download all data initially, and find you need other components later, you can access the download assistant from the Help page of the GUI or by running "g4bldata -install".

#### 1.1.1.1 Windows

##### Prerequisites:

- Java SE Runtime Environment 1.5 or later (<http://java.sun.com>) – if you have the JDK installed that includes the runtime environment.
- Root 1.24 or later (<http://root.cern.ch>).

Root is not required if you will use only ASCII output files (though you will probably find the *HistoRoot* program useful to generate plots of Root and ASCII files, and it requires a specific version of Root).

The distribution file is **G4beamline-VERSION.msi**, available from <http://g4beamline.muonsinc.com>. Simply download it and execute it with administrator privileges. By default, it will install files into "C:\Program Files\MuonsInc\G4beamline", and place icons onto your Desktop and into the Start menu. It will also place a copy of the G4beamlineExamples and the G4beamlineDocumentation into "My Documents\G4beamline Examples". Delete the downloaded file after installing it.

To use the command-line programs, you must a) install the Cygwin environment, b) use its bash shell, and c) add the G4beamline programs into your PATH:

```
source "C:/Program Files/MuonsInc/G4beamline/bin/g4bl-setup.sh"
```

You may put this into your \$HOME/.bash\_profile. There is a similar *g4bl-setup.csh*.

The first time you run G4beamline, it will launch a helper program (G4blData) to download the required Geant4 data files.

### 1.1.1.2 Linux (Intel)

#### Prerequisites:

- Java SE Runtime Environment 1.5 or later (<http://java.sun.com>) – if you have the JDK installed that includes the runtime environment.
- Root 1.24 or later (<http://root.cern.ch>).

Root is not required if you will use only ASCII output files (though you probably will find the *HistoRoot* program useful to generate plots of Root and ASCII files, and it requires a specific version of Root).

The distribution file is **G4beamline-VERSION-Linux.tgz**, available from <http://g4beamline.muonsinc.com>. Simply download it and un-tar it in your \$HOME:

```
tar -xzf g4beamline-VERSION-Linux.tgz
```

Delete the downloaded file after doing this.

This will construct an icon on your Desktop to run G4beamline:

```
g4beamline-VERSION-Linux/bin/g4bl-icon
```

You can copy this icon into any menus of your desktop manager.

To use the command-line programs, you must add the G4beamline programs into your PATH:

```
source g4beamline-VERSION-Linux/bin/g4bl-setup.sh
```

You may put this into your \$HOME/.bash\_profile. There is a similar *g4bl-setup.csh*.

The first time you run G4beamline, it will launch a helper program (G4blData) to download the required Geant4 data files.

### 1.1.1.3 Mac OS X (Intel)

#### Prerequisites:

- Java SE Runtime Environment 1.5 or later (<http://java.sun.com>) – if you have the JDK installed that includes the runtime environment. Java is normally installed on Mac OS X.
- Root 1.24 or later (<http://root.cern.ch>).

Root is not required if you will use only ASCII output files (though you probably will find the *HistoRoot* program useful to generate plots of Root and ASCII files, and it requires a specific version of Root).

The distribution file is **G4beamline-VERSION.dmg**, available from <http://g4beamline.muonsinc.com>. Simply download it, open it, and drag the *G4beamline* icon to your /Applications folder. Drag the G4beamlineDocumentation and G4beamlineExamples folders to your desktop, if desired. Then eject the installer disk image and move the downloaded file to the trash. As usual, you can then drag the application icon from /Applications to the Dock.

If you want to use the command-line programs, you must add the G4beamline programs into your PATH:

```
source /Applications/G4beamline.app/Contents/Resources/bin/g4bl-setup.sh
```

You may put this into your \$HOME/.bash\_profile. There is a similar *g4bl-setup.csh*.

The first time you run G4beamline, it will launch a helper program (G4blData) to download the required Geant4 data files.

### 1.1.2 Initial Test

After installing G4beamline, the first thing to do is to try it on some of the example files. You can double-click the G4beamline icon; on Windows/Cygwin, Linux, or Mac you can also execute the command *g4blgui*. This should bring up the G4beamline GUI window (if it doesn't, there is a problem with either your Java installation or your PATH). Once the GUI opens, click on the *Browse* button and navigate to the G4beamlineExamples directory and select *Example1.g4bl* (on Windows, go to "My Documents\G4beamlineExamples"). Clicking on the *Run* button should execute the Example1 simulation (takes just a few seconds to simulate 1000 events). Try selecting the *best* viewer, select (say) 10 events/run, and click *Run* again to see the OpenInventor display of the system and a handful of events; use the mouse to rotate and zoom the display.

If you have *tcl* and the *bash* shell available (Linux, Mac, Windows/Cygwin), you can run the full test suite of G4beamline. Simply *cd* to the install-directory/test and issue the command *./all*. Individual tests can be run singly, if desired. At present, it takes about 10 minutes to run all 97 tests on a reasonably modern computer.

### 1.1.3 Using the G4beamline Command Line

On Linux and Mac OS X, the command line is the traditional way to run programs. On Windows this is possible, but you must install the **Cygwin** environment first (<http://www.cygwin.com>) and use its *bash* shell rather than the Windows *cmd*. You can execute the G4beamline GUI in all environments via the command *g4blgui* executed from the install directory via an absolute path.

The bare *g4beamline* command specifies the input file, plus whatever additional parameters are needed (see the *param* command). It is rarely used by users (see scripts below). Its syntax is:

```
g4beamline input.file name1=value1 name2=value2 ...
```

Here *input.file* is the filename of the file describing the simulation ('-' means stdin), and *name1* and *name2* are parameters to be passed to the program or to *input.file*. Note that users will rarely (if ever) execute this command directly – most users will use the *g4bl* script instead of *g4beamline* itself; its arguments are the same:

```
g4bl input.file name1=value1 name2=value2 ...
```

The *g4bl* script will ensure that the necessary libraries from the binary distribution will be found and used, and sets G4BL\_DIR for finding *viewer.def* and the physics data files. Note that the *g4bl* script relies on being executed via an absolute pathname, so the distribution *bin* directory should be put into your PATH (or you can manually type the full path to *g4bl*). The *g4beamline* program has no such expectation, but to run it you need to have all the necessary libraries and physics data-sets available in your environment, which happens when a) you setup to build G4beamline, b) you run the *g4bl* script, c) you run the *g4blgui* script, or d) you double-click the *G4beamline* icon.

In addition, the *g4bl* command arguments can contain commands to be interpreted before reading the *input.file*. These are interpreted in the order given, after all the parameters on the command-line are defined. If the command has arguments, it must be quoted because of the spaces between arguments. The usual commands this is used for are *eventcuts* (used by *g4blgui*) and *movie* (used by users to add the movie NTuples to an existing simulation):

```
g4bl input.file movie name1=value1 name2=value2 ...
```

For visualization, the *viewer.def* file is looked for in the current directory and then in the directory from G4BL\_DIR in your environment. If you aren't using the *g4bl* script, you probably want to set G4BL\_DIR to the distribution directory, so you don't need to link *viewer.def* into every working directory (G4BL\_DIR is set when you setup to build *G4beamline*, and also by *g4bl*, *g4blgui*, and the icon).

G4beamline has two modes of operation: visualization and tracking. If the parameter *viewer* has been set to anything other than *none*, it will run in visualization mode, which displays an image of the simulated system (as specified in the *viewer.def* file for the viewer specified). For *viewer=none* (the default), G4beamline will run in tracking mode. In visualization mode any NTuples defined in the *input.file* will not be written to files. The simplest and most common way to invoke visualization mode is to put an argument *viewer=best* onto the command line, or select a viewer in the GUI. See section 2.8 for a list of the supported visualization drivers.

A very useful trick is to run G4beamline interactively, just to issue help and list commands:

```
g4bl -  
cmd: help  
cmd: help beam  
cmd: list materials
```

You may find it useful to keep this open in one window while you edit your *input.file* in another window.

Another useful trick is to add “steppingVerbose=1” to the command line. This will print one line per physics step, so you can see precisely what happens during the simulation. This is very verbose, so you want to limit the number of events to some small number (or type ^C to stop it). The prints are about 120 characters wide by default, so it's best to do this in a wide window. You can change the information printed, see section 3.2.1.

A large number of useful tips and techniques are described in section 7.

### 1.1.4 The HistoRoot Program

**NOTE:** The G4beamline distributions no longer include *HistoRoot*. It is available at <http://historoot.muonsinc.com>.

*HistoRoot* is a program to generate Root plots and histograms from Root and ASCII files containing NTuples. While everything *HistoRoot* does can be done via Root commands and code, the *HistoRoot* program makes it unnecessary to know Root and C++, and guides you through most common tasks. *HistoRoot* can read multiple Root files (*TFile*), and can generate plots from the *TNtuple*-s in them. It can

also read ASCII files (in either .csv or column format) and convert them to *TNtuple*-s for plotting (see the *Help* for details on the ASCII file format; a structured comment can associate names to the columns).

*HistoRoot* was inspired by the HistoScope program [3], though its user interface is completely different. Having a simple GUI to generate plots and histograms is quite useful, and having sliders that interactively impose cuts on the histogram is a great convenience.

## 1.2 License and Distribution

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

<http://www.gnu.org/copyleft/gpl.html>

This application includes software developed by others; see the Acknowledgements in section 10 below.

## 2 Basic Concepts

Any physical component of the simulation is called a beamline element, or just an element. Every element inherently has a geometrical shape and a material; some elements also have electromagnetic fields associated with them. Some elements are quite simple, merely defining a box or tube made of some material; other are quite complex, defining in detail the construction and electromagnetic fields of a complicated object such as an RF cavity.

Elements can often be nested inside each other; any element can be placed inside another element, but not all elements can act as the parent of others (in general, simple elements can be parents, complex ones cannot; most elements that generate an electromagnetic field cannot be parents). The notes and help for each element describe whether or not it can be the parent to other elements. This is the natural way Geant4 builds the geometry hierarchically: a “daughter” volume is said to be placed into its “mother” or “parent” volume (every element is also a volume, perhaps several volumes). The “World” volume is the ancestor of all volumes, and in G4beamline the World volume automatically grows to hold everything that is placed into it. In G4beamline, individual elements have their own material(s), and when a daughter is placed inside a parent, the daughter’s material applies inside the daughter, and the parent’s material applies inside the parent outside any daughters; many elements use their parent’s material by default.

G4beamline attempts to model the simulated world in as simple and straightforward manner as possible, so in most cases a single object you might place into a real beamline will be represented as a single beamline element in the program. So, for example, a *genericbend* element represents a (generic) bending magnet and its field; the *genericbend* command defines an individual magnet type, including all details of both its geometry and field; each placement of the magnet puts the geometrical object into the simulation and also generates the magnetic field appropriate to the magnet. Other elements behave similarly.

In the Geant4 toolkit, only simple geometrical shapes are available, and complicated objects must be built up from cylinders, boxes, etc. In *G4beamline* there is a rather large list of pre-defined elements that are much more complicated, such as an entire RF cavity (including time-dependent E- and B-fields, optional windows, etc.), or a complete absorber assembly (modeled after the absorbers of the MICE experiment, suitable for neutrino factory cooling channel simulations). *G4beamline* includes simple pipes, cylinders, spheres, boxes, extruded polygons, and polycones, so you can build up complicated objects from them; in general, it is both easier and better to use the pre-defined elements whenever they are suitable for your needs. In cases where the pre-defined elements are not sufficient, it is possible to add additional elements to the program (this requires C++ programming and the installation of the complete *G4beamline* build environment – see *BUILD.txt* in the distribution and the doxygen documentation of the classes).

### 2.1 Particle tracking in simulations

The accuracy of a simulation of tracking particles through a system depends primarily on how realistic the simulation is. This has many aspects:

- How closely the geometry of the simulation matches the real geometry
- How accurately the real electromagnetic fields are modeled in the simulation

- How accurately the material properties of the real objects are modeled in the simulation
- How well the modeled beam distributions match the real beam
- How accurately the program tracks particles
- How accurately the implementations of physical processes match the real world
- Etc.

While G4beamline provides facilities for the user to specify most of these, and gives them reasonable default values, it is really the user's responsibility to ensure that the results of a simulation are accurate enough to be useful.

In *G4beamline*, Geant4 particle tracking is used directly with the tracking-accuracy parameters of Geant4 settable by the user (see section 3.2.2).

## 2.2 Units

In G4beamline, all physical sizes and dimensions are specified in millimeters, just as in Geant4. Most other units are as in Geant4, except rotation angles are in degrees and EM fields are in Tesla and MegaVolts/meter. The units of all arguments to all commands are given in section 5 of this document and in the output from "*help commandname*".

## 2.3 Geometry

As in Geant4, geometry in G4beamline is specified at the element level and in the placement of elements. Each element has an inherent shape and size, which are usually parameterized, that determine the basic geometry of the element in isolation. Elements can be placed into the simulated World, or into other elements placed into the world, with specified geometrical relationships consisting of both rotations and offsets relative to the parent's coordinates. This will become obvious when you look at the various element definition commands (e.g. the *box* and *tubs*<sup>1</sup> commands), and the *place* command.

Because of the way Geant4 does tracking, there are restrictions on the geometrical intersections of elements:

- Every daughter element must be wholly contained in its parent
- No sibling elements can intersect each other in any way

This results in a strict hierarchical arrangement of element volumes, with the World volume as the root and outermost volume from which all other volumes are descended. The World volume, and the *group* element, will automatically expand to the size required to hold all daughter elements placed into them.

G4beamline has a geometry test that will verify these requirements for points distributed on the surfaces of every element. Due to round off in tracking, the geometrical intersections have a default accuracy of 10 microns; the geometry test has a default tolerance of 2 microns; in practice these are usually reasonable values, but they can be changed when required by simulations at a different scale.

---

<sup>1</sup> "*Tubs*" is the name Geant4 uses for a tube or a cylinder (a tube with innerRadius=0).

<sup>2</sup> This comparison can only be done statistically, not on an individual track basis, because different values of maxStep inherently use different numbers of random numbers, causing variations in stochastic processes like multiple scattering and

Note that the Geant4 toolkit has “logical volumes”, “physical volumes” and “placements”. In G4beamline, these are not visible to the user; elements are modeled as single objects, and you simply place the objects where they belong, keeping in mind the above requirements on intersections. The C++ code implementing each element deals with the Geant4 details, of course. Unlike Geant4, in G4beamline an element that generates an electromagnetic field is a single object that includes both its geometrical volume(s) and its field.

An additional limitation of the geometry in G4beamline is the requirement that any element that is to be the parent of other elements be placed after the children are placed into it. Once a given element has been placed (anywhere), it can no longer be used as the parent of other elements.

## 2.4 Coordinates

There are four types of coordinates used in G4beamline:

1. Global coordinates
2. Local coordinates
3. Centerline coordinates
4. Reference coordinates

All coordinate systems are Cartesian, with axes labeled X,Y,Z,T. For geometrical placement, T is irrelevant (but it is needed during tracking). X,Y,Z are always right handed, usually with the +Z direction in the nominal direction of the beam, X is normally beam left, and Y is normally up.

### 2.4.1 Global Coordinates

Global coordinates are merely the local coordinates of the World volume – the outermost physical volume of the simulation, which defines the entire world known to the program. All beamline elements and physical objects of the simulation are contained within the world volume. Note that the World volume automatically expands to hold any elements placed into it, so the user need not be concerned with how large it is. Any particle that reaches an edge of the World volume is killed right there.

Tracking is performed using global coordinates, and electromagnetic fields are always determined using global coordinates (but they are specified using the local coordinates of their element; each placement of an element into the simulation includes the local  $\Leftrightarrow$  global coordinate transformations needed).

### 2.4.2 Local Coordinates

There is a set of local coordinates for each and every element. If the element has an associated field, the field of the element is specified in local coordinates, and the program automatically keeps track of the global  $\Leftrightarrow$  local coordinate transforms required for each placement of the element (including rotations for vector fields).

Whenever an element is placed inside a parent element (e.g. a group or a box), the parent’s local coordinates are used in the place command, except for the world volume. When placing an element into the World volume you can specify either Centerline coordinates (the default) or global coordinates (the local coordinates of the parent, here the World volume).

### 2.4.3 Centerline Coordinates

Centerline coordinates are intended to represent the nominal centerline of a beamline, with the Z axis running down the center of the beamline, the X axis is beam left, and the Y axis is up. This includes corners generated by bending magnets. The centerline coordinates are initially identical to the global coordinates, but the *start*, *corner*, and *cornerarc* commands will change that (each bending magnet is normally immediately followed by a matching *corner* or *cornerarc* command). By default, elements placed into the World volume use centerline coordinates to specify the placement, but global coordinates can be used if desired. Centerline coordinates apply only to the World volume. They are piecewise-linear, and inside bending magnets they will differ from the theoretical reference-orbit coordinates commonly used in accelerator physics; the *cornerarc* command will approximate reference-orbit coordinates by using three corners chosen to have the same length along Z as the reference-orbit coordinates have along the arc. Note that the *corner* and *cornerarc* commands cannot have an angle greater than 90°; if necessary, this limit can be worked around by using two *cornerarc*-s in a row with half the desired angle.

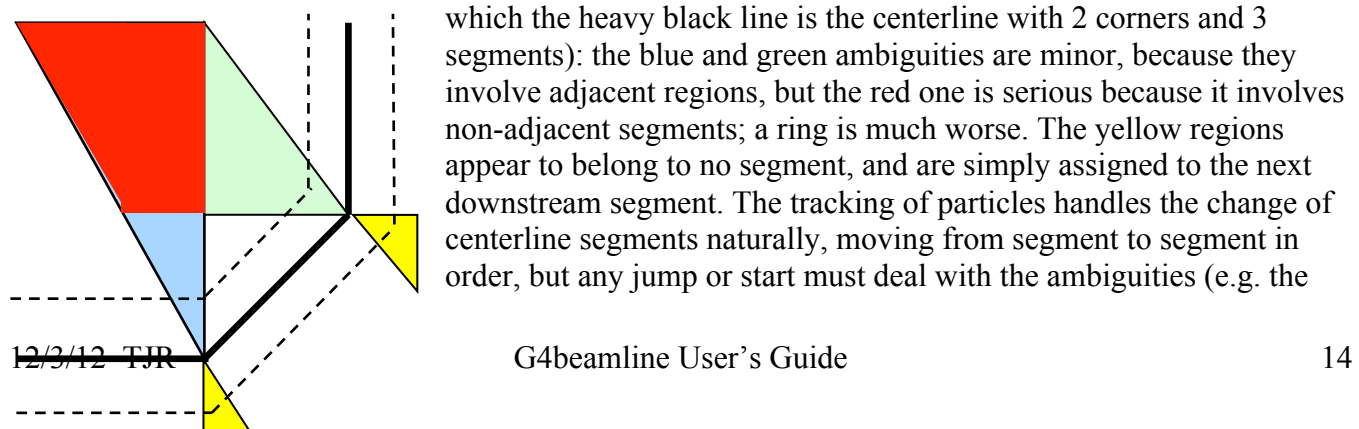
In X and Z, centerline coordinates correspond to what a surveyor might draw on the floor – a piecewise-linear centerline along which the various beamline elements will be placed. Note that just like a real beamline, in G4beamline the fields of bending magnets must be adjusted (“tuned”) so that the desired reference particle will travel along the centerline of the beamline. In general, the fringe fields of bending magnets will require them to be offset slightly from their nominal position relative to a corner of the centerline. See section 7.18 for a description of how to place and tune bending magnets.

If your *input.file* has no *start*, *corner*, or *cornerarc* commands, then the centerline coordinates will be the same as the global coordinates, everywhere.

Note that in the input file, each *start*, *corner*, or *cornerarc* command affects the centerline coordinates immediately. This means that all elements to be placed between a given pair of corners must be put into the input file between those *corner* commands. If you attempt to place an object using a Z position in front of a previous *corner* or *cornerarc*, or behind a following one, it will almost surely not do what you intend. Because of this, it is quite difficult to place any element inside a bending magnet using centerline or global coordinates; the best way to do that is to use the *genericbend* object as the parent for the placement.

Because centerline coordinates can only apply when placing elements into the World volume, bending magnets must normally be placed into the World volume, not into a *group* or other parent volume.

**NOTE:** the transform from global to centerline coordinates must be performed at the start of tracking each particle. This transform can be ambiguous in certain regions near corners – see the figure at left (in



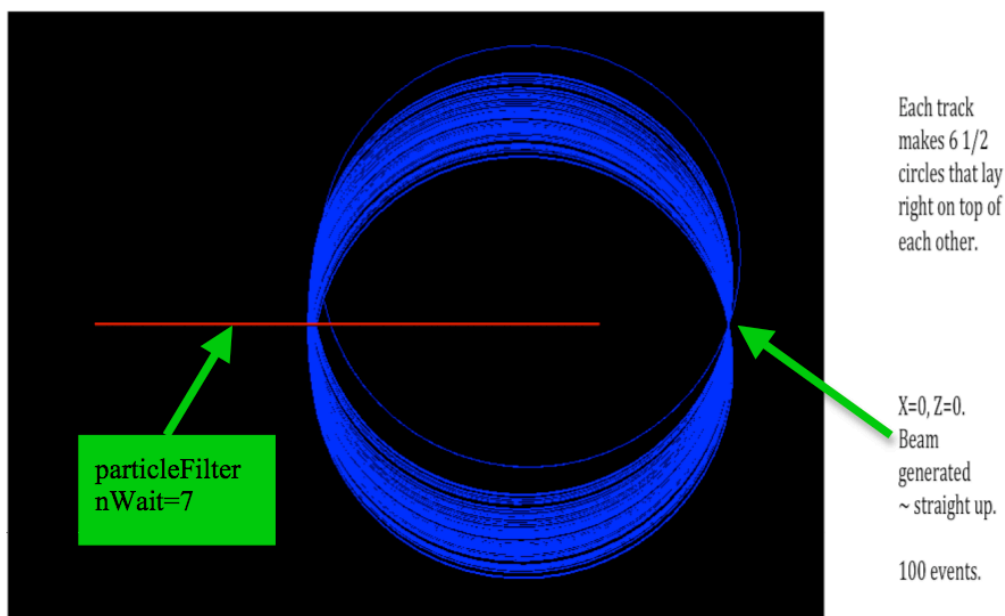
initial point of beam particles or secondaries). The solution is to specify a *radiusCut* for every *start*, *corner*, and *cornerarc* command (this is a limit on a particle's radial distance from the centerline, indicated by the dashed lines in the figure). This removes the serious ambiguities, and the remaining minor ones are resolved by using the earliest region of the centerline for which the global coordinate point is inside the *radiusCut*. Most beamlines have apertures that can be used to determine an appropriate *radiusCut* in each segment of the centerline. If no *radiusCut* is specified the ambiguities can become so bad that they prevent the use of centerline coordinates, especially if there are many *corners*, large angles, or if the *tune* command is used (which “jumps” the *tune* particle); simulating a ring may not work at all without a *radiusCut* (and needs the *ring* argument to the *start* command, which assists in resolving the ambiguities unique to a ring).

#### 2.4.4 Reference Coordinates

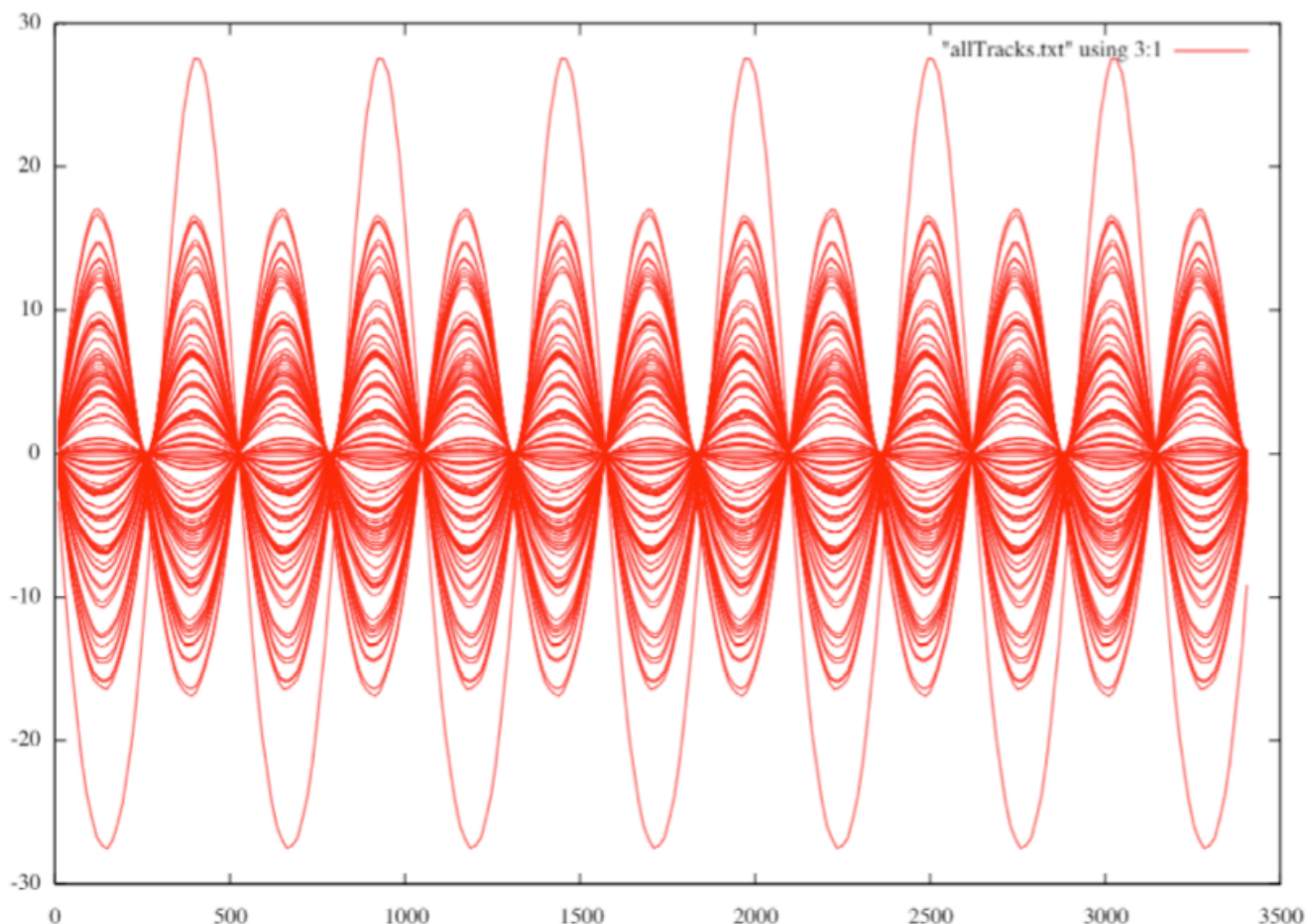
Reference coordinates are the usual coordinates used by accelerator physicists, with the Z axis along the reference orbit, X to the beam left, and Y up. In G4beamline, the actual track of the reference particle is used for the Z axis. As the reference track is stepwise linear between the physics steps of the tracking, it is usually necessary to specify a rather small *maxStep* (often 10 mm is about right). Note these coordinates can be used only for output; objects cannot be placed using them, and tracking always uses global coordinates.

As an example, consider a system having a uniform  $B_y = 8$  T with vacuum everywhere, and a Gaussian beam injected with  $P_y = 0$  centered on the +Z axis; there is a *particlefilter* to limit the number of circles each particle can take. Each particle (including the reference particle) traces  $6\frac{1}{2}$  circles that lay right on top of each other. When using global coordinates for the trace, the circles are quite evident; using reference coordinates, the tracks are sinusoids (see figures below).

#### Global Coordinates view – Z vs. X



## Reference Coordinates view – Z vs. X



### 2.5 Rotations

In G4beamline, all rotations are specified relative to the fixed coordinate axes of the parent volume into which a given element is being placed. This is the natural way to think of placing an object into a room, where the object is the element being placed, and the room is the parent volume into which it is placed.

Rotations can be specified in the *place* command, and in a few other commands. They are specified by a *rotation* parameter, the value of which is a comma-separated list of rotations around the parent's coordinate axes, which are applied in order (left-to-right) to the object; values are in degrees. So "rotation=Y90,X30" means:

1. Start with the object's local coordinate axes aligned with the parent's coordinate axes
2. Rotate the object (and its local coordinate axes) by 90 degrees around the parent's Y axis
3. Then rotate the object (and its local coordinate axes) by 30 degrees around the parent's X axis.

Negative and decimal values of the angle are permitted. The direction of a positive rotation is given by the right-hand rule: use your right hand to grasp the rotation axis with your thumb in its positive direction; a positive rotation goes in the direction of your fingers.

If both a rotation and an offset are specified when placing an element, the rotation occurs first, then the offset. This makes sense, as the location of the center of the object is specified when placing objects.

When the parent is placed with a rotation and/or an offset, the relationship between daughter and parent remains unchanged, so the effect on the daughter is cumulative, all the way back to the World volume.

When used in a *corner* or *cornerarc* command, the rotation is relative to the previous centerline coordinates. When placing into the World volume using centerline coordinates, the rotation is relative to the current centerline coordinates (as is natural). So, for example, one can simply place *genericquad* elements using centerline coordinates without rotation or offset and the nominal beam centerline will go right down their center.

**Note:** this is quite different from the way the Geant4 toolkit itself specifies rotations and placements.

## 2.6 Materials

Every physical object inherently is constructed out of some material. G4beamline uses the Geant4 methods of specifying materials. The Geant4 version of the NIST database is available, and is used to define any material that is not already defined (as long as it is known to the database). This database includes all elements with  $Z \leq 98$  plus a rather large list of other materials, including most of the ones commonly used in simulations. See “*help material*” for details (or the *material* command in section 4 below). Additional materials can be defined as needed, via the *material* command, but that is needed only for unusual materials or gases with nonstandard density. The Geant4 NIST database prepends “G4\_” to all material names, but in G4beamline that prefix is optional (G4\_Al and Al are the same material, aluminum from the NIST database).

Every element command has a *material* argument (or similar) to set the material for the physical object (some elements have more than one kind of material parameter). In many cases, the default is that of the parent volume, but refer to the individual command descriptions for the actual default values. The material of the World volume is determined by the parameter *worldMaterial*, which defaults to *Vacuum*.

Materials defined by the *material* command can be used to filter particle tracks. That is, the user can specify which types of particles will be killed whenever they enter any object made of this material. The arguments relevant to this are *keep*, *kill*, and *require*; *keep* and *kill* are lists of particle names, but *require* is an expression in terms of track variables, so quite complex filters can be constructed. For example, in a simulation of a neutrino source, it may be desirable to kill every non-neutrino track that hits a magnet or beam pipe, but let the neutrinos through to a detector far away. By defining the material “Iron” as below, and then using it as the *ironMaterial* of all magnets and the *material* of all beam pipes, only the neutrinos can escape. This way you can have a very large world, including a distant neutrino detector, and not waste time simulating uninteresting interactions in magnets or beam pipes, or tracking uninteresting particles over long distances.

```
material Iron Fe,1.0 keep=nu_mu,anti_nu_mu,nu_e,anti_nu_e
```

## 2.7 Electromagnetic Fields

In the real world, electromagnetic fields are generated by real structures. This is also true in G4beamline, where various elements include their electromagnetic fields. So a *genericbend* object includes not only the geometric shape of its iron and coils, it also includes its magnetic field. Placing a *genericbend* element into the world places both its iron and its field; any rotations or offsets used in the *place* command apply to both the iron and the field.

As G4beamline is intended to model beamlines, in many cases the field of an element is only valid inside its aperture (plus perhaps a short distance away for a fringe field). This is so for *genericbend* and *genericquad*, but *solenoid* and *fieldmap* have fields throughout space (which are truncated according to the tolerance specified or the limits of the map).

In Geant4 programs, computing the field is usually the dominant CPU time used during tracking. In order to minimize this CPU usage, G4beamline optimizes the computation of the field by using a global bounding box around the field of each element. The World volume is divided into rectangular voxels, and during initialization each voxel is given a list of all fields whose bounding boxes intersect the voxel. The voxel size is given by the parameter *fieldVoxels*, which defaults to “200,200,200” (200 mm on each side). To compute the field at a given point, the program computes the voxel containing the point, and then sums the fields from the voxel’s list.

**Note:** this is quite different from Geant4, where geometry and fields are strictly separate, and voxels are not used in the field computation.

## 2.8 Visualization

The ability to easily visualize the system being simulated is an important part of *G4beamline*. Without visually examining the system in detail, it is easy to make simple mistakes that completely invalidate the results of the simulation. Users are **strongly** urged to use visualization frequently.

In order to be visible, a given beamline element must be given a color. Every element accepts an argument “*color=1,1,1*”, where the 3 floating-point values are for Red,Green,Blue respectively, and must be between 0.0 and 1.0 (inclusive). “*color=invisible*” or “*color=*” is invisible, “*color=1,1,1*” is white, “*color=0,0,0*” is black, “*color=0,1,1*” is yellow, “*color=0.6,0.6,0.6*” is gray, etc. Most elements have a default color of *1,1,1* (white). An optional fourth value, alpha, can be given; it must be between 0.0 and 1.0 (inclusive), and represents the transparency of the element (0=transparent, 1=opaque). Transparency can be very useful, as it permits you to see tracks inside objects; a wireframe view will also permit that, but has many extraneous lines that confuse the eye.

By default, particle tracks are visible: red for negatives, blue for positives, and green for neutrals. See the *particlecolor* command to change this. If you run pions or muons you may be surprised to see all the neutrinos from their decays; see the *trackcuts* command to avoid tracking them. You may also be surprised by delta-rays, which are really low-energy electrons ( $e^-$ ); *trackcuts* can eliminate them, too. For example, when studying  $\pi^+ \rightarrow \mu^+ \rightarrow e^+$  decays, the following commands are useful to color them differently and to eliminate the neutrinos and other extraneous particles (e.g. delta rays):

```
particlecolor pi+=1,0,0 mu+=0,1,0 e+=0,0,1
trackcuts keep=pi+,mu+,e+
```

Once your beamline elements have been given colors (in your input file), and so have your particles, to actually visualize the simulation you need to select a visualization driver. G4beamline supports the following visualization drivers from Geant4:

Descriptive Name	Name	Type	Description
–	best	Interactive	Alias for the best viewer available, currently OIX/OIWin32.
OpenInventor	OIX	Interactive	Renders in 3d using either solid shading with transparency or wireframe mode (+ others), and permits the easy rotation, scaling, and movement of the image using the mouse.
OpenInventor	OIWin32	Interactive	The name of OpenInventor on Windows.
OpenGLImmediateX	OGLIX	Interactive	Basic OpenGL 3-D viewer.
OpenGLStoredX	OGLSX	Interactive	Basic OpenGL 3-D viewer.
OpenGLImmediateXm	OGLIXm	Interactive	Like OGLIX, except has Motif widgets to permit the easy rotation, scaling, and movement of the image. If you cannot get OpenInventor to work, try this one next.
OpenGLStoredXm	OGLSXm	Interactive	Like OGLSX, except has Motif widgets to permit the easy rotation, scaling, and movement of the image.
ASCIITree	ATree	Offline	A simple hierarchical description written to stdout.
DawnFile	DAWNFILE	Offline	Use the DAWN viewer.
GAGTree	GAGtree	Offline	Another geometry description written to stdout.
HepRepFile	HepRepFile	Offline	Use the WIRED viewer (Java).
HepRepFile	HepRepXML	Offline	Use the WIRED viewer (Java).
Wired	Wired	Offline	Synonym for HepRepFile.
Wired3	Wired3	Offline	Synonym for HepRepFile.
RayTracer	RayTracer	Offline	Generates JPEG files; use any image viewer for them.
VRML1FILE	VRML1FILE	Offline	Use a VRML viewer.
VRML2FILE	VRML2FILE	Offline	Use a VRML viewer.

Interactive viewers display the simulated system and events while G4beamline is running; Offline viewers write the image to a file for viewing using another program.

The parameter that controls visualization is called *viewer*, and to use a given visualization driver, simply add *viewer=best* to your command line (or name whichever driver you want from the list above). This puts G4beamline into a mode to perform tracking and visualization rather than just tracking beam through the simulated system.

Visualization requires the file *viewer.def*, which contains help text and the internal Geant4 commands required to invoke the selected viewer (it is a simple ASCII file with format documented in its comments). *viewer.def* is searched for in the current directory and then in the directory named in

`$G4BL_DIR` (which points to the G4beamline distribution directory); the file is `$G4BL_DIR/share/g4beamline/viewer.def`.

The G4beamline visualizations display all events of a run in a single image; some viewers consider each image to be an “event”, but for beamlines multiple events in an image are often useful. Once the viewer has been selected, if you are running g4beamline manually you need to start a run by waiting for the “idle>” prompt and then typing “/run/beamOn 23” (or however many events you want to display). When G4beamline presents you with an “idle>” prompt, this is the Geant4 internal command processor. *Viewer.def* uses these commands to setup and invoke the selected visualization driver, and to arrange for tracks to appear in the images. See the Geant4 User’s Guide [1] for details about these commands. Typing either “exit” or ^C (Ctrl-C) will terminate the program. The “idle>” prompt appears after the commands in *viewer.def* have been executed and the visualization viewer is set up. If you are finished, simply exit the program (^C), or issue “/run/beamOn 23” to run the next 23 events and visualize them (i.e. generate another image containing those events). The Offline drivers are smart enough to generate a new image file for every beamOn command (filenames vary for each driver). The *g4blgui* graphical user interface program automatically handles all this.

### 2.8.1 Additional Visualization Techniques

These techniques use the Geant4 user interface, via the *g4ui* command. You may want to look at the Geant4 help for these commands; to do so, go to the *examples* directory and type (any input file will do):

```
g4bl Example1.g4bl viewer=best
```

Once the viewer opens its window, you will get an *Idle>* prompt to which you can reply *help*<CR> and obtain help on any Geant4 UI command.

Note there are lots of options for the Geant4 viewers, and this is only a short list of those that users have requested in the past. If there is some aspect of drawing that you want, chances are fairly good that it is available.

#### 2.8.1.1 Different Drawing Scales

Many beamlines are much longer than they are wide or tall. This makes the transverse dimensions difficult to see in a viewer, and when magnification is increased to see them, only a very small portion of the beamline is visible at a time. This can be addressed by using different drawing scales for x, y, and z. The command to scale x by 2, y by 3, and z by 4 is:

```
g4ui when=4 "/vis/viewer/scaleTo 2 3 4"
```

Or type `/vis/viewer/scaleTo 2 3 4` to the *idle>* prompt.

#### 2.8.1.2 Changing the Background Color

To set the background color to white, use this command:

```
g4ui when=4 "/vis/viewer/set/background 1 1 1"
```

The 3 values are for red, green, and blue, and should each be a value between 0 and 1; black is “0 0 0” (the default). A fourth value for opacity can be given.

### 2.8.1.3 Drawing Axes

Drawing the coordinate axes can make it easier to find your way around the image of the system in a viewer. You can draw a set of axes at any desired location with any desired length. To draw coordinate axes of 100 mm length at x=1 y=2 z=3 (mm):

```
g4ui when=4 "/vis/scene/add/axes 1 2 3 100 mm"
```

### 2.8.1.4 Cut-Away and Section Planes

The Geant4 drawing interface includes cut-away and section planes. The interface is quirky and you'll have to search the Geant4 documentation and just play with it. Not all viewers support this. The commands are: /vis/viewer/set/sectionPlane and /vis/viewer/addCutawayPlane.

### 2.8.1.5 Starting up in Wireframe Mode

```
g4ui when=4 "/vis/viewer/set/style wireframe"
```

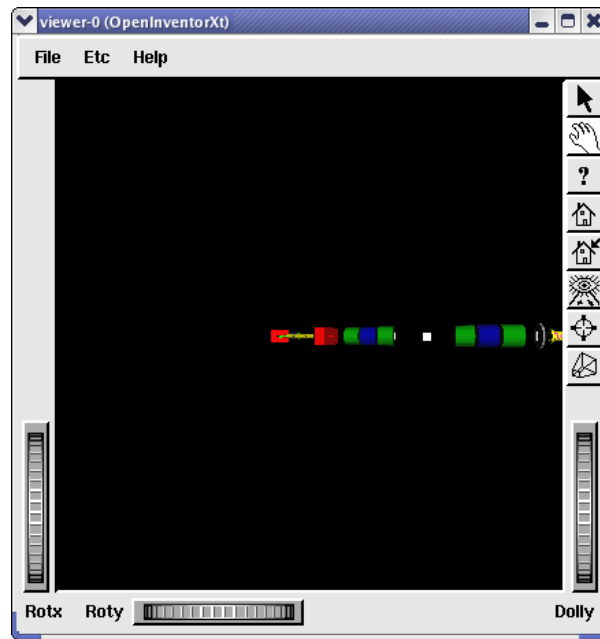
### 2.8.1.6 Filters

The Geant4 drawing interface includes filters that permit you to only draw certain particle types, or only particles that were created in a specified physical volume, or .... The interface is quirky, and you'll have to search the Geant4 documentation and just play with it.

## 2.8.2 The OpenInventor Visualization Driver

Because the OpenInventor driver (OIX/OIWin32) is so much more useful than the other drivers, it is the driver of choice (linked to the name "best"). This driver will permit you to do almost everything that any other driver can do, the exception being the Wired driver lets you modify the graphical attributes of individual objects by picking and editing (that is in the works for OpenInventor).

Like other X-windows programs, the OpenInventor driver of G4beamline can operate in either local or remote (networked) mode. The X display server (usually your desktop or laptop) must have the 3-D display capabilities of OpenGL installed, as must the X client (where G4beamline is running). When you startup the OpenInventor viewer, the window looks like this (MICE\_StageVI.g4bl):



The Rotx and Roty wheels rotate the image relative to the viewer, and the Dolly wheel moves in and out with a corresponding change of scale. The buttons along the right edge do the following:

Arrow	Select pick mode and cursor
Hand	Select move mode and cursor (default)
?	Help (not implemented)
House	Restore display to Home position
House with arrow	Set Home position to be the current display
Eye	Zoom so everything is visible
Gunsight	Not implemented
Perspective box outline	Select isometric or projected display

When the Hand cursor is active the mouse behaves like this:

- Left Button      Click and drag to rotate the display. It acts as if there were an invisible sphere surrounding the image, and the mouse movements rotate that sphere in any direction. This is quite different from the Rotx and Roty wheels, and takes some getting used to.
- Middle Button   Click and drag to pan the display – this moves the visible objects around without rotation.
- Right Button     Brings up a menu with various functions, Draw Styles, and other items. The most useful is DrawStyles/StillDrawStyle/Wireframe, which will draw in the wireframe mode rather than as solid surfaces. This permits you to look inside objects, and is considerably faster drawing.

The Etc menu has several useful items:

Set Solid	Set the G4 mode to Solid (default)
Set (G4) reduced wire frame	Set the G4 mode to wireframe, real edges only
Set (G4) full wire frame	Set the G4 mode to wireframe,

	including triangles to fill surfaces
Visible mothers + invisible daughters	Daughter volumes are invisible (default)
Visible mothers + visible daughters	Make all objects visible

The key to understanding how objects are displayed is to know that Geant4 has a solid/wireframe choice, and OpenInventor has a solid/wireframe choice, and they are different:

<b>G4 mode (Etc menu)</b>	<b>OI mode (Right click menu)</b>	<b>Description</b>
Solid	Solid (“as is”)	Draw solid surfaces
Solid	Wireframe	Draw wireframes with triangles filling visible surfaces
Reduced Wireframe	Either	Draw reduced wireframes, showing only real edges
Full Wireframe	Either	Draw wireframes with triangles filling visible surfaces

For instance, in G4 reduced wireframe mode a sphere is invisible, as it has no real edges; in G4 full wireframe mode it has triangles filling its visible surface (in such a way you can tell it is a sphere). In reduced wireframe mode, a cylinder is drawn as two circles.

Note that by default all daughter volumes are visible. You can change that in the Etc. menu.

The OpenInventor driver has many more features, but this should get you started. It will also do animation: with the hand cursor left click and drag the image, letting go of the button while moving – the image will continue to rotate until you left click on it (works best locally).

### 2.8.3 A Trick for Visually Scanning Events

The simplest way to scan events visually is to use the *g4blgui* program, but it can be done manually via the command line. The OpenInventor driver (OIX) has the ability to exit the viewer and return to the Geant4 command input. This can be used to quickly scan events visually. To do this, first create a text file called *beamon.txt* containing a large number of identical lines:

```
/run/beamOn 1
/run/beamOn 1
/run/beamOn 1
... many more identical lines
```

Then invoke *g4beamline* like this:

```
G4bl input.file viewer=best <beamon.txt
```

Once the viewer appears, it will display the first event. To see the next event, select the *File/Escape* menu item. This will suspend the OpenInventor viewer, and the Geant4 input routine will read the next line of *beamon.txt* – that causes it to track one event and refresh the viewer with it. Just keep selecting *File/Escape* to sequence through events one at a time. This is essentially what the *g4blgui* program does when a viewer is selected.

## 2.8.4 Troubleshooting

Visualization is usually the only part of G4beamline that causes problems for users getting it to work on their systems. In general, on Linux and Mac OS X a working X-windows display is required; this means on Linux you usually will run one of the common desktops (KDE or Gnome), and use a terminal window to execute the G4beamline program. On Mac OS X you may need to run the X11 application, and either run *g4bl* from an *xterm* or specify the *DISPLAY* manually (since Leopard the X11 application gets run automatically when a connection to the X-Windows display is made).

G4beamline visualization requires that both the Motif and the OpenGL extensions be configured on the display. This means that both the X-windows client (the G4beamline program) and the X-windows display server (the display+keyboard+mouse) need them installed. If you run G4beamline on your desktop Linux system, this usually happens naturally, but if you connect from your desktop to a Linux server and run G4beamline there, then you must make sure these extensions are installed on both systems. In particular, if you use the Exceed program on a Windows desktop to connect to a Linux server, be sure to install Exceed's OpenGL extension. This also applies to Mac users (G4beamline is an X-windows application, not a native Mac application).

On Windows, all necessary graphics extensions should be installed by default. Remote displays (e.g. via Citrix) should work as usual.

If you run *g4bl* with a viewer parameter, and it crashes with the error message “Xlib: extension GLX missing on display :0.0.” (or similar), this means you must install the OpenGL extension (*glx*) on the system you are using to display. That message may well be followed by a lengthy trace of function calls. If, instead, it crashes with the error message “error while loading shared libraries: libGL.so.1: cannot open shared object file: No such file or directory” (or similar), this means you must install some necessary system library on the system you are using to run *g4beamline* – see *README-\*.txt* for details.

G4beamline X-windows visualization will operate over a network, but its performance will be slower than running locally, even over a 100 Mbit/s LAN. The easiest way to do this is to use ssh with X-windows forwarding enabled (the *-X* switch, which is often supplied by default when the ssh client is run from a terminal window). In some cases (particularly on a Macintosh) the *-Y* switch may be needed instead.

## 2.9 Obtaining Results – Virtual Detectors and other NTuples

While Geant4 can be used to make quite detailed models of detectors and their readouts, at present G4beamline does not attempt to do so. The primary purpose of G4beamline is to track particles and see where they go (or don't go). The most common way to obtain output is by using a command that generates an NTuple.

### 2.9.1 Track NTuples

A Track NTuple is simply a set of tuples generated for each particle that meets the requirement for generation; the columns are fields with names, and the rows are the individual track hits. The list of fields is given below.

The following commands generate an NTuple for tracks:

- A *virtualdetector* is an idealized perfect detector that “detects” every track that hits it, and measures all of the track properties with the resolution of a 32-bit float (including position, 3-momentum, particle type, event #, etc.). It is “virtual” because any material (including Vacuum) can be used and by default it uses the material of its parent so it doesn’t affect the tracking of particles. Multiple tracks from a given event that hit the *virtualdetector* will be stored as separate rows in the NTuple (with the same EventID), as will multiple hits from a single track (with the same EventID and TrackID). The measurements for all such tracks are output to a file as an NTuple.
- A *zntuple* is similar to a *virtualdetector*, except that no physical volume is used; whenever a track reaches the specified Z position (centerline coordinates), it is sampled in the same manner as a *virtualdetector*. All *zntuple*-s are named “Z100” with the number being its Z position rounded to the nearest millimeter. Note that the *zntuple* can span multiple physical volumes, while the *virtualdetector* cannot.
- A *timentuple* samples the tracks at a specified time.
- A *newparticlentuple* samples tracks when they are created.
- A *beamlossntuple* samples tracks as they are killed.
- An *ntuple* is a collection of the data for one or more other NTuple-s, with each row of the NTuple being a single event or track, and the columns being the union of all included NTuple-s, in order. Any NTuple generated by any of the above commands, including *ntuple* itself, can be included in this NTuple, as long as the other commands precede this one in the *input.file*.
- A *trace* is an NTuple generated for a single track. The rows of the NTuple are the series of steps taken by tracking, and the columns are the same as for a *virtualdetector* plus the electromagnetic fields (B and E). Each track is a single NTuple (a separate file in ASCII formats).

Normally each *virtualdetector* generates an NTuple for its hits (singles), as does *zntuple*. The *ntuple* command can be used to combine multiple *virtualdetector* and *zntuple* data into a single NTuple, with each row of the NTuple containing all data for a single event (if multiple tracks of the event hit a given *virtualdetector* or *zntuple*, only the first is kept in the NTuple).

By default, all NTuples are written to a single Root [5] output file. A Root-based histogramming program, *HistoRoot*, is available from <http://historoot.muonsinc.com>, which can generate histograms in a very flexible and intuitive manner from Root files and ASCII files, via a graphical user interface. In the future, G4beamline may be extended to use other formats for NTuples as well.

The *format=ascii* argument to these commands writes an ASCII file for each NTuple containing one track per line, with comments at the start giving the variable names and their units. The *historoot* program can read these ASCII files, including the variable names from the initial comment. *Gnuplot* [10] can also be used to generate plots from these ASCII files.

The standard fields of an NTuple generated for tracks contains the following variables (in order):

Name	Description
x, y, z	The position of the track, in the selected coordinates. Millimeters.
Px, Py, Pz	The momentum of the track, in the selected coordinates. MeV/c.
t	The global time of the track. Nanoseconds.
PDGid	The ID of the particle, using the assigned value from the Particle

	Data Group.
EventID	The event number.
TrackID	The track identifier.
ParentID	The track identifier for this track's parent; 0 if this is a beam track.
Weight	The weight of the track (defaults to 1.0).

TrackIDs are supposed to be unique within each event; they normally start at 1 for the beam particle(s). Secondary particles' TrackIDs normally start at 1000 and increment sequentially for additional secondaries.

The *trace* command adds the following fields to the NTuple, after the standard fields:

Name	Description
Bx, By, Bz	The magnetic field at the track's position, in the selected coordinates. Tesla.
Ex, Ey, Ez	The electric field at the track's position, in the selected coordinates. Megavolts/meter.

The *format=Extended*, *format=asciiExtended*, or *format=rootExtended* argument to these commands adds the following fields to the NTuple, after the standard fields and the B and E fields from the trace format:

Name	Description
ProperTime	The proper time of the track since it was created. Nanoseconds.
PathLength	The total path length of the track since it was created. Millimeters.
PolX, PolY, PolZ	The polarization of the track in the selected coordinates.
InitialKE	The Kinetic Energy of the track when it was created. MeV.

## 2.9.2 printf

The *printf* command generates a printout in a user-specified format for every track that reaches its specified Z position; the standard fields above can be used in expressions to be printed via conversion specifications in the printf format string. Output can be to stdout or to a file, and multiple *printf*-s to the same file will be output to the file in the order tracks reach their Z positions, and in the order the *printf*-s appear in the input file for the same Z position. This permits ASCII output of track information in any desired ASCII format, perhaps to feed directly into some other program.

## 2.9.3 profile

The *profile* command prints a computation of the means, sigmas, and emittances for all tracks at a given Z position.

## 2.9.4 totalenergy

The *totalenergy* command totals up the energy deposited by all tracks of a run in selected physical volumes, and prints the results at the end of the run. Output defaults to *stdout*, but can be put into a file. With appropriate post-processing this can be used to estimate heat loads in specified volumes.

## 2.10 Random Number Generator

G4beamline uses the default Geant4 random number generator (which is HepJamesRandom from CLHEP [2] – see the CLHEP documentation for details). It is an excellent pseudo-random number generator that essentially guarantees no repeat sequences when seeded by any integer between 0 and 900 million. At the start of each event, G4beamline seeds the random number generator with the event number. This permits the user to submit multiple jobs in parallel with confidence, as long as the *beam* command arguments are arranged so no two jobs run events with the same event numbers – quite effective when using a Linux cluster. This also permits the user to re-run the same events (as long as the input file remains unchanged – the slightest change can cause tracking to use a different number of random numbers, and thus the runs of the same event number won’t be the same). A good use of this last capability is to find an “outlier” event in some histogram, determine its event number (via narrow cuts in *historoot*) and re-run it with visualization to see what actually happened.

Note that due to their internal designs, Root NTuples cannot precisely represent event numbers larger than 16 million (a float with 24 bits of mantissa). This does not affect the ability to handle larger datasets, it just means that the event numbers will be rounded to a 24-bit mantissa. ASCII formats for track NTuples do not suffer from this limitation.

## 2.11 Tuning the Beamline

Just like a real beamline, a simulated beamline must be tuned to maximize transmission and achieve various desired characteristics. G4beamline is not intended for the design of beamlines, but rather is best used for the evaluation of them. Conventional beam optics design programs can be used to determine the beamline elements and their parameters. These programs, however, usually make approximations and ignore many subtle effects that G4beamline implements (e.g. fringe fields, non-Gaussian multiple scattering tails, energy loss straggling, hadronic interactions in collimators and other apertures, non-Gaussian tails in beam profiles, etc.). The result is that just as in the real world the design programs give only approximate values.

So two methods of tweaking the tune are provided:

1. The *reference* command can tune its referenceMomentum in order to make the reference particle have a specified tuneMomentum at a specified tuneZ position (centerline coordinates). This is most useful for a beamline with material that induces a significant energy loss in the reference particle, such as muon cooling channels.
2. The *tune* command can be used to tune any *tunable* argument to any set of elements. This includes the *By* field argument of *genericbend* and *idealsectorbend*, and the *maxGradient* argument of *pillbox*.

See the online help, or section 5, for details about these commands. Tests 29-32 verify the operation of these tunes, and provide working examples.

To aid the user in fine tuning the beamline, g4beamline implements the ability to track a “Tune” particle and a “Reference” particle before tracking the beam (both are controlled by the *reference* command). These particles are tracked with all stochastic processes disabled, and nominally go right down the centerline of the beam. The Tune particle is used to tune all desired beamline elements, and the tuning process often forces it to be tracked multiple time through sections of the beamline; after tuning is

completed then the Reference particle is tracked to give a clean readout of the reference trajectory, and computation of the reference coordinates (if used).

The Tune particle can be used to:

1. Automatically set the timing of RF cavities so the Reference particle arrives at the desired RF phase.
2. Automatically set the gradient of RF cavities so the desired acceleration of the reference particle is achieved.
3. Automatically set the field of bending magnets so the Reference particle is parallel to the centerline downstream of the magnet.
4. Automatically determine the initial momentum of the Reference particle so that it has a specified momentum at some later point in the beamline.

Unfortunately, this method cannot be used to tune the position of bending magnets (because of their fringe fields, the center of a bending magnet must normally be offset by a small distance downstream of the corner in order to make the Reference particle go right down the centerline after the magnet).

As an example, consider tuning the *By* value of a *genericbend*:

```
reference referenceMomentum=200 particle=mu+ beamZ=0.0
tune B1 z0=100 z1=3000 initial=-0.6500 step=0.01 expr=Px1/Pz1 \
    tolerance=0.000001
genericbend B fieldWidth=500 fieldHeight=500 fieldLength=500 \
    ironWidth=800 ironHeight=800 ironLength=500 \
    fieldMaterial=Vacuum
place B z=2000 rename=B1 rotation=Y15 By=B1
corner C z=2000 rotation=Y30
```

The *tune* command specifies to tune the value “B1” so that at  $Z=3000$  the expression “ $P_{x1}/P_{z1}$ ” will be zero to within the tolerance of 0.000001. That is, that the Reference particle be parallel to the centerline within a microradian. The *genericbend* named “B1” uses the tune variable “B1” as its *By* argument value. What happens is that when the Tune particle reaches  $z0=100$ , it is saved and tracked to  $z1=3000$  (which goes through the *genericbend* B1). When the Tune particle reaches  $z1=3000$ , the *expr* “ $P_{x1}/P_{z1}$ ” is evaluated, and if it is within *tolerance* of zero the tuning is complete. If it is out of tolerance, a simple linear solver is used to update the value of the variable *B1*, and the saved track from  $z0=100$  is re-tracked to  $z1=3000$  and the process is repeated. If the process does not converge with 10 attempts, the tuning fails and the program exits; information about each step is written to stdout.

The other tune procedures are similar, in that the Tune particle track is saved upstream of the region to be tuned, and downstream of the region its properties are used to modify a variable that affects the tuning, and the saved track is re-tracked until the result is within tolerance.

In practice, it is often necessary to use multiple tune commands. They must be properly nested so each tuning region is either disjoint from the others, or is wholly contained in one or more other regions (separation of  $\sim 0.01$  mm is usually adequate). For instance, in a muon-cooling channel with bending magnets and RF cavities it is probably necessary to tune the reference momentum, the bending magnets, and the RF gradient(s).

Note that g4beamline can only tune based on the Tune particle, and can only tune *tunable* arguments to certain commands. These limitations are due to technical details of the implementation, and cannot

easily be relaxed. No aspect of the geometry can be tuned via the Tune particle. Note this also does not include tuning quadrupoles to achieve a specified focus, because that would require tracking many beam particles and tuning to achieve a specified distribution. Because g4beamline can be easily scripted, however, it is possible to do this manually via scripts or an external program. Such an external method can tune any parameter or argument to any element, including geometry. See sections 7.18 and 7.21 for details.

Examples: The program *triplet.sh* in the **examples** directory focuses a quad triplet for a point-to-point focus. In the **test** directory, test29, test30, test31, test32, and test35 verify the tuning of each of the above quantities.

## 2.12 The Geant4 User Interface

Geant4 has its own command line processing and user interface. For the most part this is not used in G4beamline. Some Geant4 commands can be simply placed in the *input.file* because all input lines beginning with “/” are interpreted as Geant4 commands; they are executed immediately upon being encountered in the *input.file*. In most cases, that is not what is needed, because the commands must be deferred until the geometry has been set up – see the *g4ui* command for that.

The *viewer.def* file contains Geant4 commands used to initialize and use the various visualization drivers. The selected section is automatically invoked when visualization is specified via the *viewer=...* argument to the command line.

## 2.13 Event Time Limit

Every event is subject to a per-event time limit. This is controlled by the parameter *eventTimeLimit*, which defaults to 30 (seconds), but can be set by the *param* command. As long as the program is taking physics steps, this time limit will cause the current event to be killed and the next event to be simulated (restarting the time limit). In case the code gets into an infinite loop, an alarm is set at the start of each event for *eventTimeLimit*+10, and if this fires, the entire program will be aborted (recovery is not possible); this prevents it from exhausting your CPU allocation on a cluster.

Note that systems with stable particles and electromagnetic traps can run literally forever. You may not think your system has this property, but low energy delta rays ( $e^-$ ) can surprise you. Note also that putting a medium- or high-energy electron or photon into a block of material can generate many thousands of secondaries, which can take a very long time to track.

A different type of time limit is implemented via the *maxTime* argument to the *trackcuts* command; it is a limit on the global time in the simulated world.

## 2.14 Wall Clock Limit

An additional processing limit is determined by the parameter *wallClockLimit*. It defaults to -1, indicating an infinite time limit. Setting it to a positive value will limit the run time to that many seconds. This is most useful on a supercomputer where jobs are limited by wall clock time. Note that it can take several seconds to several minutes to close up, so the value of the G4beamline parameter should be suitably

smaller than the limit of the job. This limit is checked only between events, which is another reason it should be smaller than the job limit.

## 2.15 Event and Track Numbering (EventID, TrackID)

EventID is intended to identify a particular event (which may have multiple tracks). In G4beamline, each beam track is a separate event. For internally generated beams, EventID is incremented for each beam track and is thus unique within a given run. For beams read from external files, the assignment of EventID-s to tracks is contained in the file – this means that multiple events can have the same EventID, and that EventID-s might not be monotonically increasing.

Moreover, NTuple-s use float-s to hold each data item, and a float has only 24 bits of mantissa. This implies that any event number  $\geq 16,777,216$  will be truncated. This applies to both reading and writing Root files. Two workarounds avoid this truncation for many cases:

- BLTrackFile format (ASCII) avoids the truncation internally, both reading and writing.
- Commands that generate a track NTuple will map `format=ascii` to `format=bltrackfile`. This includes: *beamlossntuple*, *newparticlentuple*, *timentuple*, *virtualdetector*, *zntuple*.

The result is that this truncation of EventID occurs only for Root NTuples and in the *ntuple* command; neither of these can be avoided without massive changes that adversely affect users.

TrackID is intended to identify a track within an event. For internally generated beams the primary track has TrackID=1, but for beams read from external files the assignment of TrackID to tracks is contained in the file – this means that multiple tracks with identical EventID-s can have the same TrackID-s.

Moreover, secondary tracks can be generated during the simulation of an event, and the TrackID-s for them can potentially conflict with TrackID-s read from a file. By default, secondary TrackID-s are assigned starting at 1001 (see *secondaryTrackID* of the *beam* command). If a beam track from an external file has a TrackID  $\geq$  *secondaryTrackID*, a warning will be printed. For beam files containing TrackID-s  $\geq 1001$ , setting *secondaryTrackID* in the *beam* command can avoid both the potential confusion and the many warnings.

## 2.16 Signals

G4beamline responds to signal *SIGUSR1* by flushing all NTuples to disk. So for a long job you can look at intermediate results.

G4beamline responds to signal *SIGUSR2* by closing up and exiting cleanly. In each worker this is detected between events. This is especially useful on Hopper, as *SIGUSR1* and *SIGUSR2* are the only signals users can send to the application; all others cause immediate termination without saving results.

## 3 Important Values that Affect the Validity and Accuracy of Simulations

Particle simulations are inherently complicated and CPU intensive. The Geant4 toolkit necessarily provides facilities to permit users to make trade-offs between accuracy and CPU time. G4beamline likewise considers this a user issue, and provides interfaces to Geant4 mechanisms as well as some new capabilities. To perform simulations that yield results you can trust, you must make some choices that affect the accuracy and validity of the results. It is often equally important to avoid becoming overwhelmed by useless information. Features related to these are described in this section.

### 3.1 Physics List

The Geant4 toolkit contains a large number of physics processes that implement the transportation, decays, and interactions of particles. In addition, Geant4 is extensible, and users can construct their own physics processes. These processes are numerous and complex, and often have complex relationships between different processes, so it is a challenge for users to become sufficiently knowledgeable to select the necessary processes and implement them in code. As a result, the Geant4 collaboration has constructed a number of “physics lists” that are crafted from the library of physics processes to be suitable for a specific user domain.

G4beamline shields the user from much of this complexity, and only offers to the user the choice among the Geant4 physics lists, plus a few additional lists constructed for special purposes. In particular, it is not possible for users to implement a new physics process in G4beamline without building the program yourself (that is rather complicated, but you can request a new feature to have us do it – see section 7.3). The physics list is selected by the *physics* command, and the set of available lists is given in the help text for that command (see section 7.4).

As a rule of thumb, for incident particles with kinetic energies above 12 GeV or so, the “standard” physics list is QGSP. For incident particles with kinetic energies above about 100 MeV, the “standard” physics list is QGSP\_BERT (it is noticeably slower than QGSP below ~12 GeV). The electromagnetic processes of these physics lists are advertised to be valid above a few hundred electron Volts; Geant4 has a set of low energy electromagnetic processes, indicated by a suffix “\_EMX” in the physics list name. The suffix “\_EMV” indicates electromagnetic processes tuned for better CPU performance with only slightly less precision.

### 3.2 Tracking Accuracy Parameters

There are a number of values in Geant4 that affect the accuracy of its tracking. The important ones are implemented in G4beamline as parameters, and they can be set anywhere in your input.file with the *param* command (the final value is what matters). These control such things as how close to a geometric boundary a track must come before it is considered to have hit the volume, how far a track can deviate from a straight line during a single step (due to a magnetic field), etc. They are:

Parameter Name	Default	Description
deltaChord	3 mm	See the Geant4 Physics manual and User's Guide for a description and discussion.
deltaIntersection	0.1 mm	
deltaOneStep	0.01 mm	
epsMax	0.05	
epsMin	2.5E-7	
maxStep	100 mm	The maximum physics step permitted to be taken. Note this is a limit on the physics step; integrating the equations of motion will take multiple "steps" within this. Can be overridden in each element.
minStep	0.01 mm	The minimum physics step permitted to be taken.
zTolerance	2 mm	How close to a specified Z position a step must occur (the two steps surrounding the Z position are then linearly interpolated).

These parameters specify the trade-off between tracking accuracy and CPU time. These are reasonable values for beamlines or detectors of ordinary sizes (with scales from roughly a few millimeters to a few tens of meters; the length of a narrow beamline can reasonably extend to a few kilometers). If you have objects smaller than a millimeter or so, or are working on scales larger than a kilometer or so, you should consider changing these values appropriately. If you have small objects spread over a large volume, in general you should select parameter values appropriate for the small objects and realize that the simulation will most likely require lots of CPU time.

Note that Geant4 tracking has two different meanings of the word 'step':

- |                  |  |
|------------------|--|
| Physics step     | Defaults to a maximum of 100 mm, and determines the maximum step for applying physics processes. At each step it is set to the smallest value of: <ul style="list-style-type: none"> <li>○ the current value of maxStep</li> <li>○ the distance to the next geometrical boundary</li> <li>○ the smallest value determined by any active physics process</li> </ul> |
| Integration step | Determined automatically from the other tracking parameters (primarily deltaChord), and determines the accuracy of the integration of the equations of motion. In a magnetic field this is of course a helix. There can be many integration steps within a single physics step, and usually are in a magnetic field.   |

In particular, the accuracy of the tracking is usually not significantly improved for small physics step sizes. For instance, tracking 250 MeV/c muons through 1 meter of liquid hydrogen in a 2 Tesla solenoid (including multiple scattering and energy loss) shows no significant difference between maxStep=1 and maxStep=1000 in position or momentum histograms<sup>2</sup> – the execution time differs by a factor of about 50, so don't set maxStep to a small value unless you have a specific reason to do so.

---

<sup>2</sup> This comparison can only be done statistically, not on an individual track basis, because different values of maxStep inherently use different numbers of random numbers, causing variations in stochastic processes like multiple scattering and ionization energy loss.

### 3.3 Electromagnetic Field Map Tolerance

Several elements that implement electromagnetic fields internally construct a field map, because evaluating the field during tracking is too CPU intensive. These are: *coil* and *fieldexpr*. They each have a *tolerance* argument that is used to determine the required grid spacing of the map. Its value obviously affects the accuracy of the simulation. This is of course also true for externally supplied EM field maps.

### 3.4 Secondary Creation Threshold in Physics Processes

An important trade-off between physics accuracy and CPU time in Geant4 is the production of low-energy delta rays ( $e^-$ ) and other particles, which diverge at low energies. The approach is to only generate a secondary when the range of the particle in the current material exceeds a user-specified minimum range cut (which is converted to a kinetic-energy threshold for each material). Particles with shorter ranges are not produced as secondaries, but are lumped into the continuous ionization energy loss in the material. In G4beamline, this is set by the *minRangeCut* argument to the *physics* command; its default value is 1 mm. If your simulation has objects smaller than that, or you are working at scales much larger than that, you should consider changing its value, unless you just don't care about these low-energy particles.

### 3.5 Track Cuts

The *trackcuts* command applies cuts to every track both before starting to track it, and at every step while tracking. These can obviously affect the realism and accuracy of the simulation; they also can be quite useful for weeding out extraneous information (e.g. the neutrinos generated by pion and muon decay). The cuts it can apply are:

Argument Name	Default	Description
kill	(empty)	Comma-separated list of particles to kill.
keep	(empty)	Comma-separated list of particles to keep (kill all others); ignored if empty.
killSecondaries	0	Set nonzero to kill all secondaries.
kineticEnergyCut	0 MeV	Minimum kinetic energy to track.
kineticEnergyMax	Infinite MeV	Maximum kinetic energy to track.
maxTime	1000000 ns	Maximum lab time to track.

In addition, there is a processing time limit implemented as a parameter, *eventTimeLimit* (defaults to 30 seconds). This applies to the real time used during the processing of all tracks in the event. For simulations in which showers are important (and are not suppressed), this may need to be increased. Unfortunately, it is implemented as a real time limit, not a CPU time limit, so on excessively loaded systems it may also need to be increased (not usually a problem).

### 3.6 Stepping and Hard Edges

If your simulation has hard-edge fields, then beware of the interaction between the stepping during tracking and these hard edges. In particular, tiny changes in track parameters can have much larger

effects on the tracking, because one more or fewer steps occurred in the field region. This can also occur for the edges of physical components, as one more or fewer steps occurs inside a material.

Usually this is not a problem, as the effects normally are both small and inherent in the monte-carlo method. This can be bothersome when making a detailed comparison with some other code or an analytic result.

### 3.7 Miscellaneous

- There is a bug in Geant4 tracking that makes gross errors when tracking a particle that stops and turns around (e.g. it is traveling exactly opposite to an E field sufficient to stop it). This has been reported as Geant4 bug #1021, and the command *bug1021* implements a work-around that resolves the issue with an accuracy of a few microns.
- A number of Geant4 processes have auxiliary commands that can vary their operation. See the Geant4 documentation for details; in most cases, these are not important, but when they are, the *g4ui* command can be used to execute the necessary Geant4 commands.
- Geant4 tracking requires that the simulated world's geometry be a rigorous hierarchical structure; violations of this requirement can cause gross tracking errors. By default, G4beamline performs a geometry test to check for invalid volume overlaps and conformance to this requirement; failures generate a warning but permit the simulation to proceed. The *geometry* command controls its parameters. Section 8.2 discusses when these rules can be violated without affecting the simulation results.
- The pseudo random number generator is normally seeded with the event number immediately before starting to process an event. The generator used has excellent properties when seeded with integers from 0 thru 900,000,000. This behavior can be changed via the *randomseed* command.
- As in real beamlines, shielding is required in realistic simulations. See section 7.9.
- Individual physics processes can be disabled via the *disable* and *doStochastics* arguments to the *physics* command. A list of the physics processes used in the current physics list is available from the *list* command.

## 4 Input File Description

A G4beamline simulation is completely specified in a single ASCII input file, plus whatever auxiliary files it references (e.g. magnetic field maps, window profiles, etc.).

Each line in the input file is either a comment or a command. Comments are blank lines, lines beginning with '#', and lines beginning with '\*' (which are printed to stdout, giving a convenient method to document your output files). Lines beginning with '/' are commands passed directly to the Geant4 user interface command interpreter (this happens before the geometry is initialized; see the *g4ui* command to defer execution of Geant4 UI commands). Lines beginning with '!' are passed directly to the shell (with the '!' stripped). On any line, '#' introduces a comment-to-end-of-line if it is preceded by a space or tab and is not enclosed in quotes.

A command is a single line beginning with the command name (optionally preceded by white space), followed by any number of arguments. Lines can be continued by preceding the ending newline with a backslash (\). Commands can have any combination of positional and named arguments, which are separated by white space (spaces or tabs). A named argument is of the form "name=value", where the name is like a C identifier except it can also contain the characters '+' and '-' (which are needed for particle names). If spaces or tabs are present in the value, the entire value must be enclosed in single- or double-quotes; this delimiter cannot appear in the value. Any argument that is not named is positional, and its value can also be enclosed in single- or double-quotes; positional arguments are conventionally given first, but can be intermixed with named arguments. The order of positional arguments is important, but the order of named arguments is not.

This command has 2 named arguments:

```
param histoFile=histoscope.hst histoInterval=100000
```

This command has 1 positional and 4 named arguments:

```
tubs Target outerRadius=10 length=100 material=W color=1,1,1
```

This command has 3 positional arguments (the last 2 are each themselves a command that will be invoked when the defined command *MyMacro* is used):

```
define MyMacro \  
    "tubs $1 outerRadius=10 length=100 material=$2 color=$3" \  
    "place $1"
```

That macro is used like this:

```
MyMacro TubsName Fe 1,0,1
```

Command files can be nested; see the *include* command.

Simple macros can be defined, with arguments; see the *define* command.

The value of a named or positional argument can come from a parameter by using a '\$':

```
param H=10.0 W=20.0 L=30.0  
box BoxName width=$W height=$H length=$L
```

Note that *\$parameter* substitution occurs only within the value of a parameter; this can be combined with arithmetic expressions for numeric arguments:

```
box Box.$H.$W.$L width=$W+10 height=$H*2.0 length=sqrt($L)
```

The *param* command is unique in that when the value of any argument is a valid numerical expression containing at least one operator, the expression will be evaluated and the parameter will be defined as the number (rather than the string of the expression). This permits the computation of rotations (the value of the *rotation* argument is a string containing multiple numerical values):

```
param angle=atan2($a,$b)*180/3.14159
place Element rotation=X$angle,Y90
```

## 4.1 Expressions

Expressions can be used for the value of any integer or double argument to any command. They are evaluated during command processing, and generally must involve only double values and the operators and functions listed below. Parameter expansion (*\$paramname*) can be used as long as the string value of *paramname* is valid where it appears in the expression. Some commands expand the expressions to include certain variables (e.g. the *Bx* argument to *fieldexpr* can be an expression involving *x,y,z*).

In the *param* command, if the value of an argument is a valid numerical expression, the parameter is set to the numerical value rather than to the expression string.

All values are doubles; for integer parameters the conversion occurs after the expression is evaluated.

The following arithmetic operators can be used in expressions: + - \* / ^ ( ). Note that ^ means exponentiation to an integer power (use *pow()* for a double exponent). Note that % is not implemented.

The following comparison operators can be used in expressions; they evaluate to a value of 1.0 (true) or 0.0 (false): < <= > >= == !=

The following logical operators behave as in C: || &&.

**DO NOT USE** the bitwise operators (& | ~), as these bitwise operations behave rather strangely for double values.

The *a?b:c* operator is not implemented but there is a similar *if(a,b,c)* function.

The following functions can be used in expressions: *abs()*, *min()*, *max()*, *sqrt()*, *pow()*, *sin()*, *cos()*, *tan()*, *asin()*, *acos()*, *atan()*, *atan2()*, *sinh()*, *cosh()*, *tanh()*, *exp()*, *log()*, *log10()*, *floor()*, *ceil()*, *if()*. All take 1 argument except *min()*, *max()*, *pow()*, and *atan2()* take 2 arguments and *if()* takes 3.

The following constants are pre-defined: *pi*, *e*, *gamma*, *radian* (=1.0), *rad* (=1.0), *degree* (=pi/180.), *deg* (=pi/180.).

## 4.2 Element Names

When you create an element (e.g. using the *tubs* command), you must specify its name as the first positional argument. When you place it into the simulated world, its name must be the first positional

argument to the *place* command (it can be renamed when placed). Element names can be any string, but conventionally obey the rules of a C identifier.

Element names are necessary to correlate the definition and placement of each element. They are also used to report errors in the geometry test, and are listed in the traces when stepping. The name of a *virtualdetector* becomes the name of its NTuple.

When elements are placed inside another element, the name of the parent is prepended to the name of the placed daughter, so the name shows its element hierarchy. This makes it desirable to capitalize the first letter of each name. The “World” volume-name is not prepended (otherwise it would apply to every element name). The *rename=* argument to the *place* command can be used to override the name of the element (the default is without the parent’s name, but *rename=+Xyz* will prepend the parent’s name to Xyz); ‘#’ can be used to automatically number multiple placements (see the *place* command in section 5 for details).

There are several independent name spaces used by various commands:

Type	Defined by	Used by
Element names	Element definition commands	<i>place</i> command
Virtualdetector names	Name of the virtualdetector, as placed	<i>ntuple</i> command
Material names	Material command (many materials are predefined).	Element definition commands, material=name argument
Coil names	<i>coil</i> command	<i>solenoid</i> command
Parameter names	<i>param</i> command	<i>param</i> command, \$name

### 4.3 Parameters

Parameters are named ASCII strings that can be used in the *input.file* to set command arguments, and some are used to control the program. They can be set either on the command line or by the *param* command. They are stored internally as ASCII strings, and when they represent an integer or floating-point value they are converted when used (invalid values are errors). When a parameter that has not been defined is used, it will be automatically defined from the environment if possible; otherwise using an undefined parameter generates an error (this is especially useful in Grid jobs, which may need \$OSG\_WN\_TMP in every filename).

A parameter can be used in the value of any argument to any command:

```
tubs Name length=$length outerRadius=$radius
```

In this example, it is necessary to define the parameters *length* and *radius* before this command is issued; that can be either earlier in the *input.file* or on the command line. Here the string definitions of both parameters *length* and *radius* must be valid floating-point numbers (or expressions):

```
# (earlier in the input.file)
param length=100.0 radius=10.0
```

Or on the command line:

```
g4bl input.file length=100.0 radius=10.0 [... ]
```

Note that once the parameter is defined, it can be used any number of times to give the values of command arguments. That can be used to avoid having to change many identical numerical values

throughout the *input.file*. It can also be used to collect all of the variable values of *input.file* in one place near the top, or even in a separate definition file (see the *include* command).

This is a limited macro facility, and can only be used to substitute the string value of a parameter into a command argument (named or positional). For integer and double (real) arguments, arithmetic expressions involving constants can be used (including common functions like `sin()` and `sqrt()`):

```
param a=3.0 b=2.0 name=Name
tubs $name length=$a+$b outerRadius=sqrt($b)*10
```

In the *param* command only, argument values that are a valid numeric expression with at least one operator are converted to the value of the expression. This permits such parameters to be used to perform computations for rotations (and colors, if desired):

```
param angle=atan2($a,$b)*180/3.14159
place Element rotation=X$angle,Y90
```

**NOTE:** *\$name* used in a macro body (the *define* command) is substituted when the macro is defined, not when the macro is expanded. *\$\$name* in a macro body will be expanded when the macro is invoked.

It is possible to re-define the value of a given parameter name within *input.file*; the value substituted for *\$name* is the most recent definition preceding its use. Because this can be confusing to humans, it is discouraged (except for the *do* and *for* commands). The *-unset* argument to *param* will set parameters only if they are not already defined; this is useful to put default values of parameters into *input.file* while permitting them to be overridden on the command line; it does not work for parameters used to control the program (they are pre-defined with default values).

### 4.3.1 Program Control Parameters

Several parameters are used internally by G4beamline to control its operation:

Name	Default	Description
viewer	"none"	Determines the mode of the program, visualization or tracking. Also determines the visualization driver to use. See section 2.8 above.
steppingVerbose	0	When nonzero causes each step to be printed to stdout.
steppingFormat	(see below)	The format of lines printed by steppingVerbose.
histoFile	g4beamline.root	The histogram output filename.
histoUpdate	0	When nonzero causes G4beamline to update the histoFile after each block of this many events have been tracked.
worldMaterial	Vacuum	The material of the World volume.
Zcl	-	Initially 0, the <i>place</i> command sets <i>Zcl</i> to the downstream <i>Z</i> position of the placed element, when using Centerline coordinates (not for Global coordinates or placements into elements other than the World). The user can set <i>Zcl</i> , but that has no effect on the <i>place</i> command.
eventTimeLimit	30	CPU Time limit on each event (sec)
fieldVoxels	200,200,200	Size of voxels for field computation (mm,mm,mm)
maxStep	100	Maximum physics step size, global. Each element can have its own value for maxStep.

wallClockLimit	-1	Limit on wall clock time for the entire job (seconds). <0 means infinite.
----------------	----	---

There are also parameters that control the values of Geant4 tracking parameters, discussed in section 3.2.

The steppingFormat parameter controls how the steppingVerbose printing is performed. It is a sequence of names of track attributes (separated by commas or spaces):

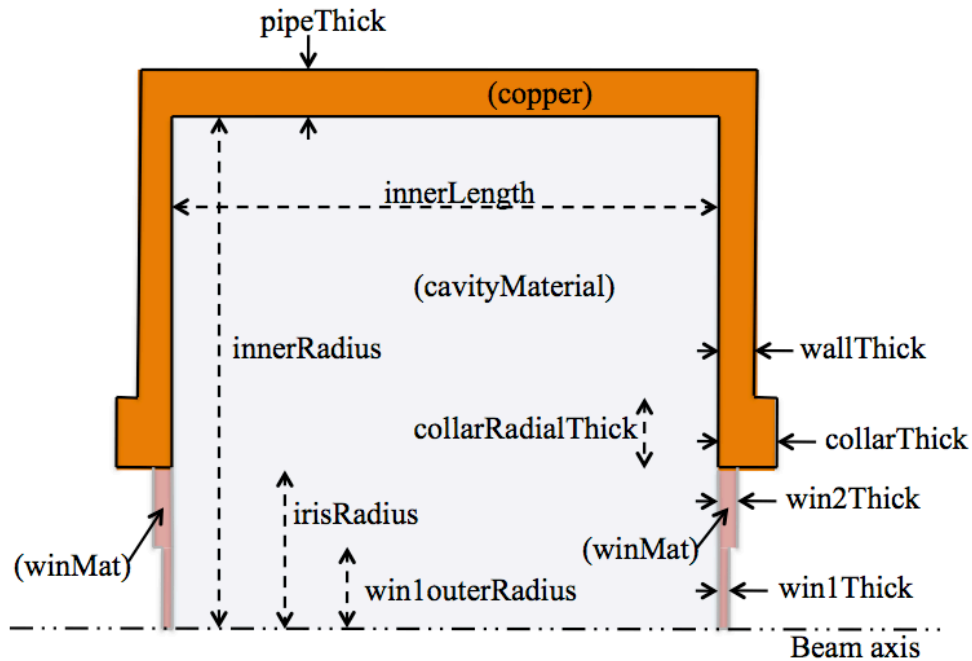
EXT	toggle extended precision (3 more digits)
TAG	print a '>' (useful to grep output)
N	step number
NSTEP	Synonym of N
GLOBAL	X,Y,Z,T in global coords
XYZT	Synonym of GLOBAL
CL	X,Y,Z,dxdz,dydz in CL coords
CLX	extended precision CL
KE	kinetic energy
STEP	step length
STEPLEN	Synonym of STEP
VOL	volume name
VOLNAME	Synonym of VOL
PROCESS	process name
B	magnetic field
E	electric field
P	3-momentum
MAT	material name
ID	event ID, track ID, parent ID
PART	particle name
SEG	centerline coord segment number
WT	weight
POLAR	polarization
NL	<newline>
NEWLINE	Synonym of NL
\n	Synonym of NL

The default format is “N GLOBAL CL KE STEP VOL PROCESS”. These names are insensitive to case.

See the *printf* command for another way to display track information, and the *trace* command for a third.

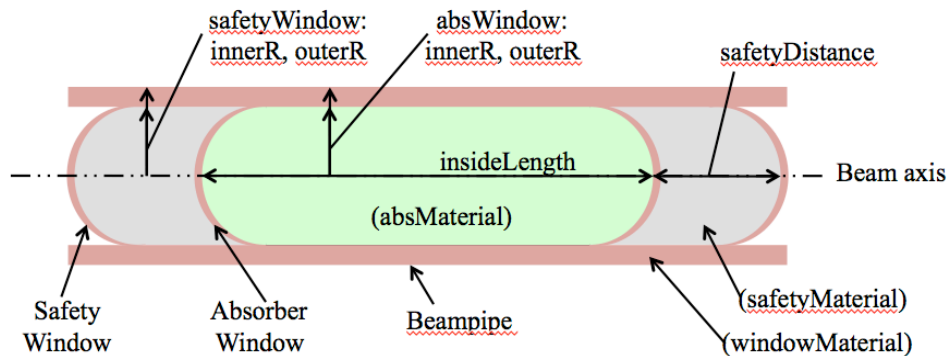
## 4.4 Pillbox Geometry and Dimensions

The geometrical dimensions of the pillbox are given in the figure below showing a radial cross-section. The pipe, walls, and collars are always made of copper; the stepped windows (win1 and win2) are made of winMat, and the interior of the cavity is cavityMaterial. The entire pillbox is cylindrically symmetric around the beam axis. The win1-s, the win2-s, the collars, the walls, and the pipe can each be omitted by setting the corresponding thickness to zero.



## 4.5 Absorber geometry and Dimensions

The absorber is cylindrically symmetric around the beam axis. The beampipe's radii come from the flange specifications in the files that define the profiles of the absWindow and the safetyWindow; the beampipe's length is extended to the same length as the safetyWindow-s on the beam axis. The safetyDistance is between the inner faces of the two windows on the beam axis. If safetyWindow is omitted, all references to it are changed to absWindow, and there are no volumes of safetyMaterial.



## 4.6 G4beamline Commands by Type

This section is just the output of the “help” command, with a little re-formatting.

The help command:

```
help           provides interactive help.
```

man            Alias for 'help'.

Program control commands:

define	defines a macro (argument-expanded set of commands).
do	Do loop.
endgroup	ends a group definition.
exit	exit a command file.
for	For loop.
g4ui	Accesses the Geant4 user interface
geometry	Arranges to perform a geometry test.
group	begins definition of a group.
if	Conditional execution of command(s), and if/elseif/else/endif.
include	includes a command file.
list	provides interactive list of interesting internal tables.
output	redirects stdout and stderr to a file
param	Defines parameter values.
randomseed	control pseudo random number generator seeds
showmaterial	Set the colors for selected materials.
trackermode	Sets mode for all trackers, manages track fitting.
tune	Tune a variable used as argument to other elements.

Centerline layout commands:

corner	Implement a corner in the centerline.
cornerarc	Implement a cornerarc in the centerline.
start	Define the initial start of centerline coordinates.

Beam definition commands:

beam	Define the Beam.
cosmicraybeam	Define a Cosmic-Ray muon 'beam'.
particlesource	Interface to the Geant4 General Particle Source.
reference	Define a reference particle.

Auxiliary definition commands:

material	construct a new material.
particlecolor	Set the colors for particle tracks.
trackcolor	Alias for 'particlecolor'.

Beamline element definition commands:

absorber	construct an absorber
box	construct a box.
coil	defines a coil (part of a solenoid)
cylinder	Alias for 'tubs'.
extrusion	construct a solid extrusion with axis along z
fieldexpr	implements a field map, E and/or B, from expressions.
fieldmap	implements a field map, E and/or B, from a file.
genericbend	construct a generic bending magnet.
genericquad	construct a generic quadrupole magnet.
helicaldipole	construct a helicaldipole magnet.
helicalharmonic	construct a helicalharmonic magnet.
idealsectorbend	construct an ideal sector bending magnet.
lilens	construct a simple Lithium lens.
multipole	construct a generic multipole magnet.
particlefilter	Will kill particles from a list, or force particles to

decay.

pillbox	Defines a pillbox RF cavity
---------	-----------------------------

polycone	construct a polycone with axis along z
rfdevice	Defines an rfdevice (RF cavity)
solenoid	defines a solenoid (a coil and current)
sphere	construct a sphere (or section of one)
torus	construct a torus.
trap	construct a solid trapezoid with axis along z.
tube	Alias for 'tubs'.
tubs	construct a tube or cylinder with axis along z.

The place command:

place	places an element into the current group (or world).
-------	--

Track and Event cuts:

eventcuts	Implements cuts on event number via lists in files.
trackcuts	Specifies per-track cuts.

Data output commands:

beamlosstuple	NTuple containing particle tracks when lost.
fieldntuple	Generates an NTuple from B and E fields at specified points.
newparticlentuple	NTuple containing particle tracks when created.
ntuple	Define an NTuple containing multiple detectors.
printf	prints track variables and expressions
printfield	Prints E or B fields, or writes FieldMap file.
probefield	Prints B and E fields at specified points.
profile	write beam profile information to a file
timentuple	Construct an NTuple of tracks at a specified time.
totalenergy	Print total energy deposited in selected volumes.
trace	Specifies tracing of tracks.
tracker	Defines a tracker.
trackerplane	Construct a tracker plane.
usertrackfilter	Construct a usertrackfilter that filters tracks via user code.
virtualdetector	Construct a VirtualDetector that generates an NTuple.
zntuple	Generate an NTuple for each of a list of Z positions.

Physics commands:

bug1021	Workaround to improve accuracy of bug1021 in E field
emfactor	Multiply multiple scattering and energy loss by factors.
muminuscapturefix	Fixes up the G4MuonMinusCaptureAtRest process.
physics	Defines the physics processes and controls them.
reweightprocess	modify the cross-section of a physics process.
setdecay	Set lifetime, decay channels, and branching ratios for a

particle's decay.

spacecharge	Beam-frame Green's function space charge computation
spacechargelw	Lienard-Wiechert space charge computation

Other commands:

collective	Monitor collective computation
demo	demo command.
fieldlines	Display magnetic field lines.
movie	Generate movie NTuple.
test	test random number seeds.

Program control Parameters:

Zcl	Last centerline Z position used (updated continuously)
-----	--

deltaChord	Geant4 tracking parameter
deltaIntersection	Geant4 tracking parameter
deltaOneStep	Geant4 tracking parameter
epsMax	Geant4 tracking parameter
epsMin	Geant4 tracking parameter
eventTimeLimit	CPU Time Limit (sec)
fieldVoxels	Size of voxels for field computation (mm)
histoFile	Default (Root) NTuple output filename
histoUpdate	Output update interval (events)
maxStep	Maximum physics step size (mm)
minStep	Minimum step size (mm)
steppingFormat	Format for printing steps
steppingVerbose	Set nonzero to print each step
viewer	Visualization driver selected (default=none)
wallClockLimit	Limit on wall clock time in seconds; -1 is infinite
worldMaterial	Material of the World volume
zTolerance	Tolerance for Z steps (mm)

steppingFormat is a space- or comma-separated list of items:

EXT	toggle extended precision (3 more digits)
TAG	print a '>' (useful to grep output)
N	step number
NSTEP	Synonym of N
GLOBAL	X,Y,Z,T in global coords
XYZT	Synonym of GLOBAL
CL	X,Y,Z,dxdz,dydz in CL coords
CLX	extended precision CL
KE	kinetic energy
STEP	step length
STEPLN	Synonym of STEP
VOL	volume name
VOLNAME	Synonym of VOL
PROCESS	process name
B	magnetic field
E	electric field
P	3-momentum
MAT	material name
ID	event ID, track ID, parent ID
PART	particle name
SEG	centerline coord segment number
WT	weight
POLAR	polarization
NL	<newline>
NEWLINE	Synonym of NL
\n	Synonym of NL

## 5 G4beamline Commands (Alphabetical)

**absorber**      construct an absorber

The absorber has two windows with beampipe and an absorber material. Optionally it has an additional two safety windows with beampipe. The WindowShape(s) are read from a file, and they determine the thickness and length of the beampipe(s). For geometry testing, acts like a cylinder enclosing the windows. For placing children, acts like a cylinder inside the central absorber.

Note that section 4.5 of the User's Guide has a dimensioned drawing of an absorber.

Named Arguments (#=cannot be changed in place cmd):

absWindow      The name of the absorber window. #  
safetyWindow   The name of the safety window. #  
insideLength   Absorber length inside windows (mm) #  
absMaterial     The material of the absorber #  
windowMaterial The material of the window(s) #  
safetyMaterial The material inside the safety windows. #  
safetyDistance Distance between absorber and safety windows(mm) #  
color           The color of the absorber (''=invisible)  
maxStep         The maximum stepsize in the element (mm)

**beam**           Define the Beam.

The beam command is: beam type arg1=v1 ...

Types are: gaussian, rectangular, ellipse, ascii, and root.

Gaussian beams are randomly generated to emanate from beamX,beamY,beamZ with the given sigmas; negative sigma means flat with |sigma| as halfwidth.

Rectangular beams are randomly generated to emanate from the rectangle beamHeight by beamWidth centered at beamX,beamY,beamZ.

Ellipse beams are randomly generated on the ellipses in (X,Xp), (Y,Yp), (T,E), with meanE determined from meanP and the sigmas used as half-widths; tracks are generated on the ellipse with uniform density when plotted with scales such that the ellipse is a circle.

ASCII beams are read from a file using the format specified; the formats supported are: BLTrackFile

Root beams are read from a .root file using the TNtuple named directory/name in the file. It must have the same fields as used in BLTrackFile format. Note that EventIDs greater than 16,777,216 will be truncated.

When reading a file (ascii or root), beamX and beamY are added to

input tracks; if beamZ is set it will overwrite the z of the track, but if it is not set the z of the track in the file is kept.

All coordinates are centerline coordinates.

Multiple beam commands can be given, and they will generate events in the order they appear in input.file.

Events are generated starting at firstEvent, until either nEvents have been generated or lastEvent would be exceeded.

For gaussian, rectangular, and ellipse beams, the beam particle can be given as either a particle name or its integer PDGid. Some common beam particle names are: proton, anti\_proton, pi+, pi-, mu+, mu-, e+, e-, kaon+, kaon-, kaon0, nu\_e, anti\_nu\_e. See the User's Guide for a complete list of particle names.

#### Named Arguments:

particle	Beam particle name
nEvents	Number of events to process (default=1 for generating events, default=ALL for reading files) set to lastEvent-firstEvent+1 if both are set.
firstEvent	First event # to process (default is the next sequential eventID, 1 if none)
lastEvent	Last (highest) event # to process
beamX	Beam location in X (mm)
beamY	Beam location in Y (mm)
beamZ	Beam location in Z (mm)
maxR	Beam maximum radius (mm)
rotation	Rotation of the beam
renumber	If nonzero, renumber events sequentially.
weight	Weight for events, overwritten by value from input file (1.0).
secondaryTrackID	The next TrackID for secondaries (1001).
meanMomentum	Gaussian Beam mean momentum (MeV/c)
meanP	Synonym for meanMomentum.
sigmaX	Gaussian Beam sigma in X (mm)
sigmaY	Gaussian Beam sigma in Y (mm)
sigmaZ	Gaussian Beam sigma in Z (mm)
sigmaXp	Gaussian Beam sigma in dxdz (slope)
sigmaYp	Gaussian Beam sigma in dydz (slope)
sigmaP	Gaussian Beam sigma in P (MeV/c)
sigmaT	Gaussian Beam sigma in T (ns)
sigmaE	Elliptical Beam sigma in E (MeV)
meanXp	Gaussian Beam mean in Xp (slope)
meanYp	Gaussian Beam mean in Yp (slope)
meanT	Gaussian Beam mean in T (ns)
polarization	Polarization 3-vector (0,0,0)
beamHeight	Rectangular Beam height (mm)
beamWidth	Rectangular Beam width (mm)
filename	input file name
file	synonym for filename.
directory	Root-file directory of NTuple
category	Deprecated synonym for directory.

uid	HistoScope uid of NTuple
name	Root name of NTuple.
format	ASCII file format (Default=BLTrackFile)
beamXp	Synonym for meanXp.
beamYp	Synonym for meanYp.

beamlossntuple NTuple containing particle tracks when lost.

The NTuple contains the usual track data:

The standard NTuple fields are:

x,y,z	(mm)
Px,Py,Pz	(MeV/c)
t	(ns)
PDGid	(11=e-, 13=mu-, 22=gamma, 211=pi+, 2212=proton, ...)
EventID	(may be inexact above 16,777,215)
TrackID	
ParentID	(0 => primary particle)
Weight	(defaults to 1.0)

The following additional fields are appended for format=Extended, format=asciiExtended, and format=rootExtended:

Bx, By, Bz	(Tesla)
Ex, Ey, Ez	(Megavolts/meter)
ProperTime	(ns)
PathLength	(mm)
PolX, PolY, PolZ	(polarization)
InitialKE	(MeV when track was created)

Valid Formats (ignore case): ascii bltrackfile dummy for009 for009.dat root trackfile Extended asciiExtended rootExtended

Named Arguments (#=cannot be changed in place cmd):

format	The NTuple format (see above for list).
filename	The filename for the NTuple.
file	Synonym for filename.
require	Expression which must be nonzero to include the track (default=1) #
coordinates	Coordinates: global, centerline, or reference (default=c).

box construct a box.

This is a direct interface to G4Box.

Named Arguments:

height	The height of the box (mm)
width	The width of the box (mm)
length	The length of the box (mm)
maxStep	The maximum stepsize in the element (mm)
material	The material of the box
color	The color of the box (''=invisible)
kill	Set nonzero to kill every track that enters.

bug1021 Workaround to improve accuracy of bug1021 in E field

When a charged particle turns around in an E field, a bug in the Geant4 transportation process can sometimes give it a wildly-incorrect kinetic energy. This workaround computes the distance to turn-around, and limits the step to half that value until minStep is reached; at that point the track is reflected.

Simulations in which there are no E fields, or no charged particle ever gets below ~0.001 MeV in an E field, have no need to apply this workaround.

#### Named Arguments:

minStep            Minimum step in space (mm, default=0.002)

coil                defines a coil (part of a solenoid)

A coil is a geometrical tube that can carry current when part of a solenoid. The field is computed for a set of nSheets infinitely-thin current sheets evenly spread radially. The solenoid specifies the actual current. For tracking the computation is too slow, so a field map on a grid in r and z is computed and written to filename (defaults to coilname.dat). While there are lots of parameters specifying the field map it is recommended to simply accept the defaults for all but innerRadius, outerRadius, length, material, and possibly tolerance. The other parameters will be determined so that the largest error is less than tolerance times the value of Bz at the center of the coil. If mapFile is given, the file is read in BLFieldMap format. The cache file contains the parameters, and is a field map in a binary format; it is automatically regenerated if any parameter changes.

Note that maxR and maxZ are by default determined to be large enough so that the field falls below tolerance; this can be quote large.

#### Named Arguments (#=cannot be changed in place cmd):

innerRadius	Inside radius of the coil (mm) #
outerRadius	Outside radius of the coil (mm) #
length	Length of the coil along z (mm) #
material	The material of the conductor (default=Cu) #
tolerance	The acceptable tolerance of the map. #
nSheets	Number of sheets used in the computation #
maxR	Maximum r value for the map (automatically determined by default). #
maxZ	Maximum z value for the map (automatically determined by default). #
dR	R interval between points of the map (automatically determined by default). #
dZ	Z interval between points of the map (automatically determined by default). #
filename	Filename for cache; defaults to name.dat. #
mapFile	Filename for map (e.g. from TOSCA). #
exactComputation	Set nonzero to use the exact computation without any field map (0).

collective     Monitor collective computation

This command computes the means and sigmas related to the time stepping in BLRunManager (global coordinates), generating a TimeStep NTuple. If the simulation has multiple bunches, this NTuple combines them all (and is thus almost useless).

This command can also generate field NTuple-s at specified points in x,y,z -- unnamed parameters should be 'x,y,z' values for monitoring E and B fields (global coordinates).

NOTE: This command must come AFTER other commands that compute collective fields; otherwise stale field values will be used from the previous time step. If deltaT is set > 0.0, this command will put the RunManager into collective mode and set its deltaT; otherwise the previous commands should do that, and this command won't modify deltaT.

Named Arguments:

deltaT             Time step (-1 ns).

format             Format of NTuples.

filenamePrefix     Prefix of NTuple filenames.

corner            Implement a corner in the centerline.

The centerline is bent by a rotation. Every track that enters the volume also gets rotated. The z value is for the corner and the front face of the volume (if any). If the corner is paired with a bending magnet or other mechanism to bend the beam, no volume should be used.

NOTE: This command is self-placing, do not use the place command; it also affects all following placements, and it cannot be issued inside a group. If radius=height=width=0 then no volume is associated with the corner, and a bending magnet should be placed nearby to bend the particles around the corner. Normally the bending magnet is placed before the corner, and is rotated by half the bend angle. For a sector bend, it's usually best to use the cornerarc command rather than this one.

NOTE: all placements before this command must have z values before the corner, and all placements after this command must have z values after the corner. Note also that the angle is limited to 90 degrees.

Note that the radiusCut is important to reduce or eliminate ambiguities in the global to centerline coordinate transform. It can also be used to 'shield' the beamline to prevent particles from taking unusual paths around the outside of beamline elements.

Named Arguments:

z                    The centerline Z of the corner (mm).

rotation            The rotation of the corner (see above).

radiusCut	The radius cut for this following segment (mm default=previous).
radius	The radius of the circular corner volume (mm).
height	The height of the rectangular corner volume (mm).
width	The width of the rectangular corner volume (mm).
length	The length of the corner volume (mm).
maxStep	The maximum stepsize in the element (mm).
material	The material of the corner volume.
color	The color of the corner volume (''=invisible).

cornerarc      Implement a cornerarc in the centerline.

The centerline is bent by a rotation. Three corners are used to approximate an arc; the total angle and path-length are equal to those for the arc. Should be used immediately after an `idealsectorbend` or a `genericbend`. The z value is for the front face of the arc.

NOTE: This command is self-placing, do not use the `place` command; it also affects all following placements, and it cannot be issued inside a group.

This command is well matched to a sector bend, but can also be used with a normal bending magnet -- normally the magnet is placed before the `cornerarc` and is rotated by half the bend angle.

The only useful rotations are those around the centerline Z.

z, angle, and centerRadius are required parameters.

NOTE: all placements before this command must have z values before the corner, and all placements after this command must have z values after the corner. Note also that the angle is limited to 90 degrees.

Note that the `radiusCut` is important to reduce or eliminate ambiguities in the global to centerline coordinate transform. It can also be used to 'shield' the beamline to prevent particles from taking unusual paths around the outside of beamline elements.

#### Named Arguments:

z	The centerline Z of the cornerarc (mm).
centerRadius	The radius of the centerline arc (mm).
angle	The angle of bend, >0=left, <0=right (degrees).
rotation	The rotation of the cornerarc (see above).
radiusCut	The radius cut for this following segment (mm).

cosmicraybeam Define a Cosmic-Ray muon 'beam'.

The muon beam is nominally headed in the +Z direction, implying that +Z is physically DOWN. The beam intersects a box defined by `beamWidth`, `beamHeight`, and `beamLength`, centered at `X=Y=0` and `beamZ`. For each event a point is selected randomly within this

box, angles theta and phi and the muon momentum are generated according to a fit to their sea-level distributions, the track is extended backwards to the 'celestial sphere', and that is the initial beam position for the event. The muon flux through the rectangle at Z=beamZ is used to display an estimate of the sea-level exposure time for the run.

**Named Arguments:**

nEvents	Number of events to process
beamZ	Beam location in Z (mm)
radius	Radius of celestial sphere (mm)
beamHeight	Rectangular Beam height (mm)
beamWidth	Rectangular Beam width (mm)
beamLength	Rectangular Beam length (mm)

cylinder      Alias for 'tubs'.

define      defines a macro (argument-expanded set of commands).

The first argument is the macro name, additional arguments become lines in the body of the expanded macro. The macro name becomes a command with up to 9 positional arguments. When the command is issued, the body is expanded and executed, with these substitutions:

\$0	MacroName
\$1-\$9	Positional arguments of the command
\$#	# macro expansions (for generating unique names)

NOTE: \$paramname is expanded in the define command, but \$\$paramname is expanded when the macro is invoked.

demo      demo command.

This demo command takes both positional and named args, and is the prototype class for all commands. All argument values are merely displayed. 'demo default name=value...' sets default values.

**Named Arguments:**

s1	a demo string argument.
s2	a demo string argument.
d1	a demo double argument.
d2	a demo double argument.

do      Do loop.

**Syntax:**

```
do i 1 10 [1]
    commands ...
enddo
```

Sets the parameter i to values 1, 2, 3, ... 10, and executes the commands. The increment is 1 by default, and negative increments are allowed (with limits reversed). 'do i 1 0' and 'do i 0 1 -1' will execute no commands. After completion, i retains its last

value. Do-s, for-s, and multi-line if-s can be nested in any manner.

Note: the do command will not work from standard-input or as a command in a single-line if.

**emfactor**      Multiply multiple scattering and energy loss by factors.

The multiple scattering and ionization energy loss processes are not really independent; the msc factor applies to scattering in both, and the eLoss factor applies to the fluctuations of energy loss in both. The 'mean' energy loss is not affected; note that in some EM models, including the one used for muons, the fluctuations only increase the loss from the 'mean', so the average energy loss is affected, but only slightly.

The deltaRay parameter is the probability of keeping any given delta-ray. The 3-momentum of each discarded delta-ray is added back into the particle's 3-momentum, and its K.E. is recomputed.

This command applies ONLY to multiple scattering and ionization energy loss. Other processes still apply, which can scatter and cause a reduction of the track's energy. For muons you may wish to disable these processes (in the physics command): muBrems, muPairProd, CoulombScat, Decay; similarly for other particles.

**Named Arguments:**

particle	Comma-separated list of particle names or patterns (default=none).
msc	Ratio for multiple scattering (1.0).
eLoss	Ratio for ionization energy loss fluctuations (1.0).
deltaRay	Probability to keep delta-rays (1.0).
Eloss	Synonym for eLoss.
eLoss	Synonym for eLoss.

**endgroup**      ends a group definition.

The group may then be placed as any other element. If the group was not given a length via an argument, endgroup computes the length and adjusts the offsets of all elements in the group so they refer to the center of the group.

**eventcuts**      Implements cuts on event number via lists in files.

The files are ASCII, with one event number per line. If the keep file is not empty, only event numbers listed in it will be analyzed (except those listed in the skip file). The skip file lists event numbers that will be skipped, and event numbers listed in both files will be skipped. When reading the files, lines beginning with '#' are ignored; blank lines are interpreted as event 0.

**Named Arguments:**

keep	The file containing a list of event numbers to
------	--

	analyze (default is all).
skip	The file containing a list of event numbers to skip.
filename	Synonym for keep.
file	Synonym for keep.

exit            exit a command file.

The exit command ceases reading the input file, and starts the simulation immediately (ignoring the remainder of the input file).

extrusion      construct a solid extrusion with axis along z

This is a basic interface to G4ExtrudedSolid. A simple polygon in the X-Y plane is extruded along z, with optional scales in XY at the two ends (which generate a linear scaling along z).

The polygon must be simple (no two sides intersect, no two vertices are equal). The vertices are listed starting from any vertex and traversing the polygon in either direction without lifting the pencil from the paper (Geant4 requires the traversal to be clockwise but this element internally reverses it if required). For an N-sided polygon give N vertices -- a side will be added from last to first to close the polygon; N is determined by counting the entries in the vertices argument.

Note that while you cannot make an extrusion with a hole, you can make such an object in two parts or by placing a daughter volume in this one.

Note the position placed is  $x=0, y=0, z=0$ , which is centered along z, but need not be near the center of the polygon in XY.

With  $scale1 \neq scale2$  this is not really an extrusion; by making one of them 0.001 or so, you can construct a sharp apex.

Any x or y value in vertices can be an expression using double constants and the usual C operators and functions.

Named Arguments (#=cannot be changed in place cmd):

length	Length of the extrusion (mm). #
vertices	List of vertices of the XY polygon (mm): 'x0,y0;x1,y1;...'; a line from last to first is added. A 2 mm square is: '-1,-1;-1,1;1,1;1,-1' #
scale1	The XY scale at the upstream (-z) end (1.0). #
scale2	The XY scale at the downstream (+z) end (1.0). #
maxStep	The maximum stepsize in the element (mm)
material	The material of the extrusion
color	The color of the extrusion (''=invisible)
kill	Set nonzero to kill every track that enters.
vertexes	Synonym for vertices. #

fieldexpr      implements a field map, E and/or B, from expressions.

A `fieldexpr` element can be either a box or a cylinder; set length and radius for cylinder, set length and width and height for a box. Units are Tesla, MegaVolts/meter, mm, and ns. Expressions for the field components can use `{x,y,z}` for a box or `{z,r}` for a cylinder; the time expression can use `{t}`. If present, the time expression multiplies all components.

Expressions can use all C operators except `?:`, and `x^n` is `x` to the `n`th power (`n` integer). The following functions are available: `abs()`, `min()`, `max()`, `sqrt()`, `pow()`, `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`, `atan2()`, `sinh()`, `cosh()`, `tanh()`, `exp()`, `log()`, `log10()`, `floor()`, `ceil()`, `if()`. `if(condition,a,b)` replaces the C `(condition? a : b)`.

A field map is used for tracking efficiency; the number of points in the map is increased until the largest map error divided by the maximum field is smaller than tolerance, or 1M points is exceeded. Similarly for the time dependence.

For time dependence: if `t-timeOffset<tmin`, the value at `tmin` is used; if `t-timeOffset>tmax`, the value at `tmax` is used.

Note that divide by zero is reported as invalid expression. For a Li lens you probably want to use an expression like this: `'if(r<100,500.0*r/100,500.0*100/(r+0.000001))'`, where 100mm is the radius, and 500T is the field at `r=100mm`.

Named Arguments (`#=cannot be changed in place cmd`) (`@=Tunable`):

<code>factorB</code>	Factor for the B-field (1.0). @
<code>factorE</code>	Factor for the E-field (1.0). @
<code>timeOffset</code>	Time offset (ns).
<code>Bx</code>	Expression for Bx (Tesla), use <code>{x,y,z}</code> . #
<code>By</code>	Expression for By (Tesla), use <code>{x,y,z}</code> . #
<code>Bz</code>	Expression for Bz (Tesla), use <code>{x,y,z}</code> or <code>{r,z}</code> . #
<code>Br</code>	Expression for Br (Tesla), use <code>{r,z}</code> . #
<code>Bphi</code>	Expression for Bphi (Tesla), use <code>{r,z}</code> . #
<code>Ex</code>	Expression for Ex (MV/m), use <code>{x,y,z}</code> . #
<code>Ey</code>	Expression for Ey (MV/m), use <code>{x,y,z}</code> . #
<code>Ez</code>	Expression for Ez (MV/m), use <code>{x,y,z}</code> or <code>{r,z}</code> . #
<code>Er</code>	Expression for Er (MV/m), use <code>{r,z}</code> . #
<code>time</code>	Expression for time-dependence factor, use <code>{t}</code> . #
<code>nX</code>	Number of grid points in x. #
<code>nY</code>	Number of grid points in y. #
<code>nZ</code>	Number of grid points in z. #
<code>nR</code>	Number of grid points in r. #
<code>nT</code>	Number of grid points in t. #
<code>tolerance</code>	Required relative accuracy (0.001). #
<code>length</code>	Length of field map (mm). #
<code>width</code>	Width of rectangular field map (mm). #
<code>height</code>	Height of rectangular field map (mm). #
<code>radius</code>	Radius of cylindrical field map (mm). #
<code>tmin</code>	Minimum value of t (ns). #
<code>tmax</code>	Maximum value of t (ns). #

`fieldlines` Display magnetic field lines.

Field lines are drawn starting within a circle specified as center and radius (global coordinates); the plane of the circle is normal to the B field at its center. Field lines are distributed within the circle with a density inversely proportional to  $|B|$ . While it is attempted to keep their spacing as uniform as possible, there are both ambiguity and randomness involved in placing the lines within the circle. Lines are placed within the circle from its center outward, and if  $|B| < \text{minField}$  then no more lines are placed.

nLines is only approximate, and the actual number of lines drawn will be within a factor of 2 of the value. Asking for fewer than 10 or more than 1000 lines is likely to be ineffective. Unnamed parameters can contain specific x,y,z values of points to start a field line.

Lines are drawn in both directions starting from the plane of the circle, and each half-line stops when it again reaches that plane. Line drawing also stops whenever  $|B|$  is less than minField or when it leaves the world. If you are interested in field lines far outside the magnets, you may need to add some object with a large value of x, y, and/or z, in order to expand the world volume. With dl=1 and subdivide=10, the accuracy of their meeting is usually better than 0.1 mm after several tens of meters.

This command does nothing if not in visualization mode. For best results, use the Open Inventor viewer, and give your magnets a transparency of about 0.3 (e.g. color=1,1,1,0.3); if necessary, use the right-button menu to set the DrawStyle/TransparencyType to 'sorted object add'. With field lines in 3-d, you will want the ability to zoom, rotate, and move the image interactively.

Setting exit=1 will display the system and the field lines, without any event tracks; exiting the viewer will then exit the program. With exit=0, field lines are drawn only at the end of the first run.

#### Named Arguments:

t	Time at which field lines are plotted (0 ns).
center	Center of circle (x,y,z) to start lines (mm, global).
radius	Radius of circle to start lines (mm).
nLines	Approximate number of field lines to plot (100).
dl	Interval between points plotted (10 mm).
color	Color of field lines (white="1,1,1").
minField	Minimum B field (0.001 tesla)
maxPoints	Max # points plotted in a line (10000).
subdivide	# field integration points between plotted points (10).
N	# grid points in x and y (128).
exit	Set nonzero to display field lines and exit (0).
square	Set nonzero to start from square rather than circle (0).
Efield	Set nonzero to draw E field (0); minField is in

MegaVolts/meter.  
 forever Set nonzero to draw lines until maxPoints is reached or  $|B| < \text{minField}$ , not stopping at the initial plane.

fieldmap implements a field map, E and/or B, from a file.

Reads an input file in BLFieldMap format to define E and/or B fields, optionally with time dependence. See the Users Guide for a description of the BLFieldMap format.

Named Arguments (#=cannot be changed in place cmd) (@=Tunable):

filename Filename for the Field Map. #  
 file Synonym for filename. #  
 current Current of the B-field. @  
 gradient Gradient of the E-field. @  
 timeOffset Time offset (ns).

fieldntuple Generates an NTuple from B and E fields at specified points.

Intended primarily for debugging. This command makes it easy to plot fields as a function of position and time, using existing NTuple plotting tools. Outputs x,y,z,t,Bx,By,Bz,Ex,Ey,Ez into an NTuple. Units are mm, ns, Tesla, and MegaVolts/meter. Runs after the reference particle is tracked. Only global coordinates are used.

The single positional argument is the name of the NTuple. Named arguments {x,y,z,t} are of two forms specifying coordinate values: x=Xmin,Xmax,dX or x=X1:X2:X3:... which generate the obvious loops (single value is OK). Expressions can be used. Omitted coordinates are held fixed at 0.0.

Named Arguments:

category The category of the NTuple.  
 format NTuple format (see above for list).  
 filename Name of file.  
 x Loop for x values: Xmin,Xmaz,dX or X1:X2:... (mm)  
 y Loop for y values: Ymin,Ymaz,dY or Y1:Y2:... (mm)  
 z Loop for z values: Zmin,Zmaz,dZ or Z1:Z2:... (mm)  
 t Loop for t values: Tmin,Tmaz,dT or T1:T2:... (ns)  
 exit Set nonzero to exit after generating NTuple (0).  
 file Synonym for filename.

for For loop.

Syntax:

```
for i v1 v2 v3 ...
  commands ...
endfor
```

Sets the parameter i to values v1, v2, v3, ..., and executes the commands. Values can be any strings, including numbers, except they cannot contain an '=' (named parameter). After completion, i retains its last value.

Do-s, for-s, and multi-line if-s can be nested in any manner.

Note: the for command will not work from standard-input or as a command in a single-line if.

**g4ui**                   Accesses the Geant4 user interface

Each positional argument is executed as a Geant4 UI command, according to the when parameter. No positional arguments means open a UI session on stdout/stdin. For a given value of when, the UI commands from all g4ui commands are executed in order.

Named Arguments:

when                   0=before reference, 1=before beam, 2=after beam,  
                      3=cannot be used, 4=visualization.

**genericbend**   construct a generic bending magnet.

The field region is a box with By specified. A fringe field computation based on the method of COSY INFINITY is included by default, extending the field in a rectangle extending the straight aperture along the local z. This is first order only, and assumes the magnet is infinitely wide; the fringe field extends outside of the magnet aperture only in a region extending the aperture in x and y. As the fringe field is first order only, it is slightly non-Maxwellian.

By default, the aperture is filled with a box volume of the fieldMaterial; this prevents placing any object inside the aperture. With openAperture=1 no aperture volume is used, and objects can be placed into the parent volume that are inside the aperture.

Named Arguments (#=cannot be changed in place cmd) (@=Tunable):

fieldWidth	The width of the field region (mm)
fieldHeight	The height of the field region (mm)
fieldLength	The length of the field region (mm)
ironWidth	The width of the iron region (mm)
ironHeight	The height of the iron region (mm)
ironLength	The length of the iron region (mm)
By	The magnetic field (Tesla) @
maxStep	The maximum stepsize in the element (mm)
fieldMaterial	The material of the field region.
fieldColor	The color of the field region.
ironMaterial	The material of the iron region.
ironColor	The color of the iron region.
kill	Set nonzero to kill particles hitting the iron.
fringe	Fringe field computation, set to 0 to disable, or a comma-separated list of 6 Enge function parameters.
fringeFactor	Fringe depth factor (1.0).
openAperture	Set nonzero to omit the aperture volume. #

**genericquad**   construct a generic quadrupole magnet.

The field region is a tube with gradient specified. A positive gradient yields a horizontally-focusing quad for positive particles. If `apertureRadius>0` the quad has a circular aperture. For a 'rounded +' aperture using circles for the poles, set `poleTipRadius`, `coilRadius`, `coilHalfwidth`. Due to visualization bugs, in the latter case you cannot see through the aperture; it is solid black. A fringe field computation based on the method of COSY INFINITY is included by default, extending the field region. This is first order only, and the fringe field extends outside of the magnet aperture only in a cylinder extending the aperture straight along local *z*. As the fringe field is first order only, it is slightly non-Maxwellian.

Named Arguments (#=cannot be changed in place cmd):

<code>fieldLength</code>	The length of the field region (mm)
<code>ironLength</code>	The length of the iron (mm)
<code>ironRadius</code>	The outer radius of the iron (mm)
<code>apertureRadius</code>	The radius of the aperture (mm)
<code>poleTipRadius</code>	The inner radius of the pole tips (mm).
<code>coilRadius</code>	The radius of the inside of the coil (mm).
<code>coilHalfwidth</code>	The halfwidth of the coil (mm).
<code>coilHalfwidth</code>	Synonym for <code>coilHalfwidth</code> .
<code>gradient</code>	The magnetic field gradient, dBy/dx (Tesla/meter)
<code>maxStep</code>	The maximum stepsize in the element (mm)
<code>ironMaterial</code>	The material of the iron region.
<code>fieldMaterial</code>	The material of the field region.
<code>ironColor</code>	The color of the iron region.
<code>kill</code>	Set nonzero to kill tracks hitting the iron.
<code>fringe</code>	Fringe field computation, set to 0 to disable, or a comma-separated list of 6 Enge function parameters.
<code>fringeFactor</code>	Fringe depth factor (1.0).
<code>openAperture</code>	Set nonzero to omit the aperture volume. #

`geometry` Arranges to perform a geometry test.

The geometry test checks `nPoints` points on the surface of each element, verifying that they are inside the parent element and that they are not inside any sibling element. The default of 100 points is usually sufficient; 0 means omit the geometry test. The first 20-40 points (depending on element) test 'corners', the rest are randomly distributed on the surface. The default tolerance is 0.002 mm.

Named Arguments:

<code>nPoints</code>	The number of surface points to test per element
<code>printGeometry</code>	Set nonzero to print the entire geometry.
<code>visual</code>	Set nonzero to display the geometry test points.
<code>tolerance</code>	Tolerance for inside/outside tests (mm).

`group` begins definition of a group.

A group is a collection of elements that can be placed together, preserving their relative positions. The group is a `LogicalVolume` in the `geant4` geometry -- this means that a group cannot overlap any other group or object, even if the overlapping portion of the

group is empty. If you need to permit overlaps, consider using a macro instead (the define command).

If the group is given a length, then children can be placed at specific z offsets relative to the center of the group. If the group is not given a length, then children can only be placed sequentially along z, and the length will be computed by the endgroup command. Width and height are computed from the largest child or the argument. If radius is set to 0, the group will be a cylinder with radius determined by the largest width or height placed into it; if >0 then that is the fixed radius. If radius is not set then a box is used.

Note: when placing objects into a group, if the rename argument is used, it should begin with a '+' to include the group's name in the object's name; otherwise there may be multiple objects with the same name -- this is only a major problem for virtualdetector-s and other output objects. The group's name is included by default if no rename is used on the place command.

Named Arguments (#=cannot be changed in place cmd):

length	Overall group length along z (mm) #
width	Overall group width (mm) #
height	Overall group height (mm) #
radius	Radius for a cylindrical group (mm) #
material	Material of the volume outside children
color	Color of the volume of the group
maxStep	The maximum stepsize in the volume (mm)

helicaldipole construct a helicaldipole magnet.

The field region is a cylinder with a helical dipole field plus a solenoid field. The simple model=1 provides just a sine and cosine transverse dependence, while the maxwellian model=2 has both dipole and quadrupole terms. Both the dipole scale bD [T] and quadrupole scale bQ [T/m] are now at rho=0; the user must determine the correct values externally.

Note that this Element generates a magnetic field only, and only within the cylinder defined by length and radius. So it has no solid associated with it, and is invisible.

Named Arguments:

radius	The radius of the field region (mm)
model	The model of field calculated(simple=1, rpj and ysd model=2, ttominaka et al model=3), modulations in bd,bq,bz= 4
length	The length of the field region (mm)
bD	The dipole magnitude at rho=0 (Tesla).
lambda	Helix period along the Z axis (mm).
phi0	The phase of the XY field at the entrance (deg).
Bsolenoid	The value of Bsolenoid (Tesla).
bQ	The quadrupole magnitude at rho=0 (Tesla).
bs	The sextupole magnitude at rho=0 (Tesla).
rr	Reference radius (mm)

psi0	The offset between the dipole term and the quadrupole term (Degrees).
ez	The base electric field inside the helix channel (GV/m).

helicalharmonic construct a helicalharmonic magnet.

Creates a cylindrical region containing the field of a magnetic helical harmonic of given order [n]. The field is defined by the value of the (n-1) order derivative [b] of the vertical field component (when the initial phase is 0) with respect to the horizontal coordinate at the center of the helix:

$b = d^{(n-1)} B_{\phi} / dr^{(n-1)} @ [r=0 \text{ \& } \phi - k*z + \phi_0 = 0]$ ,  
 where  $k = 2\pi / \lambda$  is the helix's wave number,  $\lambda$  is the length of the helix's period, and  $\phi_0$  is the initial phase. The field components in the cylindrical frame are given by:

$B_{\phi} = (2 / (n*k))^{(n-1)} * b * (I_{[n-1]}(n*k*r) - I_{[n+1]}(n*k*r)) * \cos(n*(\phi - k*z + \phi_0))$ ,  
 $B_r = (2 / (n*k))^{(n-1)} * b * (I_{[n-1]}(n*k*r) + I_{[n+1]}(n*k*r)) * \sin(n*(\phi - k*z + \phi_0))$ ,  
 $B_z = -2 * (2 / (n*k))^{(n-1)} * b * I_{[n]}(n*k*r) * \cos(n*(\phi - k*z + \phi_0))$ ,  
 where  $I_{[n]}(x)$  is the modified Bessel function of the first kind of order [n].

Note that this Element generates magnetic field only, and only within the cylinder defined by length and radius. So it has no solid associated with it, and is invisible.

Named Arguments:

radius	The radius of the field region (mm)
length	The length of the field region (mm)
n	Order of helical harmonic (i.e. n=1 for dipole)
b	(n-1)-order derivative of the field at the center (T/m <sup>(n-1)</sup> )
lambda	Helix period along the Z axis (mm).
phi0	The phase of the XY field at the entrance (rad).

help provides interactive help.

help with no arguments lists all commands. 'help command' gives more detailed help on that command. 'help \*' gives detailed help for all commands. If the first argument is -exit the program will exit after printing the help.

idealsectorbend construct an ideal sector bending magnet.

The field region is a sector with By specified. Unlike most Elements, the position of the idealsectorbend is the center of the front face of its field (aperture). angle>0 bends to the left around Y; angle<0 bends right. The only useful rotations are around the centerline Z. Should be followed immediately by a cornerarc. Note that -90<=angle<=90 degrees.

Named Arguments (@=Tunable):

angle	Angle of bend (degrees).
-------	--------------------------

fieldCenterRadius Center radius of field (mm).  
 fieldInnerRadius Inner radius of field (mm).  
 fieldOuterRadius Outer radius of field (mm).  
 fieldHeight Height of field (mm).  
 ironInnerRadius Inner radius of iron (mm).  
 ironOuterRadius Outer radius of iron (mm).  
 ironHeight Height of iron (mm).  
 By Magnetic field (Tesla). @  
 fieldMaterial Material of field.  
 fieldColor Color of field.  
 ironMaterial Material of iron.  
 ironColor Color of iron.  
 maxStep The maximum stepsize in the element (mm)  
 kill Set nonzero to kill particles hitting the iron.

if Conditional execution of command(s), and if/elseif/else/endif.

Single-line format:

```
if $i==1 CMD1 CMD2 ...
```

If the expression is true (nonzero), the commands are executed. The commands usually need to be quoted.

Multi-line format:

```

if $i==1
  CMD1 ...
elseif $i==2
  CMD2 ...
else
  CMD3 ...
endif

```

The commands are executed in the usual way; elseif and else are optional, but endif is mandatory. Any number of elseif-s and commands can be used. Do-s, for-s, and multi-line if-s can be nested in any manner.

If there are spaces in the expression, it must be quoted.

include includes a command file.

include requires one argument, the file to include.

lilens construct a simple Lithium lens.

This element consists of a current-carrying cylinder and its field. The field exists only between the end planes of the cylinder, out to adial infinity.

Named Arguments:

radius The radius of the current-carrying cylinder (5 mm)  
 length The length of the cylinder (100 mm).  
 current The current in the cylinder (100000 Amp).  
 material The material of the cylinder (Li).  
 color The color of the tube or cylinder (''=invisible)

maxStep            The maximum stepsize in the element (mm).

list               provides interactive list of interesting internal tables.

list with no arguments lists all lists except processes. 'list name' lists that specific one. 'list -exit name(s)' will exit after listing. List names are:

```
    commands      all commands
    materials      currently known materials
    physics        all physics lists
    particles       currently known particles
    processes       currently known physics processes ***
```

NOTE: the particles and processes lists are not populated until the physics list is selected (via the physics command). Different physics lists use different processes and particles.

\*\*\*NOTE: listing processes will prevent any simulating, as will a non-empty particle list.

Named Arguments:

particle           Comma-separated list of particles for which details will be printed

man                Alias for 'help'.

material           construct a new material.

This is an interface to G4Material. This command is rarely required, because elements and most common materials are available via the NIST database. Any material available from the NIST database can simply be used -- if it is unknown then it will be automatically defined from the database. Uncommon materials or nonstandard densities must be defined with this command.

The first argument to this command is the material name, which is always required; density is also required. The command to define an element (e.g. with non-standard density) is:

```
material H2 Z=1 A=1.01 density=0.000090
```

A mixture or compound is a combination of known materials and/or elements; the command is:

```
material water H,0.1119 O,0.8881 density=1.0
```

The numbers following the element names are their mass fractions (note that WATER is available from the NIST db). Either type of command can optionally have: pressure, temperature, state.

With no arguments, this command prints the current material table. Note that 'G4\_' is prepended to the names of most materials that are obtained from the NIST database; 'G4\_Al' and 'Al' refer to the same material (unless one was previously defined using this command).

The following three arguments permit track filtering for all volumes made of this material:

keep            A comma-separated list of particle names to keep.

kill A comma-separated list of particle names to kill.  
require An expression that must evaluate nonzero or the track  
is killed.

The require expression uses global coordinates and can use the  
following track variables:

x,y,z,Px,Py,Pz,t,PDGid,EventID,TrackID,ParentID,wt

The following materials are known from the NIST database, and  
will be automatically created on first use:

H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar K Ca Sc Ti V Cr Mn  
Fe Co Ni Cu Zn Ga Ge As Se Br Kr Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag  
Cd In Sn Sb Te I Xe Cs Ba La Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm  
Yb Lu Hf Ta W Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn Fr Ra Ac Th Pa  
U Np Pu Am Cm Bk Cf A-150\_TISSUE ACETONE ACETYLENE ADENINE  
ADIPOSE\_TISSUE\_ICRP AIR ALANINE ALUMINUM\_OXIDE AMBER AMMONIA  
ANILINE ANTHRACENE B-100\_BONE BAKELITE BARIUM\_FLUORIDE  
BARIUM\_SULFATE BENZENE BERYLLIUM\_OXIDE BGO BLOOD\_ICRP  
BONE\_COMPACT\_ICRU BONE\_CORTICAL\_ICRP BORON\_CARBIDE BORON\_OXIDE  
BRAIN\_ICRP BUTANE N-BUTYL\_ALCOHOL C-552 CADMIUM\_TELLURIDE  
CADMIUM\_TUNGSTATE CALCIUM\_CARBONATE CALCIUM\_FLUORIDE  
CALCIUM\_OXIDE CALCIUM\_SULFATE CALCIUM\_TUNGSTATE CARBON\_DIOXIDE  
CARBON\_TETRACHLORIDE CELLULOSE\_CELLOPHANE CELLULOSE\_BUTYRATE  
CELLULOSE\_NITRATE CERIC\_SULFATE CESIUM\_FLUORIDE CESIUM\_IODIDE  
CHLOROBENZENE CHLOROFORM CONCRETE CYCLOHEXANE 1,2-DICHLOROBENZENE  
DICHLORODIETHYL\_ETHER 1,2-DICHLOROETHANE DIETHYL\_ETHER  
N,N-DIMETHYL\_FORMAMIDE DIMETHYL\_SULFOXIDE ETHANE ETHYL\_ALCOHOL  
ETHYL\_CELLULOSE ETHYLENE EYE\_LENS\_ICRP FERRIC\_OXIDE FERROBORIDE  
FERROUS\_OXIDE FERROUS\_SULFATE FREON-12 FREON-12B2 FREON-13  
FREON-13B1 FREON-13I1 GADOLINIUM\_OXYSULFIDE GALLIUM\_ARSENIDE  
GEL\_PHOTO\_EMULSION Pyrex\_Glass GLASS\_LEAD GLASS\_PLATE GLUCOSE  
GLUTAMINE GLYCEROL GUANINE GYPSUM N-HEPTANE N-HEXANE KAPTON  
LANTHANUM\_OXYBROMIDE LANTHANUM\_OXYSULFIDE LEAD\_OXIDE  
LITHIUM\_AMIDE LITHIUM\_CARBONATE LITHIUM\_FLUORIDE LITHIUM\_HYDRIDE  
LITHIUM\_IODIDE LITHIUM\_OXIDE LITHIUM\_TETRABORATE LUNG\_ICRP M3\_WAX  
MAGNESIUM\_CARBONATE MAGNESIUM\_FLUORIDE MAGNESIUM\_OXIDE  
MAGNESIUM\_TETRABORATE MERCURIC\_IODIDE METHANE METHANOL MIX\_D\_WAX  
MS20\_TISSUE MUSCLE\_SKELETAL\_ICRP MUSCLE\_STRIATED\_ICRU  
MUSCLE\_WITH\_SUCROSE MUSCLE\_WITHOUT\_SUCROSE NAPHTHALENE  
NITROBENZENE NITROUS\_OXIDE NYLON-8062 NYLON-6-6 NYLON-6-10  
NYLON-11\_RILSAN OCTANE PARAFFIN N-PENTANE PHOTO\_EMULSION  
PLASTIC\_SC\_VINYLTOLUENE PLUTONIUM\_DIOXIDE POLYACRYLONITRILE  
POLYCARBONATE POLYCHLOROSTYRENE POLYETHYLENE MYLAR PLEXIGLASS  
POLYOXYMETHYLENE POLYPROPYLENE POLYSTYRENE TEFLON  
POLYTRIFLUOROCHLOROETHYLENE POLYVINYL\_ACETATE POLYVINYL\_ALCOHOL  
POLYVINYL\_BUTYRAL POLYVINYL\_CHLORIDE POLYVINYLIDENE\_CHLORIDE  
POLYVINYLIDENE\_FLUORIDE POLYVINYL\_PYRROLIDONE POTASSIUM\_IODIDE  
POTASSIUM\_OXIDE PROPANE lPROPANE N-PROPYL\_ALCOHOL PYRIDINE  
RUBBER\_BUTYL RUBBER\_NATURAL RUBBER\_NEOPRENE SILICON\_DIOXIDE  
SILVER\_BROMIDE SILVER\_CHLORIDE SILVER\_HALIDES SILVER\_IODIDE  
SKIN\_ICRP SODIUM\_CARBONATE SODIUM\_IODIDE SODIUM\_MONOXIDE  
SODIUM\_NITRATE STILBENE SUCROSE TERPHENYL TESTIS\_ICRP  
TETRACHLOROETHYLENE THALLIUM\_CHLORIDE TISSUE\_SOFT\_ICRP  
TISSUE\_SOFT\_ICRU-4 TISSUE-METHANE TISSUE-PROPANE TITANIUM\_DIOXIDE  
TOLUENE TRICHLOROETHYLENE TRIETHYL\_PHOSPHATE

TUNGSTEN\_HEXAFLUORIDE URANIUM\_DICARBIDE URANIUM\_MONOCARBIDE  
 URANIUM\_OXIDE UREA VALINE VITON WATER WATER\_VAPOR XYLENE GRAPHITE  
 LH2 LN2 LO2 lAr lKr lXe PbWO4 Galactic GRAPHITE\_POROUS LUCITE  
 BRASS BRONZE STAINLESS-STEEL CR39 OCTADECANOL KEVLAR DACRON  
 NEOPRENE DNA\_ADENINE DNA\_GUANINE DNA\_CYTOSINE DNA\_THYMINE  
 DNA\_URACIL DNA\_ADENOSINE DNA\_GUANOSINE DNA\_CYTIDINE DNA\_URIDINE  
 DNA\_METHYLURIDINE DNA\_MONOPHOSPHATE DNA\_A DNA\_G DNA\_C DNA\_U  
 DNA\_MU Stainless304 Stainless316 lHe

Aliases: LHe=lHe Air=AIR, H2O=WATER, Vacuum=Galactic, LH2=lH2,  
 Scintillator=POLYSTYRENE

#### Named Arguments:

a	Effective Atomic Weight of the material (g/mole)
z	Effective Atomic Number of the material
density	Density of the material (gm/cm <sup>3</sup> )
pressure	Pressure of the material (Atm)
temperature	Temperature of the material (K)
state	State of the material (g,l, or s)
A	Synonym for a (g/mole)
Z	Synonym for z
keep	A comma-separated list of particle names to keep (all others are killed; ignored if empty).
kill	A comma-separated list of particle names to kill.
require	An expression that must evaluate nonzero or the track is killed.

movie Generate movie NTuple.

This command outputs a set of NTuples suitable for generating a movie.

#### Named Arguments:

coordinates	Coordinates: global, centerline, or reference (default=r).
-------------	---

multipole construct a generic multipole magnet.

Multipole magnetic fields from dipole through dodecapole are implemented with a cylindrical field region and optional surrounding iron (ironLength=0 or ironRadius=0 omits it). All fields with positive strengths are oriented so in the X-Z plane for X>0 (beam left) the field is purely By. Negative strengths are allowed and reverse the field. The fringe field computation is not implemented.

#### Named Arguments (#=cannot be changed in place cmd):

fieldLength	The length of the field region (mm)
ironLength	The length of the iron (mm)
ironRadius	The outer radius of the iron (mm)
apertureRadius	The radius of the aperture (mm)
ironMaterial	The material of the iron region.
fieldMaterial	The material of the field region.
dipole	Strength of dipole (Tesla)
quadrupole	Strength of quadrupole (T/m)

sextupole	Strength of sextupole (T/m <sup>2</sup> )
octopole	Strength of octopole (T/m <sup>3</sup> )
decapole	Strength of decapole (T/m <sup>4</sup> )
dodecapole	Strength of dodecapole (T/m <sup>5</sup> )
ironColor	The color of the iron region.
kill	Set nonzero to kill tracks hitting the iron.
maxStep	The maximum stepsize in the element (mm)
fringe	Fringe field computation, set to 0 to disable
fringeFactor	Fringe depth factor (1.0).
openAperture	Set nonzero to omit the aperture volume. #

muminuscapturefix Fixes up the G4MuonMinusCaptureAtRest process.

This class adds extra neutrons to mu- capture. The neutrons are added with a Poisson distribution having a mean of neutronMeanNumber, and with an exponential distribution in kinetic energy:

$(1/\text{neutronMeanEnergy}) * \exp(-\text{KE}/\text{neutronMeanEnergy})$

As the muonic atom cascades to its ground state it forgets the incident mu- direction, so the extra neutrons are generated isotropically in the lab.

The extra neutrons are added only to those captures that are hadronic (i.e. not decay in orbit). The value of neutronMeanNumber should reflect this.

The default values correspond to Aluminum.

Named Arguments:

neutronMeanNumber Mean number of extra neutrons per nuclear capture (2.5).

neutronMeanEnergy Mean energy of neutron spectrum (MeV)

newparticlentuple NTuple containing particle tracks when created.

Note that initial beam particles are included, unless require is set to 'ParentID>0'.

The standard NTuple fields are:

x,y,z (mm)

Px,Py,Pz (MeV/c)

t (ns)

PDGid (11=e-, 13=mu-, 22=gamma, 211=pi+, 2212=proton, ...)

EventID (may be inexact above 16,777,215)

TrackID

ParentID (0 => primary particle)

Weight (defaults to 1.0)

The following additional fields are appended for format=Extended, format=asciiExtended, and format=rootExtended:

Bx, By, Bz (Tesla)

Ex, Ey, Ez (Megavolts/meter)

ProperTime (ns)

PathLength (mm)

PolX, PolY, PolZ (polarization)

InitialKE (MeV when track was created)

Valid Formats (ignore case): ascii bltrackfile dummy for009  
for009.dat root trackfile Extended asciiExtended rootExtended

Named Arguments (#=cannot be changed in place cmd):  
format           The NTuple format (see above for list).  
filename         The filename for the NTuple.  
file             Synonym for filename.  
require          Expression which must be nonzero to include the  
                  track (default=1) #  
coordinates      Coordinates: global, centerline, or reference  
                  (default=c).  
kill             Set nonzero to kill tracks after entering into  
                  NTuple; does not kill track if require fails (0).

ntuple           Define an NTuple containing multiple detectors.

An ntuple holds the data from multiple detectors in a single NTuple, with one row (entry) per event or per track. This permits the generation of plots that compare different detectors. Up to 64 detectors can be used. While 'detector' is used in this description, any existing NTuple can be used, generated by any command, such as: virtualdetector, zntuple, beamlossntuple, timentuple, and newparticlentuple.

There are two ways the detectors can be combined: the default method is to construct a new NTuple that combines the fields of all the detectors, each prepended with the detector name. If union=1 is given, then all detectors must have the same list of fields, and that list is used for this NTuple; any hit in any detector is simply copied to this NTuple -- this permits multiple detectors to be combined into a single NTuple (you may want noSingles=1 for the detectors). With union=1, the 'required', 'veto' and 'minHit' arguments cannot be used.

By default, an entry in this ntuple is made for each event satisfying the require, veto, and minHit conditions; if perTrack=1 then an entry is made for each track that satisfies the require, veto, and minHit conditions. Any hit in any detector matching the patterns in veto will prevent the event/track from being entered into the ntuple.

If multiple hits occur in a given detector during the event or track, only the first one is kept in this ntuple. Detectors are specified by patterns identical to UNIX file-matching, so '\*Det\*' matches any detector with 'Det' anywhere in its name, etc. The 'required' argument permits events to be omitted unless all of the matching detectors were hit at least once in the event or track. The patterns in 'required' are applied only to detectors in the ntuple, so a simple '\*' only matches detectors named in the 'detectors' argument.

If union is nonzero, the hits in each detector are entered into this NTuple as they occur; each hit in any detector is included

as a row in this NTuple. All detectors must have the same fields, which are used for this NTuple.

NOTE: the name of a detector is by default the concatenation of its ancestors' names before its own (except World), unless `rename=NAME` was used in its place command. The patterns are applied to the names of the virtualdetectors as they were placed (including rename), not the bare name of the virtualdetector command. If `'rename=det#'` was used when placing the virtualdetector-s, you probably want a `*` to match the `#`, or list them individually (`det1,det2,det3...`).

NOTE: This command does not work correctly in collective tracking mode, unless `union=1`.

Valid Formats (ignore case): `ascii bltrackfile dummy for009 for009.dat root trackfile`

Named Arguments:

<code>category</code>	The category of the NTuple.
<code>detectors</code>	A comma-separated list of detector patterns.
<code>required</code>	A comma-separated list of required detector patterns (default= <code>*</code> ).
<code>veto</code>	A comma-separated list of detector patterns, any hit cancels entry into the NTuple (default= <code>''</code> ).
<code>format</code>	NTuple format (see above for list).
<code>filename</code>	Name of file.
<code>minHit</code>	Minimum number of detectors hit (default 0).
<code>perTrack</code>	Nonzero for an entry per track; 0 for an entry per event (default 0).
<code>union</code>	Set nonzero to perform a union of the detectors, rather than combining them.
<code>require</code>	Synonym for required.
<code>minHits</code>	Synonym for minHit.
<code>file</code>	Synonym for filename.

`output`                   redirects stdout and stderr to a file

`output` requires one argument, the new output file. Any output generated before this command will not appear in the file, so this command should come at the start of the input file, preceded only by setting parameters. After the redirection, it will re-print the G4beamline version and the current parameter values to the file. The most common usage is to redirect output to a file named like the `.root` file, when that is determined by parameter values:

```
param -unset param1=1.0 param2=3.0
param histofile=$param1,$param2
output $histofile.out
```

`param`                   Defines parameter values.

Parameters are named strings used to parameterize the input file; a few are used for program control. Parameters are set by the `param` command, and on the command line (all program arguments

after the first are interpreted as name=value).

'param name=value ...' defines parameters. If no arguments are present, all parameters are displayed. If the first argument is '-unset', this command will not overwrite parameters that are already set (e.g. from the command line - this permits the input.file to set defaults that can be overridden on the command line).

Parameters are expanded only in the arguments of commands: 'cmd argname=\$paramname [...]'; real-valued expressions for arguments can use \$paramname as a value, as long as paramname contains a valid real expression.

Parameters are most useful for setting global things like viewer and histoFile, or as parameters used in the input.file.

When a parameter is used, if it has not been defined, it will be defined from the environment if possible; if it is not defined in the environment then this generates an error message.

The values of parameters are strings, but if the value of a parameter is set to a valid real expression including at least one operator {+-\*/^<>=()!~&%|?}, the parameter value will be set to the numerical value of the expression to 8 significant digits.

NOTE: pre-defined Program control parameters (listed below) are defined before the command-line and are not affected by -unset.

#### Program control Parameters:

Zcl	Last centerline Z position used (updated continuously)
deltaChord	Geant4 tracking parameter
deltaIntersection	Geant4 tracking parameter
deltaOneStep	Geant4 tracking parameter
epsMax	Geant4 tracking parameter
epsMin	Geant4 tracking parameter
eventTimeLimit	CPU Time Limit (sec)
fieldVoxels	Size of voxels for field computation (mm)
histoFile	Default (Root) NTuple output filename
histoUpdate	Output update interval (events)
maxStep	Maximum physics step size (mm)
minStep	Minimum step size (mm)
steppingFormat	Format for printing steps
steppingVerbose	Set nonzero to print each step
viewer	Visualization driver selected (default=none)
wallClockLimit	Limit on wall clock time in seconds; -1 is infinite
worldMaterial	Material of the World volume
zTolerance	Tolerance for Z steps (mm)
steppingFormat is a space- or comma-separated list of items:	
EXT	toggle extended precision (3 more digits)
TAG	print a '>' (useful to grep output)
N	step number
NSTEP	Synonym of N
GLOBAL	X,Y,Z,T in global coords
XYZT	Synonym of GLOBAL

CL	X,Y,Z,dxdz,dydz in CL coords
CLX	extended precision CL
KE	kinetic energy
STEP	step length
STEPLEN	Synonym of STEP
VOL	volume name
VOLNAME	Synonym of VOL
PROCESS	process name
B	magnetic field
E	electric field
P	3-momentum
MAT	material name
ID	event ID, track ID, parent ID
PART	particle name
SEG	centerline coord segment number
WT	weight
POLAR	polarization
NL	<newline>
NEWLINE	Synonym of NL
\n	Synonym of NL

particlecolor Set the colors for particle tracks.

Arguments are of the form 'name=1,1,0', where name is the standard name of a particle, and 1,1,0 is the R,G,B value desired for its color ('' for invisible) The special names plus, minus, and neutral will set colors for unnamed particles of each charge. The name reference will apply to the reference track (defaults to invisible).

particlefilter Will kill particles from a list, or force particles to decay.

A particlefilter will force the decay of certain particles when they enter the physical volume of the element. The list of affected particles is in the 'decay' argument, and the normal Decay process is disabled for them. In addition, the 'kill' argument is a list of particles to kill when they enter the element, and the 'keep' argument will kill all particles not named, if it is not empty.

require is an expression involving track parameters that will kill the track if it evaluates to zero (use a comparison operator). The variables available are:

x,y,z,t,Px,Py,Pz,Ptot,PDGid,EventID,TrackID,ParentID  
Units are mm, ns, MeV/c.

If nWait > 1, particles will not be killed until they hit this element nwait times; this can be used to limit the number of revolutions around a ring. Decays are unaffected by nWait. referenceWait does the same for the reference particle The element can be either a cylinder or a box: set length and radius, or set length and width and height.

Named Arguments (#=cannot be changed in place cmd):

radius	The radius of the cylindrical particlefilter (mm).
innerRadius	The inner radius of the cylindrical particlefilter (0 mm, solid).
height	The height of the rectangular particlefilter (mm).
width	The width of the rectangular particlefilter (mm).
length	The length of the particlefilter (mm).
maxStep	The maximum stepsize in the element (mm).
material	The material (default=parent's).
color	The color of the particlefilter (white).
decay	A comma-separated list of particle names to decay. #
kill	A comma-separated list of particle names to kill. #
keep	A comma-separated list of particle names to keep. #
nWait	Intersection # to do the kill (default = 1)
referenceWait	Intersection # for reference (default = 1)
require	Expression which will kill the track if zero. #
steppingVerbose	Nonzero to display track kills.
decays	Synonym for decay. #

particlesource Interface to the Geant4 General Particle Source.

The Geant4 General Particle Source (GPS) is a very flexible and general way to generate events. It is controlled by Geant4 commands which should follow this command in the input.file. If you have a macro, you can include it using either the G4beamline 'include' command or the Geant4 command '/control/execute'. Note that G4beamline only recognizes Geant4 commands when the '/' is in column 1.

Due to the design of the GPS, only one particlesource command can be used, but the GPS permits multiple sources to be combined.

NOTE: the Geant4 General Particle Source inherently uses global coordinates, so this is most useful at the beginning of a beamline when global=centerline. Note also that it is very easy to generate a beam headed in the -z direction (this command will rotate to the +z direction: '/gps/ang/rot1 -1 0 0').

To use this, see the User Manual for the GPS at <http://reat.space.qinetiq.com/gps>

Named Arguments:

nEvents	Number of events to process (default=1), set to lastEvent-firstEvent+1 if both are set.
firstEvent	First event # to process (default is the next sequential eventID, 1 if none)
lastEvent	Last (highest) event # to process
secondaryTrackID	The next TrackID for secondaries (1001).

physics Defines the physics processes and controls them.

Exactly one physics command must be present. This command implements the geant4 physics lists of physics processes. The command is 'physics QGSP' for the QGSP list, and similarly for

the other lists. With no argument it prints the available physics lists.

Note that stochastic processes are always disabled while tracking the tune and reference particles. The only non-stochastic processes are Transportation and ionization energy loss (with fluctuations disabled).

For muon beam studies you may want to disable the Decay process.

spinTracking=1 will enable tracking the spins of e+, e-, mu+, and mu- (only), including pion decays to polarized muons, polarized muon decays, and the force from the particle's magnetic moment. The rare pion decay to e nu is included, but the e are unpolarized. The muon decays give the correct distribution for the electron, but only approximate distributions for the neutrinos, and all are unpolarized.

NOTE: this command defines the particles used throughout the simulation, so this command must come before others that use particle names.

NOTE: the rare decay mode for pi+/pi- to e nu is always added, with branching ratio 1.230E-4.

The default all-around physics list for HEP is called 'QGSP\_BERT'.

NOTE: At present there is a bug in Geant4 tracking and the magnetic moment is not used. For >~ 1 GeV muons and practical fields this is a very small error.

#### Named Arguments:

disable	Comma-separated list of processes to disable (e.g. 'Decay,msc').
inactivate	Synonym of disable.
deactivate	Synonym of disable.
doStochastics	Set to zero to disable all stochastic processes.
fluct	Set to zero to disable ionization energy loss fluctuations.
minRangeCut	Minimum range cut for particle production (1 mm)
list	Nonzero to list the processes (later in output).
gammaToMuPair	Nonzero to add gamma->mu+mu- (0).
spinTracking	Nonzero to track particle spins (0).
synchrotronRadiation	Nonzero to add synchrotron radiation to the physics list for e- and e+.
synchrotronRadiationMuon	Nonzero to add synchrotron radiation to the physics list for mu- and mu+ NOTE: This is experimental, and may not work correctly.

----- PHYSICS LISTS -----

Further guidance in selecting physics lists is available at:  
<http://geant4.web.cern.ch/geant4/support/physicsLists/referencePL/index.shtml>

The default all-around physics list for HEP is called 'QGSP\_BERT'.

LHEP uses exclusively parameterized modeling.

FTF lists use the FRITIOF description of string excitation and fragmentation.

QGSP lists use the quark gluon string model

QGSC are as QGSP except applying CHIPS modeling for the nuclear de-excitation.

\_BERT uses Geant4 Bertini cascade below ~ 10 GeV.

\_BIC uses Geant4 Binary cascade below ~ 10 GeV.

\_EMV suffix indicates a faster but less accurate EM modeling.

\_EMX suffix indicates the low-energy EM processes.

\_HP suffix uses the data driven high precision neutron package (thermal to 20 MeV).

\_NQE suffix indicates a list for comparison with earlier release.

List of available physics lists:

CHIPS No synopsis available.

FTFP\_BERT For calorimetry. The FTF model is based on the FRITIOF description of string excitation and fragmentation. Uses Geant4 Bertini cascade for primary protons, neutrons, pions and Kaons below ~10GeV.

FTFP\_BERT\_TRV A variant of QGSP\_BERT where the Geant4 Bertini cascade is only used for particles below ~5.5 GeV.

FTF\_BIC No synopsis available.

LBE For low background experiments (e.g. underground)

LHEP For calorimetry -- is the fastest, when it comes to CPU. It uses the LEP and HEP parametrized models for inelastic scattering. The modeling parametrizes the final states individual inelastic reactions, so you will not see resonances, and the detailed secondary angular distributions for O(100MeV) reactions may not be described perfectly. The average quantities will be well described.

QBBC No synopsis available.

QGSC\_BERT For calorimetry and high energy physics trackers -- is as QGSP for the initial reaction, but uses chiral invariant phase-space decay (multi-quasmon fragmentation) to model the behavior of the system's fragmentation. Uses Geant4 Bertini cascade for nucleon and pion induced reactions.

QGSP For calorimetry and high energy physics trackers and high-energy and medium-energy production targets -- uses theory driven modeling for the reactions of energetic pions, kaons, and nucleons. It employs quark gluon string model for the 'punch-through' interactions of the projectile with a nucleus, the string excitation cross-sections being calculated in quasi-eikonal approximation. A pre-equilibrium decay model with an extensive evaporation phase to model the behavior of the nucleus 'after the punch'. It uses current best pion cross-section.

QGSP\_BERT Like QGSP, but using Geant4 Bertini cascade for primary protons, neutrons, pions and Kaons below ~10GeV. In comparison to experimental data we find improved agreement to data compared to QGSP which uses the low energy

parameterised (LEP) model for all particles at these energies. The Bertini model produces more secondary neutrons and protons than the LEP model, yielding a better agreement to experimental data.

QGSP\_BERT\_CHIPS No synopsis available.

QGSP\_BERT\_HP Like QGSP\_BERT but with \_HP modeling for neutrons.

QGSP\_BIC Like QGSP, but using Geant4 Binary cascade for primary protons and neutrons with energies below ~10GeV, thus replacing the use of the LEP model for protons and neutrons. In comparison to the LEP model, Binary cascade better describes production of secondary particles produced in interactions of protons and neutrons with nuclei.

QGSP\_BIC\_HP Like QGSP\_BIC but with \_HP modeling for neutrons.

QGSP\_FTFP\_BERT No synopsis available.

QGS\_BIC No synopsis available.

QGSP\_INCLXX No synopsis available.

QGSP\_INCL\_ABLA No synopsis available.

Shielding No synopsis available.

**pillbox** Defines a pillbox RF cavity

A Pillbox RF cavity is the basic RF element used to construct a linac. The phaseAcc parameter sets the phase of the tune particle at the center of the cavity, and the timing offset of the cavity is determined from that the first time that the Tune particle is tracked through the cavity. Zero degrees is the rising zero-crossing of the Ez field. If timeOffset is specified, it is used rather than setting it from the Tune particle.

The Pipe, walls, and collars are always made of copper. Pipe, wall, collar, win1, and win2 can be omitted by setting their thickness to 0. Common usage is to set the collar values so by placing multiple pillboxes sequentially the collars form a beam pipe between them.

Note that section 4.4 of the User's Guide has a dimensioned drawing of a pillbox.

Named Arguments (#=cannot be changed in place cmd) (@=Tunable):

maxGradient	The peak gradient of the cavity (MV/m) @
color	The color of the cavity
frequency	The frequency of the cavity (GHz) #
innerLength	The inside length of the cavity (mm) #
innerRadius	The inside radius of the cavity (mm) #
pipeThick	The thickness of the pipe wall (mm) #
wallThick	The thickness of the cavity walls (mm) #
irisRadius	The radius of the iris (mm) #
collarRadialThick	The radial thickness of the collar (mm) #
collarThick	The thickness of the collar along z (mm) #
win1Thick	The thickness of the central portion of the windows; zero for no window (mm) #
win1OuterRadius	The radius of the central portion of the windows (mm) #
win2Thick	The thickness of the outer portion of the windows;

```

                                zero for no window (mm) #
winMat                        The material of the windows
phaseAcc                     The reference phase of the cavity (degrees)
skinDepth                    The skin depth (mm) #
timingTolerance               Tolerance for timing tuning (ns)
maxStep                      The maximum stepsize in the element (mm).
cavityMaterial               Material of cavity volume (Vacuum).
timeOffset                   Time offset for cavity (default: tuned by tune
                                particle) (ns).
timeIncrement                Increment to timeOffset, applied AFTER tuning.
                                (ns).
fieldMapFile                 Filename for BLFieldMap (pillbox if null). #
kill                         Set nonzero to kill tracks that hit the pipe,
                                walls, or collars (0).

```

place places an element into the current group (or world).

Every element can be placed multiple times into the beamline. For most elements the geometrical centerpoint is placed; for polycone the local  $x=y=z=0$  point is placed. If front is nonzero then the front of the element is placed. If z is specified, then the element is placed at that z position relative to the center of the enclosing group. If z is not specified, then the element is placed immediately downstream (higher z) of the previous element in the group, or at the upstream edge of the group if this is the first element in the group.

The 'rename' argument can be used to change the name of the element (applies to traces and other uses of object names, such as the NTuple name of a virtualdetector). When placing into a group or other object, the rename argument should normally begin with '+' to include the parent's name; otherwise multiple placements of the parent will generate multiple objects with identical names -- that should be avoided for output objects like virtualdetector. Without a rename argument, the parent's name is included automatically.

When multiple copies are placed, z refers to the first, and the rest are placed sequentially along z. When placing an element into the World group, Centerline coordinates are used unless coordinates=global is present. When centerline coordinates are used, the parameter 'Zcl' is set to the highest Z value used; this is normally the Z value for the front of the next element when placed sequentially (i.e. with no z value given).

**Rotations:** The rotation parameter can be used to rotate this element relative to the enclosing group. The object is rotated, not the axes. Rotations are specified as a comma-separated list of axes and angles (in degrees): rotate=Z90,X45 rotates first by 90 degrees around Z and then by 45 degrees around X. The axes are the local X,Y,Z coordinate axes of the enclosing group (centerline or global coordinate axes for the World group); groups can be rotated when placed, and are rotated as a rigid unit (including children).

If parent=name is present, then the name must be an element that accepts children, and it is used as the enclosing group; in this case the size of the group is implied by the size of the parent element, and z must be given (defaults to 0). Note that a given element cannot be the parent of any other element once it has been placed, so you must place children into their parent before placing their parent.

If the special element 'OFFSET' is given, x, y, and z specify offsets for every following place command into the current group (incl. World), that gives a z position.

#### Named Arguments:

z	Z position of element's center relative to the center of the enclosing group (mm).
x	X position of element's center [default=0] (mm)
y	Y position of element's center [default=0] (mm)
parent	Parent element name (must accept children).
rename	Name to use for this placement; '#' will be substituted by the number of placements. If the value begins with '+', it is replaced with the parent's name.
copies	Number of copies (placed sequentially along z).
front	Nonzero to specify z for the front, not the center.
rotation	Rotation of this object.
coordinates	Coordinates: global or centerline (default=c).

polycone      construct a polycone with axis along z

This is a direct interface to G4Polycone. For a solid polycone, omit innerRadius and it will be filled with zeroes. The number of entries in z, innerRadius, and outerRadius must be the same. Note that a polycone is placed at its z=0,r=0 point, which need not be its geometric center.

#### Named Arguments (#=cannot be changed in place cmd):

innerRadius	Comma-separated list of inner radii (mm) #
outerRadius	Comma-separated list of outer radii (mm) #
z	Comma-separated list of z positions (mm) #
initialPhi	The initial Phi value (deg; 0 for all)
finalPhi	The final Phi value (deg; 360 for all)
maxStep	The maximum stepsize in the element (mm)
material	The material of the polycone
color	The color of the polycone (''=invisible)
kill	Set nonzero to kill every track that enters.

printf      prints track variables and expressions

This is an interface to the C printf() function. The first positional argument is the format, and the following positional arguments are double expressions printed with the % fields in the format. Up to 16 expressions can be printed. The print is performed only if the 'required' expression is nonzero, as each track reaches one of the Z positions in the 'z' argument (centerline coordinates). Multiple printf commands with the same

'file' will be combined into the file as tracks reach any of their Z positions. More than 16 expressions can be broken into multiple printf-s with noNewline=1 for all but the last.

The following variables can be used in expressions:

```
x,y,z,t,Px,Py,Pz
PDGid,EventID,TrackID,ParentId,Weight
Bx,By,Bz, Ex,Ey,Ez
tune (nonzero only for the Tune particle)
reference (nonzero only for the Reference particle)
beam (nonzero for any Beam particle)
```

Each value in z and zloop can be an expression using double constants and the usual C operators and functions.

Example:

```
printf z=0 'Momentum is %.3f GeV/c' sqrt(Px*Px+Py*Py+Pz*Pz)/1000
```

NOTE: if format begins 'Ptot=...' the parsing will think it is a named argument; put a space before the '=' to avoid that error.

Named Arguments:

```
z          Comma-separated list of Z positions for printing
           (mm)
zloop      Loop in z, first:last:incr (mm)
require    logical expression for cutting (default=true)
file       Output filename (default="-" => stdout)
filename   Synonym for file
noNewline  set nonzero to omit final newline.
coordinates Coordinates: global, centerline, or reference
           (default=c).
```

printfield Prints E or B fields, or writes FieldMap file.

Prints the value of the electromagnetic field components. For type=print, prints one component of the field in a 2-d table. Any coordinate plane can be printed (XY ... ZT). For type=grid or type=cylinder, writes a file in fieldmap format. Global coordinates are used. Units are Tesla for B and MV/meter for E.

NOTE: This command cannot handle time dependency in the output BLFieldMap file, but can in the printout.

Note: if you want to plot field vs position or time, the 'fieldntuple' command is probably better, as it is not limited to the 2-d paper, is easier to use, and lets you use existing NTuple plotting tools. If you just want to test a few points, the 'probefield' command lets you do that interactively.

Named Arguments:

```
type       print, grid, or cylinder.
exit       Set nonzero to exit after printing field
Arguments for type=print:
field      The field to print (Bx,By,Bz,Ex,Ey,Ez,Btot,Etot).
layout     Layout (RowCol) - 2 chars 'AB' each of {xyzt}.
```

x	The starting value of x (mm).
y	The starting value of y (mm).
z	The starting value of z (mm).
t	The starting value of time (ns).
drow	The incr between points in each row (mm ns).
dcol	The incr between points in each column (mm ns).
nrow	The number of rows.
ncol	The number of columns.
Arguments for type=grid:	
file	Filename to write fieldmap to.
comment	Comment for fieldmap.
X0	Initial value of X (mm, default=0).
Y0	Initial value of Y (mm, default=0).
Z0	Initial value of Z (mm, default=0).
nX	Number of points in X.
nY	Number of points in Y.
nZ	Number of points in Z.
dX	Interval in X between points (mm).
dY	Interval in Y between points (mm).
dZ	Interval in Z between points (mm).
Arguments for type=cylinder:	
file	Filename to write fieldmap to.
comment	Comment for fieldmap.
Z0	Initial value of Z (mm, default=0).
nR	Number of points in R.
dR	Interval in R between points (mm).
nZ	Number of points in Z.
dZ	Interval in Z between points (mm).

probefield    Prints B and E fields at specified points.

Intended primarily for debugging. Prints Bx,By,Bz in Tesla, and Ex,Ey,Ez in MegaVolts/meter. Each input line is x,y,z,t separated by spaces or commas; omitted values are set to 0.0.

Field Values are printed after the reference particle is tracked. Only global coordinates are used.

Positional arguments are used as input lines before reading the inputFile.

Named Arguments:

inputFile	Filename for reading list of points (- = stdin)
outputFile	Filename for output (- = stdout)
exit	Set nonzero to exit after printing field

profile        write beam profile information to a file

This command accumulates the moments of the track distributions during the run, and at the end of run prints the mean, sigma, emittance (RMS), alpha, and beta (Twiss parameters) for the tracks. Each z position generates a line in the output file.

Each value in z and zloop can be an expression using double constants and the usual C operators and functions.

NOTE: This computation has two rather serious deficiencies:

- 1) it includes all tracks, with no sigma cut, so a single outlier track can have a large effect on sigmas and emittances.
  - 2) its emittance does not include the vector potential.
- To avoid these deficiencies, it may be better to write NTuples and apply an external program such as ecalc9.

Named Arguments:

z	Comma-separated list of Z positions for profiling (mm)
zloop	Loop in z, first:last:incr (mm)
require	logical expression for cutting (default=true)
particle	Name of particle to profile (default=mu+)
file	Output filename (default=stdout)
filename	Synonym for file
coordinates	Coordinates: centerline or reference (default=c).

randomseed control pseudo random number generator seeds

This randomseed command controls the pseudo random number generator seed at the start of each event. The unnamed argument can be any of (case insensitive):

EventNumber  
None  
Time  
Set 12345  
Now 12345

EventNumber is the default and permits events to be re-run; None does not re-seed the PRNG at each event, and Time is like None after seeding with the time of day in microseconds; Set (Now) seeds the generator immediately with the value of the second argument (a long), and then acts like None.

reference Define a reference particle.

The reference particle is nominally headed in the +Z direction. Multiple reference particles can be defined, at different positions, momenta, particle types, etc. All coordinates are centerline coordinates.

If desired, the referenceMomentum will be tuned to a specific value at a later z position in the beamline by giving values for tuneZ, and tuneMomentum; tolerance can be set if desired.

Normally used in conjunction with a 'beam' command.

Named Arguments:

particle	Reference particle name
beamX	Reference location in X (mm)
beamY	Reference location in Y (mm)
beamZ	Reference location in Z (mm)
beamT	Reference time (ns)
rotation	Rotation of the beam

referenceMomentum	Reference particle momentum (MeV/c)
beamXp	Reference particle Xp (radians)
beamYp	Reference particle Yp (radians)
meanMomentum	Synonym for referenceMomentum
meanXp	Synonym for beamXp.
meanYp	Synonym for beamYp.
tuneZ	Z position for momentum tuning.
tuneMomentum	Desired momentum for momentum tuning.
tolerance	tolerance for momentum tuning (0.001 MeV/c).
noEfield	Set nonzero to make this Tune and Reference particle not respond to E fields (ICOOOL style)
noEloss	Set nonzero to make this Tune and Reference particle not respond to ionization energy loss (ICOOOL style)

reweightprocess modify the cross-section of a physics process.

This command will modify the cross-section of a physics process, modifying the track weights so that weighted histograms give the same statistical result as if the command were not used. Used properly, this can greatly reduce the variance of the result.

Care should be taken to ensure that all regions that should be sampled actually are sampled. For instance, if ratio>1 the interaction length will be reduced, and deep inside an absorber there may be no sampling because no simulated tracks ever get there, even though real tracks will. If ratio<1 then the upstream regions of absorbers will be under sampled; this is usually OK, as the desired result is to increase the sampling deep inside the absorber.

For ratio>>1 this command can be used to examine rare processes. This can induce multiple rare interactions in an event when normally none would be expected; the weights will still correctly correspond to the real interaction, even though the event topologies don't.

This command cannot reweight any continuous process (e.g. multiple scattering, ionization energy loss, etc.) -- it will issue a fatal exception if applied to such a process.

This command should not be applied to other processes that re-weight tracks (e.g. the neutrino command). Indeed it probably won't give the correct weights if any such process applies to the particle (except for itself, it cannot determine the unmodified interaction length of such processes, which is needed to compute the weight).

The re-weighting applies to both PostStep and AtRest processes, but some AtRest processes do not work with this re-weighting; for instance, Decay works properly in PostStep for a moving particle, but not for a stopped one AtRest. This is related to the Geant4 limitation that exactly one process be active AtRest, and exactly one step be taken.

Be sure to test your use of this command for a simple physical situation before believing its results.

Named Arguments:

particle	Comma-separated list of particle patterns ('' => all).
process	Comma-separated list of process patterns.
ratio	Ratio of artificial to real cross-section.

rfdevice Defines an rfdevice (RF cavity)

An rfdevice (RF cavity) is the basic RF element used to construct a linac. The G4beamline convention is that 0degRF is the positive going zero crossing of the electric field, so generally phaseAcc=90 (degRF) is on-crest.

The timeOffset parameter, if set, fixes the overall global absolute timing of the cavity relative to time=0 of the simulation. If unspecified, 0degRF is determined via the timingMethod setting. The default, timingMethod=atZlocal, defines 0degRF such that the test particle will arrive at the timingAtZ=# location then.

For longitudinal cavities, timingMethod=maxEnergyGain emulates how most cavities in linacs have their overall timing determined; while maxX would be appropriate for a horizontal transverse deflecting cavity.

Independent of how 0degRF is found, exactly two of the set of maxGradient, phaseAcc, and one fixed output quantity (fixMomentum, fixEnergyGain, fixTransitTime, fixXdeflection, or fixYdeflection) must be specified to determine the final rfdevice timing. For example, with maxGradient and phaseAcc set, the energy gain would be determined, while if maxGradient and fixEnergyGain were set, the phaseAcc would be determined.

The pipe, walls, and collars are made of copper by default. Pipe, wall, collar, win1, and win2 may be omitted by setting their thickness to 0. Common usage is to set the collar values such that, by placing multiple rfdevices sequentially, the collars form a beam pipe between them.

Note that section 4.4 of the User's Guide has a dimensioned drawing of a pillbox. Due to the presence of an (usually) invisible timing volume, care must be taken when placing objects within an rfdevice.

See the User's Guide for details on how to use this complex element.

Named Arguments (#=cannot be changed in place cmd) (@=Tunable):

maxGradient	The peak gradient of the cavity (MV/m) @
color	The color of the cavity
frequency	The frequency of the cavity (GHz) #
innerLength	The inside length of the cavity (mm) #

innerRadius    The inside radius of the cavity (mm) #  
 pipeThick     The thickness of the pipe wall (mm) #  
 wallThick     The thickness of the cavity walls (mm) #  
 wallMat       The material of all the walls [Cu]  
 irisRadius    The radius of the iris (mm) #  
 collarRadialThick The radial thickness of the collar (mm) #  
 collarThick   The thickness of the collar along z(mm) #  
 win1Thick     The thickness of the central portion of the  
                  windows; zero for no window (mm) #  
 win1OuterRadius The radius of the central portion of the windows  
                  (mm) #  
 win2Thick     The thickness of the outer portion of the windows;  
                  zero for no window (mm) #  
 winMat        The material of the windows [Be].  
 phaseAcc      The reference phase of the cavity (degrees).  
 skinDepth     The skin depth (mm). #  
 timingTolerance Tolerance for timing tuning (ns)  
 maxStep       The maximum stepsize in the element (mm).  
 cavityMaterial Material of cavity volume [Vacuum].  
 timeOffset    Time offset for cavity [set via timingMethod] (ns).  
               @  
 timeIncrement Increment to timeOffset, applied AFTER tuning.  
               (ns).  
 timingMethod   Method for determining the nominal timeOffset {atZ,  
                  maxE, noE, minE, maxT, nomT, minT, maxX, noX, minX,  
                  maxY, noY, minY}.  
 timingAtZ      Local Z location for timing (mm).  
 fixMomentum   Specify total output momentum (MeV/c).  
 fixEnergyGain   Specify energy gain (MeV).  
 fixTransitTime Specify transit time (ns).  
 fixXdeflection Specify local output XZ angle (deg).  
 fixYdeflection Specify local output YZ angle (deg).  
 fixTolerance   Specify allowable error on fixed settings [1.e-3].  
 verbose        Set nonzero to show timing volume and print info  
                  messages [1].  
 fieldMapFile   Filename for BLFieldMap (pillbox if null). #  
 kill           Set nonzero to kill tracks that hit the pipe,  
                  walls, or collars [0].

setdecay      Set lifetime, decay channels, and branching ratios for a  
 particle's decay.

The particle is specified by name as the first positional  
 argument.

The lifetime of the particle can be set, unless it is a  
 short-lived particle (for which lifetime is fixed at 0 -- these  
 are particles like quarks, Zs, and Ws). Units are ns.

Decay channels are specified 'daughter1,daughter2=BR', where the  
 daughter names are separated by commas, and the branching ratio  
 is a value between 0 and 1 (inclusive); the order of daughters  
 does not matter. The sum of all BRs must be 1.0. It is best to  
 use existing channels for the particle, because the code for the  
 decay distribution is retained; new decay channels are given a

default phase-space distribution, which is probably valid only for a 2-body decay of a spin 0 particle. New channels are limited to 4 daughters. Note that all desired decay channels must be listed.

Example to force fast decay (0.1 ns) of pi+ to a positron:  
setdecay pi+ lifetime=0.1 e+,nu\_e=1.0

showmaterial Set the colors for selected materials.

Arguments are of the form 'name=1,1,0', where name is the name of a material, and 1,1,0 is the R,G,B value desired for its color ('' for invisible) Set hideOthers=1 to make all other materials invisible. BEWARE: 'Vacuum' and 'vacuum' are different materials, as are 'Iron' and 'Fe'.

solenoid defines a solenoid (a coil and current)

A solenoid is a coil and a current. If alternate is nonzero, then each placement of the solenoid (or an enclosing group) will flip the sign of current.

Named Arguments (#=cannot be changed in place cmd):

coilName	The name of the coil (must exist) #
current	The current density in the conductor (Amp/mm <sup>2</sup> )
color	The color of the solenoid (''=invisible).
alternate	Set nonzero to alternate sign each placement.
kill	Set nonzero to kill all tracks that hit the coil.
coil	Synonym for coilName. #

spacecharge Beam-frame Green's function space charge computation

This is a space charge computation for bunched beams. It uses a grid in the beam frame to solve Poisson's equation via a Green's function with infinite boundary conditions; the E field is boosted back to the lab frame E and B for tracking.

Macro-particles are used to enable the simulation of larger bunches than can be feasibly simulated as individual particles; the macro-particles have zero radius, but are pro-rated into the nearest eight grid points when placed into the grid. This computation can handle up to about a million macro-particles, but 100,000 is more sensible for all but the simplest physical situations.

The bunch is created from the beam tracks before tracking begins. There is one bunch for each reference particle. Particles in the bunch must be the same particle as the reference, must initially be within {dx,dy,dz} of the reference particle, and when boosted to the reference particle's rest frame must initially have beta < maxBeta.

After boosting the particles to the beam frame, they are placed into the grid, pro-rating to the eight nearest grid points. The grid is dynamically re-sized to keep the 99th percentile of the

particles between 0.5 and 0.67 of the grid size. This maintains a reasonable balance between resolution of grid points within the bunch and covering all of the particles. Particles located at >85% of the grid size do not contribute to the field computation, but are tracked using the field of the rest of the bunch (and other bunches).

The grid has {nx,ny,nz} points; there is a small computational advantage to using powers of 2, but any values >1 can be used. For efficiency, the convolution of the Green's function with the charge grid is performed using FFTs; the grid is doubled in each dimension with the proper symmetry applied to the Green's function, so the cyclical convolution of the FFTs gives the proper potential with infinite boundary conditions.

Outside the grid an approximation is used. An approximation grid is constructed, with the same size of the Poisson grid, but using {nxApprox,nyApprox,nzApprox} points. Particles are placed into this approximation grid, and the mean position is kept as well as the charge. Approximation grid points with less than 1% of the total charge are consolidated with their inner neighbors. The non-zero approximation grid points are treated as point charges when computing the potential outside the grid. For reasonably Gaussian bunches, {7,7,7} are reasonable values for the approximation grid sizes.

The E field in the beam frame is computed via the derivatives of the linear interpolating function using the eight nearest grid points, and is boosted back to the lab frame E and B for tracking by the usual Geant4 routines.

Bunch particles that get destroyed cease contributing to the bunch. As the bunch particles are selected during start-up, no additional particles are ever added to a bunch. This algorithm handles multiple bunches of any particle types.

NOTE: For now, the reference MUST be parallel to the Z axis.

#### Named Arguments:

deltat	Time step (ns).
charge	Charge of macro particles (times particle charge).
nx	Number of grid points in x (65).
ny	Number of grid points in y (65).
nz	Number of grid points in z (65).
dx	Max distance of particle to reference in x (mm)
dy	Max distance of particle to reference in y (mm)
dz	Max distance of particle to reference in z (mm)
nxApprox	# bins in x in approximation (7).
nyApprox	# bins in y in approximation (7).
nzApprox	# bins in z in approximation (7).
maxBeta	Max beta (v/c) of particle in beam frame (0.1).
verbose	Non-zero for verbose prints (0).
ignoreFieldWhenTracking	For testing only (0).
useApproximationOnly	For testing only (0).
fixedGrid	Nonzero prevents re-sizing the grid (0).

percentile	Percentile of charge distribution used for grid sizing (99).
minActive	Minimum # active tracks in bunch; if < 0 is % of initial bunch size (-95).

spacechargelw Lienard-Wiechert space charge computation

This is a space charge computation that uses macro-particles to simulate more particles than is feasible to track individually. Each macro-particle is tracked as a single particle, but its charge is multiplied by the macro-particle charge when computing the field. The radius of the macro-particle is used to avoid the singularity from a point charge; outside the radius the macro-particle is treated as a point charge; inside the radius the point-charge field is multiplied by  $(r/\text{radius})^K$  (radius and K are parameters).

The trajectory of every particle is kept, and when computing the field at a point, the intersection of the point's past lightcone with the trajectory is used to determine the field from the macro-particle; there is a loop over all particles except the one currently being tracked. This computation scales as  $N^2$ , where N is the number of macro-particles; that makes it computationally infeasible for more than a few hundred macro-particles. But for the particles used, it is correct to within the following approximations: a) using macro-particles, b) linearly interpolating between steps, c) omitting the radiation term in the L-W potential.

The fields are computed using eq. 63.8-9 (p 162) of Landau and Lifshitz, Classical Theory of Fields, ignoring the radiation term. The computed fields are used in the usual Geant4 tracking. Particle creation and destruction are handled properly.

This algorithm is primarily intended to test other space charge algorithms.

#### Named Arguments:

deltaT	Time step (ns).
radius	Radius of macro-particles (mm).
charge	Charge of macro-particles (times particle charge).
trackTwice	0=linear extrapolation, 1=track (0)
verbose	Non-zero for verbose prints (0).
K	Exponent for macro-particle density (1).
ignoreFieldWhenTracking	For testing only (0).

sphere           construct a sphere (or section of one)

This is a direct interface to G4Sphere.

#### Named Arguments:

innerRadius	The inside radius of the sphere (mm)
outerRadius	The outer radius of the sphere (mm)
initialPhi	The initial Phi value (deg; 0 for all)
finalPhi	The final Phi value (deg; 360 for all)

initialTheta	The initialTheta of the sphere (deg, 0 for all)
finalTheta	The finalTheta of the sphere (deg, 180 for all)
maxStep	The maximum stepsize in the element (mm)
material	The material of the sphere
color	The color of the sphere (''=invisible)
kill	Set nonzero to kill every track that enters.

start            Define the initial start of centerline coordinates.

If used, this command must come before any other command that puts an element into the world or affects the centerline coordinates (place, beam, corner, cornerarc, and reference commands). This command may not always be needed, but it is needed to eliminate the ambiguities in the global to centerline coordinate transform, and when simulating a ring to ensure that sensible values of the centerline coordinates are used.

Note that the radiusCut is important to reduce or eliminate ambiguities in the global to centerline coordinate transform. It can also be used to 'shield' the beamline to prevent particles from taking unusual paths around the outside of beamline elements.

Named Arguments:

x	The global x position of the start.
y	The global y position of the start.
z	The global z position of the start.
initialZ	The initial centerline z value.
rotation	The initial rotation of the centerline.
radiusCut	The radius cut for the initial segment (mm).
ring	Set nonzero to indicate a ring is present.

test            test random number seeds.

Test

timentuple      Construct an NTuple of tracks at a specified time.

A time NTuple generates an NTuple of every track at a specified global time. It uses a linear interpolation in the step that straddles the required time, so accuracy will suffer for large steps. The NTuple uses centerline coordinates, if available.

The standard NTuple fields are:

x,y,z (mm)
Px,Py,Pz (MeV/c)
t (ns)
PDGid (11=e-, 13=mu-, 22=gamma, 211=pi+, 2212=proton, ...)
EventID (may be inexact above 16,777,215)
TrackID
ParentID (0 => primary particle)
Weight (defaults to 1.0)

The following additional fields are appended for format=Extended, format=asciiExtended, and format=rootExtended:

Bx, By, Bz (Tesla)  
 Ex, Ey, Ez (Megavolts/meter)  
 ProperTime (ns)  
 PathLength (mm)  
 PolX, PolY, PolZ (polarization)  
 InitialKE (MeV when track was created)

Valid Formats (ignore case): ascii bltrackfile dummy for009  
 for009.dat root trackfile Extended asciiExtended rootExtended

Named Arguments (#=cannot be changed in place cmd):  
 time           The global time of the sampling (ns).  
 format        The NTuple format (see above for list).  
 filename      The filename of the NTuple.  
 file          Synonym for filename.  
 require       Expression which must be nonzero to include the  
               track (default=1) #  
 coordinates   Coordinates: global, centerline, or reference  
               (default=c).  
 referenceParticle Set to 1 to include the Reference Particle.

torus           construct a torus.

This is a direct interface to G4Torus. The major radius is in the  
 X-Y plane, with phi=0 along X.

Named Arguments:  
 innerRadius    The inner radius of the torus (mm)  
 outerRadius    The outer radius of the torus (mm)  
 majorRadius    The major radius of the torus (mm)  
 initialPhi     The initial phi around major radius (0 degrees).  
 finalPhi       The final phi around major radius (360 degrees).  
 maxStep        The maximum stepsize in the element (mm)  
 material       The material of the torus  
 color          The color of the torus (''=invisible)  
 kill           Set nonzero to kill every track that enters.

totalenergy    Print total energy deposited in selected volumes.

At end of run, prints the total energy deposited in the selected  
 volumes.

Volume-name patterns are like UNIX filename patterns: e.g.  
 '[AB]\*' matches any name containing an A or a B.

Tracks that are killed have their kinetic energy summed into the  
 volume where they were killed, unless they are killed because  
 they leave the World.

With ancestors=1, energy deposited in matching volumes is added  
 into their ancestors; energy deposited directly into those  
 ancestors is not summed into them unless their names also match.  
 That is, if A is a child of B, but only A matches the list of  
 volume-names, energy deposited into A will be reported in both A  
 and B, but energy deposited directly into B is ignored.

Named Arguments:

volumes            Comma-separated list of Volume-Name patterns (\*)  
ancestors          Set nonzero to sum energy into all ancestor  
                    (enclosing) volumess (0).  
enclosing          Synonym for ancestors.  
filename           Filename to write summary (stdout).  
file                Synonym for filename.

trace              Specifies tracing of tracks.

Generates a separate NTuple for each track, with 1 row per step, unless oneNTuple is nonzero (in which case all tracks are put into a single NTuple). So format=ascii generates one file per track with names generated by the pattern in filename (first %d is replaced by event #, second %d is replaced by trackId); for oneNTuple, the default filename is AllTracks.txt.

Note that without a trace command no traces are generated, so to trace just the tune and reference particles include a trace command with no arguments.

In collective tracking mode, oneNTuple must be nonzero, and the entries will be generated only at collective steps (usually at a specified deltaT).

Unlike other NTuple commands, the require expression applies to entire tracks, not individual entries.

The standard NTuple fields are:

x,y,z (mm)  
Px,Py,Pz (MeV/c)  
t (ns)  
PDGid (11=e-, 13=mu-, 22=gamma, 211=pi+, 2212=proton, ...)  
EventID (may be inexact above 16,777,215)  
TrackID  
ParentID (0 => primary particle)  
Weight (defaults to 1.0)

The trace includes the following fields:

Bx, By, Bz (Tesla)  
Ex, Ey, Ez (Megavolts/meter)

The following additional fields are appended for format=Extended, format=asciiExtended, and format=rootExtended:

ProperTime (ns)  
PathLength (mm)  
PolX, PolY, PolZ (polarization)  
InitialKE (MeV when track was created)

Valid Formats (ignore case): ascii bltrackfile dummy for009  
for009.dat root trackfile Extended asciiExtended rootExtended

Named Arguments (#=cannot be changed in place cmd):

nTrace            Number of tracks to trace.  
format            Format of the NTuple (see above for list).

oneNTuple      Nonzero to put all traces into a single NTuple.  
 primaryOnly    Nonzero to trace only primary tracks.  
 traceTune      Nonzero to trace tune tracks (default=1).  
 filename      Filename (Ev%dTrk%d.txt or AllTracks.txt).  
 file           synonym for filename.  
 require        Expression which must be nonzero to trace the track  
                  (default=1) #  
 coordinates    Coordinates: global, centerline, or reference  
                  (default=c).

trackcolor      Alias for 'particlecolor'.

trackcuts       Specifies per-track cuts.

Applied to each track before tracking, and at each step.

Named Arguments:

kill            List of particles to kill (comma separated).  
 keep            List of particles to keep (kill all others).  
 killSecondaries Set nonzero to kill all secondaries.  
 kineticEnergyCut Minimum K.E. to track (0 MeV).  
 kineticEnergyMax Maximum K.E. to track (infinite MeV).  
 maxTime        Maximum lab time to track (1000000 ns).  
 keepPrimaries Set nonzero to keep tracks with ParentID==0  
                  regardless of other tests.  
 steppingVerbose Set nonzero to print kills (defaults to parameter  
                  value).

tracker         Defines a tracker.

A tracker consists of several trackerplane-s and can fit a track to wire hits and times in the trackerplanes. This is a simple algorithm that does not handle backgrounds or multiple hits. It assumes that every track hits each trackerplane at most once. It is intended to be used to explore resolutions and the effects of survey errors. A tracker is a logical combination of its trackerplanes -- the tracker cannot be placed, but its trackerplanes must be placed into the system.

The fitting algorithm used requires that all of its parameters have comparable scales, so the 'scaleX', 'scaleXp', 'scalePtot', and 'scaleT' arguments should be set to the approximate sigmas of the tracker. They should be within a factor of 10 of the actual values, but closer is better. At the end of fitting tracks a summary is printed that flags each parameter with 'RESCALE' if its scale is too different from its sigma (factor of 5 or more).

NOTE: if the tracker cannot measure Ptot, then 'scalePtot' MUST be set to zero. If the tracker cannot measure T, then scaleT MUST be set to zero. Parameters with zero scales are held fixed at their true-track values.

NOTE: the trackerplane-s of a tracker MUST be placed in the order that particles will hit them; the code does not sort them.

Usually this means that each place command of a trackerplane must have a larger z value than the previous place command. They must also come after the trackerZ value of the tracker command.

A tracker has 3 modes of operation:

true	Each track of each event is written to a TrackerHits NTuple if it hits all trackerplanes; the NTuple includes both the true track values and the individual wire hits for each trackerplane.
fit	A track is fit to the wire hits from a previous run, and the fit track is written to a TrackerFit NTuple (includes true values).
ignore	Any track is ignored.

See the 'trackermode' command to control the mode of trackers.

The TrackerHits NTuple written in 'true' mode contains:

true\_x, true\_y, true\_z, true\_Px, true\_Py, true\_Pz, true\_t,  
true\_PDGid, true\_EventID, true\_TrackID, true\_ParentID,  
true\_Weight, ... plus 1 hit and 1 time per trackerplane.  
(the first 12 are the same as a BLTrackFile.)

The TrackerFit Ntuple written in 'fit' mode contains:

x, y, z, Px, Py, Pz, t, PDGid, EventID, TrackID, ParentID,  
Weight, ChisqPerDF, nDF, nHit, nIter, true\_x, true\_y, true\_z,  
true\_Px,  
true\_Py, true\_Pz, true\_t.  
(the first 12 are from the fit and are the same as a  
BLTrackFile.)

The parameters of the fit are: x, y, dxdz, dydz, Ptot, time. You must ensure that there are at least as many data points as free parameters (scaleT=0 fixes time; scalePtot=0 fixes Ptot). Each trackerplane with nonzero wireSpacing provides a data point; each trackerplane with nonzero sigmaT provides a data point; trackerplanes that measure both provide two data points. You must have enough trackerplanes to meet this requirement, and must set minHits large enough to meet it. The TrackerFit NTuple has a field nDF that gives the number of degrees of freedom for the fit, which is defined as (#DataPoints)-(#FreeParameters); it also has nHit which gives the number of trackerplane-s hit.

Note the tracker can simulate survey errors -- see the 'trackerplane' command for details (each plane can have different errors).

Both the true and the fit tracks are reported at reportZ, which defaults to the Z position of the tracker.

Note that beamlossntuple and newparticlentuple will get many entries per track when used in mode=fit -- the fit runs the track many times through the tracker (30-100, up to maxIter).

NOTE: the trackermode command must precede all tracker commands in the input file, and each tracker command must precede all of

its trackerplane commands. The trackerZ value must also precede all trackerplane-s, but reportZ can be equal to or anywhere after trackerZ.

Note that each trackerplane must have a unique name. This means you should either have a separate trackerplane command for each one (with unique name), or use the rename= argument to the place command (again with unique name). If you use groups for trackerplane-s, use rename=+ in the group.

NOTE: This command does not work properly in collective tracking mode.

#### Named Arguments:

trackerZ	The Z position of the tracker (Centerline, mm).
reportZ	The Z position at which fit tracks are reported (Centerline, mm, default=trackerZ).
scaleX	Scale for X and Y (mm); default=1mm.
scaleXp	Scale for dxdz and dydz (radians), default=0.001.
scalePtot	Scale for Ptot (MeV), default=0. Set to 0.0 if tracker cannot determine momentum.
scaleT	Scale for t (ns), default=0. Set to 0.0 if tracker cannot determine time.
minPz	Minimum Pz for valid tracks (default=10 MeV/c)
minHits	Minimum number of hits for fitting (# planes).
tolerance	Track fitting tolerance (mm) (default=0.01 mm)
maxIter	Maximum iterations during fitting (default=200).
verbose	0=none, 1=result, 2=iterations, 3=detail, default=0.
format	Format of output NTuple.
filename	Filename of output NTuple.
for009	Set nonzero to also output TrackerFit as FOR009.Dat.
file	Synonym for filename.

trackermode Sets mode for all trackers, manages track fitting.

USAGE: trackermode mode [file=...]

mode can be any of:

true	tracks true tracks (normal operation)
fit	fits tracks to previous 'true' output
both	does both true and fit at once

'fit' requires the filename argument to be the output of a previous 'true' run (filename is ignored in other modes); each tracker processes all of its tracks in the file. 'true' mode simply denotes the standard G4beamline operation, and the simulated tracks are taken to be the 'true' tracks of the system; the response of the tracker(s) to these tracks is then simulated in 'fit' mode. 'both' tracks a 'true' event, and then a fit is performed in each tracker for which its first track hit all trackerplane-s.

Note that in 'true' mode every track that hits all trackerplane-s is considered, but in 'both' mode only the first track of the event can be considered.

In fit mode, the filename argument MUST be different from the parameter 'histoFile', because this run must not overwrite the Root file from the previous (true) run.

One trackermode command controls the mode of all trackers. 'true' mode is normal G4beamline operation, and is the same as if no trackermode command was present.

Note that the geometry of the system must not change between a 'true' run and a 'fit' run. You can, however, make small variations in fields to explore how errors in setting them will affect the fitted tracks. The trackerplane command can simulate survey errors.

NOTE: the trackermode command must precede all tracker commands in the input file.

Named Arguments:

filename	Filename to read for fitting tracks.
file	Synonym for filename.

trackerplane Construct a tracker plane.

A trackerplane belongs to a specific tracker, and represents one measuring element of the tracker. While the term 'wire' is used, a trackerplane can model any planar measuring device that measures one dimension using equally spaced detectors that are either on or off (hit or not hit). A trackerplane can be circular (specify radius and possibly innerRadius) or rectangular (specify height and width).

The wires are at angle theta from the vertical, so theta=0 means vertical wires that measure x; theta=90 means horizontal wires that measure y; theta=180 also measures x, but increasing x means decreasing wire #.

Setting wireSpacing=0 means there are no wires, which is useful for a trackerplane that models a scintillator used for track timing (set sigmaTime >= 0 to indicate that).

sigmaTime>0 means this plane can measure the time of the track with that resolution. A Gaussian random number is added to the true track's time at this plane when reading the TrackHit NTuple.

Survey errors can be modeled using the err-arguments -- the values they specify are applied during trackfitting (but not for true tracks). errType: 'fixed' means error values given are the actual values, 'gaussian' means error values are the sigma of a Gaussian random number, 'rect' means error values are the half-width of a uniform random number. The random number is picked before the run begins; the random-number seed is set from the clock so every run will have different random errors.

trackerplane has 3 modes:

true	The trackerplane reports the hit wire and time to the tracker;
fit	The trackerplane reports the Chisq contribution of the fit track distance to the true track's hit wire center, plus survey errors (if any). The track time also contributes to the chisq.
ignore	Any track is ignored.

NOTE: the trackerplane-s of a tracker MUST be placed in the order that particles will hit them; the code does not sort them. Usually this means that each place command of a trackerplane must have a larger z value than the previous place command. They must also come after the trackerZ value of the tracker command.

Named Arguments:

tracker	The tracker to which this plane belongs; REQUIRED.
radius	The radius of the circular tracker plane (mm).
innerRadius	The inner radius of the circular tracker plane (0 mm).
height	The height of the rectangular tracker plane (mm).
width	The width of the rectangular tracker plane (mm).
length	The length of the tracker plane (mm).
theta	Wire angle in X-Y plane (deg). 0=>x, 90=>y...
wireSpacing	Wire spacing (mm).
wireOffset	Wire # 0 offset (mm).
errType	Error type: fixed, gaussian, rect.
errTheta	Error in wire angle (deg).
errSpacing	Error in wire spacing (mm).
errOffset	Error in wire offset (mm).
sigmaTime	Sigma for timing plane (ns), default=-1.
maxStep	The maximum stepsize in the element (mm).
material	The material of the tracker plane.
color	The color of the tracker plane (''=invisible).

trap           construct a solid trapezoid with axis along z.

This is a direct interface to G4Trap. The trapezoid is symmetrical left-right, but upper or lower width can be larger or smaller.

Named Arguments:

height	The height of the solid trapezoid (mm)
upperWidth	The upper width solid trapezoid (mm)
lowerWidth	The lowerWidth of the solid trapezoid (mm)
Xul	X position of upper left corner (mm)
Xur	X position of upper right corner (mm)
Xll	X position of lower left corner (mm)
Xlr	X position of lower right corner (mm)
length	The length of the solid trapezoid (mm)
maxStep	The maximum stepsize in the element (mm)
material	The material of the trapezoid
color	The color of the trapezoid (''=invisible)
kill	Set nonzero to kill every track that enters.

tube           Alias for 'tubs'.

tubs                   construct a tube or cylinder with axis along z.

This is a direct interface to G4Tubs, which can implement a tube or cylinder; either can subtend less than 360 degrees in phi.

Named Arguments:

innerRadius	The inside of the tube, 0.0 for cylinder (mm)
outerRadius	The outer radius of the tube or cylinder (mm)
initialPhi	The initial Phi value (deg; 0 for all)
finalPhi	The final Phi value (deg; 360 for all)
length	The length of the tube or cylinder (mm)
maxStep	The maximum stepsize in the element (mm)
material	The material of the tube or cylinder
color	The color of the tube or cylinder (''=invisible)
kill	Set nonzero to kill every track that enters.
radius	Synonym for outerRadius (mm)

tune                   Tune a variable used as argument to other elements.

This command samples the Tune particle track at z0, samples it again at z1, and varies its tune variable in order to bring the expression to zero. The tune variable is the first positional argument, and can be used in the argument expression(s) for tunable arguments located after z0. Due to the simple solver used, there should be an approximately linear dependence between the tune variable and the expression. This is suitable for tuning the By field of a genericbend or the maxGradient of a pillbox. At each solver step the saved Tune particle is re-started from z0, and when it reaches z1 the next step in the solver is taken.

Note that multiple tune commands can be used together, as long as their z0-z1 regions are properly nested by at least 0.5 mm (i.e. each pair of regions must either not overlap at all, or one must be wholly contained in the other). This command can be complicated to use; see the User's Guide for more description and examples.

Named Arguments:

z0	The starting z position in CL coordinates.
z1	The ending z position in CL coordinates.
initial	Initial value of the variable 'name'
initialStep	Initial step (0 to disable tuning)
step	Synonym for initialStep
start	An expression that must be nonzero to start tuning (default=1)
expr	The expression to tune to zero
tolerance	The tolerance for expr to be zero
maxIter	The maximum number of iterations (10).

usertrackfilter Construct a usertrackfilter that filters tracks via user code.

...

Named Arguments:

radius	The radius of the circular element (mm).
innerRadius	The inner radius of the circular element (0 mm, solid).
height	The height of the rectangular element (mm).
width	The width of the rectangular element (mm).
length	The length of the element (mm).
maxStep	The maximum stepsize in the element (mm).
material	The material of the element.
color	The color of the element (''=invisible).
filterName	Name of the UserTrackFilter.
filter	Synonym for filterName.
init	Initialization string passed to user setup().

virtualdetector Construct a VirtualDetector that generates an NTuple.

A VirtualDetector generates an NTuple of any track when it enters the physical volume of the VirtualDetector. It may be placed via multiple place commands (usually with a 'rename=det#' argument to distinguish the different placements). If material is not specified, it uses the material of the enclosing element. Every placement creates an individual NTuple. For a circular VirtualDetector give radius; for a rectangular one give height and width; length is usually left at 1 mm, but can be set to correspond to the length of a physical detector. The NTuple by default uses centerline coordinates. The NTuple of the virtualdetector can be included in an ntuple command by including a pattern that matches its name in the 'detectors' argument to the ntuple command. Note that must match the name as placed (i.e. includes rename=), not the name given to this command. The noSingles argument may be useful in this case to avoid a huge NTuple of singles (an empty NTuple may be created).

Note that secondary particles created within the virtualdetector will not get an entry until they have taken one step. They are guaranteed to do so.

The standard NTuple fields are:

- x,y,z (mm)
- Px,Py,Pz (MeV/c)
- t (ns)
- PDGid (11=e-, 13=mu-, 22=gamma, 211=pi+, 2212=proton, ...)
- EventID (may be inexact above 16,777,215)
- TrackID
- ParentID (0 => primary particle)
- Weight (defaults to 1.0)

The following additional fields are appended for format=Extended, format=asciiExtended, and format=rootExtended:

- Bx, By, Bz (Tesla)
- Ex, Ey, Ez (Megavolts/meter)
- ProperTime (ns)
- PathLength (mm)
- PolX, PolY, PolZ (polarization)
- InitialKE (MeV when track was created)

Valid Formats (ignore case): ascii bltrackfile dummy for009  
for009.dat root trackfile Extended asciiExtended rootExtended

Named Arguments (#=cannot be changed in place cmd):

radius	The radius of the circular VirtualDetector (mm).
innerRadius	The inner radius of the circular VirtualDetector (0 mm, solid).
height	The height of the rectangular VirtualDetector (mm).
width	The width of the rectangular VirtualDetector (mm).
length	The length of the VirtualDetector (mm).
maxStep	The maximum stepsize in the element (mm).
material	The material of the VirtualDetector.
color	The color of the VirtualDetector (''=invisible).
noSingles	Set to 1 to omit the NTuple for singles.
format	NTuple format: (see above for list).
filename	filename ('' uses name to determine filename)
file	alias for filename
require	Expression which must be nonzero to include the track (default=1) #
referenceParticle	Set to 1 to include the Reference Particle.
coordinates	Coordinates: global, centerline, or reference (default=c).
kill	Set to 1 kill all tracks after entering them into NTuple(s).

zntuple      Generate an NTuple for each of a list of Z positions.

Generates an NTuple like a virtualdetector, but without a physical volume. Tracks are forced to take steps within 2mm surrounding each desired z position, and they are interpolated to the desired z position. Each z position generates a separate NTuple named Z123 (etc.). z accepts a list of z positions, and zloop can generate a set of equally spaced z positions; both can be used.

Each value in z and zloop can be an expression using double constants and the usual C operators and functions.

The standard NTuple fields are:

- x,y,z (mm)
- Px,Py,Pz (MeV/c)
- t (ns)
- PDGid (11=e-, 13=mu-, 22=gamma, 211=pi+, 2212=proton, ...)
- EventID (may be inexact above 16,777,215)
- TrackID
- ParentID (0 => primary particle)
- Weight (defaults to 1.0)

The following additional fields are appended for format=Extended, format=asciiExtended, and format=rootExtended:

- Bx, By, Bz (Tesla)
- Ex, Ey, Ez (Megavolts/meter)
- ProperTime (ns)
- PathLength (mm)

PolX, PolY, PolZ (polarization)  
InitialKE (MeV when track was created)

Valid Formats (ignore case): ascii bltrackfile dummy for009  
for009.dat root trackfile Extended asciiExtended rootExtended

Named Arguments (#=cannot be changed in place cmd):  
z                    Comma-separated list of Z positions (mm)  
zloop                Loop in z, first:last:incr (mm)  
noSingles            Set to 1 to omit the NTuple for singles.  
format                NTuple format (see above for list)  
file                  Output filename ( ' ' uses name to determine  
                      filename)  
filename              Synonym for file  
require               Expression which must be nonzero to include the  
                      track (default=1) #  
referenceParticle    Set to 1 to include the Reference Particle.  
coordinates          Coordinates: global, centerline, or reference  
                      (default=c).

## 6 Examples

On Linux and Mac OS, the examples are found in the “examples” directory under the install directory. On Windows, they are found there also, and a copy is placed into “My Documents\G4beamline Examples” (“My Documents” => “Documents” on Vista).

The annotated output from Example1 is given in Appendix 5.

### **README.txt:**

G4beamline Examples.

Look at the comments in each input file.

Example1.g4bl

A simple first example. Puts a Gaussian beam into 4 detectors.

ExampleN02.g4bl

Implements the Geant4 example/N02.

FieldLines.g4bl

Displays field lines for 4 solenoids.

Study2Cooling.g4bl

Four cells of the Study 2 cooling channel (ca. 2003).

MultipleScattering.g4bl

A simple simulation of multiple scattering and ionization energy loss in various materials.

TungstenTarget.g4bl

An 8 GeV proton beam on a tungsten target.

MICE\_StageVI.g4bl

A simulation of the MICE beamline and cooling channel in Stage VI, from August 2005. Includes field maps for the bending magnets and realistic window profiles for the absorbers. This is the largest and most realistic example.

Idealized\_g-2.g4bl

A simulation of an idealized version of the BNL g-2 experiment. 100% polarized 3.094 GeV/c mu- circle in a 1.4513 T uniform B field, while their spins precess, which affects the energies of their decay e-.

SpaceCharge.g4bl

A parallel-to-point quad triplet with space charge. With a bunch charge of 1E10 the effects of space charge start to become visible; with a bunch charge of 1E12 the focus is unrecognizable.

SampleMovie.g4bl

Example1.g4bl with a movie command added. Use movie.in with g4blmovie.

Movie.in

An input file for g4blmovie, works with SampleMovie.g4bl.

```

triplet.sh
  A script to tune a quad triplet for point-to-point or parallel-to-point
  focusing.
  Uses gminuit, tclsh, wish, and gnuplot.

emittancematch.sh
  A script that demonstrated emittance matching.
  Uses gminuit, tclsh, wish, and gnuplot.

Magnets/ and Windows/
  Magnet field maps and window profiles for MICE_StageVI.g4bl.

```

## 6.1 Example1 – Simple Tracking and Virtualdetectors

```

*      Example1.g4bl - simple first example
*
* Simple example g4beamline input file:
*      a 200 MeV mu+ Gaussian beam is tracked through 1-meter drift
*      spaces into four detectors

# QGSP is the "default" physics use-case for High Energy Physics
# but QGSP_BERT is better for low-energy simulations
physics QGSP_BERT

# the beam is nominally headed in the +Z direction
beam gaussian particle=mu+ nEvents=1000 beamZ=0.0 \
    sigmaX=10.0 sigmaY=10.0 sigmaXp=0.100 sigmaYp=0.100 \
    meanMomentum=200.0 sigmaP=4.0 meanT=0.0 sigmaT=0.0

# a reference particle
reference particle=mu+ referenceMomentum=200

# BeamVis just shows where the beam comes from
box BeamVis width=100.0 height=100.0 length=0.1 material=Vacuum color=1,0,0
place BeamVis z=0

# define the detector
virtualdetector Det radius=1000.0 color=0,1,0

# place four detectors, putting their number into their names
place Det z=1000.0 rename=Det#
place Det z=2000.0 rename=Det#
place Det z=3000.0 rename=Det#
place Det z=4000.0 rename=Det#

```

## 6.2 ExampleN02.g4bl – The Geant4 ExampleN02

```

#      ExampleN02.g4bl - mimic Geant4 examples/novice/N02, APPROXIMATELY
#
physics QGSP
material Pb A=207.19 Z=82 density=11.35
box Target width=50 height=50 length=50 material=Pb color=1,0,0

```

```

material Xe A=131.29 Z=54 density=0.005458
virtualdetector Det1 width=480 height=480 length=100 material=Xe color=1,1,1
virtualdetector Det2 width=1344 height=1344 length=100 material=Xe color=1,1,1
virtualdetector Det3 width=2208 height=2208 length=100 material=Xe color=1,1,1
virtualdetector Det4 width=3072 height=3072 length=100 material=Xe color=1,1,1
virtualdetector Det5 width=3936 height=3936 length=100 material=Xe color=1,1,1
beam gaussian beamZ=-200 sigmaX=0 sigmaY=0 sigmaXp=0 sigmaYp=0 \
    meanMomentum=3824 sigmaP=0 particle=proton nEvents=100
place Target z=0
place Det1 z=800
place Det2 z=1600
place Det3 z=2400
place Det4 z=3200
place Det5 z=4000

```

### 6.3 FieldLines.g4bl

```

*           FieldLines.g4bl - display magnetic field lines.
*
*           Display magnetic field lines for 5 solenoids.

# uncomment the following line to get a white background
#g4ui when=4 "/vis/viewer/set/background 1 1 1"

# display magnetic field lines
fieldlines exit=1 center=0,0,0 nLines=100

# five solenoids (note transparency to show lines inside them)
coil C innerRadius=100 outerRadius=120 length=100
solenoid S coil=C current=100 color=1,0,0,0.3
place S z=0
place S z=500
place S z=-500
place S z=1000
place S z=-1000

# physics is required, even though it is not used.
physics QGSP_BERT

```

### 6.4 Study2Cooling.g4bl

```

*   Study2Cooling.g4bl
*
*   Simple example g4beamline file:
*   There are 4 Study2 cooling cells.
*   This version uses a Gaussian beam

# trace the first 10 tracks
trace nTrace=10

# QGSP is the "default" physics use-case for HEP
physics QGSP disable=Decay

beam gaussian meanMomentum=200 sigmaP=-10 sigmaXp=0.01 sigmaYp=0.01 \
    nEvents=100 beamZ=0

```

```

reference particle=mu+ referenceMomentum=200.0 beamZ=0
trackcuts kineticEnergyCut=50.0 killSecondaries=1

# define the solenoids (use individual Focus solenoids for alternate=1 to work)
coil default material=Cu dR=5.0 dZ=5.0
solenoid default alternate=1 color=1,1,0
coil Focus1 innerRadius=330.0 outerRadius=505.0 length=167.0 maxR=330.0
coil Coupl1 innerRadius=770.0 outerRadius=850.0 length=330.0 maxR=770.0
solenoid USFocus coilName=Focus1 current=75.20
solenoid DSFocus coilName=Focus1 current=-75.20
solenoid Coupl coilName=Coupl1 current=-98.25

# define a detector for the center of each absorber
virtualdetector Det radius=179.9 length=1

# define the absorber with flat Al windows
tubs Win1 outerRadius=180.0 length=0.360 material=Al color=0.0,1.0,0.0
tubs LH2 length=350.0 outerRadius=180.0 color=1.0,0.0,1.0 material=LH2

# place the virtualdetector into the absorber, so its front is in the center
place Det z=0.5 parent=LH2 color=1,1,1

group Abs radius=0
  place Win1
  place LH2
  place Win1
endgroup

# tune the RF Gradient
tune Grad z0=100 z1=11300 initial=15 step=0.1 expr=Pz1-Pz0 tolerance=0.001

# define the pillbox RF cavity, and put 4 of them into a linac
pillbox RF innerLength=466.0 frequency=0.20125 maxGradient=Grad \
  irisRadius=160.0 win1Thick=0.300 win2Thick=0.700 wallThick=5.0 \
  winMat=Be collarThick=5.0 phaseAcc=40.0 maxStep=10.0
group Linac1 radius=0
  place RF rename=RF# copies=4
endgroup

# define one cooling cell
group Cell length=2750.0
  place Abs z=-1033.0
  place USFocus z=-1291.5 rename=Focus
  place DSFocus z=-774.5 rename=Focus
  place Coupl z=342.0 rename=Coupl
  place Linac1 z=342.0 rename=''
endgroup

tubs Spacer length=200 outerRadius=300 material=Vacuum
place Spacer z=100

# place 4 cells
place Cell copies=4 rename=C#

place Spacer

```

## 6.5 MultipleScattering.g4bl

```
*      MultipleScattering.g4bl   TJR   1-FEB-2006   mu+ scattering
#
#      Multiple scattering and ionization energy loss in materials.
#
#      lengths are mm; momentum is MeV/c, density is gm/cm^3

physics QGSP_BIC
beam gaussian particle=mu+ meanMomentum=172 nEvents=100000
trackcuts keep=mu+

#param histoFile=Li
#material Li A=6.941 Z=3 density=0.53
#tubs Target outerRadius=100 material=Li length=12.78 color=1,0,0

## material is Be2 because Be is already defined with slightly different density
#param histoFile=Be
#material Be2 A=9.012182 Z=4 density=1.85
#tubs Target outerRadius=100 material=Be2 length=3.73 color=1,0,0

#param histoFile=H2
#material H2 A=1.00794 Z=1 density=0.0755
#tubs Target outerRadius=100 material=H2 length=159 color=1,0,0

param histoFile=C
material C A=12.011 Z=6 density=1.69
tubs Target outerRadius=100 material=C length=2.5 color=1,0,0

virtualdetector Det radius=1000 color=0,1,0

place Target z=100
place Det z=200
```

## 6.6 TungstenTarget.g4bl

```
#      TungstenTarget.g4bl - 8 GeV proton beam into a tungsten target

# The "default" physics list is QGSP_BERT
physics QGSP_BERT

# the beam is 8 GeV kinetic energy, the mass of a proton is 938.272 MeV/c^2
param M=938.272 KE=8000.0
param P=sqrt((M+KE)*(M+KE)-M*M)

# a zero-emittance beam is unrealistic, but simple; it easily fits through
# a 1 mm hole in the backward detector. It emanates from z=0.
beam gaussian meanMomentum=P nEvents=1000 particle=proton

# the target is a tungsten rod 20 cm long and 1 cm in diameter; make it red
cylinder Target outerRadius=5 length=200 material=W color=1,0,0
place Target z=200

# These three virtualdetector-s catch everything that comes out, except for
```

```

# a 1mm hole for the incoming beam. Note the 0.5 mm clearance at each end of
# the target, and the 201 mm length of the cylinder to match corners.

# This virtualdetector catches what comes out of the target to the back;
# note the hole for the incoming beam; make it yellow
virtualdetector DetBackward innerRadius=0.5 radius=1000 length=1 color=1,1,0
place DetBackward z=99

# This virtualdetector catches what comes out of the target to the side,
# one meter away; make it blue
virtualdetector DetSideways innerRadius=1000 radius=1001 length=201 color=0,0,1
place DetSideways z=200

# This virtualdetector catches what comes out of the target in the forward
# direction; make it green
virtualdetector DetForward radius=1000 length=1 color=0,1,0
place DetForward z=301

```

## 6.7 MICE\_StageVI.g4bl

```

* MICE_StageVI.g4bl 2/5/2006 TJR
#
# This is an old model of MICE Stage VI, as of Aug05.
#
# Command-line parameters (optional - default as follows):
#     first=0 last=1000 viewer=none
#
# NOTES:
# A. dp/p = 2.5% is only 11 MeV/c, which seems WAY too small!
#     (old value was 75 MeV/c; tests show we need +- 17 MeV.c)
#
# The MICE beamline is from bmdata_AUG2105(p3v3).xls
#
# The vacuum windows and pipes are my best GUESS -- this needs to be
# properly designed. APPROXIMATION: the vacuum windows are flat.
#
# The MICE magnetic lattice for the cooling channel is Stage VI, flip,
# beta=42 cm, from UB-09-04-A-TJR (I added the column "Center Z").
#
# The downstream PID detector layout is from cm9_palladino_pidupdate.pdf.
# Except I have used a total of 1" scintillator for TOF2.
#
# APPROXIMATION: The DecaySolenoid map is for 3.7 T, scaled to 4.172 T.
#
# APPROXIMATION: The RF layout is from Study2, not updated.
#
# APPROXIMATION: I have an old spreadsheet for computing the absorber
# and safety window profiles.
#
# APPROXIMATION: the outer shapes of quads and bends are wrong; but there
# are no particles of interest out there, so that's OK.
#
# APPROXIMATION: neglect the varying density of the ISIS beam over
# the target.
#
# APPROXIMATION: the ISIS vacuum and pipe do not implement ISIS at all,

```

```

#           but are OK for our pions.
#
# APPROXIMATION: the vacuum between Q3 and B1 is a box without any pipe
#
# APPROXIMATION: the vacuum between B1 and DecaySolenoid is a box
#           without any pipe

# command-line parameter default values
param -unset first=0 last=10000 viewer=none

# Absorber material, LH2 or Vacuum
param -unset absMaterial=Vacuum

# HistoScope filename from first event #
param histoFile=$first histoUpdate=100000
param steppingFormat="TAG CL STEP VOL MAT PROCESS B"

###
### Beamline tune parameters, AUG05
###
# piMomentumRef is the pion reference momentum at target
# muMomentum is the muon design momentum at center of B2
# muMomentumRef is the muon reference momentum at center of DecaySolenoid
# (account for Eloss in vacuum window, air, and ProtonAbsorber)
# slight change in B2 field and position (compared to Kevin's AUG05), due to
# Eloss in air
param -unset piMomentumRef=440.42 muMomentum=255.24 muMomentumRef=266.0
param -unset DecaySolenoidCurrent=86.02
param -unset Q1=1.155989 Q2=-1.445 Q3=1.005937
param -unset B1=-1.41384 B1z=7918.02
param -unset B2=-0.424270 B2z=15791.47
# Kevin's values: param B1=-1.413935 B2=-0.424573
param -unset Q4=-1.13004 Q5=1.477795 Q6=-0.80022
param -unset Q7=-0.98765 Q8=1.520426 Q9=-1.43098
param -unset DiffuserThickness=7.6

###
### Colors
###
param -unset vacuumPipeColor=0.6,0.6,0.6

###
### general definitions
###
param worldMaterial=Air maxStep=100.0
physics QGSP_BIC
trackcuts keep=proton,pi+,mu+,e+,e- kineticEnergyCut=30 maxTime=1000
particlecolor proton=1,0,0 pi+=0,1,0 mu+=0,0,1 plus=1,0,1 minus=1,1,0 \
    neutral=0,1,1 reference=1,1,1
coil default material=Cu dR=5.0 dZ=5.0 tolerance=0.001
solenoid default color=1,1,0

# Target is 1mm wide and 10mm long:  $(1*\cos(25*\text{deg})+10*\sin(25*\text{deg}))/2 = 2.57$ 
# For now we assume the target dips 2mm into the ISIS beam.
# sigmaP, sigmaXp, and sigmaYp are determined empirically to be larger than
# the actual beamline acceptance.

```

```

# (In the beam command, sigma<0 means a flat distribution with that halfwidth.)
beam gaussian sigmaX=-2.57 sigmaY=-1.0 meanMomentum=$piMomentumRef \
  particle=pi+ sigmaXp=-0.040 sigmaYp=-0.020 sigmaP=-17 \
  firstEvent=$first lastEvent=$last

###
### define the beamline magnets and components
###
virtualdetector TargetDet radius=100 length=1 material=Vacuum noSingles=1
genericquad QuadTypeIV fieldLength=853.4 apertureRadius=101.5 ironRadius=381 \
  ironLength=914 ironColor=0,.6,0 kill=1
# Type I bend shimmed to 200 mm gap
genericbend BendTypeI fieldWidth=660 fieldHeight=200 fieldLength=1038 \
  ironColor=1,0,0 ironWidth=1828 ironHeight=1320 ironLength=990
# PSI solenoid
coil Decay innerRadius=60.0 outerRadius=97.0 length=5000.0 maxR=60.0 \
  mapFile=Magnets/DecaySolenoid
solenoid DecayS coilName=Decay current=$DecaySolenoidCurrent color=1,1,0
tubs SolenoidBody innerRadius=97.0 outerRadius=180 length=5000 kill=1 \
  color=1,1,0
tubs DecayEnd innerRadius=48 outerRadius=180 length=68 material=Fe kill=1 \
  color=1,1,0
# invisible shield to kill junk from B1 that misses the aperture
tubs DecayShield innerRadius=180 outerRadius=1000 length=2 kill=1 \
  color=invisible
# the assembled DecaySolenoid
group DecaySolenoid length=5138 radius=0 material=Vacuum
  place DecayShield z=-2568
  place DecayEnd z=-2534
    place DecayS z=0
  place SolenoidBody z=0
  place DecayEnd z=2534
endgroup
# Proton Absorber.
material H Z=1 A=1.00794 density=0.0838
material C Z=6 A=12.011 density=2.265
material polyethylene density=0.93 C,0.856 H,0.144
# 50mm * cos(15 deg) = 48.29
box ProtonAbsorber width=400 height=400 length=50 material=polyethylene \
  color=1,0,1
# Type QC Quads - includes mirror plates
genericquad QuadTypeQC fieldLength=660 poleTipRadius=171.5 coilRadius=236 \
  coilHalfwidth=57 ironRadius=700 ironLength=1046 ironColor=0,.6,0 \
  kill=1
# Diffuser
tubs Diffuser outerRadius=200 material=Pb length=$DiffuserThickness color=1,0,1
virtualdetector DiffuserDet radius=200 length=1
# Collimator -- place twice, on beam left and right
box Collimator width=600 height=500 length=150 material=Fe color=.8,.8,.8 kill=1

###
### The beamline vacuum components
###
### GUESS: vacuum windows are 0.5mm Al, flat.
### GUESS: 0.5mm Al flat isolation windows at each end of DecaySolenoid.
### NOTE: vacuum3 and vacuum4 intersect B1; this is OK as all are Vacuum.

```

```

tubs ISISVacuum outerRadius=100 length=600 material=Vacuum
tubs vacuumWindow outerRadius=120 length=0.5 material=Al color=$vacuumPipeColor
tubs ISISPipe innerRadius=100 outerRadius=103 length=600 material=Al \
    color=$vacuumPipeColor
tubs vacuum1 outerRadius=100 length=2143 material=Vacuum
tubs pipe1 innerRadius=100 outerRadius=103 length=2143 material=Al \
    color=$vacuumPipeColor
tubs vacuum2 outerRadius=100 length=486 material=Vacuum
tubs pipe2 innerRadius=100 outerRadius=103 length=486 material=Al \
    color=$vacuumPipeColor
box vacuum3 width=300 height=200 length=1643 material=Vacuum
box vacuum4 width=300 height=200 length=1745.2 material=Vacuum

```

```

###
### Lay out the beamline
###

```

```

# reference particle for manually tuning B1, and the RF
reference referenceMomentum=$piMomentumRef particle=pi+ beamZ=0

```

```

place TargetDet z=152 parent=ISISVacuum
place ISISVacuum z=0
place ISISPipe z=0
place vacuumWindow z=300.25
place vacuumWindow z=399.75
place vacuum1 z=1471.5
place pipe1 z=1471.5
place QuadTypeIV rename=Q1 gradient=$Q1 z=3000
place vacuum2 z=3700
place pipe2 z=3700
place QuadTypeIV rename=Q2 gradient=$Q2 z=4400 ironColor=0,0,.6
place vacuum2 z=5100
place pipe2 z=5100
place QuadTypeIV rename=Q3 gradient=$Q3 z=5800
place vacuum3 z=7078.5

```

```

place BendTypeI rename=B1 By=$B1 z=$B1z x=200 rotation=Y30
cornerarc z=7434.2 angle=60 centerRadius=1038

```

```

place vacuum4 z=8722.6
place vacuumWindow z=9595.45
place DecaySolenoid z=12164.7
place vacuumWindow z=14733.95

```

```

# reference particle for manually tuning B2
reference referenceMomentum=$muMomentumRef particle=mu+ beamZ=12164.7

```

```

place ProtonAbsorber z=15070.9

```

```

place BendTypeI rename=B2 By=$B2 z=$B2z x=100 rotation=Y15 fieldMaterial=Air
cornerarc z=15281.7 angle=30 centerRadius=2005.2

```

```

place Collimator z=16760 x=353
place Collimator z=16760 x=-353

```

```

place QuadTypeQC rename=Q4 gradient=$Q4 z=17361.6 fieldMaterial=Air

```

```

place QuadTypeQC rename=Q5 gradient=$Q5 z=18521.6 fieldMaterial=Air
ironColor=0,0,.6
place QuadTypeQC rename=Q6 gradient=$Q6 z=19681.6 fieldMaterial=Air
place QuadTypeQC rename=Q7 gradient=$Q7 z=22993.7 fieldMaterial=Air
place QuadTypeQC rename=Q8 gradient=$Q8 z=24153.7 fieldMaterial=Air
ironColor=0,0,.6
place QuadTypeQC rename=Q9 gradient=$Q9 z=25313.7 fieldMaterial=Air

place Diffuser z=27165.3
place DiffuserDet

### NOTE - the entire cooling channel has moved upstream
### End coil 1.1 upstream face is now z=27169.1

###
### Define and place the MICE upstream particleID elements
###
material scintillator density=1.032 C,0.918 H,0.082
virtualdetector TOF0 height=500 width=500 length=50.8 color=1,1,1 \
    material=scintillator
virtualdetector TOF1 height=480 width=480 length=50.8 color=1,1,1 \
    material=scintillator
place TOF0 z=20537.0
# TOF1 and TOF2 are placed with the cooling channel

material F Z=9 A=18.998 density=1.5
material C6F14 density=1.7 C,0.213 F,0.787
material Si Z=14 A=28.086 density=2.33
material O z=8 A=15.999 density=0.001
material quartz density=2.20 Si,0.467 O,0.533
box Cherenkov1Front width=400 height=400 length=1 color=1,0,0 material=Al
virtualdetector Cherenkov1 width=400 height=400 length=20 color=0,0,1 \
    material=C6F14
box Cherenkov1Window width=400 height=400 length=2 color=0,1,0 material=quartz
box Cherenkov1Light width=550 height=550 length=478 material=Air color=1,1,1
place Cherenkov1Front z=20762.9
place Cherenkov1
place Cherenkov1Window
place Cherenkov1Light

###
### The MICE cooling channel solenoids and IronShield
### all currents give Bz>0; rotation=Y180 is used for Bz<0.
###
# TRD 09/09/2005 Table 4.1-1
coil Focus innerRadius=263 outerRadius=347 length=210 maxR=263
coil Coupl innerRadius=725 outerRadius=841 length=250 maxR=725
coil Match1 innerRadius=255 outerRadius=355 length=202 maxR=255
coil Match2 innerRadius=255 outerRadius=312 length=202 maxR=255
coil End1 innerRadius=255 outerRadius=393 length=120 maxR=255
coil Center innerRadius=255 outerRadius=305 length=1260 maxR=255
coil End2 innerRadius=255 outerRadius=404 length=120 maxR=255
# TRD 09/09/2005 Flip mode, Case 1 Stage VI
solenoid Focus coilName=Focus current=113.95
solenoid Coupl coilName=Coupl current=96.21
solenoid Match1 coilName=Match1 current=56.30
solenoid Match2 coilName=Match2 current=66.79

```

```

solenoid End1 coilName=End1 current=62.80
solenoid Center coilName=Center current=64.44
solenoid End2 coilName=End2 current=67.11
# IronShield is the downstream magnetic shield for the detectors
tubs IronShield innerRadius=250 outerRadius=750 length=100 material=Fe \
    color=0.7,0.7,0.7

###
### The MICE Trackers
###
### APPROXIMATION: each SciFi station is approximated as 2.0mm scintillator
### GUESS: the vacuum window is 1mm Al, flat; placed in the center of End2.
### APPROXIMATION: the tracker pipe ends at the vacuum window (make room for
### the Diffuser)
virtualdetector SciFi radius=150 length=2.0 color=0,0,1 material=scintillator
tubs TrackerPipe innerRadius=200 outerRadius=255 length=2491 material=Al \
    kill=1 color=1,0,1
tubs TrackerWindow outerRadius=200 length=1.0 material=Al color=1,0,0
group Tracker1 material=Vacuum length=2491 radius=0
    place TrackerWindow z=-1245.0
    place SciFi rename=Tracker1a z=-1045.5
    place SciFi rename=Tracker1b z=-595.5
    place SciFi rename=Tracker1c z=-245.5
    place SciFi rename=Tracker1d z=-45.5
    place TrackerPipe z=0
    place SciFi rename=Tracker1e z=54.5
endgroup
group Tracker2 material=Vacuum length=2491 radius=0
    place SciFi rename=Tracker2a z=-54.5
    place TrackerPipe z=0
    place SciFi rename=Tracker2b z=45.5
    place SciFi rename=Tracker2c z=245.5
    place SciFi rename=Tracker2d z=595.5
    place SciFi rename=Tracker2e z=1045.5
    place TrackerWindow z=1245.0
endgroup

###
### The MICE Absorbers
###
### APPROXIMATION: I used an old spreadsheet to compute window profiles
absorber Abs absWindow=Windows/abs300 safetyWindow=Windows/safe320 \
    insideLength=350 absMaterial=$absMaterial windowMaterial=Al \
    safetyDistance=133 color=0,1,0
tubs AFCPipe innerRadius=160 outerRadius=263 length=850 material=Al \
    color=$vacuumPipeColor
group Absorber radius=0 material=Vacuum length=850
    place Abs rename='' z=0
    place AFCPipe z=0
endgroup

###
### The MICE RF Linacs
###
### APPROXIMATION: the drawings in the Mar28 TRD gave no dimensions, so these
### are Study2 RF cavities.
### APPROXIMATION: the RF windows are flat, 0.38mm thick.

```

```

#pillbox Pbox maxGradient=16 frequency=0.20125 innerLength=430 pipeThick=3 \
# wallThick=3 irisRadius=210 collarRadialThick=25 collarThick=12.5 \
# win1Thick=0.38 win1OuterRadius=140 win2Thick=0.38 winMat=Be \
# phaseAcc=40 skinDepth=0.1 timingTolerance=0.001 \
# tuneEnd=RF2C1 goalFactor=1.0 initialStep=0.5 tuneTolerance=0.001
# for empty absorber and no RF
pillbox Pbox maxGradient=0 frequency=0 innerLength=430 pipeThick=3 \
    wallThick=3 irisRadius=210 collarRadialThick=25 collarThick=12.5 \
    win1Thick=0.38 win1OuterRadius=140 win2Thick=0.38 winMat=Be \
    skinDepth=0.1 innerRadius=608.5
tubs RFPipe innerRadius=700 outerRadius=725 length=1900 material=Al \
    color=$vacuumPipeColor
group RF radius=0 material=Vacuum length=1900
    place Pbox z=-690 rename=C1
    place Pbox z=-230 rename=C2
    place RFPipe z=0
    place Pbox z=230 rename=C3
    place Pbox z=690 rename=C4
endgroup

###
### Cooling channel vacuum components
###
tubs vacuumPipeA innerRadius=200 outerRadius=255 length=285 material=Al \
    color=$vacuumPipeColor
group vacuumA radius=0 material=Vacuum
    place vacuumPipeA
endgroup

###
### Lay out the MICE cooling channel
### (use z=0 for the center of Absorber2, Tracker1End2 US edge is at z=29486.0)
###
### AUG05 value, Cooling Channel TRD 09/09/2005
place OFFSET z=33180.1

place TOF1 z=-6611
place IronShield z=-6461

place End2 z=-5951 rotation=Y180 current=0
place Center z=-5201 rotation=Y180 current=0
place Tracker1 z=-4705.5
place End1 z=-4451 rotation=Y180
place Match2 z=-4052 rotation=Y180
place Match1 z=-3611 rotation=Y180
place vacuumA z=-3317.5
place Focus z=-2955 rotation=Y180
place Absorber rename=Abs1 z=-2750
place Focus z=-2545
place Coupl z=-1375
place RF rename=RF1 z=-1375
place Focus z=-205

place Absorber rename=Abs2 z=0

place Focus z=205 rotation=Y180

```

```

place RF rename=RF2 z=1375
place Coupl z=1375 rotation=Y180
place Focus z=2545 rotation=Y180
place Absorber rename=Abs3 z=2750
place Focus z=2955
place vacuumA z=3317.5
place Match1 z=3611
place Match2 z=4052
place End1 z=4451
place Tracker2 z=4705.5
place Center z=5201 current=0
place End2 z=5951 current=0

place IronShield z=6461

###
### Downstream PID counters
### (OFFSET for cooling channel still applies; DS edge of Tracker2End2=6011)
###
### APPROXIMATION: the octagonal Cherenkov2 is approximated as a circle
### APPROXIMATION: an additional window is used in place of the mirror
virtualdetector TOF2 height=480 width=480 length=50.8 color=1,1,1 \
    material=scintillator
material aerogel density=0.2 Si,0.292 O,0.666 H,0.042
tubs Cherenkov2Window outerRadius=400 length=1.0 material=Al color=0,0,1
virtualdetector Cherenkov2 radius=425 length=100 material=aerogel color=0,0,1
tubs Cherenkov2Light outerRadius=640 length=367 material=Air color=1,1,1
material calorimeter density=3.7 Pb,0.85 scintillator,0.15
virtualdetector Calorimeter width=1200 height=1200 length=160 \
    material=calorimeter color=0,1,1

place TOF2 z=6586.4
place Cherenkov2Window z=6710.5
place Cherenkov2
place Cherenkov2Light
# This Cherenkov2Window is standing in for the mirror, so there are 2
place Cherenkov2Window
place Cherenkov2Window
place Calorimeter z=7308

###
### an NTuple for good particles
###
ntuple GoodParticle category=NTuples \
    detectors=TargetDet,TOF0,Cherenkov1,TOF1,SciFi,TOF2,Cherenkov2,Calorimeter

```

## 6.8 Idealized\_g-2.g4bl

```

# Idealized_g-2.g4bl
#
# This is an idealized simulation of the BNL g-2 experiment:
# 3.094 GeV/c mu- are started with polarization=(0,0,1) in a
# perfectly uniform B field of 1.4513 T; their decay electrons
# are put into an NTuple for histogramming. This requires hundreds
# of millions of events, so you probably need to run it on a grid

```

```

# or a cluster. the .root and .out files are named for $first (may
# need to be changed on a grid).

param -unset first=0 last=999999

# set name of .root and .out files
param histoFile=$first
output $histoFile.out

# set steppingFormat to include the polarization
param steppingFormat=GLOBAL,P,KE,POLAR,B

# physics includes spin tracking; keep only mu- and e-
physics QGSP_BERT spinTracking=1
trackcuts keep=mu-,e-

# beam is polarized.
beam gaussian polarization=0,0,1 particle=mu- meanMomentum=3094 \
    firstEvent=$first lastEvent=$last

# NTuple for electrons
newparticlentuple Electrons require=PDGid==11 kill=1

# B is used to expand the world to 100 meters square (more than enough)
box B height=1 width=100000 length=100000
place B z=0 y=500

# F gives the field.
fieldexpr F height=1000 width=100000 length=100000 By=1.4513
place F z=0

```

## 6.9 SpaceCharge.g4bl

```

# SpaceCharge.g4bl
#
# The beamline is a quad triplet tuned to do parallel to point focusing
# at z=6000mm.
#
# To plot sigmaX and -sigmaY using gnuplot, do:
#     plot "profile.txt" using 1:4 with lines, \
#         "profile.txt" using 1:(-$6) with lines
# HistoRoot can also be used to plot profile.txt.
#
# Note the simulation ends when tracks start leaving the world, making
# the number of tracks in the bunch drop below 95% of the initial
# number, around meanZ=10000.
#
# NOTE: the spacecharge command currently has the limitation that the
# reference particle must be parallel to the z axis. The spacechargelw
# command uses no reference and has no such limitation, but is
# computationally limited to about 1,000 macro-particles.
#
# nMP=5000 takes ~ 3 min on my Mac, gives same plot as nMP=50000.
#
# The default deltaT is 0.100 ns, which may well be too large for
# practical use.

```

```

#
# At the default P=200, the effects of space charge start to become
# visible around totalCharge=1E10. At 1E12 the focus is very washed out,
# and at 1E13 is gone. The default bunch has sigmaX=sigmaY=sigmaZ=20mm.
#
# The default quadrupole strengths are scaled with momentum, so the
# focus remains at z=6000 independent of P.
#
# These parameters can be given on the command line to explore variations:
# Bunch parameters:
#     nMP          # macro particles
#     totalCharge  total charge of the bunch (number of e+).
#                  (totalCharge=1 makes spacecharge negligible)
# Beam parameters:
#     P            momentum of the reference and bunch center
#     sigmaP       sigmaP of the initial beam
#     sigmaX,sigmaY,sigmaZ,sigmaXp,sigmaYp - other beam parameters
# spacecharge parameters:
#     deltaT       time step (ns)
#     nGrid        # points in each dimension of the grid
#     nApprox      # points in each dimension of the approximation
#     maxBeta      max beta in reference frame for bunch
#     percentile   percentile of charge distribution used for
#                  grid sizing
#     verbose      verbose print control

param -unset P=200 sigmaP=0 sigmaX=20 sigmaY=20 sigmaZ=20 sigmaXp=0 sigmaYp=0
param -unset deltaT=0.100 nGrid=65 nApprox=7 maxBeta=0.1 percentile=99 verbose=0
param -unset nMP=5000 totalCharge=1
# parallel-to-point focus at z=6000, determined using triplet.sh, scaled by P:
param -unset QF=2.79310*$P/200 QD=-4.95221*$P/200

physics QGSP_BERT disable=Decay
spacecharge deltaT=0.100 charge=$totalCharge/$nMP dx=3*$sigmaX dy=3*$sigmaY \
    dz=3*$sigmaZ verbose=0 deltaT=$deltaT nx=$nGrid ny=$nGrid nz=$nGrid \
    nxApprox=$nApprox nyApprox=$nApprox nzApprox=$nApprox maxBeta=$maxBeta \
    percentile=$percentile verbose=$verbose
collective
param worldMaterial=Vacuum

beam gaussian particle=mu+ nEvents=$nMP meanMomentum=$P sigmaP=$sigmaP \
    sigmaX=$sigmaX sigmaY=$sigmaY sigmaZ=$sigmaZ sigmaXp=$sigmaXp sigmaYp=$sigmaYp
reference particle=mu+ referenceMomentum=$P

genericquad Quad fieldLength=100 ironLength=100 apertureRadius=100 \
    ironRadius=200 kill=1
box Box height=50 width=50 length=1

place Box z=0 rename=Beam color=0,1,0
place Quad z=2000 rename=Q1 gradient=$QF
place Quad z=3000 rename=Q2 gradient=$QD
place Quad z=4000 rename=Q3 gradient=$QF
place Box z=10000 rename=End color=1,0,0

profile file=profile.txt particle=mu+ zloop=0.01:10000:100

```

## 6.10 SampleMovie.g4bl

```
* SampleMovie.g4bl
*
* Just like Example1.g4bl but with movie command added
*
* Simple example g4beamline input file:
* a 200 MeV mu+ Gaussian beam is tracked through 1-meter drift
* spaces into four detectors
movie

# QGSP is the "default" physics use-case for High Energy Physics
# but QGSP_BERT is better for low-energy simulations
physics QGSP_BERT

# the beam is nominally headed in the +Z direction
beam gaussian particle=mu+ nEvents=1000 beamZ=0.0 \
  sigmaX=10.0 sigmaY=10.0 sigmaXp=0.100 sigmaYp=0.100 \
  meanMomentum=200.0 sigmaP=4.0 meanT=0.0 sigmaT=0.0

# a reference particle
reference particle=mu+ referenceMomentum=200

# BeamVis just shows where the beam comes from
box BeamVis width=100.0 height=100.0 length=0.1 material=Vacuum color=1,0,0
place BeamVis z=0

# define the detector
virtualdetector Det radius=1000.0 color=0,1,0

# place four detectors, putting their number into their names
place Det z=1000.0 rename=Det#
place Det z=2000.0 rename=Det#
place Det z=3000.0 rename=Det#
place Det z=4000.0 rename=Det#
```

## 6.11 Movie.in

```
# Movie.in - input file to g4blmovie command
# for SampleMovie.g4bl with movie command -- outputs a Flash movie
setup outputFile=SampleMovie.swf tMin=0.001 tMax=18 duration=10

# e+=red, mu+=blue, others=gray (neutrinos)
particle -11=#FF0000 -13=#0000FF 0=#808080

plot filename=g4beamline.root xmin=-800 xmax=800 ymin=-800 ymax=800 x=x y=y
plot xmin=-200 xmax=200 ymin=-200 ymax=50 x=z y=Pz
row
plot xmin=-800 xmax=800 ymin=-50 ymax=50 x=x y=Px
plot xmin=-800 xmax=800 ymin=-50 ymax=50 x=y y=Py
row height=0.2
sideview xmin=-100 xmax=5000 ymin=-2000 ymax=2000 markerSize=2.0
row height=0.1
time textHeight=12
space title=Example6 textHeight=12
position textHeight=12
```

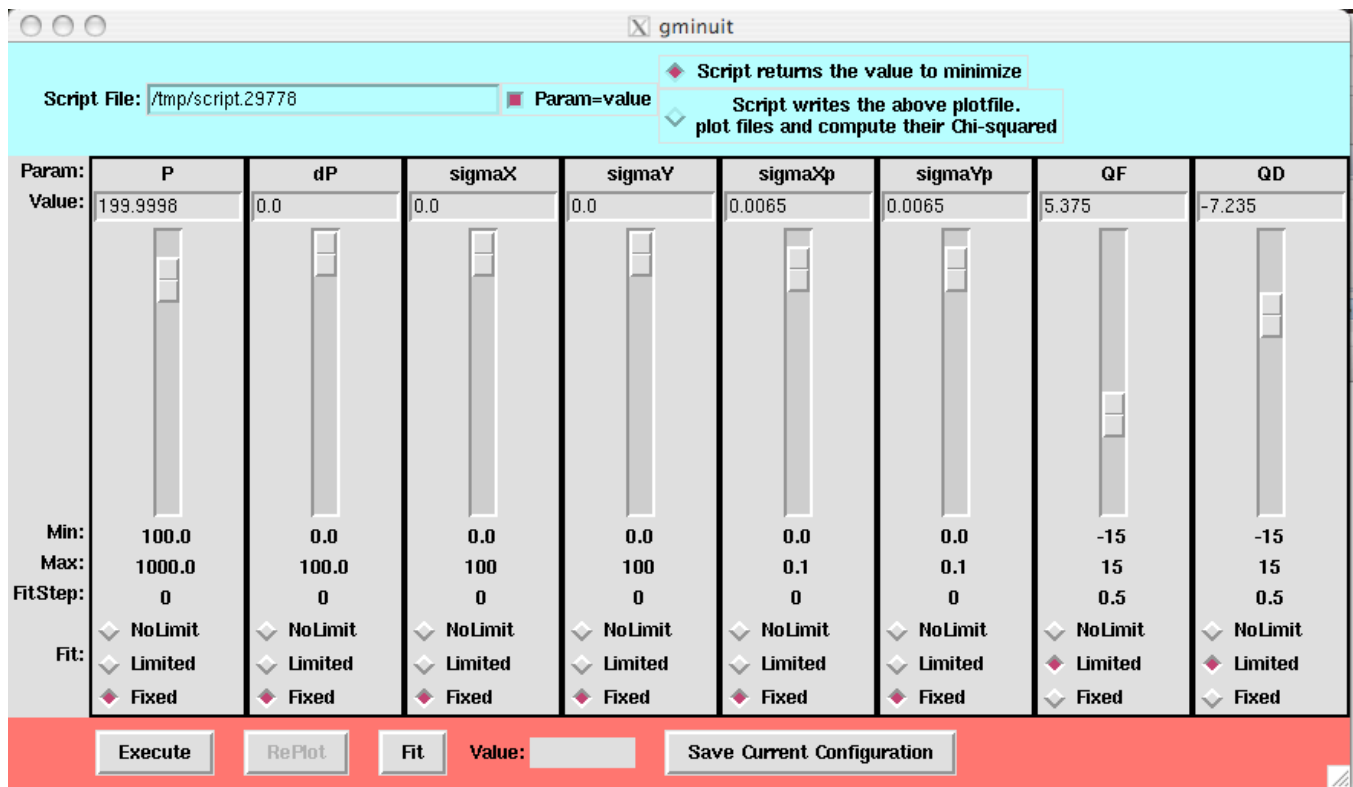
## 6.12 triplet.sh – tune a quad triplet for point-to-point focus

This shell script requires the *bash* shell and the following programs:

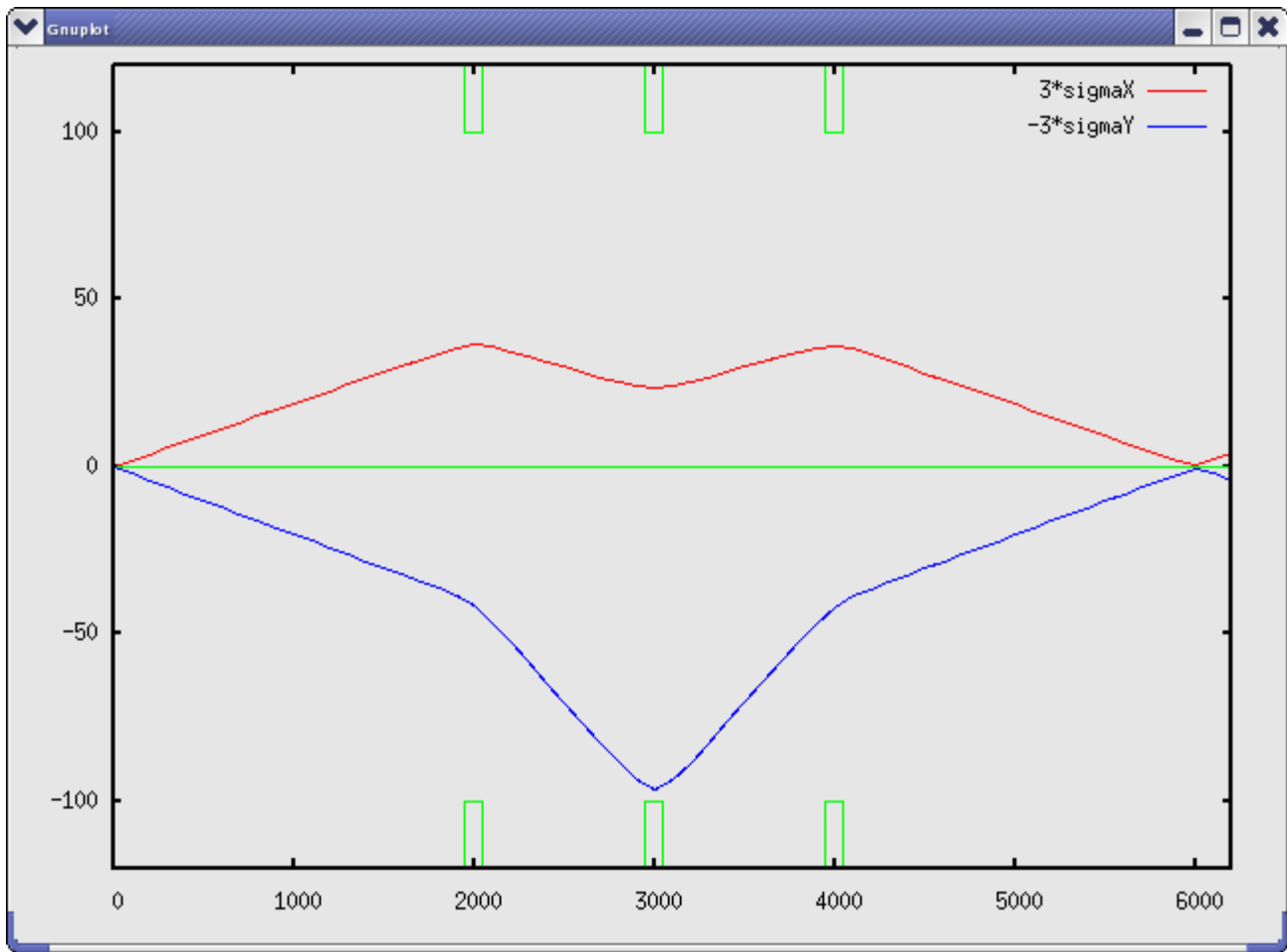
- Gbeamline (<http://g4beamline.muonsinc.com> )
- gminuit (<http://muonsinc.com> ComputerPrograms/gminuit)
- gnuplot (available from your Linux distribution or <http://sourceforge.net> )
- tclsh and wish (available from your Linux distribution)

The gminuit program is configured to have the following parameters: P, dP, sigmaXp, sigmaYp, QF, QD. The first four determine the initial beam (which emanates from a point), and the last two determine the gradients of the three quads. The Chisquared to be minimized is  $(\sigma_X^2 + \sigma_Y^2)$  evaluated at  $Z=6000$ , so the minimum Chisquared will be a point focus. Because of the front-to-back symmetry, the two horizontal-focusing quads have the same gradient.

Gminuit presents the user with a window containing sliders and edit-boxes for the parameters, plus buttons to execute the script once and to fit the parameters (see the gminuit documentation for details):



The script run by gminuit computes the Chisquared to be minimized, and also plots the profile of the beam (as traditional,  $3 \cdot \sigma_X$  is on top and  $-3 \cdot \sigma_Y$  is below the axis, with the apertures of the quads plotted in green):



In practice, a human user can find the best tune more quickly than the minuit minimization engine. This is primarily because the human can watch the effect of parameter changes on the entire profile plot, and use experience and knowledge that minuit does not have.

### 6.13 `emittancematch.sh` – attempt to match a quad triplet into a solenoid

This script is similar to *triplet.sh*, except it adds a solenoid after the quad triplet. This is incomplete, but demonstrates the gnuplot commands necessary to plot the profile, emittance, and beta of the beam.

## 7 Tips and Techniques

These generally useful tips and techniques have been found useful while using G4beamline to simulate several dozen different systems, including suggestions from numerous users.

### 7.1 Getting Help on Using G4beamline

The primary means of obtaining assistance in using G4beamline is our user's forum, [http://www.muonsinc.com/tiki-view\\_forum.php?forumId=1](http://www.muonsinc.com/tiki-view_forum.php?forumId=1). You can also send email to <mailto:inquiries@muonsinc.com>, or to Tom Roberts, the primary author of G4beamline <mailto:tjrob@muonsinc.com>.

Some questions are more about Geant4 than G4beamline. In particular, detailed questions about physics processes and/or physics lists are in this category. For such questions, you can use the previous paragraph, but we will most likely redirect you to the Geant4 forums:

<http://hypernews.slac.stanford.edu/HyperNews/geant4/cindex>. It's quite appropriate for users of G4beamline to join the Geant4 forums as well.

### 7.2 Reporting Bugs in G4beamline

The primary means of reporting bugs in G4beamline is our user's forum, [http://www.muonsinc.com/tiki-view\\_forum.php?forumId=1](http://www.muonsinc.com/tiki-view_forum.php?forumId=1). You can also send email to Tom Roberts, the primary author of G4beamline <mailto:tjrob@muonsinc.com>.

### 7.3 Requesting New Features in G4beamline

The primary means of requesting new features in G4beamline is our user's forum, [http://www.muonsinc.com/tiki-view\\_forum.php?forumId=1](http://www.muonsinc.com/tiki-view_forum.php?forumId=1). You can also send email to Tom Roberts, the primary author of G4beamline <mailto:tjrob@muonsinc.com>.

### 7.4 Getting Help on Individual G4beamline Commands

The source code of each G4beamline command includes help text for the command and its individual arguments; it is printed by the *help* command. This text is the primary source of information about the command, and is copied into this User's Guide and into the help text of the G4beamline GUI. You can obtain help on each command in several ways:

1. From a command line type "*g4bl* --" on one line, and then "*help command*" on a new line in response to the *cmd:* prompt.
2. Run the G4beamline GUI, and if the help text is not displayed, push the *Help* button. You can then scroll around and find the desired command.
3. Look in section 5 of this document.

Note that the definitive source is number 1, so if there is a question about changes (e.g. in different versions of G4beamline), that is the method to use. In particular, the help text in the GUI and in this document can get out of date.

## 7.5 In what Directory should I Work?

In general, it is a poor idea to put your simulations inside the G4beamline install directory – that makes it difficult to update G4beamline; similarly, the G4beamlineExamples directory is not a good place to work (except to run the examples). It is usually a good idea to work in your \$HOME directory, and to make a separate directory for each simulation; this is especially true if you are using the GUI. I have a single directory in my \$HOME called *g4work* under which I create a new directory for each new simulation, using a descriptive name.

## 7.6 Files Common to Multiple Simulations

Many G4beamline commands reference additional files. In some cases there will be a collection of different simulations that share common files, such as field maps, window profiles, etc. It may be convenient to put these files into a single directory called “Common” at the same level as the simulation directories. Then in each input file, reference these common files as “../Common/filename.ext”.

## 7.7 Basic Execution in a Command-Line Environment

Most modern desktop environments permit the user to have multiple command-line windows open simultaneously. This can significantly improve your productivity while developing a new G4beamline simulation, or optimizing parameters of an existing one. Multiple windows are not necessary, and what I describe in parallel can be performed sequentially in a single command-line window; it is noticeably more efficient with multiple windows, however.

Usually when developing a new simulation, or optimizing the parameters or configuration of a simulation, one wants to see histograms or plots from each run before tweaking the *input.file* and running another test run. The idea is to run enough events to get statistically meaningful plots, but to not take too long so one can iterate rapidly through choices. For a simple system being simulated, I can often perform the loop of “edit, simulate, histogram, choose new values” in less than a minute per step. I do this with four windows open simultaneously, all running *bash* in the same directory:

- W1. A command-line window in which I run G4beamline simulations.
- W2. A command-line window in which I run “*g4bl* –“ and to which I type “*help command*” whenever I need help on a command or its arguments.
- W3. A command-line window in which I edit my current *input.file*. I happen to use *vi*, but you should use the editor you are most familiar with.
- W4. A command-line window in which I run *hstoroot*. This can be window 1 if *hstoroot* is put in the background.

It’s best to use a monitor that is big enough to see all four windows at once, but it works well if you can see all of one and parts of the others, so you can rapidly switch windows with a mouse click. The basic sequence is:

1. W3 (editor): If it’s not open already, open *input.file* in the editor and prepare the next version to try. Use W2 (help) for help if necessary. When finished, save the file to disk without exiting the editor.
2. W1 (command prompt): run “*g4bl input.file*” and wait for it to finish. If there was an error, go back to step 1.

3. W4 (historoot): do *File/OpenFile* and open *g4beamline.root* (or whatever file the simulation wrote). Select the desired NTuple and set the expressions to plot (if necessary – the plot configuration will remain from the previous iteration), and then push *CreatePlot*.
4. W4 (historoot): do *File/CloseAllFiles*. This is important to avoid confusion. On Windows this is essential, because if *historoot* has the Root file open, *g4beamline* won't be able to write to it and the next step 2 will fail for that reason.
5. Examine the plot and determine what to do next. Go back to step 1.

By using separate command-line windows, the command-line history in each will permit you to avoid re-typing commands, reducing typos and speeding things up considerably. By keeping *historoot* and the editor open, you avoid the delay of re-configuring them each time.

When constructing a new geometry, plots are usually not needed, and the GUI is better suited to viewing the system easily. So in window 1 I type “*g4blgui input.file*” and push its *Run* button instead of running *g4bl* directly (with the best viewer selected). W4 (historoot) is not used. I find that the GUI is easier to use than the command-line when using visualization (though it works fine to type “*g4bl input.file viewer=best*”, and then “*/run/beamOn 10*” to the Geant4 Idle> prompt).

## 7.8 Basic Execution in a GUI Environment

Execution in a GUI environment, such as Windows, is similar to that in a command-line environment discussed above. The basic idea is to keep windows open so you don't constantly have to re-configure them.

Usually when developing a new simulation, or optimizing the parameters or configuration of a simulation, one wants to see histograms or plots from each run before tweaking the *input.file* and running another test run. The idea is to run enough events to get statistically meaningful plots, but to not take too long so one can iterate rapidly through choices. For a simple system being simulated, I can often perform the loop of “edit, simulate, histogram, choose new values” in less than a minute per step. I do this with four windows open simultaneously, all in the same directory:

1. The G4beamline GUI window.
2. (Optional) The G4beamline User's Guide in a PDF viewer. A PDF viewer makes it easier to scan the text for command names. Can be omitted and the help text of W1 can be used instead.
3. An editor window to edit the current *input.file*.
4. The *historoot* window.

It's best to use a monitor that is big enough to see all four windows at once, but it works well if you can see all of one and parts of the others, so you can rapidly switch windows with a mouse click. The basic sequence is:

1. W3 (editor): If it's not open already, open *input.file* in the editor and prepare the next version to try. Use W2 (help) for help if necessary. When finished, save the file to disk without exiting the editor.
2. W1 (G4beamline GUI): If *input.file* is not already selected, do so with the *Browse* button. Push the *Run* button to run the simulation. If there was an error, go back to step 1.

3. W4 (historoot): do *File/OpenFile* and open *g4beamline.root* (or whatever file the simulation wrote). Select the desired NTuple and push *CreatePlot* (the plot configuration will remain from the previous iteration).
4. W4 (historoot): do *File/CloseAllFiles*. This is important to avoid confusion. On Windows this is essential, because if *historoot* has the *.root* file open, *g4beamline* won't be able to write to it and the next step 2 will fail for that reason.
5. Examine the plot and determine what to do next. Go back to step 1.

## 7.9 Putting Shielding into a Simulation

G4beamline is a realistic simulation program, and just like a real beamline, one must provide appropriate shielding. While obviously there are no radiation issues in a simulation, it is necessary to provide shielding that is sufficient to prevent particles from traveling in unexpected places. This is particularly troublesome for secondary particles from decays and interactions, as they often have sufficient transverse momentum to quickly leave many beamline apertures, and even get outside the outer dimensions of magnets. Particles that go in unexpected places and directions can confuse plots and histograms, and can waste CPU time on irrelevant particles.

Probably the best and simplest approach is to use the *radiusCut* argument to the *start*, *corner*, and *cornerarc* commands. This puts a virtual cylinder around the centerline of the beamline, killing all particles that exceed the current *radiusCut*. You may need to use an unusually large *radiusCut* inside a bending magnet paired with a *cornerarc*, so be sure to check this visually. If everything within the *radiusCut* is either the beam aperture or elements with *kill=1*, then this will ensure that unusual paths outside the nominal beam aperture only extend between elements, which is usually sufficient. Make sure the elements fill the *radiusCut*, or particles could go around them. Note that a *corner* without object or rotation can be used to change the *radiusCut* at a specified value of *z* (centerline coordinates).

Another common approach is to use the *kill=1* argument to bending magnets, quadrupole magnets, beam pipes, etc. This makes them kill any particles that hit their solid parts, providing shielding for many situations. You can add specific objects just for shielding, such as a *box*, *cylinder*, or *tubs*. If you enclose the entire beamline with elements and pipes having *kill=1*, this is probably sufficient.

In many cases, the beam pipes are not of interest in a simulation, and putting them in is often more effort than it is worth. Then you must take care that particles do not take unintended paths. For instance, it is rather common for a particle to take a “shortcut” around a bending magnet, so be sure to check for this. Rings are obviously subject to this. Shielding can be any element with *kill=1*, most often *box* or *tubs* (the latter is particularly convenient as it can be given a hole for the intended beam); it is often convenient to set *color=invisible*, but don't do that until you have looked at it to know it is appropriately sized and placed. A *tubs* of any radius, a length of 0.01 mm or more, and with *kill=1*, is a perfect shield (i.e. kills every particle that hits it).

If you have varying magnetic fields, sometimes they create a trap that can store particles for a long time (1 microsecond is a long time). These are usually uninteresting, but can become CPU hogs. Using the *maxTime* argument to the *trackcuts* command can reduce their effect.

An exception is a Neutrino Factory decay ring, in which muons are stored, but the desired beam is neutrinos. The difficulty is that the far detector may be very far away, which makes the World be quite large, and tracking non-neutrino particles is wasteful. The best solution is to surround the storage ring with beam pipes and magnets having a special material that kills all particles except neutrinos. As the neutrinos spread out, and ones that miss the far detector are uninteresting, one can separate the source and far detector by a *cylinder* with *material=Rock*, and surround it with a thin *tube* with *kill=1*.

## 7.10 How to Debug a Simulation

An important debugging aid is the geometry test performed by G4beamline. The results appear in the output just before tracking. In the real world, two objects cannot occupy the same space; the simulated world does not have this restriction, and input-file errors can generate unphysical and invalid overlaps. The Geant4 optimization of tracking requires a strict hierarchical arrangement of volumes. Usually any geometry errors listed should be corrected in your input file before believing the simulation results; in some cases this is difficult or impossible, and some exceptions are listed in section 8.2 on Advanced Topics.

The most common debugging technique is to add *steppingVerbose=1* to the command line (or the Parameters field of the GUI). This causes G4beamline to print (to stdout) the track variables at every step. That usually lets you figure out what is going wrong. Note that by setting *steppingFormat* you can control what track variables are printed (see the help for details) – this can be very useful when debugging E and B fields, or *spinTracking*. In most cases it is best to widen your display’s window before running, so lines are not wrapped – 140 columns is usually sufficient.

Sometimes the problems are subtle, and only rare events look crazy. If you can select those “crazy” events with sliders in *historoot*, then you can set its *EventID* field to cause it to write the *histo\_event.txt* file. You can then check the *HistoRootEvents* checkbox in the GUI, or put an *eventcuts* command into your *input.file*. Running with a viewer will then permit you to visualize just the “crazy” events – seeing them will often let you figure out what’s wrong.

Note that G4beamline simulations are quite realistic. This means that pions and muons will decay, particles will interact in material objects, and tracks will propagate wherever they can go (even if you don’t expect it). Decay neutrinos can look “crazy” if you’re not expecting them (so cut on *PDGid* in the plots, or eliminate them at the source with a *trackcuts* in your *input.file*). You may be surprised at the delta-rays (low energy electrons) produced in vacuum windows and other materials (eliminate them with a *trackcuts* command with *kineticEnergyCut=1* or higher, or perhaps *kill=e-*). Tracks can occasionally go in the most amazing places, and you need to install shielding just as in a real accelerator (use the *radiusCut* as described above, or add a *box* or *tubs* with *kill=1* and place it appropriately; when complete you can use *color=invisible* to avoid cluttering your visualizations). It is often appropriate to surround the beam line with beam pipes having *kill=1* – this avoids spending CPU time on unwanted tracks, and prevents them from taking “shortcuts” and re-appearing as crazy events in plots; add *kill=1* to quads and bends, etc.

In my experience, the most common errors in G4beamline simulations and interpreting results are:

1. Failure to cut on *PDGid* in plots and histograms, and thus confusing other particles for the desired ones.

2. Failure to put shielding in appropriate places and thus having particles go in unexpected places, causing “crazy” events in plots and other confusion.
3. Mistakes in the input file that cause invalid overlaps of objects.
4. Mistakes in the input file that unintentionally omit objects or put them in the wrong place.
5. Inappropriate values of tracking parameters (see section 3) that cause Geant4 to optimize tracking in an unsuitable or insufficiently accurate manner.

## 7.11 Geant4 Commands

Geant4 has an internal command processor with a large number of commands. For the most part, G4beamline users don’t need to use these commands, as G4beamline controls all aspects of the simulation. Nevertheless, there are situations for which a user might want to issue a Geant4 command. There are two rather different ways to do this:

1. Simply put the Geant4 command into the *input.file*, putting its initial “/” in column 1. Such commands are issued when they are read – that happens before the system is constructed, and for many commands that is too early.
2. Use the *g4ui* command. The *when* argument controls when the command is issued, and all values are after the system is constructed.

The Geant4 command interpreter is quirky, but there is a way to obtain a list of all its commands: */control/manual*. This command can use either method above.

The most common reason to issue a Geant4 command is to set some verbose level, for debugging or investigating some aspect of Geant4 (most especially its physics processes). G4beamline sets all verbose levels to 0 initially. Here is the list of commands that G4beamline runs to do that:

```
/control/verbose 0
/run/verbose 0
/event/verbose 0
/tracking/verbose 0
/hits/verbose 0
/material/verbose 0
/process/setVerbose 0 all
/process/verbose 0
/process/eLoss/verbose 0
```

Another reason to use a Geant4 command is to control a viewer. Use “*g4ui when=4*” for that. The selected viewer’s initialization commands (from *viewer.def*) will be executed before such commands.

## 7.12 Obtaining Plots and Histograms

There are a number of different ways to generate plots and histograms from G4beamline output files. I generally find *HistoRoot* to be most convenient, but each user should use the method they are most comfortable with. Both *HistoRoot* and PAW can fit functions to histograms or plots; excel and some other spreadsheets can do so, perhaps with more overhead.

## General Remarks

In most cases, you must cut on *PDGid* when generating a plot or histogram (*PDGid* is the Particle Data Group's ID for the particle of a track, so this is selecting the type of particle to plot). G4beamline simulations are realistic, and particles will decay and interact, generating secondary particles. Without such a cut, secondary particles can greatly confuse the interpretation of a plot. *HistoRoot* has a specific feature to display the particle types: put "PDGid" into the "Particle Type" field, and whenever the NTuple is scanned to generate the plot, a list of the PDGid-s contained within the slider cuts will be printed, with names for common particles.

To display a bundle of tracks, put "*trace nTrace=100 format=ascii oneNTuple=1*" into your input.file, and then use *gnuplot* to plot the *AllTracks.txt* file that G4beamline wrote. *gnuplot* omits the back-trace line between tracks because the *trace* command puts a blank line between tracks, specifically for this. *gnuplot* is available for Linux as part of the distribution, for Mac OS via Fink, and for Windows via <http://www.gnuplot.info/>.

## HistoRoot

*HistoRoot* requires *Root* to be installed, and for best results, install the version of *Root* available from <http://historoot.muonsinc.com>. By default, all NTuples generated by G4beamline are placed into a single *Root* file. Note, however, that *HistoRoot* can also read the ASCII NTuple files written by G4beamline (and other programs – see the *HistoRoot* Help for the file formats accepted). Generating plots and histograms is straightforward, and the help for *HistoRoot* should permit you to get started right away. You can quickly and easily plot arbitrary expressions of the NTuple's fields, with cuts applied based on expressions; there are four sliders which implement cuts that affect the plot in real time. For more complicated analyses (e.g. computing masses of multi-particle states), *Root* can be used as an analysis platform; this is advanced usage and is beyond the scope of this tip.

## Gnuplot

*Gnuplot* can directly read the ASCII NTuples written by G4beamline (add *format=ascii* to the command). Note that *Gnuplot* does not generate histograms; you'll have to use another program to do that. *Gnuplot* is especially good at displaying multiple trajectories from "*trace format=ascii oneNTuple=1*". *gnuplot* is available for Linux as part of the distribution, for Mac OS via Fink, and for Windows via <http://www.gnuplot.info/>.

## Excel and other Spreadsheet Programs

Excel and most other spreadsheet programs can read the ASCII NTuples written by G4beamline (add *format=ascii* to the command). Each line is a row of the NTuple with tabs separating the columns; the first 2-3 lines give comments, column names, and column units (optional).

## PAW

PAW can read the ASCII NTuples written by G4beamline (add *format=ascii* to the command), possibly after a format conversion.

## 7.13 Obtaining Pictures of the System and Events

G4beamline is a complicated and flexible program, and like all such programs it is subject to "garbage in, garbage out". So it is important to check and verify that your *input.file* actually represents the

simulation you want, and not some other (perhaps unphysical) system. An important tool for this is the visualization implemented by G4beamline – many errors can be found simply by looking at the system to be simulated.

In the command-line environment, you need to have an active X-Windows environment (i.e. the DISPLAY variable must be set to a valid display server). On Windows this is automatic; on Linux it is automatic as long as you run X windows (i.e. bring up a GUI); on a Mac you may need to run the X11 application and use an xterm window (in Leopard and later OSs you can simply use a normal Terminal window, as the X11 application is launched automatically when needed). Then all you need do is append “viewer=best” to your *g4bl* command, and the OpenInventor viewer will be displayed. Other viewers can be used; get their names by using viewer=best and looking near the end of the output for a list of supported viewers. In the command-line environment, you need to type the Geant4 command “/run/beamOn 10” to run 10 events and display them and the system.

In the GUI environment, simply select Viewer:best. Each run will be displayed in the viewer as a single image, so setting events/run determines how many events are shown, and selecting #runs is how many different runs (pictures) will be made. In the OpenInventor viewer, select File/Escape to quit this image, simulate the next run, and display its image. Other viewers can be selected by selecting and setting *Other*.

To save the image into a file, some viewers have a File menu item that can save in various formats (e.g. OpenInventor can save as PostScript). In any viewer, you can take a screenshot of the viewer window and save it to a file:

- Windows: Ctrl-PrtScrn followed by running a program to save the Clipboard in a graphics format (e.g. Paint)
- Linux: a program like ksnapshot
- Mac OS X: the Grab utility (menu: Capture), the Preview utility (menu: File/Grab), or type shift-command-4 and select the desired region of the screen

Note that screenshots all save a bitmap, but for viewers that implement it the File/Save menu item can generate a vector file that is smaller and scales better.

## 7.14 Warning and Error messages – Which Ones can be ignored

G4beamline errors are all unified within the Geant4 G4Exception framework. This gives them a common look, and ensures that in a MPI environment they are all tabulated in rank 0 and summarized at the end of the job.

Errors in the input file generate an error message immediately while reading and listing the input file. They must be fixed before the simulation will run. These are such things as syntax errors, invalid parameter names, invalid arguments to commands, etc.

Errors and warnings generated during execution are handled by the *G4Exception* function; they are printed as a 6-line message starting and ending with a row of asterisks – that makes them stand out visually in a long printout. Those that are issued by a routine beginning with “G4” came from Geant4 code, while those issued by a routine beginning with “BL” came from G4beamline code; those that are

issued by a command names are also from G4beamline code. It is important to distinguish these two cases, as how you obtain help on them may differ.

The *severity* of the message is a strong indicator of whether it can be ignored, or indicates a serious problem. Note that a Warning issued by many events probably means the simulation cannot be trusted. Warnings on a tiny fraction of the events can probably be safely ignored (unless they are related to specific rare events of interest). Even Errors that abort the track or event might well be ignorable if they occur only rarely. Note that G4beamline suppresses many similar exception printouts, so look at the Exception Summary at the end of the output to see how many actually occurred.

Errors issued by G4beamline code can be discussed in our user's forum, [http://www.muonsinc.com/tiki-view\\_forum.php?forumId=1](http://www.muonsinc.com/tiki-view_forum.php?forumId=1). You can also send email to <mailto:inquiries@muonsinc.com>, or to Tom Roberts, the primary author of G4beamline <mailto:tjrob@muonsinc.com>.

Errors issued by Geant4 code can be discussed in the Geant4 forums (linked to the Geant4 home page, <http://geant4.cern.ch>). You can also discuss them as in the previous paragraph (choose one method, not both).

See also section 2.8.4 on troubleshooting visualization problems. A reasonably complete list of G4beamline error messages is given in Appendix 7. Here are some notes on specific exceptions:

### **Stuck Track**

If you track very low-energy tracks ( $< 100$  eV), and have materials in your simulated world, then you will probably see a number of “Stuck Track” exceptions that kill the track. This occurs because very low energy tracks can get stuck on the surface, taking many extremely small steps ( $< 1$  micron). Such tracks would really get absorbed into the material, so killing the track is a reasonable thing to do. Such error messages can usually be ignored, unless they dominate your simulation.

### **Large Primary TrackID**

This exception has a specific cause and cure. Remember that tracking in Geant4 (and thus G4beamline) will often generate secondary tracks, and these must each be assigned a TrackID. To minimize the collision of multiple tracks within an event having identical TrackID-s, secondary tracks by default have TrackIDs starting with 1001 and incrementing thereafter. If the input beam file has a TrackID  $\geq 1001$ , then confusion can result, so this exception is issued. You could edit your beam file to not have such large TrackID-s, or you can add the following to the beam command that reads the file: `secondaryTrackID=2001`. This avoids the overlap, and thus the exception; larger values can be used if necessary.

## **7.15 Secondary Tracks and Particles**

Unlike most accelerator physics programs, G4beamline includes particle decays and interactions of particles with matter. This means that during a simulation new particle tracks can, and usually will, be created. For instance, whenever the beam goes through a vacuum window, delta-rays (low-energy  $e^-$ ) will be produced; a beam of unstable particles will generate decay products, etc. In some simulations such secondaries are essential (e.g. for a muon beam produced by pion decay), but in others these

secondary particles are uninteresting and are merely a nuisance (as, perhaps, are the neutrinos associated with that muon beam). G4beamline has several mechanisms to let you deal with secondaries:

- *physics* – doStochastics=0 will turn off all stochastic processes, which has major implications, but this implicitly inhibits the creation of secondaries
- *physics* – disable=process will turn off specific physics processes
- *trackcuts* – killSecondaries=1 will kill all secondaries; keep=list will keep only those particles, killing all others; kill=list will kill those particles
- *particlecolor* – assigns colors to different particles for the viewers

Note that when generating histograms and plots, you usually need to cut on PDGid (the Particle Data Group’s assigned ID for the particle of the track). It is all too easy to look at a histogram, see an outlier, and wonder what is happening – in many cases the outlier is merely a secondary track from a decay or interaction, and is unrelated to the phenomenon of interest. So be warned.

*HistoRoot* has a specific feature to display the particle types: Put “PDGid” into the “Particle Type” field, and whenever the NTuple is scanned to generate the plot, a list of the PDGid-s contained within the slider cuts will be printed, with names for common particles.

Here is a list of common (stable) particle PDGid-s (negative for anti-particle):

PDGid	Particle		PDGid	Particle		PDGid	Particle
11	$e^-$		12	$\nu_e$		13	$\mu^-$
14	$\nu_\mu$		16	$\nu_\tau$		22	$\gamma$
111	$\pi^0$		211	$\pi^+$		311	$K^0$
321	$K^+$		2112	n		2212	p

A much more complete list of particle IDs is in Appendix 6.

## 7.16 Finding Example Input Files using the XXX command

First look in the examples directory of the distribution; see section 6. There is at least one test for every G4beamline command – look in the *test* directory under the G4beamline installation directory. You can also send an inquiry our user’s forum, [http://www.muonsinc.com/tiki-view\\_forum.php?forumId=1](http://www.muonsinc.com/tiki-view_forum.php?forumId=1). You can also send email to <mailto:inquiries@muonsinc.com>, or to Tom Roberts, the primary author of G4beamline <mailto:tjrob@muonsinc.com>.

## 7.17 Parameterizing the Input File

This is a general technique applicable to many different situations.

Parameters in G4beamline are names that represent strings. Whenever *\$name* appears in an argument to a command, the current value of the parameter *name* will be substituted. Note this is not a general macro substitution, and it only occurs in the values of command arguments. In addition, for all real- and int-valued arguments and in the *param* command, argument values that are numerical expressions are evaluated after *\$name* substitution; the standard C math functions can be used. See section 4.3 on expressions and section 4.3 on parameters.

Parameters can be defined by the *param* command, on the command line, and in the *Parameters:* field of the GUI. These last two methods permit different runs using the same input.file to be different. In all cases a parameter is set with the syntax name=value, without spaces; the value can be enclosed in single- or double-quotes to include whitespace. These examples will use the command-line interface, but the *Parameters* field of the GUI can be used in the same way.

The basic idea is to use a parameter rather than hard-coding a specific value into the input.file. Usually one wants to provide a default value for the parameter, so it is never undefined. As an example, imagine you want to compare the effects of several different target materials, all with the same size. The input.file could look like this:

```
param -unset MATERIAL=Cu
...
box Target material=$MATERIAL ...
...
```

Note the initial “param -unset” command – this provides a default value for the parameter. Commands to run this simulation for several materials would be:

```
g4bl input.file MATERIAL=Cu
g4bl input.file MATERIAL=W
g4bl input.file MATERIAL=Be
```

Note that in the previous example the Root output file will be overwritten each time, so one must be careful to obtain all desired output from one run before starting the next. A better input.file will use different output files for each run:

```
param -unset MATERIAL=Cu
param histoFile=$MATERIAL
output $histoFile
...
box Target material=$MATERIAL ...
...
```

The second *param* command sets the name of the Root output file to include the material (this command cannot be combined with the first *param* command and must not have *-unset*). The *output* command then re-directs stdout and stderr to files “Cu.out”, “W.out”, “Be.out”, etc. It’s best to put only *param* commands before the *output* command, so all output gets re-directed into the output file.

This can obviously be extended to as many different parameters as desired. It is useful to put them all into the *histoFile* value, perhaps separated by commas.

By putting the parameter values into the output filenames, multiple runs can be executed in parallel, in the same directory. This is especially useful on a system with multiple cores, or on a cluster or grid.

## 7.18 Setting Fields of Magnets

If you have a specified field for a solenoid, you need to convert that into a current (density), as that is the argument of the *solenoid* command that determines the field. You can compute the ampere-turns required, but that is both time consuming and error prone. A better way is to use the *probedfield* command with its input set to a file containing the global coordinates of the solenoid’s center. Just guess

a value for the current and run the simulation; from the printed value you can take the appropriate ratio and quickly determine the current required to give the desired field. You can also use the *printfield* command for this.

If, instead, you have a specified current for a given magnet, then you will need the magnet's specifications to determine the field corresponding to that current. Except for a solenoid without iron, there is no simple way to relate field and current. If you have a drawing of the magnet, you could in principle construct a model of the magnet using an EM modeling tool (Tosca, Opera3D, Ansys, ...); that is usually a major effort – the drawing probably gives a value for the integral of  $B \, dl$ , which should be enough for you to compute the field.

## 7.19 Tuning Bending Magnets

G4beamline is a realistic simulation program, and like a real beamline, the bending magnets in a simulated beamline must be tuned for position, orientation, and field (current). To model a real beamline that has the corners of the beam centerline marked on the floor, a *corner* or *cornerarc* command will be used to model each corner. This implies that the corners are fixed and the bending magnet must be positioned and tuned appropriately. The standard criterion is that a reference particle coming into the magnet along the centerline of the beam should exit the magnet along the centerline, both measured outside the fringe fields of the magnet. If there are solenoids or other magnets near the bending magnet such that their fringe fields overlap, then the tuning of the bending magnet may depend on their settings. Note that if quadrupoles are used to steer the beam, these remarks apply to them, too.

See also section 2.11 (Tuning the Beamline).

To begin, make sure you have a *reference* command that puts a reference particle down the centerline into the bending magnet. This implies that if you have multiple bends in your beamline, you should start tuning with the first bend, and proceed downstream.

The usual orientation for a rectangular bending magnet is with its front and rear faces angled at half the total bending angle to the centerlines. It must be placed a short distance behind the corner in order that its fringe fields are accounted for, and must be placed a bit to the outside of the bend so the two centerlines are equidistant from the left and right sides of the aperture.

The usual orientation for a sector bending magnet is with its front and rear faces perpendicular to their centerlines.

Other orientations are possible, and can be used to affect the vertical focusing of the bending magnet. The beamline designer will specify the appropriate angle of the magnet relative to the centerlines.

G4beamline can automatically tune the field of a bending magnet, but cannot tune its position. So the best approach is to put the field tuning into the input file and then tune the position manually – this reduces a 3-parameter manual problem to two parameters, which goes much faster. Once the position is correct for one reference momentum, other reference momenta can be handled by simply re-tuning the field (or manually adjusting it). Here is an example for a 60-degree left bend:

```
Param -unset Zoffset=0 Xoffset=0
```

```
tune B1Field z0=1000 z1=3000 initial=-2.000 step=0.05 \
    expr=Pxl/Pz1 tolerance=0.000001
place B1 z=2000+$Zoffset x=$Xoffset rotation=Y30 By=B1Field
corner z=2000 angle=60
```

The basic idea is that the *tune* command will vary the value of B1Field and re-track the tune particle from  $z_0=1000$  until it is parallel to the centerline at  $z_1=3000$ , within 1 microradian (tolerance). The user will manually vary Zoffset and Xoffset until the magnet is properly placed so that reference particle is on the centerline at  $z_1=3000$  and is equidistant from the horizontal edges of its aperture (these are not independent, due to the rotation of the magnet and the presence of its fringe field; both will be reasonably close to 0, within ~200 mm or so). Note the initial value in the *tune* command must be close enough so the initial tune particle actually reaches  $z_1=3000$  (i.e. does not hit any aperture beforehand).

The idea is to put “Zoffset=0 Xoffset=0 steppingVerbose=1” onto the command line (or Parameters in the GUI), and re-run the simulation while varying their values until the reference particle is on the centerline at  $z_1=3000$ . To speed this up, it is useful to temporarily set the *beam* command(s) to run one event (nEvents=1). To avoid laborious scanning of a long output file with many steps, it is useful to use the command-line program and pipe it into *grep* with an appropriate pattern to select a step near or at  $z=3000$  (because step-lengths can vary, it may help to temporarily place an object at  $z=3000$  and grep for it). NOTE: be sure to look at the reference particle, and not any tune or beam particle.

First keep Xoffset=0 and vary Zoffset. You will quickly learn how changes in value relate to distance from the centerline. This usually executes in about 3-5 seconds, and you ought to be able to find the required value of Zoffset in a few minutes. Using the command-line history in your shell greatly speeds this up. To tune Xoffset you must add this command to make the reference particle visible and white: “trackcolor reference=1,1,1” (I put it just after the beam command; you can leave it in). Once you have the value for Zoffset, then add viewer=best to the end of the command and re-run it, typing “/run/beamOn 1” to the Geant4 Idle> prompt (as indicated by the comments preceding it). In OpenInventor, select wireframe mode (right click, DrawStyles/StillDrawStyle/wireframe). Now rotate the image using the mouse until you can see the white reference track inside the bending magnet (ignore the beam track(s) that are other colors); zoom in if necessary, and use a ruler or judge by eye whether or not it is centered left-right. If it is not centered, guess how much change in Xoffset is needed. Then go back to tuning Zoffset with this new value (no viewer).

Once you have values for Zoffset and Xoffset, edit the input.file and put them in.

If you want to handle beams with different momenta, you can parameterize the momentum and B1Field like this (assuming you initially tuned at 200 MeV/c and B1Field for 200 MeV/c was 2.10525 Tesla):

```
param -unset MOMENTUM=200
beam Gaussian meanMomentum=$MOMENTUM ...
reference referenceMomentum=$MOMENTUM ...
tune B1Field ... initial=2.10525*$MOMENTUM/200 ...
...
```

## 7.20 Setting the Phase of RF Cavities (pillbox)

The phase of an RF cavity (*pillbox*) is determined by its *timeOffset* and *timeIncrement* arguments. If *timeOffset* is not specified it is automatically set to *phaseAcc* by the Tune particle.

The basic method used to set the phase: when the Tune particle enters the volume of the pillbox its track parameters are saved and an initial value of *timeOffset* is estimated (based on its current velocity and the distance to the center of the cavity). Then it is tracked to the center of the cavity and its phase is checked (the volume is split in two to ensure the track takes a step ending in the center of the cavity). If the phase is correct within tolerance, nothing further is done and the Tune particle continues being tracked. If the phase is incorrect, the *timeOffset* value is adjusted via a simple linear solver, the current Tune particle is killed, and a new Tune particle is generated from the saved values. This usually converges in one iteration.

If *timeIncrement* is nonzero, it is added to *timeOffset* after all tuning is complete. The Tune particle is not re-tracked, so large values will completely confuse the reference particle. *timeIncrement* is intended for exploring small phase errors of the RF.

See also section 2.11 (Tuning the Beamline).

## 7.21 Tuning the maxGradient of RF Cavities (pillbox)

When RF cavities are used, it is generally necessary to carefully tune their *maxGradient* so the output beam has the desired energy or momentum. The operators of the machine arrange for this (either manually or via an automated feedback system). In G4beamline this requires the *tune* command and is done while tracking the Tune particle, usually in parallel with setting their phases (see previous section).

The *tune* command defines a “tune variable” and will arrange to save the Tune particle at one Z-position (*z0*), track the Tune particle to a second Z-position (*z1*), and then evaluate an expression using track fields. It then varies its tune variable in an attempt to make the expression evaluate to 0.0. This of course requires that some relevant beamline elements between *z0* and *z1* use the tune variable to change their behavior appropriately. In this case we’ll vary the *maxGradient* of all RF cavities between *z0* and *z1*, and use an expression at *z1* that makes the Tune particle have the desired momentum, which we’ll take as 1 GeV/c to within 1 MeV/c:

```
tune Grad z0=990 z1=41010 initial=10.0 step=0.5 \  
    expr=Pz1-1000 tolerance=0.001  
pillbox Cavity maxGradient=Grad frequency=0.201 \  
    innerLength=400 ...  
place Cavity z=1000 copies=100
```

Note that *z0* must come before the first Cavity is placed, and *z1* must be after the last Cavity is placed, at the location where the 1 GeV/c beam is desired.

See also section 2.11 (Tuning the Beamline).

## 7.22 Multiple Jobs in Parallel

In many cases you will want to run multiple simulations, typically scanning one or more parameters. In this case it is often not necessary to run multiple jobs in parallel (with or without MPI), as you can simply run the different simulations on the available CPUs. In this case nothing special is needed, as long as you keep the different output files from clashing (parameterizing the input file can help, section 7.17).

There certainly are situations in which multiple jobs working on a single simulation are appropriate.

There are two ways to run multiple G4beamline jobs in parallel, all working on a single simulation:

1. Manually or via scripts
2. Via the Message Passing Interface (MPI)

### 7.22.1 Manually or via Scripts

G4beamline is a realistic simulation program, and runs considerably slower than many accelerator simulation programs. While simple simulations can run as fast as ~1000 events per second, a moderately complicated system might run ~100 events per second, and complex simulations can run less than 1 event per second. With the advent of Linux clusters and multi-core CPUs, it is now quite common to be able to run multiple jobs in parallel. The key to doing that in G4beamline is that it always seeds the pseudo-random number generator with the event number, just before starting to process the event. The PRNG used comes from CLHEP [2], and has excellent properties for event numbers from 0 to 900,000,000.

These basic capabilities are needed to run multiple jobs in parallel:

1. **The ability to submit multiple jobs.**

This is system dependent. On a multi-core or multi-CPU system, you can simply use multiple terminal windows or multiple GUI windows to run multiple jobs simultaneously. Be sure to keep the jobs separate, including output files. On Linux, Mac OS, and Windows/Cygwin you can use a single terminal window to run several jobs simultaneously “in the background”:

```
g4bl input.file first=1000000 last=1199999 >1000000.out &
g4bl input.file first=1200000 last=1299999 >1200000.out &
g4bl input.file first=1300000 last=1399999 >1300000.out &
```

On a Linux cluster or other multi-system, you will need to learn how to submit multiple “batch” jobs.

2. **The ability to assign a unique range of event numbers to each job.**

Do this with parameters in your input.file. I use *first* and *last*, and the commands are:

```
param -unset first=1 last=1000
beam .... firstEvent=$first lastEvent=$last
```

3. **The ability to assign a unique Root output file to each job.**

Use *first* in the name of *histoFile*:

```
param histoFile=$first
```

4. **The ability to combine all of the Root output files into histograms and plots.**

Use *HistoRoot* to read all the root files, and simply select the same NTuple from each file. *HistoRoot* will combine all selected NTuples into each plot.

Note that this works well up to perhaps 32 parallel jobs; above that it becomes quite cumbersome.

### 7.22.2 Via the Message Passing Interface (MPI)

On systems that implement MPI, it can be used to speed up a simulation by a factor that is almost the number of CPUs available. The advantage is that all outputs from the different instances are combined into a single output (typically one Root file, one stderr, and one stdout). See section 8.5. The program scales well up to at least 24 CPUs, and often more (depends in detail on the system being simulated and on the computer hardware used; more complex systems will scale efficiently to more CPUs). The basic command is:

```
mpirun -np 5 g4bl input.file [... parameters]
```

This will run 5 instances of G4beamline cooperating on a single simulation. As one instance performs only management tasks, it is usually appropriate to start one more instance than the number of available CPUs. Section 8.5 has many more implementation details and suggestions.

### 7.23 Performing a Scan of Values of Some Parameter

It often happens that one wants to scan over the values of some parameter, looking for maximum transmission, the desired emittance at a given location, or some other aspect of the beamline. The idea is to parameterize the input.file so the parameter can be specified on the command line, and then either manually scan or write a shell script to do the scan. For example:

```
Param -unset VALUE=1  
... rest of input, using $VALUE
```

One can simply do this:

```
g4bl input.file VALUE=1.01  
g4bl input.file VALUE=1.02  
g4bl input.file VALUE=1.03  
g4bl input.file VALUE=1.04
```

and keep track of the desired property. If the desired property is printed to stdout (e.g. it appears normally, or via a *printf* command), one can use *grep* to greatly simplify the search for its value. This method gives a lot of output, and it can be time consuming to scan it all for the desired property of interest. So it is often better to write a short shell script to run G4beamline (with VALUE passed) and then *grep* for the desired property, or otherwise compute it, and just output that one item. Often the command history of your shell is sufficient.

One can also write a simple shell script to run the jobs in the background (see previous tip). On a 4-core system, it is usually best to run four jobs in the background, do a *wait*, run four jobs in the background, etc. In the *bash* shell, *wait* (no arguments) waits until all background jobs have completed, so this does a reasonably good job of keeping four jobs running, but does not overload the system with too many simultaneous jobs.

### 7.24 Visually Scanning Events via the Command Line

The simplest way to scan events visually is to use the *g4blgui* program, but it can be done manually via the command line. The OpenInventor driver (OIX) has the ability to exit the viewer and return to the

Geant4 command input (*File/Escape*). This can be used to quickly scan events visually. To do this, first create a text file called *beamon.txt* containing a large number of identical lines:

```
/run/beamOn 1
/run/beamOn 1
/run/beamOn 1
... many more identical lines
```

Then invoke *g4beamline* like this:

```
g4bl input.file viewer=best <beamon.txt
```

Once the viewer appears, it will display the first event. To see the next event, select the *File/Escape* menu item. This will suspend the OpenInventor viewer, and the Geant4 input routine will read the next line of *beamon.txt* – that causes it to track one event and refresh the viewer with it. Just keep selecting *File/Escape* to sequence through events one at a time. This is essentially what the *g4blgui* program does when a viewer is selected.

## 7.25 Optimizing the Value of Some Parameter(s)

When designing a new machine or facility, there is frequently the need to optimize some parameters. The program *gminuit* is available for this, and interfaces well to *G4beamline* (and to other programs). Download it from <http://muonsinc.com> in the Computer Programs link at the left.

The basic idea is to write the *input.file* to have command-line parameters for the parameters to be optimized, and to configure *gminuit* to vary them. A shell script must be written to run *G4beamline* and then compute a Chi-squared for *gminuit* to minimize. See the *gminuit* documentation for details how to do this. *Gminuit* will optimize any quantity that can be computed in a shell script, and is not limited to *G4beamline*.

In practice, one finds that program overheads are rather large, so there is a big advantage to requiring fewer iterations. A human can often converge on an optimum faster (fewer iterations) than can *gminuit*'s minimizer. This is especially true if the shell script can present a useful graph to the user rather than just a single value of the Chi-squared. Note also that minimizing an integer-valued expression, such as the # of particles transported to a given location, is difficult; in *gminuit* one should set the granularity of MINUIT to 1.0 or more, and run enough particles so that the statistics are reasonably good (at least 1,000 particles, and 10,000 is better).

The *examples* directory of the *G4beamline* distribution contains *triplet.sh* which tunes a quad triplet to achieve a focus at  $z=6000$ . Note that it presents the user with a plot of the beam's horizontal and vertical sizes as a function of  $z$  along the beamline – this additional information permits a human to find a solution much faster than the minimizer (the locations of the horizontal and vertical zero-crossings tell you which way to change the quads' values). Even so, using *gminuit* in manual mode is significantly easier than it would be to do this by manually typing each command (*gminuit* presents the user with a simple way to change the values of the variables and to execute the program once).

## 7.26 Using Two or More Reference Particles

Unlike most accelerator programs, G4beamline realistically handles particle decays and interactions. This makes it suitable to simulate muon beams from a primary proton beam all the way to a muon experiment or cooling channel. This means that the portion of the beamline just after the production target should be optimized for pions, and the portion at the end should be optimized for muons. The best way to do this is to use two reference particles, one starting at the production target with `particle=pi+`, and one starting at a suitable downstream point with `particle=mu+`. The key point is that no tuning should be done in the region where both reference particles are tracked. In exampleAUG05.in, the pion reference hits an aperture and is killed. If this does not happen naturally, you could use a *particlefilter* to kill the pion reference at an appropriate place. Tuning will get confused if multiple reference particles enter the tune region.

## 7.27 Fitting to Plots and Histograms in HistoRoot

A major advantage of using *HistoRoot* is the fact that it is based on Root [5] and has all of the Root features at its disposal. This includes the ability to fit functions to plots and histograms. The Root user interface is quirky, but quite functional.

To fit a function to a histogram, first generate the plot containing the histogram. Right-click on the histogram itself (one of its bins, not the canvas and not any axis or other object – the cursor changes from cross to arrow when clicking will select the histogram; that’s what you want), and then select FitPanel. The FitPanel lets you select from among a number of pre-defined functions and a user-defined function:

Name	Description
gaus	A Gaussian distribution.
gausn	A normalized Gaussian distribution.
expo	An exponential distribution.
landau	A Landau distribution.
landaun	A normalized Landau distribution.
pol0 – pol9	A polynomial of order 0 through 9.
user	A user-defined function: use "[0]", "[1]", ... for the parameters to be fitted, and make it a function of the variable "x" (or "x" and "y"). For instance, here is a 1-d Gaussian: $[0]*\exp(-0.5*(x-[1])*(x-[1])/([2]*[2]))$ (parameters are: constant, mean, sigma). Note that after typing a user expression you MUST do "Set parameters", or you will get an error "Function with name ... does not exist".

There is a button to set the initial values and ranges of the parameters for the selected function. There are a number of options. If you select “Use range”, then the double-slider below becomes active and will set the range of the fit (displayed visually in the plot as you move the slider ends). When fitting, it is useful to check the plot’s menu item Options/FitParameters so the fit results are displayed in the statistics panel of the plot (you can drag its edges to make it larger, and its font will scale). For further details, see the Root documentation.

Note that if you want to play with backgrounds, you can select Options/CanEditHistograms – then with the mouse you can drag any histogram bin’s value up or down as you wish. You can then re-fit to see how your changes affect the results.

## 7.28 Interfacing to Other Programs

Many different file formats are used in physics today, and no program can implement all of them. G4beamline supports ASCII file formats that are suitable for interfacing to other programs, including both track files and field maps. A conversion program in C or Java can easily be written to read an ASCII file and write another in a different format (e.g. new order of items, different units, etc.). Root files can also be converted – Root [5] is quite powerful, and can easily be scripted using C++ macros. It is straightforward to open a root file, select a TNTuple from it, loop over its rows, and write a new Root file and TNTuple (or ASCII file) in a new format. The Root documentation and examples contain numerous macros of this sort. If you have need for an interface that cannot easily be implemented, you can suggest a new feature for G4beamline to implement it directly (see section 7.3).

## 7.29 EventID and TrackID, and Encoding Information in them

In G4beamline, EventID is an integer between -2 and 2,147,483,647 (0x7FFFFFFF), inclusive. Note, however, that EventID-s greater than 16777215 (0xFFFFF) cannot be precisely represented in a float and their values in any NTuple will be rounded. Values -2 and -1 are reserved for the Tune and Reference particles.

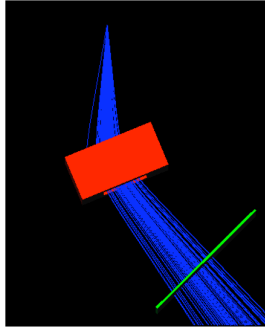
In G4beamline, TrackID is an integer between 1 and 2,147,483,647 (0x7FFFFFFF), inclusive. Similarly, TrackID-s greater than 16777215 (0xFFFFF) cannot be precisely represented in a float and their values in any NTuple will be rounded. In addition, the *beam* command will issue a warning for every primary TrackID greater or equal to the value of *secondaryTrackID* (such a primary TrackID could be confused with a secondary’s automatically-generated TrackID).

Note there is no requirement that either EventID or TrackID be unique or sequential. When G4beamline generates events, the EventID-s are assigned sequentially from 1, the primary TrackID-s are assigned sequentially from 1 within each event, and secondary TrackID-s are assigned sequentially from the *secondaryTrackID* argument to the *beam* command (default=1001).

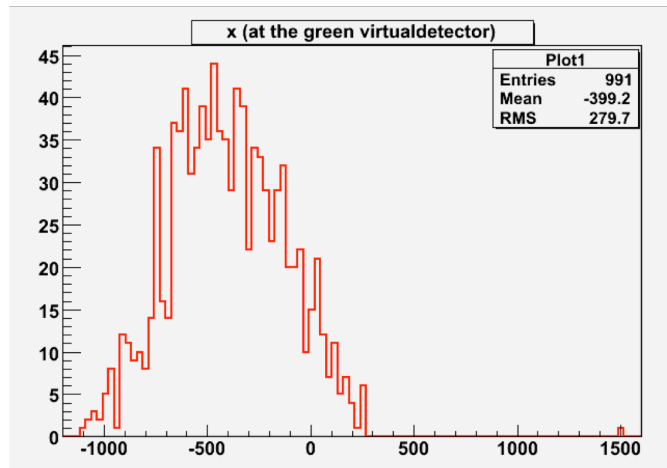
This means that when generating a beam file to be read into G4beamline, you can assign both EventID-s and TrackID-s in whatever manner you like, subject to the above constraints. For instance, you could assign bins to the phase space of a generated beam and set either EventID or TrackID to the bin number of each track in the beam file. After running G4beamline to determine which tracks are accepted by the system, a plot of the acceptance can be made by post-processing the output file with knowledge of how many tracks were generated in each bin.

## 7.30 Examining Outlier Events

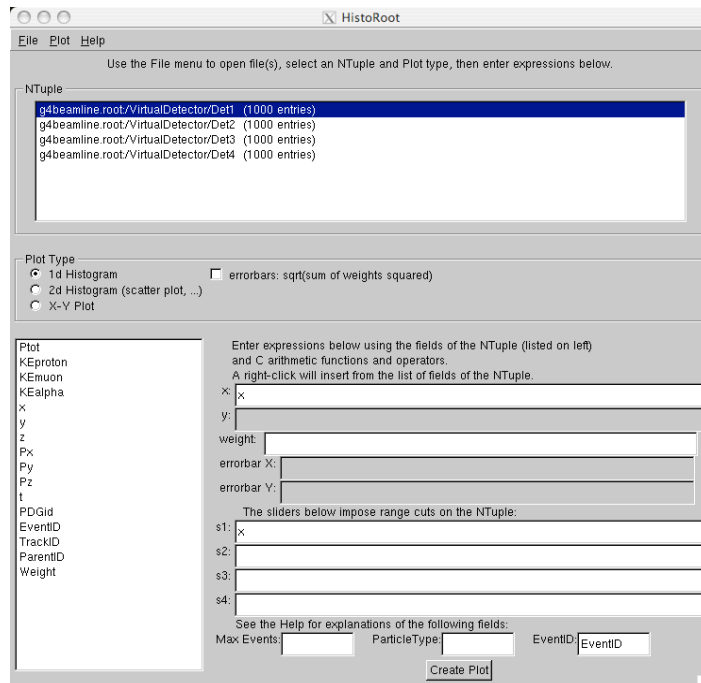
It often happens that some sort of puzzling event occurs, or an outlier entry appears in some histogram. Here is a method to find such events and then visualize them. As a simple example, consider this bending magnet (red) followed by a virtual detector (green), with 100 beam tracks shown (blue):



This looks fine, until one runs 1,000 events and histograms  $x$  in the virtualdetector – clearly there is an outlier track at  $x \sim 1500$ .



The way to find this event is to use a slider and the EventID features in *hisoroot* to select the outlier event and write a *histo\_events.txt* file:



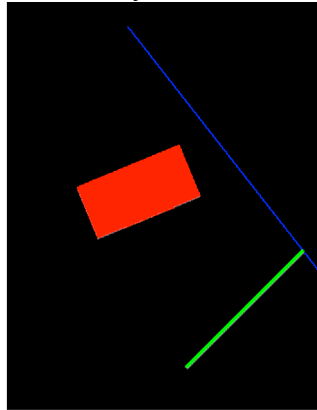
Note that slider S1 has the same expression as the histogram  $x$  – this permits you to use S1 to select just the outlier event. As the EventID field contains an expression that evaluates to the event # of the NTuple

row, historoot will write all selected EventID-s to the file *histo\_events.txt* (re-writing it repeatedly as the slider is changed). After selecting just the outlier entry, its EventID will be the only entry in the file.

It is then straightforward to run G4beamline in visualization mode with the *eventcuts* command to select just those events listed in the *histo\_events.txt* file:

```
g4bl file.in 'eventcuts file=histo_events.txt' viewer=best
```

(The *eventcuts* command could be edited into file.in instead of being placed on the command-line; in *g4blgui* the checkbox “HistoRoot Events” puts that *eventcuts* command into the command-line.) The result is clear and obvious -- this simulation clearly needs shielding around the bending magnet:



**Note:** this assumes the default treatment of pseudo-random numbers: the generator is seeded with the event # before each event. This won't work if you use the *randomseed* command to change that.

**Note:** If your input beam is a file, and that file has multiple tracks with the same event number, G4beamline will treat each track as a separate event with that event number (see the description of beams above). That means that if an event selected via *eventcuts* has (say) three beam tracks, only one of them need satisfy the conditions used to select the event. The other two tracks with that event # will be selected by *eventcuts* but may well look puzzling, as they probably don't satisfy the conditions, and will appear as separate tracks in the visualization. Just ignore them.

## 7.31 Increasing the Number of Events Displayed Visually

By default, the maximum number of events displayed in a viewer is 100. To increase this limit to 1000 events, execute this command:

```
g4ui when=4 "/vis/scene/endOfEventAction accumulate 1000"
```

## 7.32 Building G4beamline, Adding Your Own Code

There are several reasons why you might want to build G4beamline from source:

- You want to run it on an unsupported system or OS.
- You want to optimize it for your system (e.g. a 64-bit build).
- You want to add your own new feature or code.

Before you can add your own code to G4beamline, you must build it from source. Read BUILD.txt and README-\*.txt (in the doc directory of any distribution) for details. Note the required tools and the details of the build procedure differ for different OSs.

Once you have built G4beamline from source, test that *g4blmake* works on your system:

```
cd test
./test72
```

If this says “test-72: compiling user code omitted” then there is an error in your configuration. If test72 takes 20-30 seconds and prints nothing but its 1-line synopsis, then it succeeded (all tests succeed silently but print an error on failure).

Now you are ready to add your own code. First, create a directory in which to work; for this example I’ll call it testing:

```
cd
mkdir testing
cd testing
... copy or create your new code in a file such as testing.cc
g4blmake
```

By default this compiles \*.cc \*.cpp \*.C into a new build of G4beamline. Add “-v” to display the details.

If you need an additional library, list all of the files you need on the command line:

```
G4blmake [-v] *.cc /some/other/library.a
```

You can put the corresponding “-I/some/other/include/dir” either in the environment variable CPPFLAGS, or directly on the *g4blmake* command line (before the .cc files that use it).

If you have some Fortran source, then you will need a Fortran compiler that is compatible with the C++ compiler on your system. You will also have to write the C++ that calls the Fortran routine(s) correctly (functions are extern “C” in lower-case with an underscore appended, scalar arguments are passed by reference, multiple array indexes are in different order, etc.; Google “C++ calling Fortran” for suggestions). In this case, it is easiest to construct a Makefile to build the pieces (this uses the system .f.o rule to compile the file fortran.f, then link it into G4beamline):

Makefile:

```
# Compile *.cc and fortran.f into G4beamline.
g4beamline: *.cc *.hh fortran.o
    g4blmake *.cc fortran.o /usr/local/lib/libgfortran.a

clean:
    rm -f *.o g4beamline
```

Once your private build of G4beamline succeeds, to run it just follow the directions *g4blmake* printed:

```
export G4BEAMLINE=`pwd`/g4beamline
```

Now the *g4bl* script will run your private version. You can verify this by looking at the heading that *g4beamline* prints out during startup: the version will show both the G4beamline version and your login id. You can also do “*help mycommand*”, or just use *mycommand*.

Some brief suggestions for writing G4beamline code:

- Point your browser to `<install>/doc/html/index.html` – this will browse the Doxygen documentation for the G4beamline classes.
- Class names beginning “BL” are G4beamline infrastructure classes.
- Class names beginning “BLCMD” are G4beamline commands.
- Filenames are based on the class names by appending `.hh` or `.cc`.
- Because commands are modular and independent, your new feature should be packaged in a command. That is, your code should not do anything until and unless the input file uses your command. As a bonus, arguments to that command can pass user-specified values to your code.
- There are many commands in the source. Base your command on an appropriate one.
- Multiple features should be in multiple commands. Note that even such complex features as a space charge computation are contained in a command.
- In general, G4beamline commands are contained in a single file not referenced by any other code (BLCMDphysics and BLCMDcoil are exceptions). In particular, there are no `.hh` files for command classes; the declarations are at the top of the `.cc` file.
- All classes that implement a command are derived from `BLCommand`; they register themselves as commands via a static initializer and their default constructor. When properly written, simply compiling your *MyCommand.cc* via *g4blmake* will enable its command to be used in any *input.file*.
- Because commands are so modular, multiple developers can be working on multiple commands in parallel, without conflicts. Only modifications to infrastructure classes require coordination.
- Contributing a command to the G4beamline distribution is very simple, and consists merely of having the development team test it, and then copy its source file into the build directory.
- An exception to using a command is *usertrackfilter*. This command was specifically designed to use simplified user-supplied code. See section 8.4.

### 7.33 Displaying Magnetic Field Lines

The *fieldlines* command will display magnetic field lines. Due to the difficulty of finding fields without user assistance, you must specify a point (in global coordinates) near which to start the field lines; this point is used as the center of a circle of specified radius. The plane of the circle is normal to the  $B$  field at its center. Within the circle, field lines are placed with density that is inversely proportional to  $|B|$ . You must also specify the approximate number of field lines to draw; due to the algorithm used this is only approximate.

Within the circle, the initial points of the field lines are allocated as follows: an  $N \times N$  square grid is placed so it circumscribes the circle; initially all points outside the circle are excluded. An initial value of scale is guessed. As each line is placed, all grid points within  $\text{scale}/|B|$  of its location in the circle are excluded. The next point is placed at the un-excluded grid point that is closest to the center of the circle, and nearby grid points are excluded. This continues until all grid points have been excluded. If the total number of field lines placed is within a factor of 2 of the desired number, this set is accepted and the field lines are drawn. If the number placed is too small, scale is reduced and the procedure is repeated; if the number placed is too large, scale is increased and the procedure is repeated. At most 10 iterations are permitted. This implies that asking for fewer than 5 lines, or more than 1,000 lines, is likely to be ineffective. This algorithm gives a good distribution of field lines for solenoids and other simple fields.

Once the lines are placed within the circle, for each one the B field is traced in both directions; there are three criteria for stopping drawing a given half-line:

- a. The line intersects the plane of the circle (i.e. it has come back to the plane, but outside the magnet).
- b. The value of  $|B|$  falls below *minField*.
- c. The field line leaves the simulated world.

There are additional arguments to the *fieldlines* command that control the drawing.

Remember the field lines are drawn in 3 dimensions, so they won't look like the pictures in a textbook. This is a feature, not a bug, and using the OpenInventor viewer to examine the system from different angles can give a good understanding of the field lines.

### 7.34 Setting the Phase of RF Cavities (rfdevice)

A *rfdevice* may describe either just single RF cell or a whole multicell RF structure which shares a common RF phase. It is important to note that G4beamline defines **0° RF to be the rising slope of the zero crossing** of the RF waveform, so 90° RF is on-crest for a positive particle. This is a common, but not universal convention.

The phase of an RF device (*rfdevice*) may be set in a number of ways. The timing process is local to a single *rfdevice*; tuning uses other elements to adjust parameters. Timing is done first and is described here, while tuning is described in the next section. Examples and many details on timing RF cavities are given in section 8.6; this is just a very brief summary.

The most basic way to time an RF cavity is to explicitly set its *timeOffset*, the absolute time by which the fields are translated. *timeOffset* includes all phase information. The electric field is of the form:

$$E(x,y,z) * \sin( \omega (t - timeOffset) )$$

and the magnetic field:

$$B(x,y,z) * -\cos( \omega (t - timeOffset) )$$

The *rfdevice* will try to find a consistent solution based on what it is told; both under- or over-specifying the system are fatal errors. If both *timeOffset* and *maxGradient* are specified, nothing more needs to be determined.

If *timeOffset* is unspecified, a *timingMethod* must be used to determine the global (“BASE”) timing of 0° RF. For most applications *timeMethod=maxE* is preferred, in which the time of maximum energy gain of a positive particle is taken to be 90° RF. The default *timeMethod=atZ* emulates the old *rfdevice* behavior, in which *timeOffset* is chosen such that  $\omega(t-timeOffset)=phaseAcc$  at the *timingAtZ* (usually the *rfdevice* center) location. In most cases, the *maxStep* ought to be set to ~1 mm, as the default step size is too large. The method launches Tune (Event# -2) particles and uses them to find a solution. The timing of deflecting cavities is discussed in section 8.6.

```
rfdevice ... timingMethod=maxE maxStep=1 ...
```

maxGradient=( $\Delta V/L$ ) MV/m	phaseAcc=( $\phi$ ) degRF
maxGradient=( $\Delta V/L$ ) MV/m	fixEnergyGain=( $\Delta E$ ) MeV
maxGradient=( $\Delta V/L$ ) MV/m	fixMomentum=( $ P_{out} $ ) MeV/c
timeOffset=( $t_{absolute}$ ) ns	maxGradient=( $\Delta V/L$ ) MV/m

### Common phase-setting option combinations.

The arguments to the `rfdevice` command are described in section 5, and section 8.6 describes setting it up in more detail, but briefly one can fix some parameters while the `rfdevice` routine will attempt to fill in the remaining parameters; if it fails, G4beamline reports a fatal error. The final result of the timing for each `rfdevice` is printed to stdout

See also section 2.11 (Tuning the Beamline).

## 7.35 Tuning the maxGradient of RF Cavities (rfdevice)

When RF cavities are used, it is generally necessary to carefully adjust their *maxGradient* so the output beam has the desired energy or momentum. The operators of the machine arrange for this either manually or via an automated feedback system. In G4beamline this can be done two ways.

The first is the *autoTiming* technique and it is local to just its own `rfdevice`, requiring no information from downstream locations. In this method, generally both the appropriate *timingMethod=**maxE* and the step size *maxStep* ought to be specified, as the defaults may not be good choices.

```
rfdevice ... timingMethod=maxE maxStep=1 ...
```

timeOffset=( $t_{absolute}$ ) ns	fixMomentum=( $ P_{out} $ ) MeV/c
phaseAcc=( $\phi$ ) degRF	fixEnergyGain=( $\Delta E$ ) MeV
phaseAcc=( $\phi$ ) degRF	fixMomentum=( $ P_{out} $ ) MeV/c

### Common gradient-finding option combinations

The second technique requires the *tune* command and is done while tracking the Tune (Event# -2) particle, usually in parallel with setting their phases (see previous section). The *tune* command defines a “tune variable” and will arrange to save the Tune particle at one Z-position (*z0*), track the Tune particle to a second Z-position (*z1*), and then evaluate an expression using track fields. It then varies its tune variable in an attempt to make the expression evaluate to 0.0. This of course requires that some relevant beamline elements between *z0* and *z1* use the tune variable to change their behavior appropriately. In this case we’ll vary the *maxGradient* of all RF cavities between *z0* and *z1*, and use an expression at *z1* that makes the Tune particle have the desired momentum, which we’ll take as 1 GeV/c to within 1 keV/c:

```
tune Grad z0=990 z1=41010 initial=10.0 step=0.5 \
  expr=Pz1-1000 tolerance=0.001
```

```
rfdevice Cavity maxGradient=Grad frequency=0.201 \
```

```
innerLength=400 ...
```

```
place Cavity z=1000 copies=100
```

Note that  $z_0$  must come before the first Cavity is placed, and  $z_1$  must be after the last Cavity is placed, at the location where the 1 GeV/c beam is desired. See also section 2.11 (Tuning the Beamline).

## 8 Advanced Topics

### 8.1 Writing Scripts Using G4beamline

By using parameters in your *input.file* you can automate running G4beamline programs, without having to script the editing of your *input.file*.

#### 8.1.1 Using a Linux Cluster or Multi-CPU System

Because the pseudo-random number generator is seeded with the Event number before the start of each event, all that is necessary to run jobs in parallel is to ensure that they do not run the same events and that they write to different output files. The arguments *firstEvent* and *lastEvent* to the *beam* command facilitate this. The basic idea is to use parameters for *firstEvent* and *lastEvent* and pass them in on the command-line, and write a shell script to submit multiple jobs to the cluster with disjoint ranges for each job. Here the output file corresponds to the first event number. The *input.file* looks like this:

```
# give defaults for ease in testing
param -unset first=0 last=1000
# use $first for the Root file (.root will be appended)
param histoFile=$first
output $histoFile.out
beam Gaussian firstEvent=$first lastEvent=$last ...
```

Any other output files (e.g. *format=ascii* for *virtualdetector*, *zntuple*, *timentuple*, etc.) should be set to *\$first* as well, so multiple instances of the program won't overwrite output files.

The simulation is run like this on a 4-CPU system (*bash* shell):

```
g4bl input.file first=0 last=999999 &
g4bl input.file first=1000000 last=1999999 &
g4bl input.file first=2000000 last=2999999 &
g4bl input.file first=3000000 last=3999999 &
tail -f 0.out # watches the output from the first instance
```

Or:

```
tail -1 *.out # gives a snapshot of where each instance is
```

The output files will be *0.root*, *1000000.root*, *2000000.root*, and *3000000.root*. The *historoot* program can be used to generate histograms:

```
historoot *.root
```

You can select the 4 instances of a given NTuple and *historoot* will generate histograms and plots containing results from all 4 million events.

The example *MICE\_StageVI.g4bl* is a real-world example using this technique.

#### 8.1.2 Using the gminuit Program to Tune Values

The *gminuit* program (<http://www.muonsinc.com> [ComputerPrograms/Gminuit]) was specifically designed to permit the easy tuning of values in an *input.file* for G4beamline. Because any argument to any G4beamline command can be set from a parameter, and any parameter can be set on the G4beamline command line, *gminuit* can be used to tune any value in your *input.file* based on any

criterion you can script. It is possible to determine the *gminuit* value to minimize by either reading the output from G4beamline with `steppingVerbose=1` (tailor `steppingFormat` to suit), or by reading the file generated by *virtualdetector format=ascii* or *trace format=ascii*. Remember that most shells only do integer arithmetic, so it is appropriate to use a `tel` or `perl` script to interpret the G4beamline output and return a value for *gminuit* to minimize. See the *gminuit* documentation for details. The file *examples/triplet.sh* is an example that uses *gminuit* to tune a quad triplet.

## 8.2 Violating the Rules on Geometrical Intersections

Geant4 tracks particles through the geometry hierarchy by looking for the next geometrical boundary along the track's path, and arranging so there is a step that ends at each boundary crossed by the track. To keep the search for boundaries efficient, Geant4 only looks for the outer boundary of the current volume and for boundaries of all daughter volumes of the current volume. So if sibling volumes A and B overlap, if the track enters A first, then it will be tracked in A through the region of intersection; but if it enters B first then it will be tracked in B through the region of intersection. Similarly, if a daughter A1 of A extends outside volume A, then if the track is in A1 it can be tracked in A1 even outside of A; but if it comes in from the sibling or parent of A it won't enter A1 until it enters A.

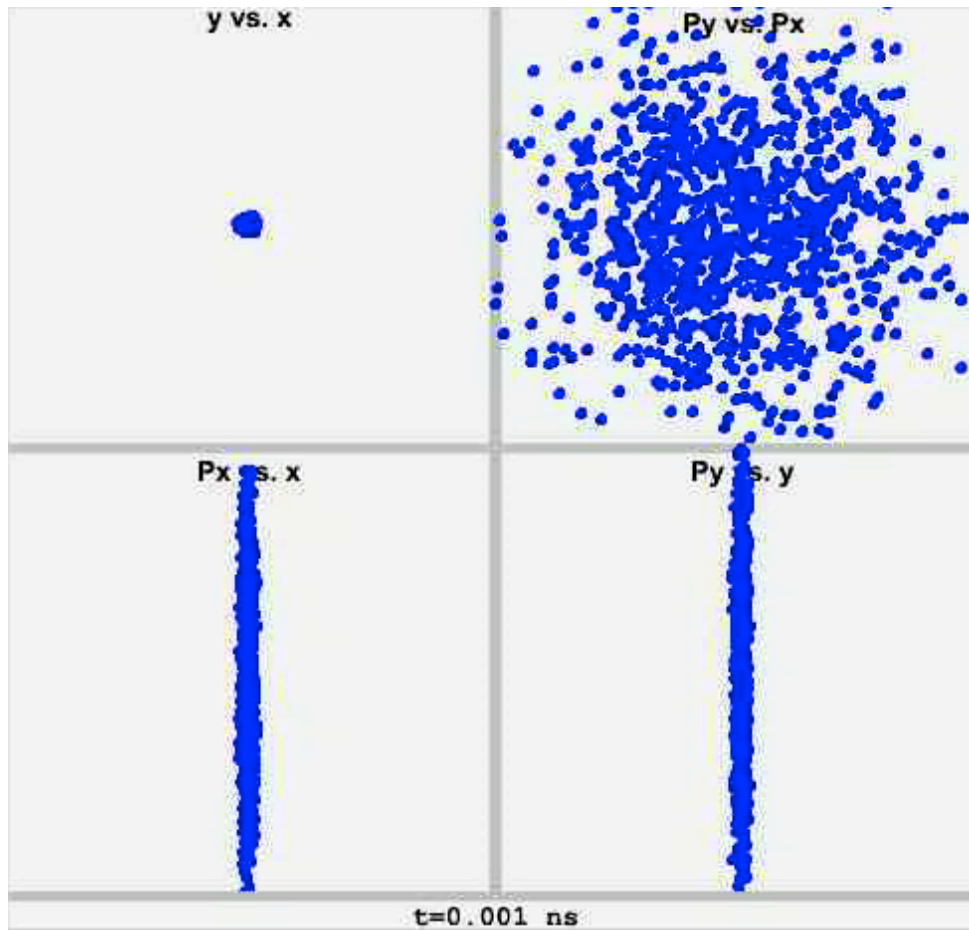
While violating the rules is risky and discouraged, if you are careful they can be violated when necessary (e.g. it is not possible or too difficult to describe the geometry correctly). If the invalid intersections don't matter, then they won't affect the result of the simulation. They won't matter if all volumes involved have the same material and none of them is "active" (in G4beamline that means they aren't a *virtualdetector*). They also won't matter if the regions of intersection are in places where no particles are tracked (e.g. far from the beamline aperture). And they won't matter if either of the intersecting volumes has `kill=1`.

The automatic geometry test in G4beamline verifies the geometry hierarchy by testing points on the surface of each element's volume, verifying that each point is inside the parent volume of the element, and is outside every sibling volume. The points are selected to test "corners" first, and then points are randomly distributed over the faces of the surface of the volume. This is not perfect, but it does catch nearly all invalid intersections.

Geant4 also has other methods of verifying the geometry hierarchy. See the Geant4 user's manual [1]. They involve shooting geantinos through the detector and verifying that the boundaries each one crosses are ordered properly. They can be executed using the *g4ui* command.

## 8.3 Making a Movie

G4beamline now supports the ability to make a movie of the beam, with the camera sitting on the reference particle watching the other beam particles "dance" around nearby. The implementation is quite flexible, and permits the user to specify multiple simultaneous panes, each displaying an arbitrary pair of expressions using the track variables. An example movie is viewable at <http://g4beamline.muonsinc.com>. Here is its first frame:



This movie is of *Example1.g4bl*, in which a Gaussian muon beam propagates through free space into four virtual detector-s (which are invisible in the movie). The movie is 10 seconds long, during which the “y vs. x” pane’s particles grow considerably, and the two lower panes’ ellipses rotate in the usual manner. A handful of decay electrons and neutrinos become visible.

Making a movie requires the following prerequisites:

**1. Installation/configuration requirements:**

- a. G4beamline version 1.16 (or later).
- b. Root version 5.20 (or later).
- c. A C++ compiler that Root can use in ACLiC mode.
- d. You must use the command-line to run `g4blmovie`; on Windows this requires Cygwin [7].
- e. The `ffmpeg` program [8] must be installed (it converts a series of frame images into a movie).

**2. G4beamline simulation requirements:**

- a. There must be a reference particle.
- b. The `movie` command must be included. Note that once you have a working simulation, this can be given on the command line:  
`g4bl input.file movie ...`

**3. g4blmovie command-line program**

- a. It is included in the G4beamline distribution (1.16 and later).

- b. Note that so far this has been used only on Mac OS X; it is expected to work on other platforms but has not yet been tested.

The basic approach is to first run a G4beamline simulation, with the *input.file* containing the *movie* command, and with a Root output file. The *movie* command arranges to generate several NTuples in the Root file that will be plotted to make the movie. This is quite similar to the *trace* command that traces all tracks into a single NTuple, so the output file can grow quite large. Note that a movie containing more than a few hundred particles tends to be so crowded that details are not visible. Once the simulation is complete, the *g4blmovie* command is run, which has its own input file to describe the movie. This program uses Root to read the movie NTuples and generate a series of image files; it then runs *ffmpeg* to convert them into the desired movie, and then deletes the image files. Note that movies can get large, and the production process requires about triple the disk space of the final movie; a 10-second movie at 24 frames/second is typically about a megabyte long (these rather un-natural images don't compress very well).

The new *g4blmovie* command has one argument, an input file to describe the movie. This gives great flexibility in organizing the picture as a number of panes, each displaying a pair of expressions using the track variables. The panes are laid out in rows, with no inherent limit on either the number of panes in a row or on the number of rows, but too many of either will probably make the movie unwatchable. The input file is quite similar to that of G4beamline, except that quoted arguments are not allowed, spaces are not permitted (or needed) in any argument, and there are no parameters.

g4blmovie input file commands		
Command	Argument	Description
setup		This command must be first. It sets basic parameters for the movie.
	outputFile	The output movie file. Its extension determines the format – any format supported by ffmpeg can be used. The following extensions are known to work: .mov, .swf, .avi. <b>Beware:</b> spaces are not allowed.
	windowWidth	The movie window width, <u>in pixels</u> . All panes scale to this value.
	windowHeight	The movie window height, <u>in pixels</u> . All panes scale to this value.
	tMin	The simulation's start time of the movie (ns).
	tMax	The simulation's end time of the movie (ns).
	duration	The real time duration of the movie in seconds (10).
	frameRate	The frame rate of the movie, frames/sec (24).
	textHeight	The default height of text, <u>in pixels</u> (15). Changeable in other commands.
	background	The background color (#F3F3F3 – light gray). Changeable in other commands.
	borderColor	The color of the borders between panes (#C0C0C0).
	borderSize	The size of the borders, <u>in pixels</u> (5).
	marker	The marker to use (21 – the square marker). This marker is used for all particles (but see the particle command below to color them). Note that other markers, such as 20 (filled circle), can greatly slow down the movie production.
	markerSize	The relative size of the marker (1.0).

	pictureType	The extension of the pictures used to generate the movie (jpg). Other types known to both Root and ffmpeg can be used, but jpg has given the best performance.
particle		Sets the color for each particle type.
	*	Arguments are of the form: 13=#0000FF, where 13 is the PDGid of the particle, and #0000FF is the color (6 hex digits, 2 each for red, green, and blue; #000000 is black, #FFFFFF is white, and #0000FF is bright blue). The '#' is required. PDGid 0 applies to all unset particles (defaults to black).
plot		Displays a plot of two expressions.
	filename	The name of the Root file to use. If omitted, uses the most recently opened Root file (i.e. from the previous command). Note the NTuple names to use are fixed: Movie/Reference and Movie/Tracks.
	title	Title of the plot (defaults to "y vs. x", where x and y are the expressions).
	x	Expression involving the track variables to plot horizontally. Variables: x, y, z, Px, Py, Pz, t, PDGid, EventID, TrackID, ParentID, Weight. The standard C functions and operators can be used.
	y	Expression involving the track variables to plot vertically.
	xMin	Left limit of the plot, in whatever coordinates are used by x.
	xMax	Right limit of the plot, in whatever coordinates are used by x.
	yMin	Bottom limit of the plot, in whatever coordinates are used by y.
	yMax	Top limit of the plot, in whatever coordinates are used by y.
	background	Color of the background ( <i>setup</i> value)
	textHeight	Height of text <u>in pixels</u> . If omitted, the value from the <i>setup</i> command is used.
sideview		Displays a side view of the apparatus using Reference Coordinates, with a moving marker that indicates where the reference particle is located at the current time.
	markerSize	The relative size of the marker (1.0).
	filename	The name of the Root file to use. If omitted, uses the most recently opened Root file (i.e. from the previous command). Note the NTuple names to use are fixed: Movie/Reference and Movie/Elements.
	zMin	Left limit of the plot, in Reference Coordinates (synonym is xMin).
	zMax	Right limit of the plot, in Reference Coordinates (synonym is xMax).
	yMin	Bottom limit of the plot, in Reference Coordinates.
	yMax	Top limit of the plot, in Reference Coordinates.
	title	Title of the view ("").
	textHeight	Height of text <u>in pixels</u> ( <i>setup</i> value).
	background	Color of the background ( <i>setup</i> value)
row		Initiates a new row of plots.
	height	The relative height of this row. The first row has a height of 1.0, which sets the scale.
space	(none)	Displays like a plot, but is empty.
	title	Title of the pad ("").

	textHeight	Height of text in pixels ( <i>setup</i> value).
	background	Color of the background ( <i>setup</i> value)
time	(none)	Displays like a plot, but contains the current time (in ns).
	title	Title of the pad ("").
	textHeight	Height of text in pixels ( <i>setup</i> value).
	background	Color of the background ( <i>setup</i> value)
Position	(none)	Displays like a plot, but contains the current Z position of the reference particle (mm).
	title	Title of the pad ("").
	textHeight	Height of text in pixels ( <i>setup</i> value).
	background	Color of the background ( <i>setup</i> value)

Within a row, all plots share equally the total width of the window. The rows are laid out to use the entire height of the window, but the fraction of each row is determined by its *height/totalHeight* (*totalHeight* is the sum of *height* for all rows). There can be a different number of plots in each row. You can use *windowWidth*, *windowHeight*, the row *height*-s, and the number of panes per row to have both large and small panes in one movie. Usually each pane should be approximately square. To play with the layout, it is helpful to temporarily set *tMax* and *duration* to small values to speed up the production until the layout is what you want. Many simulations start at *t*=0, and for them it is best to set *tMin* to a small value (e.g. 0.001) so the initial tracks show up in the viewer when the file is opened but not playing.

Here is the *examples/Movie.in* file that generated the *example1* movie referenced above:

```
# Movie from example1.in with movie command -- outputs a Flash
setup outputFile=example1.swf tMin=0.001 tMax=18 duration=10
# e+=red, mu+=blue, others=gray (neutrinos)
particle -11=#FF0000 -13=#0000FF 0=#808080
plot filename=g4beamline.root xMin=-800 xMax=800 yMin=-800 \
    yMax=800 x=x y=y
plot xMin=-50 xMax=50 yMin=-50 yMax=50 x=Px y=Py
row
plot xMin=-800 xMax=800 yMin=-50 yMax=50 x=x y=Px
plot xMin=-800 xMax=800 yMin=-50 yMax=50 x=y y=Py
row height=0.1
time
```

Here are the commands used to generate and display that movie:

```
cd g4beamline-1.16-Darwin-g++/examples
g4bl example1.in movie
g4blmovie movie.in
open example1.swf
```

## 8.4 User Code for the usertrackfilter Command

The usertrackfilter command requires that the user compile code for use with G4beamline. The usertrackfilter element permits the user to supply code that applies to every track that enters the physical volume of the element; this code can:

- Kill tracks, based on arbitrary criteria.
- Modify the momentum, time, and/or weight of tracks.
- Create secondary tracks.

Doing this requires:

- G4beamline built from source on your system.
- A basic understanding of how to write and compile C++ code.
- On Windows, the Cygwin system must be installed.
- On all OSs, a command-line window is used.

The basic idea is to compile your code directly into G4beamline. An example is provided in the directory UserCode in the G4beamline installation directory (if you don't know where that is, type the command "g4bl -dir"). It contains the following files:

Filename	Description
ExampleFilter.cc	Code for the example filter.
test.in	Input file to test the example filter.

These instructions are for the bash shell; modify them appropriately for other shells. These are all basic UNIX commands, and if you don't understand them you probably need the assistance of someone more experienced in UNIX and C++ code development. After installing and building G4beamline from source (see the appendices, BUILD.txt, and the README files), you should first try running the example:

```
cd directory-for-your-simulation
mkdir UserTrackFilter
cp `g4bl -dir`/UserCode/* .
g4blmake
export G4BEAMLINE=$PWD/g4beamline
g4bl test.in
```

This last command should end by printing:

```
Run complete 10 Events 0 seconds
ExampleFilter::complete killed 6 tracks, created 51 tracks
ExampleFilter::complete killed 6 tracks, created 51 tracks
```

Once the example works, you can modify it to suit your needs (the example is quite silly and unphysical; it is intended only to show what can be done). Things to remember:

- The source files to be compiled are arguments to g4blmake; they default to \*.cc \*.cpp \*.C (i.e. the common extensions for C++ source files).
- Any additional libraries your code requires can be passed in the environment variable EXTRALIBS, or can be simply appended to the g4blmake command line.
- Any additional compiler flags (e.g. -I/some/dir) can be passed in the environment variable CPPFLAGS.
- Your filter code must implement a class derived from UserTrackFilter, and must have a static instance (to register it with G4beamline).

- Read the comments in `UserTrackFilter.hh` to understand the interface to G4beamline, and to see what types of things your code can do.
- More than one derived class can be defined in the source files, either in a single `.cc` file or in separate `.cc` files. They must have different `filterName()`-s.
- You can add other libraries if you wish. For example, your class could query an external database indexed by `eventID` and `trackID`, and kill unwanted tracks as indicated by the database. (To do that your class would connect to the database in `setup()`, query it in `filter()`, and disconnect in `complete()`.)
- It's best to keep your code simple, as it is rather difficult to debug large programs like G4beamline.

## 8.5 Multiple Instances of G4beamline using MPI

The Message Passing Interface (MPI) is a standard interface that permits multiple jobs on multiple computer systems to cooperate in a single computation. One executes a single MPI job that simulates the events specified in the *input.file*; internally it uses multiple jobs running simultaneously, collecting their output into a single set of NTuples (in a single Root file by default). Only the tracking is parallelized, but that is “trivially parallelizable” and the overheads are low; it scales quite well to many processors depending on the details of both the system being simulated and the computer hardware. On a Mac Pro running Mac OS X with 4 cores, the management and communication overhead is only a few percent. On a supercomputer at NERSC (Hopper), a large simulation can profitably use hundreds or even thousands of cores.

Note that MPI is not configured for the distributions of G4beamline. It is too system-specific for that. You must install some MPI implementation and then build G4beamline from source.

G4beamline implements MPI when the argument “—enable-mpi” is given to *configure* when building the program. While it is a general implementation compliant with MPI 2, it has been tested only on the following systems:

- Mac OS X 10.7.5 with OpenMPI [11] installed, on a Mac Pro with 4 cores.
- The supercomputer Hopper, at NERSC [12].

On other systems it may be necessary to modify the *g4blmpi* script. Using MPI on a Linux box or cluster ought to be very similar to Mac OS X; Hopper runs a supercomputer version of Linux.

G4beamline is written such that if you just run it by itself, via the *g4bl* script, via the *g4blgui* program, or via an icon, it runs without MPI as a single, local process. But if it is run via the *g4blmpi* script, then it runs the specified number of instances and operates in MPI mode, via either *mpirun* or *aprun* (depending on which is available).

In MPI terminology, each instance of the program is called a “rank”; the ranks are enumerated by a non-negative integer unique to each instance of the program, assigned by the MPI system sequentially from 0. For G4beamline, rank number 0 is the master-controlling instance, and all other ranks are worker instances (nodes). The usual command is:

```
g4blmpi 5 input.file [... parameters]
```

This command acts very nearly like a single G4beamline program simulating *input.file*, except it uses 5 processes to speed up the computation. The basic operation is:

1. Setup the environment as in *g4bl*. This includes locating the shared objects and the *Geant4Data*.
2. Start 5 instances of the g4beamline program, using *mpirun*'s internal mechanism to start them.
3. Each instance of g4beamline determines that it was run via *mpirun*, and gets its rank number from the MPI library.
4. The rank 0 instance does the following:
  - a. Reads *input.file*, constructs the geometry, the physics list, etc.
  - b. Creates an output file for each NTuple (one file for all Root NTuples).
  - c. It loops, processing messages from worker nodes, of which there are three basic types:
    - i. A message indicating the worker is idle; rank 0 responds by sending it a block of beam tracks to process, sending 0 tracks if none are left.
    - ii. A message containing NTuple rows; rank 0 responds by writing them to the appropriate NTuple file.
    - iii. A message indicating the worker took an exception; rank 0 prints it (throttled as usual for many identical exceptions). If the exception is fatal, that is communicated to all worker nodes, and the program terminates gracefully.
  - d. When all events have been processed, and all worker nodes have indicated they have exited, rank 0 terminates.
5. The instances with rank > 0 do the following:
  - a. Reads *input.file*, constructs the geometry, the physics list, etc.
  - b. Configures all NTuples to send their rows to rank 0 rather than writing to any file.
  - c. Sends a message to rank 0 indicating it is idle.
  - d. Waits for a message from rank 0 containing a block of beam tracks to process.
  - e. If the message contains 0 beam tracks, sends any un-sent NTuple rows to rank 0 (i.e. flushing their buffers), and then exits.
  - f. Otherwise, processes the received beam tracks.
  - g. Loops back to c.

Error and exception handling is not mentioned above; it significantly complicates the code. G4beamline makes a valiant attempt to always closeup gracefully, but when the system terminates the job that may not be possible. Results are usually lost when a worker node runs out of memory, or when the job exceeds its wall clock limit.

The following parameters control the MPI implementation:

<b>MPI_PackTracks</b>	Number of beam tracks per message (default=100).
<b>MPI_PackNTuples</b>	Number of NTuple rows per message (default=1000).
<b>MPI_IdleSleep</b>	Sleep time (milliseconds) of the sleep in a loop calling <code>MPI_Iprobe()</code> ; see below.
<b>MPI_WorkerOutput</b>	Controls whether each worker writes to rank#.out (0 = not). Default = 1. Rank 0 always writes to stdout and stderr; worker nodes never do.
<b>MPI_Debug</b>	Controls MPI debugging output. Default = 0.

In addition, the parameter **MPI\_rank** is set to the rank of the process; as each rank processes the *input.file*, this can be used for debugging or distinguishing output files. It is set to -1 if MPI was built but the run is in non-MPI mode. It is not set if MPI was not configured and built.

Rank 0 writes to stdout and stderr in the usual way; all other ranks have both stdout and stderr redirected to files ‘rank#.out’, or closed if *MPI\_WorkerOutput=0*. G4Exceptions will be displayed on both rank#.out and stdout, and the rank 0 process totals them over all ranks. stdin is normally readable by rank 0, but that won’t work as an *input.file* (because all ranks must read it).

Note that rank 0 only does management tasks, and performs no tracking of beam. On a Mac Pro with 4 cores, it is appropriate to run 5 ranks, as rank 0 needs only a few percent of a core. On a supercomputer using dozens or hundreds of cores, rank 0 should have its own core. By default, MPI message passing does not block, in order to reduce message latency. This means that rank 0 will use 100% of a core, even though it does very little processing. On a supercomputer with hundreds of ranks working that is appropriate, but it is wasteful on a small computer with just a handful of cores. Setting *MPI\_IdleSleep=1* will make rank 0 sleep for 1 millisecond when it is idle (i.e. is not processing a message from a worker). On a 4-core computer running 5 ranks, this can improve throughput by about 20%. Larger values only reduce throughput.

There is little point in changing either *MPI\_PackTracks* or *MPI\_PackNTuples*; larger values do not help, and smaller values only reduce throughput.

The parallel performance of G4beamline on your simulation will depend on a) the speed of the CPUs, b) the speed and latency of the network among them, and c) the size of the system being simulated. See section 8.5.2 below.

Some aspects of G4beamline have their operation modified in MPI mode:

<i>steppingVerbose=1</i>	Performed independently on each rank, writing to files ‘rank#.out’ (unless <i>MPI_WorkerOutput=0</i> ).
<i>fieldntuple</i> <i>printfield</i> <i>probefield</i>	Performed only in rank 1, to avoid duplication. (These commands are not normally used in MPI mode, as they are primarily for debugging.)
<i>trace</i>	Performed only in rank 1, to avoid duplication. In ASCII format, the annotations are lost (blank lines between tracks, and a comment giving EventID and TrackID).
<i>profile</i> <i>totalenergy</i>	Sum up over all workers at the end of the run, before printing from rank 0.
<i>printf</i>	Always prints from rank 0 (via messages from workers). As long as each <i>printf</i> command is < 1 kByte, and all <i>printf</i> -s for a single event are < 32 kBytes, the output from different events will not be intermixed (but the order of events is not preserved).
<i>spacecharge</i> <i>spacechargelw</i> <i>collective</i>	Collective mode is incompatible with MPI. This will probably change in the future.
Visualization	Visualization is incompatible with MPI. A single build can enable both visualization and MPI, but they cannot be used together.
Windows	MPI is not supported on Windows.

### 8.5.1 Signals

G4beamline responds to signal *SIGUSR1* by flushing all NTuples to disk. So for a long job you can look at intermediate results.

G4beamline responds to signal *SIGUSR2* by closing up and exiting cleanly. In each worker this is detected between events. This is especially useful on Hopper, as *SIGUSR1* and *SIGUSR2* are the only signals users can send to the application; all others cause immediate termination without saving results.

### 8.5.2 Estimating how many workers to use

In general, if your simulation runs >100 events/sec on a typical core, then MPI will probably max out around 10-20 ranks, because the serial operations in rank 0 will saturate. For a simulation that runs 13 ev/sec on a Hopper worker node, rank 0 saturates around 800 workers. From the parallelization plots in section 8.5.4 below, it's clear that on Hopper rank 0 saturates around 8,000 events per second. There are several design and code changes that could be done to mitigate the rank 0 bottleneck, but for now this is not considered serious enough to warrant development.

Note that a larger number of messages per event will reduce the number of workers that can be used effectively, because the serial I/O in rank 0 will dominate. So for high performance, avoid *printf*, *steppingVerbose=1*, *trace*, and a large number of NTuples.

There can be rather large overheads in starting each MPI process; on Hopper, 1600 ranks can take several minutes to start up (from rank 0 starting to the last worker processing its first event). Startup is loosely but not strictly in rank order.

The best way to estimate how many workers to use is to plot the performance (events per second) as a function of the number of workers. When rank 0 becomes saturated, the events per second will likewise saturate. Usually a few minutes will suffice for each point, and you can use the *wallClockLimit* parameter to terminate the jobs. Be sure it is large enough for all workers to start up – if they don't all start up, the symptom is a droop in the Events/sec vs. nWorkers plot (as in Benchmark2 on hopper, below, for 1600 workers). Another symptom of not all workers starting up would be an exception “Not all Workers Contributed Data”, for the *profile* or *totalenergy* commands.

### 8.5.3 MPI on Mac OS X with OpenMPI

Here is the build of OpenMPI:

```
./configure --disable-dlopen --enable-static --disable-shared \  
--disable-mpi-f77 --disable-mpi-f90 --prefix=/usr/local  
make  
make install
```

If you have other systems on a network that could be used via MPI, install OpenMPI on them and configure it to use them. My instance of OpenMPI only uses the cores on my machine, and message passing is via shared memory.

After installing OpenMPI, build G4beamline from source:

```
mkdir G4beamline-2.14-mpi  
cd G4beamline-2.14-mpi  
../G4beamline-2.14-source/configure --enable-mpi  
make
```

Note the make system must be able to find MPI on its own; if *mpicxx* is available, the Makefile uses it.

The program runs in single-core mode with visualization (via *g4bl*, *g4blgui*, or *icon*), or in MPI mode (via *g4blmpi*). Using a visualization viewer in MPI mode is not supported.

The best performance on my Mac Pro with 4 cores is with 5 ranks and `MPI_IdleSleep=1`.

### 8.5.4 MPI on [hopper.nersc.gov](http://hopper.nersc.gov)

On Hopper you must select which compiler to use, both for building and for running. For G4beamline, the best compiler to use is gnu:

```
module load PrgEnv-gnu
export CC=cc CXX=CC CPPFLAGS="-dynamic"
mkdir G4beamline-2.14-mpi
cd G4beamline-2.14-mpi
../G4beamline-2.14-source/configure --enable-mpi --disable-visual
make
```

Note the export of variables CC and CXX, to prevent the Makefile from using the defaults (which are an old version of gcc and g++ which you do NOT want to use). MPI is already known by the compiler.

Be sure to do “module load PrgEnv-gnu” in your batch script to *qsub* (it is needed to find shared objects).

On Hopper, use the default `MPI_IdleSleep=0`.

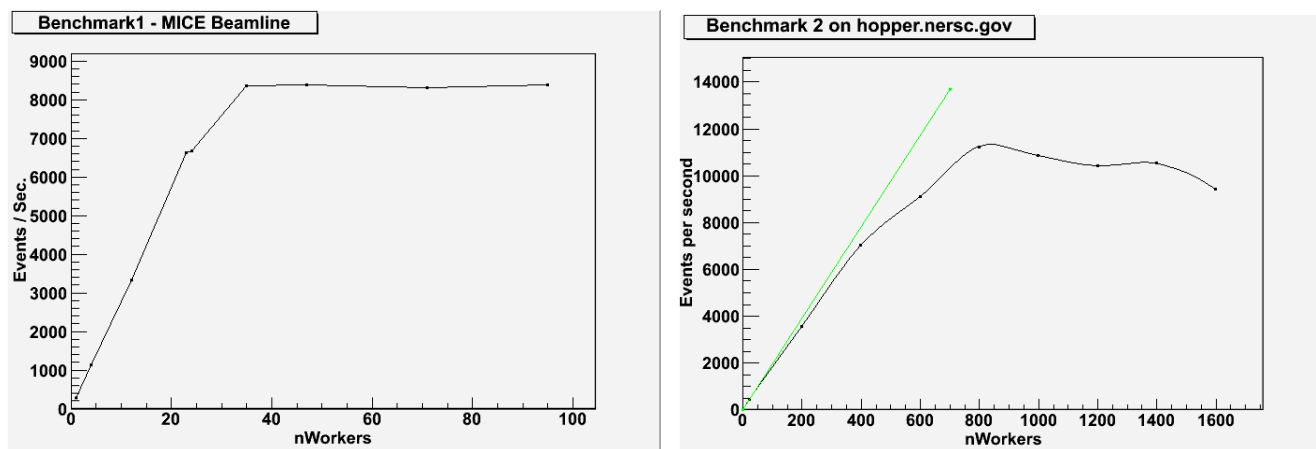
**NOTE:** Hopper does not have a very good method of detecting when memory is exhausted on a worker node (even though G4beamline itself does). You really want to avoid running out of memory, as Hopper does not permit the job to closeup gracefully, and all results will probably be lost. The message “OOM killer terminated this process” indicates the job was killed due to a worker node being Out Of Memory.

If you are part of a project that uses G4beamline, there may already be an installation of G4beamline for your project. For the Muon Accelerator Program (MAP), look in `/project/projectdirs/map/Codes`.

Here is an example script for submission via *qsub*:

```
#PBS -q debug
#PBS -l mppwidth=24
#PBS -l walltime=00:30:00
#PBS -N Benchmark
#PBS -j eo
source /project/projectdirs/map/bin/compiler gnu
source /project/projectdirs/map/Codes/G4beamline/bin/g4bl-setup.sh
cd $PBS_O_WORKDIR
g4blmpi 24 Benchmark1.g4bl wallClockLimit=240
```

These parallelization plots were made on [hopper.nersc.gov](http://hopper.nersc.gov), using the scripts in the Benchmark directory of the distribution.



**Hopper Parallelization Graphs: Benchmark1 and Benchmark2.**

## 8.6 G4beamline Helper Programs and Scripts

G4beamline requires a large number of environment variables to be set, so users rarely run the *g4beamline* command directly, but rather execute it via a helper program or script.

The helper programs are all written in Java and run in all environments that G4beamline itself can run. Their Java *.class* files are located in *share/g4beamline* of the installation. The Java runtime environment (JRE) must be installed.

The helper scripts are usable in these environments:

1. Linux
2. Mac OS X
3. Windows with Cygwin

The scripts rely on the *bash* shell, and are therefore not runnable on Windows alone. They are located in the *bin* directory of the installation. Note, however, that the Java programs provide the same functionality as the scripts, except for MPI (which is inherently command-line).

### 8.6.1 Helper Programs

- |                 |  |
|-----------------|--|
| <i>G4blGui</i>  | Provides a Graphical User Interface to G4beamline. It sets up all required environment variables, and runs <i>g4beamline</i> with or without visualization. The standard output and error streams are displayed in a scrollable pane in the window. This is the standard interface on Windows and Mac OS X, and is available on Linux. |
| <i>G4blData</i> | Manages the <i>Geant4Data</i> directory, which contains the Geant4 data files necessary for the different physics lists. It provides a GUI that lets the user specify the location of the directory, select and download the data sets from the CERN website, and unpack them into the specified destination directory.                |

### 8.6.2 Scripts

- |                      |   |
|----------------------|---|
| <i>g4bl-setup.sh</i> | A script to put the G4beamline scripts into the user's PATH, for the sh class of shells ( <i>sh</i> , <i>bash</i> , <i>ksh</i> , <i>zsh</i> , etc.); <u>must</u> be source-d. |
|----------------------|---|

<i>g4bl-setup.csh</i>	A script to put the G4beamline scripts into the user's PATH, for the csh class of shells ( <i>csh</i> , <i>tcsh</i> , etc.); <u>must</u> be source-d.
<i>g4bl</i>	This is the standard script to run G4beamline. It determines the install directory from the path by which it is run. It then sets up the required environment variables and runs <i>g4beamline</i> with or without visualization. The standard output and error streams are as usual in a UNIX environment.
<i>g4blmpi</i>	This script runs G4beamline in an MPI environment, applying multiple instances of <i>g4beamline</i> to a single problem. Its first argument is how many instances (ranks) to use; the remaining arguments are the same as for <i>g4bl</i> . See section 8.5.
<i>g4blgui</i>	A script to run the <i>G4blGui</i> program from the command-line.
<i>g4bldata</i>	This script manages the <i>Geant4Data</i> directory and files. Runs the <i>G4blData</i> program for installation.
<i>g4blmake</i>	A user command to build <i>g4beamline</i> with additional user-written code.
<i>g4bl-config</i>	A script for querying the G4beamline configuration. Rarely if ever run directly by users.
<i>g4blmovie</i>	A script to create a movie. See section 8.3.
<i>g4bl_debug</i>	A script for debugging G4beamline. Useful only when built from source.

### 8.6.3 Implementation Details

Each script or program must determine various aspects of the environment required to run the *g4beamline* program:

<i>Install Directory</i>	The environment variable <i>G4BL_DIR</i> points to the install directory. Java programs use <code>Util.findG4blDir()</code> , while it is hard coded into each script.
<i>ROOTSYS</i>	The Root library works better when this is set to <i>G4BL_DIR</i> .
<i>Physics Datasets</i>	The Geant4 physics processes need various data. The <i>g4bldata</i> script manages this, as do the Java programs via <code>Util.findGeant4Data()</code> . Once installed, the <i>.data</i> file in the install directory points to the directory containing the Geant4Data. That directory contains a <i>setup.sh</i> script which sets the environment variables the Geant4 library expects, plus directories containing the various datasets.
<i>Library Search Path</i>	On Linux and Mac OS X the <i>lib</i> directory contains shared libraries. On Windows they are in <i>bin</i> .

## 8.7 Setting up an rfdevice

It is worth repeating that G4beamline defines  $0^\circ$  RF to be the rising slope of the zero crossing of the RF waveform, so  $90^\circ$  RF is on-crest for a positive particle, while  $270^\circ$  RF is on-crest for a negative one. This is a common, but not a universal convention.

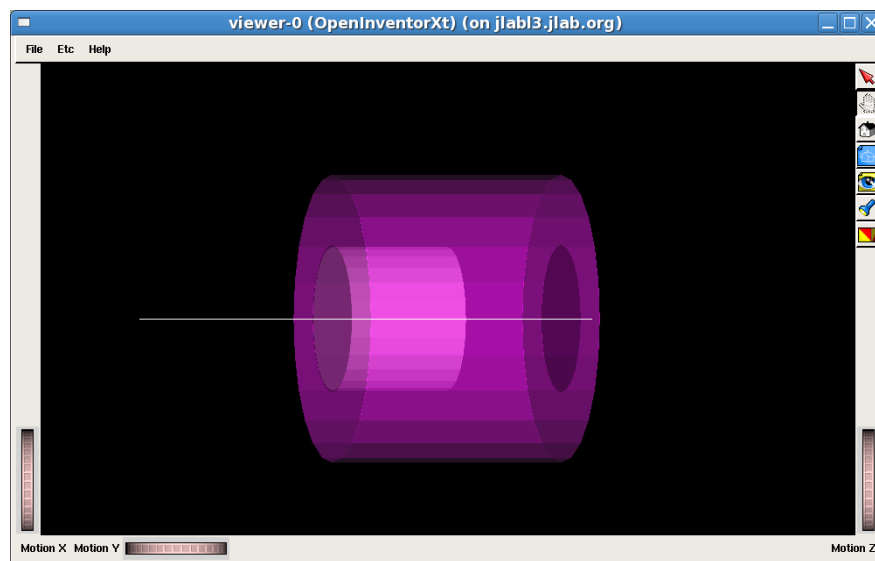
For many applications, especially very relativistic beams, setting up the RF is very straightforward. For others, such as muon cooling, the beams are relatively slow, phase and gradient are coupled, and compensating for energy loss in absorbers is the goal rather than simple acceleration.

There are a number of ways to setup the RF. This autoTiming process is local to a single *rfdevice*, while the *tune* command uses information downstream of the rfdevice. A brief how-to on timing the RF is in section 7.34 and adjusting the gradient using *tune* is in section 7.35.

Everything is totally determined once *timeOffset* and *maxGradient* and the fields are known. The timing process described here may be used to determine *timeOffset* and/or *maxGradient* of a single RF device, and may then derive *phaseAcc* and other quantities.

Inside every rfdevice is a *timingVolume*. This object is a cylinder with one face close to the entrance and the other at the point that will be used to determine the timing; almost always either at the center of the cavity (the default when using *timingMethod=atZlocal*) or at the exit (all other methods). This *timingVolume* is immaterial, but it is an object and ought not overlap any other object. It will be visible in the graphics if *timingDisplay* is nonzero.

```
rfdevice RF1 innerLength=744 color=0.8,0,0.8,0.4 \  
  frequency=0.20125 maxGradient=25 phaseAcc=70 \  
  innerRadius=450 pipeThick=1 wallThick=1 irisRadius=230 \  
  collarRadialThick=0 collarThick=0 win1Thick=0 win2Thick=0 \  
  winMat=Vacuum winMat=Vacuum cavityMaterial=Vacuum \  
  maxStep=1 timingMethod=atZ timingDisplay=1
```



The timing process ignores everything outside of the *timingVolume*. The timing process allows one to specify 2 out of 3 conditions (*maxGradient*, *timeOffset*, and fixed outputs) and it will determine the 3<sup>rd</sup> condition, if possible. If it cannot find a solution, G4beamline will exit with an error message.

For deflecting cavities, the *timingVolume* is still aligned so the Tune particle enters one face and exits the other, but the trajectory needs not be normal to the face.

The relationship of the particle and the RF fields is a delicate one; the particle must arrive at the correct point in the RF cycle and the RF must have the correct voltage. For slow particles, there is an interrelationship between the phase and voltage of a cavity, so changing the *maxGradient* may require readjusting the timing.

The most basic way to setup an RF cavity is to just explicitly set its *timeOffset*, the absolute time by which the fields are translated. *timeOffset* includes all phase information. The electric field is usually of the form:

$$E(x,y,z) * \sin( \omega (t - timeOffset) )$$

The rfdevice's *UserSteppingAction* routine will try to find a consistent solution based on what it is told; both under- or over- specifying the system are fatal errors. A Tune (Event# -2) particle is launched from a spot near the entrance of the *timingVolume* for each pass within the rfdevice.

If explicitly specified, *timeOffset* may not be altered by any subsequent requirement, and any attempt to do so will result in an error message. If unspecified, *timeOffset* will be determined according to the set *timingMethod* and the subsequent requirements given to *rfdevice*, and then finally shifted by *timeIncrement*.

A method (*timingMethod*) launches multiple Tune particles (Event# -2) from a spot near the entrance of the *timingVolume* and uses them to determine the time associated with 0° RF. Once that is complete, a Tune particle then makes one pass at that setting. Messages for each step in the procedure will be printed if *timingDisplay=1*, and additional messages will be shown if *timingDisplay=2*. The process will continue until changes in the base timing are less than an associated tolerance (*timeTolerance*); the default is 0.001 ns. The tracking step size within the rfdevice ought to be set reasonably small; usually 1 mm works well.

Several methods are provided to determine the base timing; only for special cases of deflecting cavities or very slow particles will the methods give much differing results. The default method is *timingMethod=atZlocal*, where the timing is adjusted so the particle would arrive at the (local to the rfdevice) *timingZlocal* location (default is 0 mm) inside the rfdevice at the appropriate phase (this was used in earlier versions of G4beamline). In actual practice, however, most people adjust a real cavity for maximum energy gain (*timingMethod=maxEnergyGain*), corresponding to 90° RF for a positive particle and 270° RF for a negative particle. When using a multicell cavity, the latter method is usually preferable.

For special cases, including deflecting cavities, other methods of determining the base timing are available as shown in the following table. Here the transit time is considered relative to the drift time, so *timingMethod=noTransitTime* corresponds to the same transversal time as a simple drift.

### Available *timingMethods* for determining 0° RF

method	abbreviation	default <i>timingAtZ</i>	description
<i>atZlocal</i>	<b><i>atZ</i></b>	middle	used by G4beamline 2.08 and earlier, generally not used with fix* output requirements
<i>maxEnergyGain</i>	<b><i>maxE</i></b>	exit	most common method for most RF
<i>noEnergyGain</i>	<i>noE</i>	exit	for RF bunchers
<i>minEnergyGain</i>	<i>minE</i>	exit	for decelerating RF
<i>maxTransitTime</i>	<i>maxT</i>	exit	for special applications
<i>noTransitTime</i>	<i>noT</i>	exit	for special applications
<i>minTransitTime</i>	<i>minT</i>	exit	for special applications
<i>maxXdeflection</i>	<i>maxX</i>	exit	for deflecting cavities - X in local rfdevice's coordinates
<i>noXdeflection</i>	<i>noX</i>	exit	for deflecting cavities – X in local rfdevice's coordinates
<i>minXdeflection</i>	<i>minX</i>	exit	for deflecting cavities – X in local rfdevice's coordinates
<i>maxYdeflection</i>	<i>maxY</i>	exit	for deflecting cavities – Y in local rfdevice's coordinates
<i>noYdeflection</i>	<i>noY</i>	exit	for deflecting cavities – Y in local rfdevice's coordinates
<i>minYdeflection</i>	<i>minY</i>	exit	for deflecting cavities – Y in local rfdevice's coordinates

After the base timing is determined, the RF phase (section 7.20) may be explicitly set via *phaseAcc* or be determined by specifying *maxGradient* and a single fixed output quantity (*fixEnergyGain*, *fixMomentum*, *fixTransitTime*, *fixXdeflection*, or *fixYdeflection*); the phase is then adjusted to satisfy the conditions. If a fixed output quantity and the *phaseAcc* are set, but not *maxGradient*, both *timeOffset* and *maxGradient* will be adjusted to satisfy the requirements (section 7.21).

### Fixed output conditions

fixed output request	units	description
----------------------	-------	-------------

<i>fixEnergyGain</i>	MeV	energy gain during transit of timingVolume
<i>fixMomentum</i>	MeV/c	momentum at the exit of the timingVolume
<i>fixTransitTime</i>	nS	total transit time of the timingVolume
<i>fixXdeflection</i>	deg	change of direction toward local X
<i>fixYdeflection</i>	deg	change of direction toward local Y

The Tune particle is repeatedly tracked through the cavity as many times as necessary to determine the timing to the *timingTolerance* set accuracy. Independent of which method is used to determine the base timing, *timeOffset* may be further adjusted to satisfy the requested conditions. The total number of Tune particles/rfdevice is typically between 5 and ~200. To avoid logging them when using *trace*, specify *trace keepTune=0*.

For deflecting cavities, the deflections are in the coordinate system used to define a cavity, not the global coordinate system, to avoid ambiguities when placing a cavity in the world.

If the final error on the requested output condition exceeds *fixTolerance* (the default is 1.E-3 in the same units as the fixed output), G4beamline will exit with an error message.

Though it is rarely used, *timeIncrement* is intended for exploring small phase errors of the RF. If *timeIncrement* is set nonzero, it is added to *timeOffset* after all other timing is complete, and the Tune particle is then re-tracked. If the error on a fixed output condition then exceeds *fixTolerance*, G4beamline will not report it as an error.

## Timing Algorithm

G4beamline launches a Tune particle; after each step when it is inside the *rfdevice* (section 5.49) *RfdeviceField::UserSteppingAction* is called. That routine may make changes to the *rfdevice* and then push the particle back to the point where it first entered the *rfdevice* and the repeat the process. Often multiple Tune (Event# -2) particles, or *passes*, are used to setup the RF device. The routine's only points of interest are the entrance and exit of the *timingVolume*. The routine is a state machine associated with a single RF device described by *rfdevice*; the state is set before returning to G4beamline.

*UserSteppingAction* goes through a number of states. Initially, upon first entering the *rfdevice*'s *timingVolume*, it is in the *ATWORKING\_UNKNOWN* state; if sufficient information exists to completely define the system, i.e. *timeOffset* and *maxGradient* both set explicitly, the state is set to *ATWORKING\_DONE*. Otherwise, if *timeOffset* is set explicitly, the state is set to *ATWORKING\_BYPASS*, and if not it is set to *ATWORKING\_OFFSET*. If no *maxGradient* is specified, it is set to an estimated reasonable working value.

Upon entering the *rfdevice*'s *timingVolume*, various initializations are done, a snapshot is taken of particle and the track is saved at that point. Control is returned to G4beamline and the particle continues, with *UserSteppingAction* called each step, but it does nothing until the particle exits the *timingVolume*. Then another snapshot is taken, various calculations are done, the conditions and

state changed, and the track returned to the spot just inside the entrance of the *timingVolume* via the saved track. Generally only one Tune particle enters the *timingVolume*, but many leave it.

It is important the *maxStep* used by the tracking not be too large. The default value is usually too large, while ~1mm seems to work well.

If the state is `ATWORKING_OFFSET`, the 1<sup>st</sup> pass is done with *timingZero*=0, the 2<sup>nd</sup> at 90° RF later. The output  $\Delta E$ ,  $\Delta(t-t_{\text{drift}})$ , or  $\Delta\theta_{x,y}$  may be approximated as:

$$O = A \cdot \sin(\omega t + \varphi)$$

so taking the first two passes:

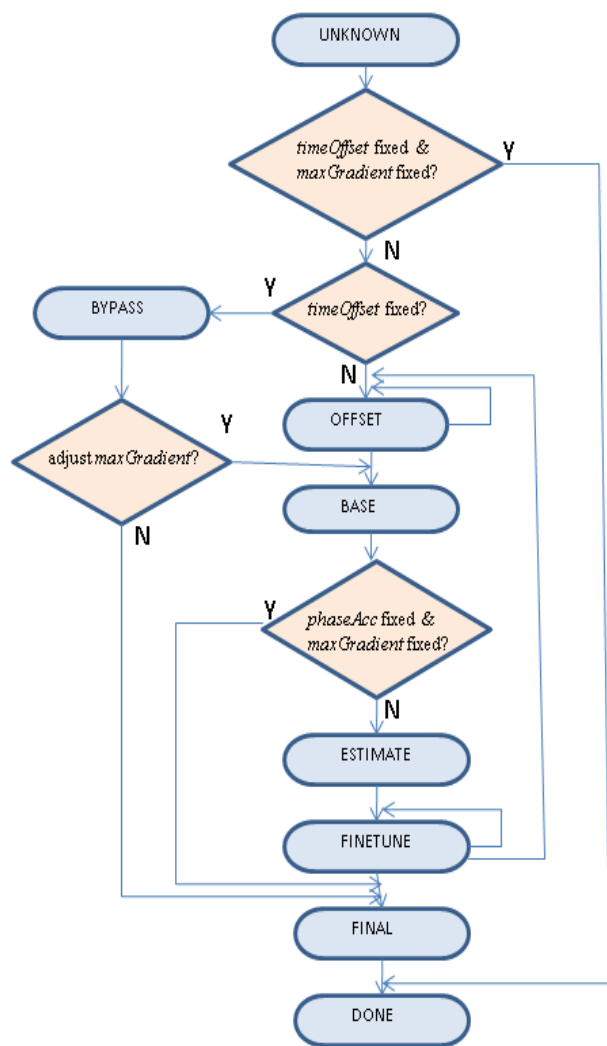
$$\begin{aligned} O_1 &= A \cdot \sin(\varphi) \\ O_2 &= A \cdot \sin(\pi/2 + \varphi) = A \cdot \cos(\varphi) \end{aligned}$$

gives:

$$\begin{aligned} \varphi &= \text{atan}(O_1/O_2) \\ A &= \sqrt{(O_1^2 + O_2^2)} \end{aligned}$$

`rfdeviceField::UserSteppingAction` algorithm logic states

timing state	description
<code>ATWORKING_UNKNOWN</code>	initial state – no processing done yet
<code>ATWORKING_OFFSET</code>	multiple pass search defines 0° RF
<code>ATWORKING_BYPASS</code>	iff total timing explicitly fixed
<code>ATWORKING_BASE</code>	single pass at 0° RF
<code>ATWORKING_ESTIMATE</code>	1 <sup>st</sup> estimate to match requirements
<code>ATWORKING_FINETUNE</code>	fine adjustments to match requirements
<code>ATWORKING_FINAL</code>	final pass with final parameters
<code>ATWORKING_DONE</code>	complete state – all processing done



When using *timingMethod=atZlocal*, the 3<sup>rd</sup> pass onward during state=ATWORKING\_OFFSET adjusts the arrival time at *timingAtZ* to be 90° RF for a positive particle and 270° RF for a negative one. For all other methods, the 3<sup>rd</sup> pass places the system near the method's condition and a simple search is conducted around it. The search continues until the time steps are smaller than *timingTolerance*.

These are used to place the system near the *timingMethod*'s desired condition; a simple search then adjusts the value of *timingZero* to satisfy the condition to the desired *timingTolerance* accuracy, and then does a single pass for state ATWORKING\_BASE. While the algorithm is working, *timeOffset* = *timingZero*+*timingPhase*/ω.

If the state is ATWORKING\_BYPASS and *maxGradient* was fixed, the state is changed to ATWORKING\_FINAL and a final pass is done; otherwise, it is changed to ATWORKING\_BASE.

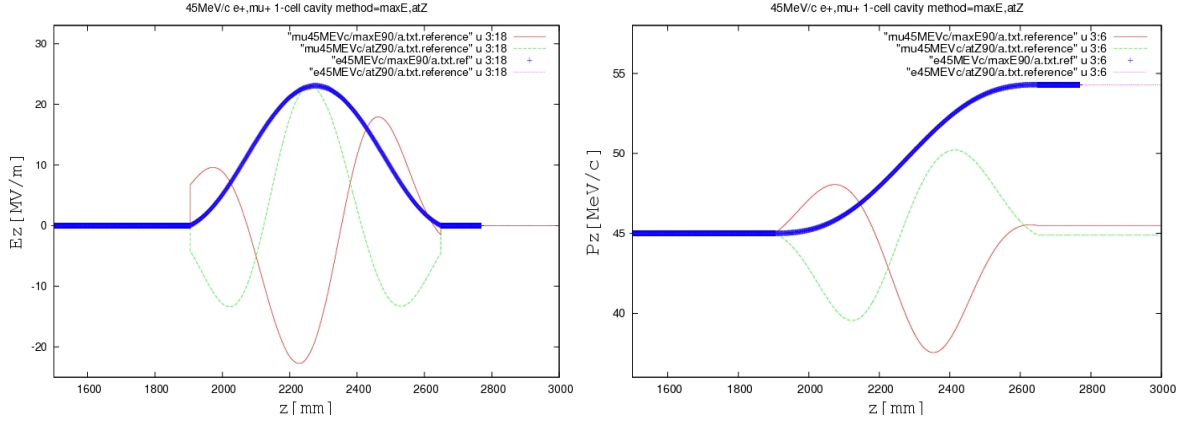
State ATWORKING\_BASE just shows the path of the particle at 0° RF for verification. If *phaseAcc* was set, it is added onto *timeOffset* and the state changed to ATWORK\_FINAL; otherwise, an estimate based on the output amplitude and phase from the first 2 passes is used to generate appropriate values for ATWORKING\_ESTIMATE.

After a single pass of state ATWORKING\_ESTIMATE, the state is changed to ATWORKING\_FINETUNE and the routine searches for a nearby solution. If one is found, the state is changed to ATWORKING\_FINAL. If *maxGradient* was a guess during the search for 0° RF, the whole process is repeated from ATWORKING\_OFFSET using the new value of *maxGradient* as there may be some dependence of the phasing on the gradient.

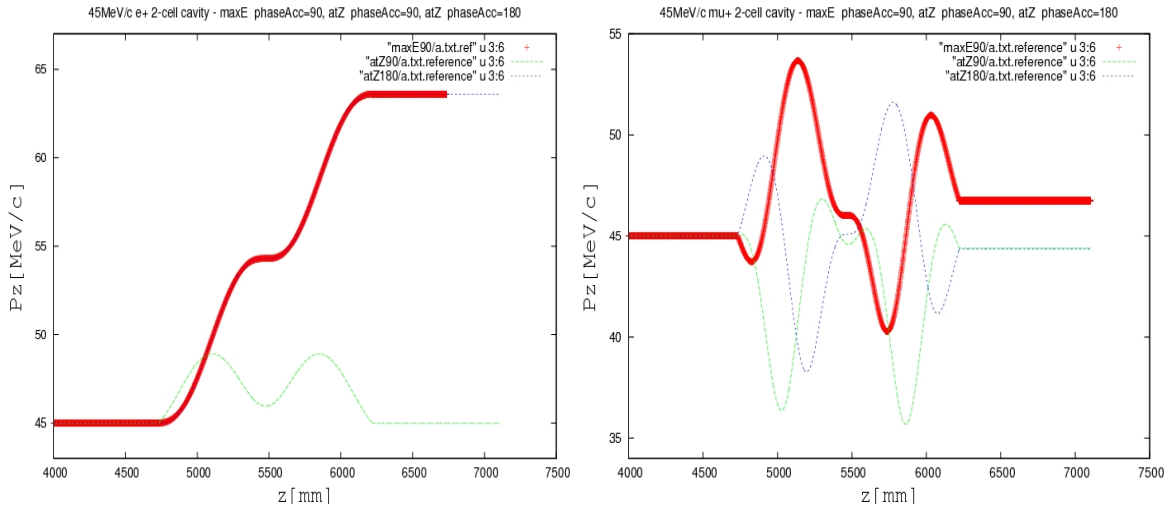
In the very rarely used case that *timeIncrement* is set nonzero to study timing errors, *timeOffset* is changed by that amount before the last pass without any further adjustments and the final *fixTolerance* test is disabled for the ATWORKING\_FINAL.

The state ATWORKING\_FINAL just tracks the particle through the rfdevice using the final timing and *maxGradient*; this is important to ensure the next device sees the correct Tune particle. The state is then changed to ATWORKING\_DONE. If the state is ATWORKING\_DONE, the routine always returns and does nothing.

The choice of *timingMethod* becomes important when particles are moving slowly. For illustration, compare the reference particle tracks through a single 201.25 MHz, 744 mm cavity for 45 MeV/c ( $\beta \approx .39$ )  $\mu^+$  and ( $\beta \approx 1$ )  $e^+$  with *maxGradient*=25 MV/m and *phaseAcc*=90° RF using *timingMethod*=*maxE* and *atZ*. The  $e^+$  trajectories are virtually identical for both methods, while the  $\mu^+$  trajectories display a distinct difference.



For a multicell cavity with an even number of cells the differences are much more pronounced, as the center of the cavity will usually be a node in the field map. For example, consider a similar 2-cell cavity again with 45 MeV/c  $\mu^+$  and  $e^+$ . For  $e^+$ , *timingMethod=atZ atZlocal=0 phaseAcc=90* results in no energy gain while *timingMethod=atZ atZlocal=0 phaseAcc=180* gives the maximum energy gain, the same as one gets using *timingMethod=maxE phaseAcc=90*. For  $\mu^+$ , the situation gets complicated, but *timingMethod=maxE phaseAcc=90* still returns the highest energy gain:



## Examples

The old rfdevice set timing based on the middle of the cavity (for backward compatibility *timingMethod=atZ* is still the default, but usually ought not to be used with fixed output options), setting the maximum gradient over the RF cycle to 25 MV/m and placing a positive particle on-crest:

```
rfdevice rf1 ... maxStep=1 maxGradient=25 phaseAcc=90
```

A typical case where a positive particle will see some synchrotron motion (later particles see more acceleration, 20° before crest):

```
rfdevice telsa9cell ... maxStep=1 timingMethod=maxE \
    phaseAcc=70 maxGradient=40
```

while to get the same synchrotron motion for a negative particle:

```
rfdevice tesla9cell ... maxStep=1 timingMethod=maxE \
    phaseAcc=250 maxGradient=40
```

To specify the 8° before (negative) crest for e<sup>-</sup>, but request that the output momentum be set to 3 GeV/c:

```
rfdevice cebaf7cell ... maxStep=1 timingMethod=maxE \
    phaseAcc=262.0 fixMomentum=3000
```

Success ensures that the found *maxGradient* satisfies both  $|\Delta timeOffset| < timingTolerance$  and  $||P| - fixMomentum| < fixTolerance$ .

## Building a Cryomodule

Complicated assemblies may be constructed in G4beamline using groups and macros. Since various quantities often must be often passed to internal components, macros are used here to construct a cryomodule.

A rfdevice need not have material walls – it may serve as merely a place to put a known RF field map. This example rfdevice has no physical components, but does have a field and a *timingVolume*:

```
rfdevice RF2 innerLength=$Ldoublecell_iris2iris-$c1 \
    frequency=$freqGHz fieldMapFile=RF2_1MV_wot.BLfmt \
    innerRadius=$innerPipe-$c1 pipeThick=0 wallThick=0 \
    irisRadius=$innerPipe-$c1 collarRadialThick=0 \
    collarThick=0 win1Thick=0 win2Thick=0 winMat=Vacuum \
    timingTolerance=0.000001 cavityMaterial=Vacuum \
    maxStep=1 timingMethod=maxE timingDisplay=1
```

Rfdevice RF2 represents a cylindrical space of length \$Ldoublecell\_iris2iris (reduced by a tiny clearance \$c1 to avoid overlap) with a RF field described by the file RF2\_1MV\_wot.BLfmt. A G4beamline visualization of RF2 would only show the *timingVolume* if *timingDisplay*!=0, and would not show anything at all with *timingDisplay*=0.

The physical SRF cavity wall may be described as a *polycone* (section 5.51) made of niobium:

```
polycone doublecell \
    z=$z2i0,$z2i1,$z2i2,$z2i3,$z2i4,$z2i5,...,$z2i21 \
    innerRadius=$r2i0,$r2i1,$r2i2,$r2i3,...,$r2i21 \
```

```

outerRadius=$r2o0,$r2o1,$r2o2,$r2o3,...,$r2o21 \
material=Nb

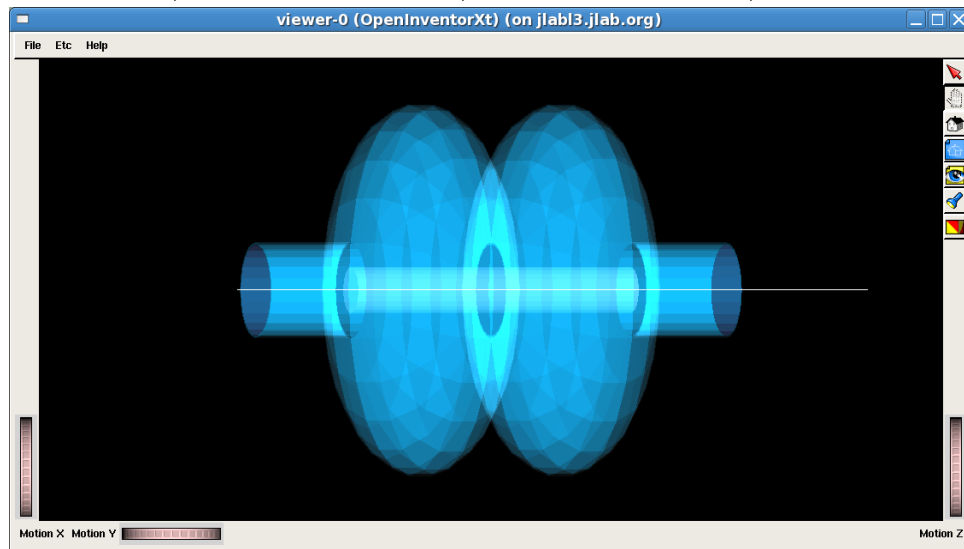
```

Both the *rfdevice* and *polycone* may be placed into a macro to keep them into the correct relationship. In this case both the *timeOffset* and *phaseAcc* will be determined by the timing algorithm to satisfy the condition that the exiting Tune particle have total momentum  $P_{out}$ .

```

param RFcolor=0,0.8,0.8,0.6
#                               $1           $2           $3
# doublecellAssembly <z{mm}>   <Pout{MeV/c}>   <V{MV/m}>
#
define doublecellAssembly \
    "place doublecell z=$1 color=$cellcolor " \
    "place RF2 z=$1 fixMomentum=$2 maxGradient=$3 rename=rf2_#

```



doublecellAssembly

The macro may be used as:

```

#                               <z{mm}>   <Pout{MeV/c}>   <maxGradient {MV/m}>
doublecellAssembly             1000       250.0          23.09

```

Another macro describing a superconducting solenoid, its field map, counterwound conductors, and iron shield may also be defined:

```

# solenoidAssembly <z{mm}>   <current{A}>
solenoidAssembly   500       1.5

```

Both macros, as well as a couple predefined *tubs* (section 5.77), may be placed inside yet another macro that constructs a whole cryomodule:

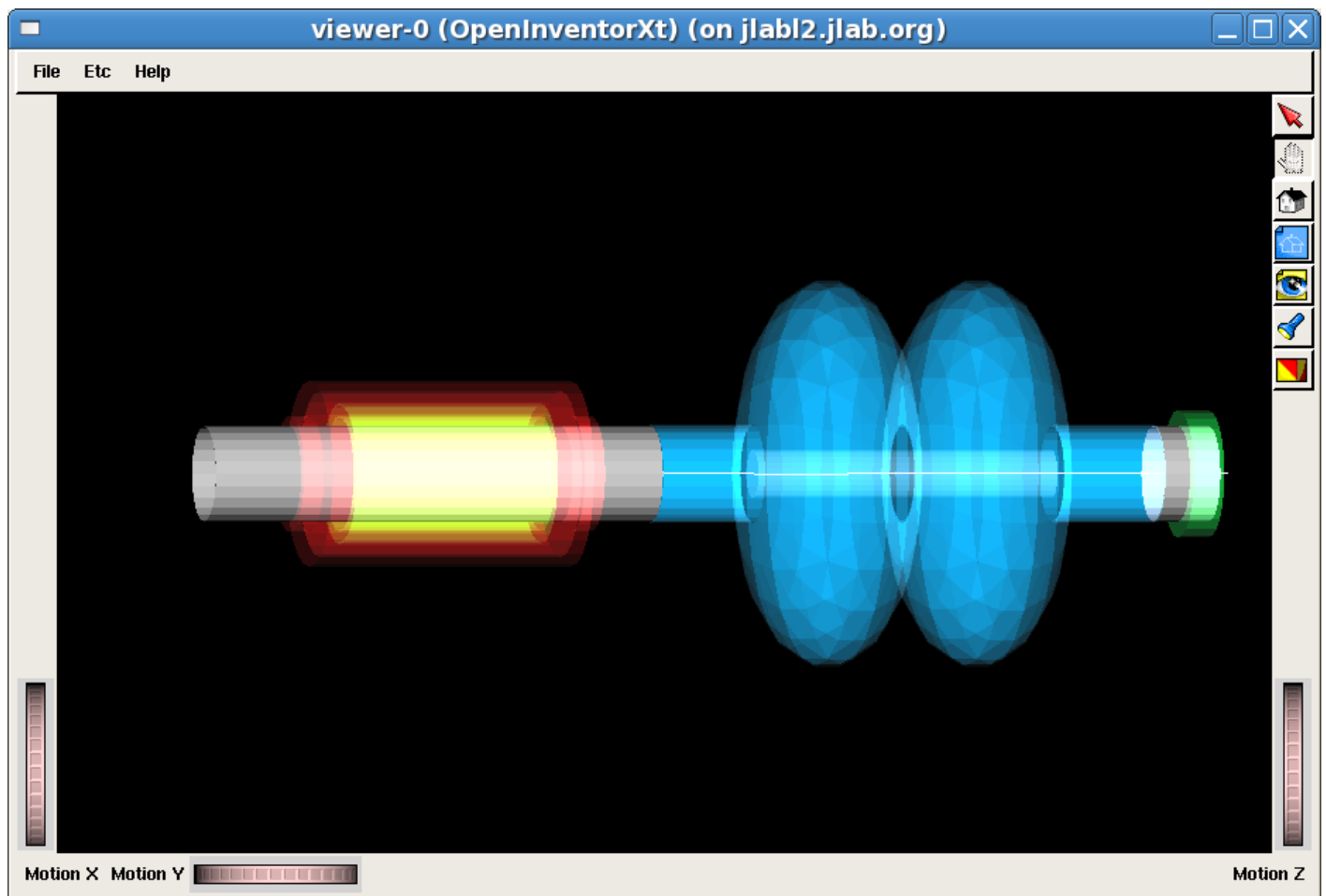
```

#                               $1           $2           $3           $4

```

```
# middleCryomodule <z{mm}> <current{A}> <Pout{MeV/c}> <maxGradient{MV/m}>
#
define middleCryomodule \
  "place beamPipe z=$1+$zo_miPipe length=$LmiPipe-$c1 rename=miPipe#\
  "solenoidAssembly $1+$zo_misolensoid $2 $5"\
  "doublecellAssembly $1+$zo_midoublecell $3 $4"\
  "place beamPipe z=$1+$zo_mivaPipe length=$LmivaPipe-$c1 rename=mivaPipe#\
  "place valveAssembly z=$1+$zo_mivalve
```

Note that macros are invoked, while objects are placed. For example, to put this cryomodule off the Z axis, one would need to modify the macro that defines the cryomodule to pass the x and y parameters to each of the individual *place* commands within each macro. Also note that when using field maps, the *current* and *maxGradient* quantities multiply the quantities within the field maps, rather than being absolute numbers.



middleCryomodule

To actually use the cryomodule:

```
#
#           <z>           <current> <Pout>   <maxGradient>
```

```
middleCryomodule $Zcryo9 $j9 $Po9 $Vlmax  
#
```

Other details, such as virtual detectors, cryostats, and absorbers may be added easily as required. In this example, the *polycone* lay outside of the *rfdevice*, however, if tracking particles through the cavity far off axis were required, the *rfdevice* could have been made larger and the *polycone* placed inside the *rfdevice*, but outside of the *timingVolume*.

## 9 File Formats

Note that ASCII file formats begin with comments ('#' in column 1) that give both the field names and their units. The *historoot* program is able to use these comments to give names to the fields of such files (it ignores the units, which are intended for humans).

### 9.1 BLTrackFile (generated by *virtualdetector*, read by *beam*)

```
*      The file format is ASCII:
*      Lines beginning with # are comments
*      First line is a structured comment (if not present the input
*      routine issues a warning):
*          #BLTrackFile ... user comment...
*      Second line is a comment giving the column names:
*          #x y z Px Py Pz t PDGid EvNum TrkId Parent weight
*      Third line is a comment giving the units:
*          #cm cm cm MeV/c MeV/c MeV/c ns - - - -
*
*      OR:
*          #mm mm mm MeV/c MeV/c MeV/c ns - - - -
*      (When writing, mm are used; on reading, mm are assumed, but a
*      comment line containing "cm cm cm" switches to cm.)
*      Thereafter follow the tracks, one per line. While the input
*      routine can handle initial spaces in the first column, it is
*      STRONGLY suggested you not put any there (so cut/grep will
*      work). Any fixed or floating-point format will do; PDGid,
*      EV#, TrkId, and Parent are integers (but .00 can be appended).
*      Common PDGid-s:
*
*          e-      11          e+      -11
*          mu-     13          mu+     -13
*          pi+     211         pi-     -211
*          proton  2212        anti_proton -2212
*          neutron 2112        anti_neutron -2112
*          gamma   22
```

### 9.2 Trace File

When writing track traces with *format=ascii* in the *trace* command, the output is as follows:

The first 12 columns are identical to those of BLTrackFile

Columns 13,14,15 give Bx,By,Bz in Tesla

Columns 16,17,18 give Ex,Ey,Ez in MegaVolts/meter

The first 3 comment lines are similar to those of BLTrackFile, giving a comment, the column names, and their units. All values are in the selected coordinates from the *trace* command (centerline, global, or reference).

### 9.3 FOR009.DAT

```
*      NOTE: Because the FOR009.DAT format requires the particles be sorted
*      by region (JSRG), all tracks are stored in memory until close() is
*      called; the file is written at that time. The ICOOL region is
*      determined from the Z position of the track, and is simply
*      sequentially assigned as tracks at different Z positions are written.
```

```

*   The region is +-REGION_SIZE in Z from the first track of each region.
*
*   XP[], PP[], BFLD[], and EFLD[] should all be converted to
*   Centerline coordinates before calling write().
*
*   This data format comes from ICOOL v 2.77, User's Guide section 4.2.3.
*   The first line of the file is the title.
*   The second and third lines are comments, supposedly units and column
*   labels.
*   There follow the tracks, one per line, sorted by "region". The first
*   track of each region should be the "reference" particle in ICOOL
*   parlance; in g4beamline it is the reference particle -- if multiple
*   reference particles intersect the region, the first will be
*   "reference", and the following ones will be considered "beam".
*   The variables for each track are:
*       IEVT          (I) event #
*       IPNUM         (I) track # for this event
*       IPTYP         (I) particle type: >0 for positive charge,
*                       <0 for negative
*                       1=e, 2=mu, 3=pi, 4=K, 5=proton
*       IPFLG         (I) "flag", always 0
*       JSRG          (I) region number (see above)
*       TP            (F) time (sec)
*       XP[3]         (F) position (meters)
*       PP[3]         (F) momentum (GeV/c)
*       BFLD[3]       (F) Magnetic field (Tesla)
*       EVTWT         (F) weight
*       EFLD[3]       (F) Electric field (V/meter)
*       SARC          (F) arclength (meter) -- set to 0.0
*       POL[3]        (F) spin -- set to 0.0
*
*   Note that event 0 will be ignored, as that value of IEVT is
*   reserved for the reference particle.

```

## 9.4 BLFieldMap

```

*   Blank lines, and lines beginning with # or * are comments. Lines
*   beginning with * are printed to stdout. Units are mm for coordinates,
*   Tesla for B, and MegaVolts/meter for E; use normB and normE if the
*   data points use different units.
*
*   The input file starts with a set of commands to define the parameters
*   of the map, followed by blocks of lines containing the values of the
*   field components. The field component names depend on the type of map
*   (grid: Bx,By,Bz,Ex,Ey,Ez; cylinder: Br,Bz,Er,Ez).
*   Each command has a specific list of arguments to define parameters
*   of the map.
*
*   BEWARE: the parsing is not exhaustive. For instance, invalid arguments
*   are silently ignored (which means you must verify the spelling and
*   capitalization of argument names). Correct inputs will yield correct
*   results, but invalid inputs may not be detected and may yield
*   seemingly-correct but unintended results.
*
*   The first command is usually a param command, which has the following
*   arguments:

```

```

*      maxline      The maximum number of characters per line (default=1023)
*      current      The current corresponding to this map (default=1.0)
*      gradient     The gradient corresponding to the map (default=1.0)
*      normE        A normalization factor for E components (default=1.0)
*      normB        A normalization factor for B components (default=1.0)
*
* Two types of maps are implemented: grid and cylinder.
*
* grid maps are a 3-D grid, with each block of data being a single
* X-Y plane; within a block the lines are Y and the columns of each line
* are values along X.
* The grid command has the following arguments:
*      X0           The X value for the first value in each line
*      Y0           The Y value for the first line in each block
*      Z0           The Z value for the first block of each field component
*      nX           The number of columns per line
*      nY           The number of lines per block
*      nZ           The number of blocks per field component
*      dX           The X increment between values in each line
*      dY           The Y increment between lines
*      dZ           The Z increment between blocks
*      tolerance    The tolerance for pointwise data (default=0.01 mm)
* After the grid command, the following optional commands can be given:
*      extendX flip=...
*      extendY flip=...
*      extendZ flip=...
* These commands permit a half-map to be extended to the full map
* around X=0, Y=0, or Z=0 respectively. The optional flip argument is a
* comma-separated list of field components whose signs will be inverted
* for negative values of the coordinate. For example, "extendZ flip=Bx,Ex"
* means the map from Z=0 to Z=(nZ-1)*dZ is extended symmetrically around
* Z=0 to negative Z values, flipping the signs of Bx and Ex when Z<0.
* This could be followed by "extendX flip=Bx,Ex", and the field flips
* will be the products of both commands.
*
* cylinder maps are a 2-D map with rotational symmetry around the Z axis.
* Each field component has a single block with lines being Z and the
* columns being R.
* The cylinder command has the following arguments:
*      Z0           The Z value for the first line in each block
*      nR           The number of columns per line
*      nZ           The number of lines per block
*      dR           The R increment between columns
*      dZ           The Z increment between lines
*      tolerance    The tolerance for pointwise data (default=0.01 mm)
* After the cylinder command, the following optional commands can be
* given:
*      extendZ flip=...
* This command behaves the same as for the grid map.
*
* After the commands, each block consists of a line containing the
* name of the field component, followed by the lines of the block.
* The values within a line can be separated by whitespace or a ','
* followed by optional whitespace.
* Field components that are not given are set to 0.0 everywhere.
* Missing values will be considered to be 0.0.
* For grid maps the first block is for Z=Z0, and successive blocks

```

```

* increment Z by dZ; the first line in a block is for Y=Y0 and the
* first column in each line is for X=X0.
* For cylinder maps, the first line in each block is for Z=Z0 and the
* first column in each line is for R=0.
*
* Instead of the blocked input format, a pointwise data format can be
* used. This is introduced by a line containing the command "data",
* followed by the individual points of the map, one per line.
* for a grid field, each line contains values for
* X,Y,Z,Bx,By,Bz,Ex,Ey,Ez separated by either a comma and optional
* whitespace or by whitespace. for a cylinder field each line contains
* values for R,Z,Br,Bz,Er,Ez. The order of the points does not matter;
* omitted grid points will be 0.0, and for duplicates the last entry
* wins. If there is no E field, the Ex,Ey,Ez or Er,Ez entries should
* be omitted on every line. NOTE: every line's X,Y,Z or R,Z must be
* on a grid point as specified by the arguments to the grid or cylinder
* commands, to within the tolerance specified; if not, an error message
* is printed and the input line is ignored.
*
* For time=dependent fields, the "time" command is used:
*     time [period=12]
* period, if given, is in nanoseconds, and causes the interval [0,period)
* to be extended forever (before and after the values given). Because of
* the interpolation used, at least two points beyond the interval
* boundaries should be provided; there need not be a point at either
* boundary (but usually there are).
* Following the time command are lines containing 2 or 3 doubles:
*     t B E
* where t is the time (nanoseconds), and B and E are factors for the
* fields. If E is omitted, the value for B is used. These values will
* be interpolated in time with a cubic spline that can handle either
* uniform or non-uniform spacing of points along t.
* The time command can come either before or after the cylinder or grid
* commands, but not within either of their sequences.
* Note a cubic spline is used to interpolate between points, and that
* can cause over/under-shoot near an abrupt change. Combined with period=
* this gives an excellent representation of sinewave/cosinewave.
*
* Note that time dependence can currently only be specified via the
* time command in an input file (i.e. not programmable method exists).
*
* Example block input file:
* * this is an example BLFieldMap input file, suitable for a solenoid
* # grid interval is 1 cm.
* # The region of validity is -390<=Z<=390 and 0<=R<=90
* param normB=1.0 current=1.0
* cylinder Z0=0.0 nR=10 nZ=40 dR=10.0 dZ=10.0
* extendZ flip=Br
* Bz
* ... 40 lines of 10 values, Z=0 thru Z=390
* Br
* ... 40 lines of 10 values, Z=0 thru Z=390
* --EOF--
*
* Example pointwise input file:
* * this is an example BLFieldMap input file, suitable for a solenoid

```

```

*   # grid interval is 1 cm.
*   # The region of validity is -390<=Z<=390 and 0<=R<=90
*   param normB=1.0 current=1.0
*   cylinder Z0=0.0 nR=10 nZ=40 dR=10.0 dZ=10.0
*   extendZ flip=Br
*   data
*   ... 400 lines of 4 values, giving R,Z,Br,Bz
*   --EOF--

```

## 9.5 Window Files

Used by the *absorber* command.

- \* The file format is a series of lines:
- \* First character # means comment, \* means printed comment.
- \* Comment and blank lines are ignored. Units are mm.
- \* The first line contains the 4 flange variables:
- \* innerR outerR insideZ outsideZ
- \* The remaining lines contain 3 values for a given radius:
- \* r z t
- \* The first line must have r=0.0 and have the largest z value
- \* and the largest z+t value. z is the inside of the window,
- \* z+t is the outside of the window. All values must be positive.
- \* Successive r values must increase by at least 0.010 mm.
- \* Successive z values and z+t values must decrease by at least
- \* 0.010 mm.
- \* flangeInnerRadius should equal the last r value.
- \* Any z origin may be used (it will be subtracted away).
- \* This is intended to be easy to interface to window design
- \* spreadsheets (export a list of 3 columns delimited by spaces, then
- \* add the appropriate comments and flange values at the top).

## 9.6 Root Files

The standard Root file format is used. All G4beamline outputs are TNtuple-s combined into a single Root file whose name is given by the parameter *histoFile*. The TNtuple fields are the same as for BLTrackFile and the Trace file above.

## 10 Acknowledgments

The genesis of G4beamline was my desire to learn modern physics simulation codes, and Geant4 seemed to be the best choice. Parts of G4beamline's design were inspired by the Beam Tools developed at Fermilab [6] by Daniel Elvira, Paul Lebrun, Panagiotis Spentzouris, and others, even though almost none of their code remains. The LISAPhysicsList came from the Geant4 collaboration, and the MICEPhysicsList came from G4MICE developed by the MICE collaboration. The source files from these external sources retain the original authors' comments and disclaimers; they total less than 1% of the G4beamline source code.

Geant4 [1] is a flexible and versatile toolkit for simulating the passage of particles through matter and electromagnetic fields. This program literally would not have been possible without it. Thanks to the entire Geant4 collaboration. This product includes software developed by Members of the Geant4 Collaboration (<http://cern.ch/geant4>).

CLHEP [2] is a comprehensive class library for High Energy Physics, and G4beamline literally would not have been possible without it. Thanks to all of the CLHEP editors and authors.

The HistoScope program [3] is a marvelous program for generating and manipulating histograms. It has a fine graphical user interface and can do just about everything one might want to do with histograms, X-Y plots, scatter plots, 2-d histograms, and NTuples. Much thanks to the authors: Mark Edel, Konstantine Iourha, Joy Kyriakopoulos, Paul Lebrun, Jeff Kallenbach, and Baolin Ren; thanks also to Fermilab for making it available. Unfortunately, it is no longer supported. It was the inspiration for HistoRoot.

Root [5] is a programming environment intended for data analysis. It supports data files with a tree structure, 1-D and 2-D histograms, X-Y plots, and NTuples. Accompanying G4beamline is *historoot*, a root program that provides a graphical user interface to the histogram capabilities of Root – it was inspired by the capabilities of HistoScope, with a completely different user interface. Of course, the native Root tools can be used as well. Thanks to all of the Root developers.

The Coin 3-D graphics library [9] is licensed under the Gnu Public License, maintained by Kongsberg SIM AS. This implements *by far* the best visualization driver for simulated systems and events.

The X-windows graphics libraries and the basic gcc libraries and compiler are also released under the GPL and/or LGPL. Thanks to the entire open source community for making such a comprehensive software development environment freely available.

# Appendix 1 – README.txt

G4beamline  
by Tom Roberts  
Copyright (C) 2003–2012 by Tom Roberts.  
All rights reserved.

<http://g4beamline.muonsinc.com>

## LICENSE

-----

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

<http://www.gnu.org/copyleft/gpl.html>

## GENERAL

-----

The general reference for G4beamline is the User's Guide, located in the doc directory named G4beamlineUsersGuide.pdf.

It is also available on the web: <http://g4beamline.muonsinc.com>

## INSTALLATION

-----

There is a "Quick Start" section in the User's Guide.

For details, see the README-\*.txt file for your OS (appendices of the User's Guide).

## INITIAL TESTING

-----

Be sure you have added the G4beamline programs to your PATH (see README-\*.txt).

Execute the following commands (in a Cygwin shell on Windows):

```
source G4beamline-VERSION/bin/g4bl-setup.sh
cd G4beamline-VERSION/test
./loop
```

This will run a series of tests, with 1 line of description for each, hopefully ending with "All Tests Passed" and starting over. If this fails,

see README-\*.txt for a description of what is required to run G4beamline.  
For further assistance, you can join the G4beamline forum at  
<http://g4beamline.muonsinc.com> .

NOTE: On Windows you will need to setup the Cygwin environment. The Cygwin environment is not needed to run G4beamline itself, it is needed just for building the program and running the tests.

#### RUNNING THE PROGRAM -- GUI

-----  
Simply double-click the G4beamline icon. On Windows it is placed on your desktop and in the Start/G4beamline menu. On Mac OS X it is placed where you dragged (copied) it when you installed it. On Linux the g4bl-icon script will create one on your desktop.

To run the GUI program via the command line:

```
source G4beamline-VERSION/bin/g4bl-setup.sh
# optionally cd where your input file is located
g4blgui [input.file]
```

This requires Java and a display (X-Windows on Linux and Mac OS X, nothing special on Windows). Its opening screen describes how to use it.

To run the examples, simply push the Browse button, navigate to the install directory / examples, and select Example1.g4bl (or other \*.g4bl file). Then select the desired viewer (if any), and push the Run button. On Windows, a copy of "G4beamline Examples" is put into "My Documents" (Windows Xp) or "Documents" (Windows Vista). On Mac, there is a copy of G4beamlineExamples on the install .dmg.

#### RUNNING THE PROGRAM -- COMMAND-LINE

-----  
For command-line use (Linux, Mac OS X, and Windows with Cygwin), you must add G4beamline's bin directory to your PATH:

```
source G4beamline-VERSION/bin/g4bl-setup.sh
```

You can put this into \$HOME/.bash\_profile. There is also g4bl-setup.csh.

Then cd to whatever directory you plan to use for developing your simulation(s), and execute:

```
g4bl -
```

After a few seconds it should type (with obvious variations):

```
G4BL_DIR=/Users/g4bl/G4beamline
G4LEDDATA=/Users/g4bl/G4beamline/data/G4EMLOW6.2
G4LEVELGAMMADATA=/Users/g4bl/G4beamline/data/PhotonEvaporation2.0
DYLD_LIBRARY_PATH=/Users/g4bl/G4beamline/LibraryBinaries/Darwin-g++:
G4beamline Process ID 68927
```

```
*****
```

```
g4beamline version: 2.10 (17-Dec-2011,14:14)
Copyright : Tom Roberts, Muons, Inc.
```

```

                License : Gnu Public License
                WWW : http://g4beamline.muonsinc.com
*****
Geant4 version Name: geant4-09-04-patch-03      (9-December-2011)
                Copyright : Geant4 Collaboration
                Reference : NIM A 506 (2003), 250-303
                WWW : http://cern.ch/geant4
*****

geometry                nPoints=100 printGeometry=0 visual=0
                        tolerance=0.002

cmd:

```

The program is now ready for input. Type "help" to get a short list of the input-file commands, "help beam" to get help on the beam comand, or "help \*" for a detailed description of all commands. This is a useful way to get help on commands when editing your input file(s). Type ^C to exit back to a shell prompt.

#### EXAMPLE 1

-----

The first example input file is Example1.g4bl. It is a simple file to track muons through 1-meter drift spaces into 4 detectors. To visualize its geometry using OpenInventor, execute:

```

cd G4beamline-VERSION/examples
g4bl example1.g4bl viewer=best

```

To run the beam through the geometry, execute:

```

g4bl example1.g4bl

```

This will generate a Root file named g4beamline.root. To view it do:

```

hisoroot g4beamline.root

```

#### OTHER EXAMPLES

-----

There are other examples in the examples directory. See README.txt for details.

## Appendix 2 – README-Linux.txt

G4beamline on Linux (Intel)  
G4beamline 2.14 November 2012 TJR

Note that the distributed 32-bit version of G4beamline for Linux was built on Scientific Linux Fermi 4.8 (SLF 4.8). This is a derivative of RedHat Enterprise Linux 4. It should run as is on any RedHat-derived Linux equal or later than that. There is also a 64-bit distribution built on SLF 5.5.

Before installing G4beamline, you should first install:

openmotif -- should be available in your Linux distribution. Some distributions install this automatically, some don't. It provides libXm.so.

java - <http://java.sun.com>  
Select "Java SE", and all you need is the Java Runtime Environment (JRE); get the latest. If you have installed the development kit (JDK) that includes the JRE.  
Most distributions install this automatically.

You may need to install these:

x11-deprecated-libs -- should be available in your Linux distribution, but the name may be different. Some distributions install this automatically, some don't. It provides libXp.so.

compat-libstdc++.i386 -- should be available in your Linux distribution, but the name may be different. Provides an older version of /usr/lib/libstdc++.so.

You probably want to install the following:

Root - <http://root.cern.ch>  
You may install any version of Root, because G4beamline is built with its own instance of Root and is independent of whichever Root you install. If you want to use HistoRoot, however, you should install the version of Root from <http://historoot.muonsinc.com>.

You may want to install the following:

tcl - should be available in your Linux distribution. Required by most of the tests, but g4beamline itself will run without it.  
tcl can be installed after G4beamline, if desired.  
Many distributions install this automatically.

To install G4beamline, download the tarball and un-tar it in your HOME directory:

```
tar -xzf G4beamline-2.14-Linux.tgz
```

OR

```
tar -xzf G4beamline-2.14-Linux64.tgz
```

this will create a directory \$HOME/G4beamline-2.14 .

You need to put the G4beamline programs into your PATH:

```
source G4beamline-VERSION/bin/g4bl-setup.sh
```

You can put this into your \$HOME/.bash\_profile. There is also g4bl-setup.csh.

You can create an icon on your Desktop via:

```
g4bl-icon
```

You can then copy or move it into any menu. This works for both KDE and Gnome, and it probably works for other window managers.

Once g4beamline has been put into your PATH, to run G4beamline simply do:

```
cd ..wherever...
```

```
g4bl input.file [name=value [...]]
```

You can also run the GUI version:

```
g4blgui [input.file]
```

The default format for NTuples is now Root. To create histograms from NTuples you can use Root in the usual way. That is rather complicated and un-obvious, so you can download the HistoRoot program from <http://historoot.muonsinc.com> -- it provides a GUI interface to Root plots and histograms.

For visualization, the OpenInventor viewer works fine, use viewer=best or viewer=OIX. Other viewers are available.

## TROUBLESHOOTING

Here is the output of ldd on Scientific Linux Fermi 4.8 (where the distribution was built), showing which shared libraries are used, and where they come from.

```
$ ldd g4beamline
libGLU.so.1 => /usr/X11R6/lib/libGLU.so.1 (0x0342a000)
libGL.so.1 => /usr/X11R6/lib/libGL.so.1 (0x00d72000)
libXm.so.3 => /usr/X11R6/lib/libXm.so.3 (0x00794000)
libXpm.so.4 => /usr/X11R6/lib/libXpm.so.4 (0x0065f000)
libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x0077d000)
libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x006fc000)
libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x003c6000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x002e5000)
libXi.so.6 => /usr/X11R6/lib/libXi.so.6 (0x00111000)
libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x00416000)
libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x003e8000)
libCint.so => /home/tjrob/G4beamline-2.14/lib/libCint.so (0x00de0000)
libCore.so => /home/tjrob/G4beamline-2.14/lib/libCore.so (0x04a22000)
libMathCore.so => /home/tjrob/G4beamline-2.14/lib/libMathCore.so
(0x0043d000)
libNet.so => /home/tjrob/G4beamline-2.14/lib/libNet.so (0x00186000)
libRIO.so => /home/tjrob/G4beamline-2.14/lib/libRIO.so (0x009cb000)
libThread.so => /home/tjrob/G4beamline-2.14/lib/libThread.so (0x00119000)
libTree.so => /home/tjrob/G4beamline-2.14/lib/libTree.so (0x00b45000)
libexpat.so.0 => /usr/lib/libexpat.so.0 (0x0028a000)
libdl.so.2 => /lib/libdl.so.2 (0x002df000)
libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x0297d000)
libm.so.6 => /lib/tls/libm.so.6 (0x002ba000)
```

```

libgcc_s.so.1 => /lib/libgcc_s.so.1 (0x0015c000)
libc.so.6 => /lib/tls/libc.so.6 (0x06242000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x00402000)
libXxf86vm.so.1 => /usr/X11R6/lib/libXxf86vm.so.1 (0x00101000)
libXp.so.6 => /usr/X11R6/lib/libXp.so.6 (0x003d6000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x005e0000)
/lib/ld-linux.so.2 (0x0016e000)

```

Note that some motif/X11 shared libraries are included in the G4beamline distribution (and remain local to it). This reduces the need for users to install precisely the same version of these libraries as was used to build g4beamline.

Here are the RPMs that provide these libraries in Scientific Linux Fermi 4.8:

glibc.i686	/lib/tls/libpthread.so.0
glibc.i686	/lib/libdl.so.2
xorg-x11-Mesa-libGLU.i386	/usr/X11R6/lib/libGLU.so.1
xorg-x11-Mesa-libGL.i386	/usr/X11R6/lib/libGL.so.1
openmotif.i386	/usr/X11R6/lib/libXm.so.3
xorg-x11-libs.i386	/usr/X11R6/lib/libXpm.so.4
xorg-x11-libs.i386	/usr/X11R6/lib/libXmu.so.6
xorg-x11-libs.i386	/usr/X11R6/lib/libXt.so.6
xorg-x11-libs.i386	/usr/X11R6/lib/libXext.so.6
xorg-x11-libs.i386	/usr/X11R6/lib/libX11.so.6
xorg-x11-libs.i386	/usr/X11R6/lib/libXi.so.6
xorg-x11-libs.i386	/usr/X11R6/lib/libSM.so.6
xorg-x11-libs.i386	/usr/X11R6/lib/libICE.so.6
libstdc++.i386	/usr/lib/libstdc++.so.6
glibc.i686	/lib/tls/libm.so.6
libgcc.i386	/lib/libgcc_s.so.1
glibc.i686	/lib/tls/libc.so.6
xorg-x11-libs.i386	/usr/X11R6/lib/libXxf86vm.so.1
xorg-x11-deprecated-libs.i386	/usr/X11R6/lib/libXp.so.6

## Appendix 3 – README-Windows.txt

G4beamline on Windows XP, Windows Vista, and Windows 7  
G4beamline 2.14 November 2012 TJR

Before installing G4beamline, you must first install:

Java - <http://java.sun.com>  
Select "Java SE", and all you need is the Java Runtime Environment (JRE); get the latest. If you have installed the development kit (JDK), that includes the JRE.

You may want to install:

Root - <http://root.cern.ch> or <http://historoot.muonsinc.com>  
You may install any version of Root, because G4beamline is built with its own instance of Root and is independent of whichever Root you install. If you want to use HistoRoot, however, you should install the version of Root from <http://historoot.muonsinc.com>.

Cygwin - <http://cygwin.com>  
This is a UNIX-like command-line environment for Windows. Select "Install Cygwin now". This will download Cygwin's setup.exe. Run it, select a mirror site, and you will have the opportunity to select packages. You should select at least one editor of your choice (vi, emacs, or ...). You must also select tcltk, which is in the Libs category. You can add or remove packages at any time by re-running the Cygwin setup.exe. Install it into the default directory, C:\cygwin

To install G4beamline, download G4beamline-2.14.msi and run it. Follow the usual directions for installing packages into Windows. The installer will install shortcuts for G4beamline and Historoot onto your desktop and into the Start menu.

If you have installed Cygwin, then if you want to run G4beamline from its command line, do this in a Cygwin shell:

```
source "C:/Program Files/MuonsInc/G4beamline/bin/g4bl-setup.sh"
The puts G4beamline into your PATH. Then do:
cd ...wherever...
g4bl input.file [name=value [...]]
```

The default format for NTuples is now Root. To create histograms from NTuples you can use Root in the usual way. That is rather complicated and un-obvious, so you can download the HistoRoot program from <http://historoot.muonsinc.com>.

For visualization, the OpenInventor viewer works fine, simply select "best" from the viewer panel of the G4beamline window. Other viewers are available.

## Appendix 4 – README-MacOSX.txt

G4beamline on Mac OS X (Intel)  
G4beamline 2.14 November 2012 TJR

This version of G4beamline was built and tested on Snow Leopard (Mac OS X 10.6.8); it runs on Snow Leopard (Mac OS X 10.6.8) and Lion (10.7.2). G4beamline has been used in the past on Tiger (10.4.x) and Leopard (10.5.x), but now must be built from source to run there (see BUILD.txt).

Note: PowerPC binaries are not available. It is expected that installing G4beamline from source should work on older Macs running Tiger or later, and at least one user has had success doing this.

Before installing G4beamline, you probably want to install this:

Java - installed by default as part of Mac OS X.

Root - <http://root.cern.ch> or <http://historoot.muonsinc.com>  
You may install any version of Root, because G4beamline is built with its own instance of Root and is independent of whichever Root you install. If you want to use HistoRoot, however, you should install the version of Root from <http://historoot.muonsinc.com>.

To install G4beamline, download the installer .dmg from  
<http://g4beamline.muonsinc.com>

Open it, and drag the application icon into /Applications (requires administrator privileges; if you can't do that, drag it into your Home or to your Desktop). You can then drag it into the Dock, if desired.

You can run it in the usual way by double-clicking on the icon (single-click in the dock).

If you want to run the programs from the command line, you need to put the G4beamline programs into your PATH:

```
source /Applications/G4beamline.app/Contents/Resources/bin/g4bl-setup.sh
```

This can be put into your \$HOME/.bash\_profile. There is also g4bl-setup.csh.

Once g4beamline has been put into your PATH, to run G4beamline simply do:

```
cd ...wherever...  
g4bl input.file [name=value [...]]
```

You can also run the Graphical User Interface:

```
g4blgui [input.file]
```

The default format for NTuples is now Root. To create histograms from NTuples you can use Root in the usual way. That is rather complicated and un-obvious, but you can download the HistoRoot program from <http://historoot.muonsinc.com> -- it provides a GUI interface to Root plots and histograms.

For visualization, the OpenInventor viewer works fine, use viewer=best or viewer=OIX. Other viewers are available.

## TROUBLESHOOTING

-----

Here is the result of "otool -L" on Mac OS X 10.6.8 (Intel). This shows what shared libraries are used. Libraries in /usr are from the OS; those in /sw are from Fink.

```
$ otool -L g4beamline
```

```
g4beamline:
```

```
    @rpath/libCint.so (compatibility version 0.0.0, current version 0.0.0)
    @rpath/libCore.so (compatibility version 0.0.0, current version 0.0.0)
    @rpath/libMathCore.so (compatibility version 0.0.0, current version 0.0.0)
    @rpath/libNet.so (compatibility version 0.0.0, current version 0.0.0)
    @rpath/libRIO.so (compatibility version 0.0.0, current version 0.0.0)
    @rpath/libThread.so (compatibility version 0.0.0, current version 0.0.0)
    @rpath/libTree.so (compatibility version 0.0.0, current version 0.0.0)
    /usr/X11/lib/libGLU.1.dylib (compatibility version 1.3.0, current version
1.3.0)
    /usr/X11/lib/libGL.1.dylib (compatibility version 1.2.0, current version
1.2.0)
    /sw/lib/libXm.2.dylib (compatibility version 3.0.0, current version 3.1.0)
    /usr/X11/lib/libXpm.4.dylib (compatibility version 16.0.0, current version
16.0.0)
    /usr/X11/lib/libXmu.6.dylib (compatibility version 9.0.0, current version
9.0.0)
    /usr/X11/lib/libXt.6.dylib (compatibility version 7.0.0, current version
7.0.0)
    /usr/X11/lib/libXext.6.dylib (compatibility version 11.0.0, current
version 11.0.0)
    /usr/X11/lib/libX11.6.dylib (compatibility version 9.0.0, current version
9.0.0)
    /usr/X11/lib/libXi.6.dylib (compatibility version 7.0.0, current version
7.0.0)
    /usr/X11/lib/libSM.6.dylib (compatibility version 7.0.0, current version
7.0.0)
    /usr/X11/lib/libICE.6.dylib (compatibility version 10.0.0, current version
10.0.0)
    /usr/lib/libexpat.1.dylib (compatibility version 7.0.0, current version
7.2.0)
    /usr/lib/libstdc++.6.dylib (compatibility version 7.0.0, current version
7.9.0)
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version
125.2.11)
```

Note that libXm.2.dylib is included in the G4beamline distribution and should not be a problem. If it is a problem, it comes from Fink in the package "lesstif" (the Fink implementation of the Motif graphical widgets). The @rparh libraries are for Root, and are also included in the distribution.

# Appendix 5 – Annotated Output from Example1.g4bl

Note that Geant4 outputs considerable detail to stdout. Annotations below are headings inserted into the output text.

## First, the g4bl script outputs environment variables:

```
G4BL_DIR=/Users/g4bl/G4beamline-2.11
G4ABLAData=/Users/g4bl/G4beamline-2.11/Geant4Data/G4ABLA3.0
G4LEData=/Users/g4bl/G4beamline-2.11/Geant4Data/G4EMLOW6.23
G4NEUTRONHPData=/Users/g4bl/G4beamline-2.11/Geant4Data/G4NDL4.0
G4NEUTRONXSData=/Users/g4bl/G4beamline-2.11/Geant4Data/G4NEUTRONXS1.1
G4PIIData=/Users/g4bl/G4beamline-2.11/Geant4Data/G4PII1.3
G4LEVELGAMMADATA=/Users/g4bl/G4beamline-2.11/Geant4Data/PhotonEvaporation2.2
G4RADIOACTIVEDATA=/Users/g4bl/G4beamline-2.11/Geant4Data/RadioactiveDecay3.4
G4REALSURFACEDATA=/Users/g4bl/G4beamline-2.11/Geant4Data/RealSurface1.0
DYLD_LIBRARY_PATH=/Users/g4bl/G4beamline-2.11/lib:
G4beamline Process ID 30447
```

## G4beamline outputs its identification:

```
*****
g4beamline version: 2.11 (19-Apr-2012-17:00)
      Copyright : Tom Roberts, Muons, Inc.
      License : Gnu Public License
      WWW : http://g4beamline.muonsinc.com
*****
Geant4 version Name: geant4-09-05-patch-01 (20-March-2012)
      Copyright : Geant4 Collaboration
      Reference : NIM A 506 (2003), 250-303
      WWW : http://cern.ch/geant4
*****
geometry          nPoints=100 printGeometry=0 visual=0
                  tolerance=0.002
```

## Here are some lines beginning with ‘\*’ from the input file:

```
*      Example1.g4bl - simple first example
*
* Simple example g4beamline input file:
*      a 200 MeV mu+ Gaussian beam is tracked through 1-meter drift
*      spaces into four detectors
```

## Some output from the physics command comes before the command itself:

```
G4PhysListFactory::GetReferencePhysList <QGSP_BERT> EMoption= 0
<<< Geant4 Physics List simulation engine: QGSP_BERT 3.4

<<< Reference Physics List QGSP_BERT is built

physics          QGSP_BERT      disable= deactivate= doStochastics=1
                                minRangeCut=1 list=0 gammaToMuPair=0 spinTracking=0
                                synchrotronRadiation=0 synchrotronRadiationMuon=0
```

## Next is the beam command:

```
beam      GAUSSIAN particle=mu+ nEvents=1000 firstEvent=-1 lastEvent=2147483647
          beamX=0.0 beamY=0.0 beamZ=0.0 maxR=1000000.0
          meanMomentum=200.0 weight=1.000000
```

```
sigmaX=10.0 sigmaY=10.0 sigmaZ=0.0 sigmaXp=0.10000 sigmaYp=0.10000
sigmaP=4.0 sigmaT=0.000 meanXp=0.00000 meanYp=0.00000 meanT=0.000
```

### The rest of the commands from the input file:

```
reference          particle=mu+ beamX=0 beamY=0 beamZ=0 beamT=0
                  rotation= referenceMomentum=200 beamXp=0 beamYp=0
                  meanMomentum=200 meanXp=0 meanYp=0 tuneZ=-3.7e+21
                  tuneMomentum=-3.7e+21 tolerance=0.001 noEfield=0
                  noEloss=0
box               BeamVis height=100 width=100 length=0.1 maxStep=100
                  material=Vacuum color=1,0,0 kill=0
place BeamVis copies=1 x=0.0 y=0.0 z=0.0
virtualdetector Det radius=1000 innerRadius=0 height=2000 width=2000
                  length=1 maxStep=100 material= color=0,1,0
                  noSingles=0 format= filename= file= require=
                  referenceParticle=0 coordinates=Centerline kill=0
place Det copies=1 x=0.0 y=0.0 z=1000.0 rename='Det#'
place Det copies=1 x=0.0 y=0.0 z=2000.0 rename='Det#'
place Det copies=1 x=0.0 y=0.0 z=3000.0 rename='Det#'
place Det copies=1 x=0.0 y=0.0 z=4000.0 rename='Det#'
```

### The World size is printed:

```
World size (before incrementing by 201.357 mm): 2000.0 H 2000.0 W 8001.0 L
```

### The final values of the parameters are printed:

```
PARAMETERS:
      Zcl=4000.5
      deltaChord=3.0
deltaIntersection=0.1
      deltaOneStep=0.01
      epsMax=0.05
      epsMin=2.5e-7
eventTimeLimit=30
      histoFile=g4beamline
      histoUpdate=0
      maxStep=100.0
      minStep=0.01
steppingFormat=N GLOBAL CL KE STEP VOL PROCESS
steppingVerbose=0
      viewer=none
worldMaterial=Vacuum
      zTolerance=2.0
```

### Now a line for the construction of each Element is printed:

```
BLCMDbox::Construct BeamVis parent= relZ=0.0 globZ=0.0
      zmin=-0.1 zmax=0.1
BLCMDvirtualdetector::Construct Det1 parent= relZ=1000.0 globZ=1000.0
BLCMDvirtualdetector::Construct Det2 parent= relZ=2000.0 globZ=2000.0
BLCMDvirtualdetector::Construct Det3 parent= relZ=3000.0 globZ=3000.0
BLCMDvirtualdetector::Construct Det4 parent= relZ=4000.0 globZ=4000.0
```

### A comment from the Geant4 Physics List:

```
### Adding tracking cuts for neutron TimeCut(ns)= 10000 KinEnergyCut(MeV)= 0
```

### The Geometry test results:

```

Geometry test nPoints=100 tolerance=0.002 mm:
Testing geometry for children of group 'World':
Total geometry errors = 0 0 seconds

```

### **Now the Tune Particle is prepared and simulated (nothing to tune in this simulation):**

```

===== Prepare Tune Particle(s) =====
*** NOTE: All stochastic processes are disabled!
===== Begin Tune Particle(s) =====
Run complete 1 Events 0 seconds

```

### **The Reference Particle is tracked:**

```

===== Begin Reference Particle(s) =====
Run complete 1 Events 0 seconds

```

### **Stochastic processes are enabled after the reference particle:**

Stochastic processes are enabled.

### **Preparation to track the beam:**

```

===== Prepare Tracking Beam =====
Stochastic processes are enabled.

```

### **And the beam is tracked (timing summaries are spaced out to avoid too much output):**

```

===== Begin Tracking Beam =====
Event 1 Completed 1 events realTime=1 sec 1.0 ev/sec
Event 2 Completed 2 events realTime=1 sec 2.0 ev/sec
Event 3 Completed 3 events realTime=1 sec 3.0 ev/sec
Event 4 Completed 4 events realTime=1 sec 4.0 ev/sec
Event 5 Completed 5 events realTime=1 sec 5.0 ev/sec
Event 6 Completed 6 events realTime=1 sec 6.0 ev/sec
Event 7 Completed 7 events realTime=1 sec 7.0 ev/sec
Event 8 Completed 8 events realTime=1 sec 8.0 ev/sec
Event 9 Completed 9 events realTime=1 sec 9.0 ev/sec
Event 10 Completed 10 events realTime=1 sec 10.0 ev/sec
Event 20 Completed 20 events realTime=1 sec 20.0 ev/sec
Event 30 Completed 30 events realTime=1 sec 30.0 ev/sec
Event 40 Completed 40 events realTime=1 sec 40.0 ev/sec
Event 50 Completed 50 events realTime=1 sec 50.0 ev/sec
Event 60 Completed 60 events realTime=1 sec 60.0 ev/sec
Event 70 Completed 70 events realTime=1 sec 70.0 ev/sec
Event 80 Completed 80 events realTime=1 sec 80.0 ev/sec
Event 90 Completed 90 events realTime=1 sec 90.0 ev/sec
Event 100 Completed 100 events realTime=1 sec 100.0 ev/sec
Event 200 Completed 200 events realTime=1 sec 200.0 ev/sec
Event 300 Completed 300 events realTime=1 sec 300.0 ev/sec
Event 400 Completed 400 events realTime=1 sec 400.0 ev/sec
Event 500 Completed 500 events realTime=1 sec 500.0 ev/sec
Event 600 Completed 600 events realTime=1 sec 600.0 ev/sec
Event 700 Completed 700 events realTime=1 sec 700.0 ev/sec
Event 800 Completed 800 events realTime=1 sec 800.0 ev/sec
Event 900 Completed 900 events realTime=1 sec 900.0 ev/sec
Event 1000 Completed 1000 events realTime=1 sec 1000.0 ev/sec

```

### **Finally the run is finished; a summary of NTuples is printed:**

```

Run complete 1000 Events 1 seconds
NTuple Det1 1002 entries
NTuple Det2 1003 entries
NTuple Det3 999 entries

```

```
NTuple Det4                      953 entries
NTuple wrote Root File 'g4beamline.root'
```

**A summary of all Exceptions is printed (a list of how many of each type is also printed):**

Exceptions: 0 Fatal, 0 Events Aborted, 0 Stuck Tracks (stopped), 0 Warnings

**The simulation is over:**

```
g4beamline: simulation complete -- exiting
```

## Appendix 6 – Particle IDs

```
B+ PDGid=521
B- PDGid=-521
B0 PDGid=511
Bs0 PDGid=531
D+ PDGid=411
D- PDGid=-411
D0 PDGid=421
Ds+ PDGid=431
Ds- PDGid=-431
GenericIon PDGid=0
He3 PDGid=1000020030
J/psi PDGid=443
N(1440)+ PDGid=12212
N(1440)0 PDGid=12112
N(1520)+ PDGid=2124
N(1520)0 PDGid=1214
N(1535)+ PDGid=22212
N(1535)0 PDGid=22112
N(1650)+ PDGid=32212
N(1650)0 PDGid=32112
N(1675)+ PDGid=2216
N(1675)0 PDGid=2116
N(1680)+ PDGid=12216
N(1680)0 PDGid=12116
N(1700)+ PDGid=22124
N(1700)0 PDGid=21214
N(1710)+ PDGid=42212
N(1710)0 PDGid=42112
N(1720)+ PDGid=32124
N(1720)0 PDGid=31214
N(1900)+ PDGid=42124
N(1900)0 PDGid=41214
N(1990)+ PDGid=12218
N(1990)0 PDGid=12118
N(2090)+ PDGid=52214
N(2090)0 PDGid=52114
N(2190)+ PDGid=2128
N(2190)0 PDGid=1218
N(2220)+ PDGid=100002210
N(2220)0 PDGid=100002110
N(2250)+ PDGid=100012210
N(2250)0 PDGid=100012110
a0(1450)+ PDGid=10211
a0(1450)- PDGid=-10211
a0(1450)0 PDGid=10111
a0(980)+ PDGid=9000211
a0(980)- PDGid=-9000211
a0(980)0 PDGid=9000111
a1(1260)+ PDGid=20213
a1(1260)- PDGid=-20213
a1(1260)0 PDGid=20113
a2(1320)+ PDGid=215
```

```

a2(1320)- PDGid=-215
a2(1320)0 PDGid=115
  alpha PDGid=1000020040
  anti_B0 PDGid=-511
  anti_Bs0 PDGid=-531
  anti_D0 PDGid=-421
anti_N(1440)+ PDGid=-12212
anti_N(1440)0 PDGid=-12112
anti_N(1520)+ PDGid=-2124
anti_N(1520)0 PDGid=-1214
anti_N(1535)+ PDGid=-22212
anti_N(1535)0 PDGid=-22112
anti_N(1650)+ PDGid=-32212
anti_N(1650)0 PDGid=-32112
anti_N(1675)+ PDGid=-2216
anti_N(1675)0 PDGid=-2116
anti_N(1680)+ PDGid=-12216
anti_N(1680)0 PDGid=-12116
anti_N(1700)+ PDGid=-22124
anti_N(1700)0 PDGid=-21214
anti_N(1710)+ PDGid=-42212
anti_N(1710)0 PDGid=-42112
anti_N(1720)+ PDGid=-32124
anti_N(1720)0 PDGid=-31214
anti_N(1900)+ PDGid=-42124
anti_N(1900)0 PDGid=-41214
anti_N(1990)+ PDGid=-12218
anti_N(1990)0 PDGid=-12118
anti_N(2090)+ PDGid=-52214
anti_N(2090)0 PDGid=-52114
anti_N(2190)+ PDGid=-2128
anti_N(2190)0 PDGid=-1218
anti_N(2220)+ PDGid=-100002210
anti_N(2220)0 PDGid=-100002110
anti_N(2250)+ PDGid=-100012210
anti_N(2250)0 PDGid=-100012110
  anti_b_quark PDGid=-5
  anti_c_quark PDGid=-4
  anti_d_quark PDGid=-1
anti_dd1_diquark PDGid=-1103
anti_delta(1600)+ PDGid=-32214
anti_delta(1600)++ PDGid=-32224
anti_delta(1600)- PDGid=-31114
anti_delta(1600)0 PDGid=-32114
anti_delta(1620)+ PDGid=-2122
anti_delta(1620)++ PDGid=-2222
anti_delta(1620)- PDGid=-1112
anti_delta(1620)0 PDGid=-1212
anti_delta(1700)+ PDGid=-12214
anti_delta(1700)++ PDGid=-12224
anti_delta(1700)- PDGid=-11114
anti_delta(1700)0 PDGid=-12114
anti_delta(1900)+ PDGid=-12122
anti_delta(1900)++ PDGid=-12222
anti_delta(1900)- PDGid=-11112

```

```

anti_delta(1900)0 PDGid=-11212
anti_delta(1905)+ PDGid=-2126
anti_delta(1905)++ PDGid=-2226
anti_delta(1905)- PDGid=-1116
anti_delta(1905)0 PDGid=-1216
anti_delta(1910)+ PDGid=-22122
anti_delta(1910)++ PDGid=-22222
anti_delta(1910)- PDGid=-21112
anti_delta(1910)0 PDGid=-21212
anti_delta(1920)+ PDGid=-22214
anti_delta(1920)++ PDGid=-22224
anti_delta(1920)- PDGid=-21114
anti_delta(1920)0 PDGid=-22114
anti_delta(1930)+ PDGid=-12126
anti_delta(1930)++ PDGid=-12226
anti_delta(1930)- PDGid=-11116
anti_delta(1930)0 PDGid=-11216
anti_delta(1950)+ PDGid=-2218
anti_delta(1950)++ PDGid=-2228
anti_delta(1950)- PDGid=-1118
anti_delta(1950)0 PDGid=-2118
    anti_delta+ PDGid=-2214
    anti_delta++ PDGid=-2224
    anti_delta- PDGid=-1114
    anti_delta0 PDGid=-2114
    anti_k(1460)0 PDGid=-100311
anti_k0_star(1430)0 PDGid=-10311
    anti_k1(1270)0 PDGid=-10313
    anti_k1(1400)0 PDGid=-20313
    anti_k2(1770)0 PDGid=-10315
anti_k2_star(1430)0 PDGid=-315
anti_k2_star(1980)0 PDGid=-100315
anti_k3_star(1780)0 PDGid=-317
anti_k_star(1410)0 PDGid=-100313
anti_k_star(1680)0 PDGid=-30313
    anti_k_star0 PDGid=-313
    anti_kaon0 PDGid=-311
    anti_lambda PDGid=-3122
anti_lambda(1405) PDGid=-13122
anti_lambda(1520) PDGid=-3124
anti_lambda(1600) PDGid=-23122
anti_lambda(1670) PDGid=-33122
anti_lambda(1690) PDGid=-13124
anti_lambda(1800) PDGid=-43122
anti_lambda(1810) PDGid=-53122
anti_lambda(1820) PDGid=-3126
anti_lambda(1830) PDGid=-13126
anti_lambda(1890) PDGid=-23124
anti_lambda(2100) PDGid=-3128
anti_lambda(2110) PDGid=-23126
    anti_lambda_c+ PDGid=-4122
    anti_neutron PDGid=-2112
        anti_nu_e PDGid=-12
        anti_nu_mu PDGid=-14
        anti_nu_tau PDGid=-16

```

```

    anti_omega- PDGid=-3334
    anti_omega_c0 PDGid=-4332
    anti_proton PDGid=-2212
    anti_s_quark PDGid=-3
    anti_sd0_diquark PDGid=-3101
    anti_sdl_diquark PDGid=-3103
    anti_sigma(1385)+ PDGid=-3224
    anti_sigma(1385)- PDGid=-3114
    anti_sigma(1385)0 PDGid=-3214
    anti_sigma(1660)+ PDGid=-13222
    anti_sigma(1660)- PDGid=-13112
    anti_sigma(1660)0 PDGid=-13212
    anti_sigma(1670)+ PDGid=-13224
    anti_sigma(1670)- PDGid=-13114
    anti_sigma(1670)0 PDGid=-13214
    anti_sigma(1750)+ PDGid=-23222
    anti_sigma(1750)- PDGid=-23112
    anti_sigma(1750)0 PDGid=-23212
    anti_sigma(1775)+ PDGid=-3226
    anti_sigma(1775)- PDGid=-3116
    anti_sigma(1775)0 PDGid=-3216
    anti_sigma(1915)+ PDGid=-13226
    anti_sigma(1915)- PDGid=-13116
    anti_sigma(1915)0 PDGid=-13216
    anti_sigma(1940)+ PDGid=-23224
    anti_sigma(1940)- PDGid=-23114
    anti_sigma(1940)0 PDGid=-23214
    anti_sigma(2030)+ PDGid=-3228
    anti_sigma(2030)- PDGid=-3118
    anti_sigma(2030)0 PDGid=-3218
    anti_sigma+ PDGid=-3222
    anti_sigma- PDGid=-3112
    anti_sigma0 PDGid=-3212
    anti_sigma_c+ PDGid=-4212
    anti_sigma_c++ PDGid=-4222
    anti_sigma_c0 PDGid=-4112
    anti_ssl_diquark PDGid=-3303
    anti_su0_diquark PDGid=-3201
    anti_sul_diquark PDGid=-3203
    anti_t_quark PDGid=-6
    anti_u_quark PDGid=-2
    anti_ud0_diquark PDGid=-2101
    anti_udl_diquark PDGid=-2103
    anti_uul_diquark PDGid=-2203
    anti_xi(1530)- PDGid=-3314
    anti_xi(1530)0 PDGid=-3324
    anti_xi(1690)- PDGid=-23314
    anti_xi(1690)0 PDGid=-23324
    anti_xi(1820)- PDGid=-13314
    anti_xi(1820)0 PDGid=-13324
    anti_xi(1950)- PDGid=-33314
    anti_xi(1950)0 PDGid=-33324
    anti_xi(2030)- PDGid=-13316
    anti_xi(2030)0 PDGid=-13326
    anti_xi- PDGid=-3312

```

```

    anti_xi0 PDGid=-3322
    anti_xi_c+ PDGid=-4232
    anti_xi_c0 PDGid=-4132
    b1(1235)+ PDGid=10213
    b1(1235)- PDGid=-10213
    b1(1235)0 PDGid=10113
    b_quark PDGid=5
    c_quark PDGid=4
chargedgeantino PDGid=0
    d_quark PDGid=1
    ddl_diquark PDGid=1103
    delta(1600)+ PDGid=32214
    delta(1600)++ PDGid=32224
    delta(1600)- PDGid=31114
    delta(1600)0 PDGid=32114
    delta(1620)+ PDGid=2122
    delta(1620)++ PDGid=2222
    delta(1620)- PDGid=1112
    delta(1620)0 PDGid=1212
    delta(1700)+ PDGid=12214
    delta(1700)++ PDGid=12224
    delta(1700)- PDGid=11114
    delta(1700)0 PDGid=12114
    delta(1900)+ PDGid=12122
    delta(1900)++ PDGid=12222
    delta(1900)- PDGid=11112
    delta(1900)0 PDGid=11212
    delta(1905)+ PDGid=2126
    delta(1905)++ PDGid=2226
    delta(1905)- PDGid=1116
    delta(1905)0 PDGid=1216
    delta(1910)+ PDGid=22122
    delta(1910)++ PDGid=22222
    delta(1910)- PDGid=21112
    delta(1910)0 PDGid=21212
    delta(1920)+ PDGid=22214
    delta(1920)++ PDGid=22224
    delta(1920)- PDGid=21114
    delta(1920)0 PDGid=22114
    delta(1930)+ PDGid=12126
    delta(1930)++ PDGid=12226
    delta(1930)- PDGid=11116
    delta(1930)0 PDGid=11216
    delta(1950)+ PDGid=2218
    delta(1950)++ PDGid=2228
    delta(1950)- PDGid=1118
    delta(1950)0 PDGid=2118
    delta+ PDGid=2214
    delta++ PDGid=2224
    delta- PDGid=1114
    delta0 PDGid=2114
    deuteron PDGid=1000010020
    e+ PDGid=-11
    e- PDGid=11
    eta PDGid=221

```

```

    eta(1295) PDGid=100221
    eta(1405) PDGid=9020221
    eta(1475) PDGid=100331
    eta2(1645) PDGid=10225
    eta2(1870) PDGid=10335
    eta_prime PDGid=331
    f0(1370) PDGid=10221
    f0(1500) PDGid=9030221
    f0(1710) PDGid=10331
    f0(600) PDGid=9000221
    f0(980) PDGid=9010221
    f1(1285) PDGid=20223
    f1(1420) PDGid=20333
    f2(1270) PDGid=225
    f2(1810) PDGid=9030225
    f2(2010) PDGid=9060225
f2_prime(1525) PDGid=335
    gamma PDGid=22
    geantino PDGid=0
    gluon PDGid=21
    h1(1170) PDGid=10223
    h1(1380) PDGid=10333
    k(1460)+ PDGid=100321
    k(1460)- PDGid=-100321
    k(1460)0 PDGid=100311
k0_star(1430)+ PDGid=10321
k0_star(1430)- PDGid=-10321
k0_star(1430)0 PDGid=10311
    k1(1270)+ PDGid=10323
    k1(1270)- PDGid=-10323
    k1(1270)0 PDGid=10313
    k1(1400)+ PDGid=20323
    k1(1400)- PDGid=-20323
    k1(1400)0 PDGid=20313
    k2(1770)+ PDGid=10325
    k2(1770)- PDGid=-10325
    k2(1770)0 PDGid=10315
k2_star(1430)+ PDGid=325
k2_star(1430)- PDGid=-325
k2_star(1430)0 PDGid=315
k2_star(1980)+ PDGid=100325
k2_star(1980)- PDGid=-100325
k2_star(1980)0 PDGid=100315
k3_star(1780)+ PDGid=327
k3_star(1780)- PDGid=-327
k3_star(1780)0 PDGid=317
    k_star(1410)+ PDGid=100323
    k_star(1410)- PDGid=-100323
    k_star(1410)0 PDGid=100313
    k_star(1680)+ PDGid=30323
    k_star(1680)- PDGid=-30323
    k_star(1680)0 PDGid=30313
        k_star+ PDGid=323
        k_star- PDGid=-323
        k_star0 PDGid=313

```

```

        kaon+ PDGid=321
        kaon- PDGid=-321
        kaon0 PDGid=311
        kaon0L PDGid=130
        kaon0S PDGid=310
        lambda PDGid=3122
lambda(1405) PDGid=13122
lambda(1520) PDGid=3124
lambda(1600) PDGid=23122
lambda(1670) PDGid=33122
lambda(1690) PDGid=13124
lambda(1800) PDGid=43122
lambda(1810) PDGid=53122
lambda(1820) PDGid=3126
lambda(1830) PDGid=13126
lambda(1890) PDGid=23124
lambda(2100) PDGid=3128
lambda(2110) PDGid=23126
    lambda_c+ PDGid=4122
        mu+ PDGid=-13
        mu- PDGid=13
    neutron PDGid=2112
        nu_e PDGid=12
        nu_mu PDGid=14
        nu_tau PDGid=16
        omega PDGid=223
omega(1420) PDGid=100223
omega(1650) PDGid=30223
    omega- PDGid=3334
omega3(1670) PDGid=227
    omega_c0 PDGid=4332
opticalphoton PDGid=0
    phi PDGid=333
    phi(1680) PDGid=100333
phi3(1850) PDGid=337
pi(1300)+ PDGid=100211
pi(1300)- PDGid=-100211
pi(1300)0 PDGid=100111
    pi+ PDGid=211
    pi- PDGid=-211
    pi0 PDGid=111
pi2(1670)+ PDGid=10215
pi2(1670)- PDGid=-10215
pi2(1670)0 PDGid=10115
    proton PDGid=2212
rho(1450)+ PDGid=100213
rho(1450)- PDGid=-100213
rho(1450)0 PDGid=100113
rho(1700)+ PDGid=30213
rho(1700)- PDGid=-30213
rho(1700)0 PDGid=30113
    rho+ PDGid=213
    rho- PDGid=-213
    rho0 PDGid=113
rho3(1690)+ PDGid=217

```

```

rho3(1690)- PDGid=-217
rho3(1690)0 PDGid=117
  s_quark PDGid=3
sd0_diquark PDGid=3101
sd1_diquark PDGid=3103
sigma(1385)+ PDGid=3224
sigma(1385)- PDGid=3114
sigma(1385)0 PDGid=3214
sigma(1660)+ PDGid=13222
sigma(1660)- PDGid=13112
sigma(1660)0 PDGid=13212
sigma(1670)+ PDGid=13224
sigma(1670)- PDGid=13114
sigma(1670)0 PDGid=13214
sigma(1750)+ PDGid=23222
sigma(1750)- PDGid=23112
sigma(1750)0 PDGid=23212
sigma(1775)+ PDGid=3226
sigma(1775)- PDGid=3116
sigma(1775)0 PDGid=3216
sigma(1915)+ PDGid=13226
sigma(1915)- PDGid=13116
sigma(1915)0 PDGid=13216
sigma(1940)+ PDGid=23224
sigma(1940)- PDGid=23114
sigma(1940)0 PDGid=23214
sigma(2030)+ PDGid=3228
sigma(2030)- PDGid=3118
sigma(2030)0 PDGid=3218
  sigma+ PDGid=3222
  sigma- PDGid=3112
  sigma0 PDGid=3212
  sigma_c+ PDGid=4212
  sigma_c++ PDGid=4222
  sigma_c0 PDGid=4112
ssl_diquark PDGid=3303
su0_diquark PDGid=3201
sul_diquark PDGid=3203
  t_quark PDGid=6
  tau+ PDGid=-15
  tau- PDGid=15
  triton PDGid=1000010030
  u_quark PDGid=2
ud0_diquark PDGid=2101
ud1_diquark PDGid=2103
uul_diquark PDGid=2203
xi(1530)- PDGid=3314
xi(1530)0 PDGid=3324
xi(1690)- PDGid=23314
xi(1690)0 PDGid=23324
xi(1820)- PDGid=13314
xi(1820)0 PDGid=13324
xi(1950)- PDGid=33314
xi(1950)0 PDGid=33324
xi(2030)- PDGid=13316

```

```
xi(2030)0 PDGid=13326
  xi- PDGid=3312
  xi0 PDGid=3322
  xi_c+ PDGid=4232
  xi_c0 PDGid=4132
```

## Appendix 7 – Error Messages

Errors and warnings generated during execution are handled by the *G4Exception* function; they are printed as a 7-line message starting and ending with a row of asterisks – that makes them stand out visually in a long printout. Those that are issued by a routine beginning with “G4” came from Geant4 code, while those issued by a routine beginning with “BL” came from G4beamline code; those that are issued by a command name are also from G4beamline code. This list may not be complete; it does not attempt to list all error messages from libraries such as Geant4, CLHEP, GSL, OpenInventor, OpenGL, FFTW, Xwindows, etc.

Exception	Severity	Discussion
001	Warning	G4HadronicProcess – this is a coding error and is being investigated by the Geant4 team. It is rare and can be ignored unless a large number of tracks are affected
Alarm Signal	Fatal	The alarm timer fired. This timer is set 10 seconds longer than the eventTimeLimit, and will only fire if no steps were taken during those extra 10 seconds. Usually indicates an infinite loop somewhere in the program.
Missing material	Fatal	The specified material cannot be found.
UnknownParticle	Fatal	The specified particle does not exist.
Cannot Tune	Fatal	The pillbox cannot be tuned properly.
Tune Iteration Limit	Fatal	The pillbox tuning did not converge.
Invalid Step	Fatal	The pillbox tuning made an invalid step.
Tuning failed to converge	Fatal	Tuning did not converge.
Overwriting input file	Fatal	The trackermode filename is the same as the parameter histoFile, which will overwrite the previous run’s Root file.
Duplicate trackerplane-s	Fatal	Two or more trackerplane-s have the same name.
No Trackers	Fatal	The trackermode command has no trackers to control.
Invalid Start Expr	Fatal	The tune command has an invalid argument.
Iteration Limit	Fatal	The tune command did not converge within the limit.
Invalid Tune Expr	Fatal	The tune command has an invalid argument.
Failed to Converge	Fatal	The tune command did not converge.
Out of Memory	Fatal	Not enough memory.
Failed to achieve required accuracy	Fatal	The coil command could not achieve the required accuracy in constructing the field map, given the limits on its size.
Material not found	Fatal	The named material does not exist.
Invalid Coordinate Type	Fatal	Invalid coordinate argument
Invalid init during tracking	Event	Internal coding error.
Invalid start	Fatal	The start command is not valid.
Cannot determine reference coordinate segment	Event	The reference coordinates are not unique at the start of this track. Probably means that radiusCut is omitted or too large in some <i>corner</i> or <i>cornerarc</i> command.
Coordinate Object Missing	Event	Internal coding error.

Reference Coordinates not available	Fatal	Reference coordinates specified, but no applicable reference particle was tracked.
Output File Exists	Fatal	To prevent confusion BLFieldMap will not overwrite an existing file.
Nesting > 64	Fatal	Groups can only be nested to 64 levels.
Object Already Exists	Fatal	The name of an object is not unique.
Already Initialized	Fatal	Internal coding error.
No Physics Registered	Fatal	No <i>physics</i> command in the input file.
No beam Registered	Fatal	No <i>beam</i> command (or other beam).
G4VIS_USE not defined	Fatal	Attempt to use visualization, but the program has no visualization compiled into it.
Coordinates got lost	Event	Internal coding error.
Stuck Track	Track	A track has taken >100 steps without moving significantly. This is usually a very low energy track on a boundary.
Event time limit	Event	Simulating the event has taken more real time than the parameter eventTimeLimit.
Erroneous call to ...	Fatal	Internal coding error.
Cannot find NTuple	Fatal	NTuple name not found in Root file.
Cannot run beam	Fatal	Internal coding error.
Geometry not closed	Fatal	Internal coding error.
Cannot read viewers.def	Fatal	Visualization requested, but cannot find the file <i>viewers.def</i> that defines the viewers (it is normally in the G4beamline install directory).
Viewer not defined	Fatal	A viewer was requested that does not exist.
Large Primary TrackID	Warning	A beam track was read with TrackID > 1000, which can be confused with new secondary tracks having such TrackID-s. See section 7.14.

# References

- [1] Geant4 – <http://geant4.cern.ch>
- [2] CLHEP – <http://proj-clhep.web.cern.ch/proj-clhep>
- [3] HistoScope – <http://fermitools.fnal.gov/abstracts/histoscope/abstract.html>
- [4] The MICE Collaboration – <http://mice.iit.edu>
- [5] Root – <http://root.cern.ch>
- [6] “Beam simulation tools for GEANT4 (and neutrino source applications)”, hep-ex/0210057
- [7] Cygwin – <http://www.cygwin.com>
- [8] ffmpeg – <http://www.ffmpeg.org>
- [9] Coin – The Coin 3D graphics library – <http://coin3d.org>
- [10] Gnuplot – <http://www.gnuplot.info/>
- [11] OpenMPI – <http://openmpi.org>
- [12] NERSC: <http://nersc.gov>, the supercomputer itself is <http://hopper.nersc.gov>.