Manual of musrSim

Kamil Sedlák[1], Toni Shiroka[1], Zaher Salman[1], Tom Lancaster[2], Thomas Prokscha[1], Taofiq Paraiso[1]
[1] *Laboratory for Muon Spin Spectroscopy, Paul Scherrer Institut, CH-5232 Villigen PSI, Switzerland*
[2] *Clarendon Laboratory, Department of Physics, Oxford University, Parks Road, Oxford OX1 3PU, UK*

"musrSim" is a simulation program based on Geant4, optimised for the μSR instruments. This document describes some of the commands used in the user macros of the simulation program for the μSR instruments based on the GEANT4. The root output variables are also described.

1 Scope of the musrSim program

The program "musrSim" is a relatively general program that can be used to simulate the response of a μSR [1] instruments (detectors) to muons and their decay particles (electrons, positrons and gammas). Even though musrSim is tailored to the needs of the μSR technique [7], it has been used also in the detector development studies without any muons involved, e.g. to test the response of an APD-based scintillator counters to the irradiation of Sr radioactive source [2].

The program is based on the Geant4 [3] and Root [4] libraries. Geant4 is Monte Carlo toolkit used (not only) in particle physics to simulate the passage of particles through the detectors. Root is an analysis tool that allows to manipulate and analyse the simulated data, namely to plot histograms and other graphical output.

The aim of the musrSim is to provide an easy-to-use simulation program, which does not require a deep knowledge of Geant4 in order to simulate a (μSR) detector. In our view, the main advantages of musrSim are:

- Simple way how to define or modify the instrument geometry.
- Limited (ideally no) need to modify and/or recompile the source code.
- Implementation of the μSR-specific classes (muon spin rotation in magnetic fields, muonium formation and decay, ...).
- Possibility to read in the output files of the TURTLE [6] program for the beam-line simulation.
- Simple way how to define (overlapping) electromagnetic fields.
- Output in the Root tree.

On the other hand, there are also some drawbacks and limitations:

- The user has to have an installation of Geant4 and Root before installing musrSim.
- It is supposed the user will analyse the data with Root, therefore Root has to be installed and some knowledge of it is needed. Even though is relatively easy to simulate a μSR instrument, and to create the output Root file without even knowing the c++ programming language, some c++ programming is needed to analyse the simulation output and to plot graphs.
- At present time the program does not simulate any muon-spin related physics processes happening in the sample, except for the muon spin rotation.
- The simulation of one event takes much more time than the measurement of a real event. The simulation time depends on the complexity of the instrument geometry and on the presence of electromagnetic fields. To simulate one million of muons in the case of the PSI high-field instrument took about 10 hours of a computer time of a desktop PC.

2 How to install and run musrSim

To install and run musrSim, one has to install Geant4 and Root first. The present version of musrSim has been tested with Geant version 4.9.1, patch no. 3 and Root version 5.20.00.

Once Geant4 has been successfully installed and some of the default Geant4 exmples has been run, the musrSim installation package can be downloaded from the web page http://lmu.web.psi.ch/simulation/index.html. Usually the "env.sh" script has to be run to set-up the environment variables appropriatelly before the musrSim or any other Geant4 application can be compiled or run. The simulation can be executed by *"musrSim RUNNUMBER.mac"*, where RUNNUMBER.mac is a "macro file" containing the information about the instrument setup. The string "RUNNUMBER" represents the integer run number.

In order to simulate a new instrument, the user has to define the following blocks of information in the macro file:

- Define the geometry (the so-called "volumes") of the new instrument. Note that in Geant4 volumes can be included inside other volumes (a "daughter" volume is positioned inside the "mother" volume), and it is therefore necessary to distinguish between the global (world) coordinates and the coordinates of the daughter volumes (local coordinates). It is not allowed to overlap any two different volumes partially.
- Define the electric and magnetic fields.
- Define physics processes relevant for your case.
- Define the initial muon parameters (more generally – initial particle parameters).
- Define some other parameters influencing the execution of the simulation.
- Define which variables should be written out to the output Root tree.
- In case it is required to visualise the geometry, define the visualisation attributes.

By default, the output of the simulation is written out in the subdirectory "data" with the name "musr_RUNNUMBER.root". (Note that the execution of the simulation can be terminated gently by creating a file "RUNNUMBER.stop" in the working directory.)

3 Conventions

The default units of the musrSim in both the macro file (RUNNUMBER.mac) and in the Root tree are summarised in table 1.

Quantity	**Default unit**
Length	mm
Time	μs
Energy	MeV
Momentum	MeV/c
Magnetic field	tesla
Electric field	keV/mm

Table 1. The default units in musrSim.

4 Detector construction

/musr/command rotation *matrixName* α β γ
/musr/command rotation *matrixName vx vy vz angle*
These commands define a rotation matrix of the name *matrixName* that can be used later on during the definition of the detector geometry (see command "/musr/command construct"). It can be defined either by the Euler angles (if there are three float parameters behind the *matrixName*) or by the vector *(vx,vy,vz)* and an *angle* of rotation around this vector (if the fourth float parameter behind the *matrixName* is non-zero). All angles are specified in degrees.

/musr/command construct *solid name dimensions ... material x y z motherVolume matrixName sensitiveClass idNumber*

This command defines a volume in GEANT4 (It comprises three steps of GEANT4: defines a solid, logical volume and physical volume. Details can to be found in GEANT4 manual).

- *solid* (string) can be one of the G4VSolid.cc particular types, presently "tubs", "cons", "box", "trd", "sphere", "para", or it can be one of the specifically implemented solids by our program as "uprofile" (an U-profiled bar), "alcSupportPlate" (shape specific to ALC support plate), "tubsbox" (a tube with a rectangular hole along its axis), "tubsboxsegm" (a volume that looks like an intersection of tube and box) and "trd90y" (a trd volume rotated by 90 degrees around y axis in addition to the rotation requested by *matrixName*). Not all G4VSolids are presently supported, but it is relatively easy to implement a new kind of solids in the musrDetectorConstruction.cc class.
- *name* (string) stands for the name of the volume. As the command "/musr/command construct" constructs three kinds of classes/volumes (the solid, logical volume and physical volume), there are three names of the concrete volume used internally inside musrSim: sol_*name*, log_*name* and phys_*name*. The main volume, inside which all other volumes are positioned, has to be called "World".
- *dimensions* (floats) define the size of the required solid. They are kept equivalent to the dimensions of solids as used in GEANT4. For example the "box" is defined by its **halflengths** along x, y and z coordinates. Note that the number of *dimensions* varies for each type of solid. The units are mm for lengths and degrees for angles.
- *material* (float) one of the materials defined in GEANT4, namely in the file $G4INSTALL/source/materials/src/G4NistMaterialBuilder.cc (e.g. "G4_Galactic" for vacuum, "G4_Cu" for copper, "G4_AIR" for air, "G4_PLASTIC_SC_VINYLTOLUENE" for a scintillator, ...). One can also define a new material inside the function musrDetectorConstruction::DefineMaterials(). Presently "Mylar", "Brass" "Steel", "Macor", "MCPglass", "MgO", "SiO2", "K2O" and "B2O3" are defined there.
- *x, y, z* (floats) – coordinates of the volume, used to position the volume within its mother volume (as used by the G4PVPlacement). Thus these coordinates are interpreted in the local coordinate system of the *motherVolume*.
- *motherVolume* (string) – name of the mother volume, in which the given volume should be positioned. Note that the mother volume has to be defined first (before its daughter), and that the name of mother starts with a string **log_***name*, following the naming convention defined above. When the "World" volume is defined, its *motherVolume* should be set to "no_logical_volume".
- *matrixName* (string) – name of the rotation matrix that will be used to position the volume inside its mother volume (as used in member function G4PVPlacement). Use string "norot" if no rotation is required for the given volume. Otherwise the rotation matrix has to be defined by the command line "/musr/command rotation" **before** the given volume is defined.
- *sensitiveClass* (string) – specifies whether the volume is sensitive detector or just a piece of a "dead" material. Use the string "dead" for the latter, and the string "musr/ScintSD" for a scintillator (a sensitive volume, i.e. a volume where hits are observed). No other detector type (other than "dead" and "musr/ScintSD") is supported at the moment, but the program might be extended in the future (e.g. to properly include also the semiconductor tracking detectors, etc.).
- *idNumber* (int) – serves as a unique identifier of the volume. It is primarily used in the output Root tree to identify the volume: 1) in which a muon stopped (tree variable "muDecayDetID"), 2) in which hits were deposited in case of sensitive volume (the variable "det_ID[det_n]").

/musr/command region define *regionName logicalVolume*

The "G4Region" can be created using this command, and a logical volume of the name *logicalVolume* will be assigned to it. If the G4Region of the name *regionName* does not exist (i.e. the command "/musr/command region define" is called for the first time for this particular *regionName*), the G4Region will be created first, otherwise the logical volume *logicalVolume* will be just assigned to the already existing G4Region.

G4Region can be useful namely for setting some special Geant4 parameters (production cuts) just in some part of the detector (e.g. where a finer simulation is needed). See Geant4 manual for more details.

/musr/command region setProductionCut *regionName gammaCut electronCut positronCut*
Set the so-called "production cuts" in the G4Region called *regionName*. The variables *gammaCut, electronCut* and *positronCut* are given in mm.

Three special volumes "Target, M0, M1 and M2".

5 Electric and magnetic fields

/musr/command globalfield *fieldName half_x half_y half_z* uniform *X Y Z logicalVolume Bx By Bz Ex Ey Ez*
or
/musr/command globalfield *fieldName X Y Z* fromfile *fieldTableType fieldInputFileName logicalVolume fieldValue [fieldValueFinal] [fieldNrOfSteps]*
This command specifies the electric and/or magnetic fields, which are (in some sense) independent of any logical volume and can overlap with each other. In the case of tabulated field read in from and external field map file the field values used internally by the Geant4 are linearly interpolated using eight (3D) or four (2D) grid points surrounding the point of interest.

- *fieldName* (string) – name of the field (important mainly for the user and print-out messages of the musrSim.
- *half_x*, *half_y*, *half_z* (floats) – the (half) dimensions of the box, within which the uniform field is defined.
- **uniform / fromfile** – specifies whether the field is uniform within some volume or whether it is read in from an external file as a field-map.
- *X*, *Y*, *Z* (floats) – position of the centre of the field in the **global** coordinate system. IMPORTANT: For some technical internal Geant4 reasons, this POSITION HAS TO LAY WITHIN THE *logicalVolume*! (Note that the logical volume may be positioned somewhere deep in a volume structure, not directly within the "World" volume, and therefore the (local) coordinates in the definition of the the logical volume do not have to match the (global) coordinates *X*, *Y* and *Z*.
- *logicalVolume* (string) – specifies the logical volume, to which the field is "assigned". One may ask, why a logical volume is needed for a field "independent" of any Geant4 volume? The reason is purely technical - the logical volume is used to allow the field to be rotated the same way as the assigned logical volume. The field can be smaller or larger than the logical volume, to which it is assigned. The field extending out of the logical volume will also be used in the Geant4 calculations (will be not truncated). The only limitation is that the centre of the volume has to lay within the assigned logical volume (see above). Sometimes it might be useful to create a very small volume (e.g. of the order of 0.01 mm) to position a rotated field into a (differently rotated or unrotated) larger volume. The volume can also be made of vacuum (i.e. G4_Galactic).
- *Bx*, *By*, *Bz*, *Ex*, *Ey*, *Ez* (float) – the vector of the uniform electromagnetic field. The units are tesla, for the first three components, and kilovolt/mm for the last three components.
- *fieldTableType* (string) – specifies the format in which the field map is written in the file. In general, the field is specified in a grid of three space coordinates x, y and z (3D). Sometimes it is convenient to use the symmetry of the field and to reduce the field description to R and z (2D) only. In the following, we use this terms:
  nx, ny, nz or *nR, nz* – the number of divisions of the grid in *x, y, z* or *R, z*. *length unit* – the unit in which the grid coordinates are specified, usually cm or m.
  field normalisation factor – a multiplicative factor applied to the values of the field map to normalise the field (usually to 1 tesla or to 1 kV/mm in the centre of the field map).
  minimumx, maximumx, minimumy, maximumy, minimumz, maximumz – the minimum and maximum value in x, y and z coordinates. These values can be usually easily calculated from the field map itself, however the field can also be specified in a compact format in which case

the x, y and z coordinates are removed from the field map file, and the maxima and minima of coordinates have to be specified.
The following formats are supported:
3DB, 3DE – magnetic or electric field specified in x, y and z coordinate system. The first line of the file has to contain the information about *nx, ny, nz, length unit* and *field normalisation factor*. Optionally, a compact form of the field map can be specified, in which case *minimumx, maximumx, minimumy, maximumy, minimumz, maximumz* has to be added to the first line of the file. The next few lines of the field map file beginning with the character "%" are comments. The following lines specify the *x, y, z, Field_x, Field_y, Field_z* values (non-compact format) or *Field_x, Field_y, Field_z* values (compact format).
3DBOpera, 3DEOpera – 3D magnetic field in the form of OPERA output. It is expected that the *length unit* is 1 m, and the *field normalisation factor* is 1. (Note that this default normalisation is different from 2DBOpera and 2DBOperaXY options). However, a different *field normalisation factor* can be specified in the field map file using the keyword "fieldNormalisation *number*" before the line started with 0.
It is expected that the we first loop over the z coordinate of the field map, then (when z changed from minimum to maximum) it is looped over y coordinate, and the highest-lever loop goes over x coordinate. Hoever, if the order of looping is reversed in the field map, it can be specified using the keyword "variableIncreasingOrder xyz" placed in the field map before the line started with 0.
The *length unit* can be changed to 1 cm by specifying "[CENTIMETRE]" after the "0" character in the field map file.
It is expected that the field map is defined in the full volume of the field. Sometimes (due to the symmetry of the field), it is enough to define the field in just one octant of the Kartesian coorinate system (e.g. for positive x, y and z). In such cases, the user can specify this in the field map file using the keyword "symmetryType *number*", where the *number* specifies how the field should be extrapolated to other octants. The "symmetryType 1" case means that the planes of symmetry are (x,y) and (x,z), i.e. if the field at point (x, y, z) is (F_x, F_y, F_z), the field in different octants will look like this: $(-x, y, z) \rightarrow (F_x, F_y, F_z)$; $(x, -y, z) \rightarrow (F_x, -F_y, F_z)$; $(-x, -y, z) \rightarrow (F_x, -F_y, F_z)$; $(x, y, -z) \rightarrow (F_x, F_y, -F_z)$; $(-x, y, -z) \rightarrow (F_x, F_y, -F_z)$; $(x, -y, -z) \rightarrow (F_x, -F_y, -F_z)$; $(-x, -y, -z) \rightarrow (F_x, -F_y, -F_z)$.
Similar case is the "symmetryType 2", where the planes of symmetry are (x,y) and (y,z). These two symmetry types are realised in a the spin rotator oriented along the z axis.
Example of the beginning of the field map file:

```
2 2 55
1 X
2 Y
3 Z
4 BX
5 BY
6 BZ
7 DUMMY
fieldNormalisation -22.5733634
symmetryType 2
0 [METRE]
-0.2 -0.2 -1.35 0. 0. 0. 0.
-0.2 -0.2 -1.30 0. -0.0002 0. 0.
-0.2 -0.2 -1.25 0. -0.0002 0. 0.
-0.2 -0.2 -1.20 0. -0.005 0. 0.
...
```

2DB, 2DE – magnetic or electric field specified in R and z coordinate system. The first line of the file has to contain the information about *nR, nz, length unit* and *field normalisation factor*. The compact form of the field map (see 3DB case) is not supported. The next few lines of the field map file beginning with the character "%" are comments. The following lines specify the *R, z, Field_R Field_z* values.
2DBOpera – 2D magnetic field in the form of OPERA output. It is expected that the *length unit* is 1 cm, and the *field normalisation factor* is 0.00001 (Note that this default normalisa-

tion is different from 3DBOpera option). See example of **3DBOpera** for the usage of keyword "fieldNormalisation *number*". The data in the field map OPERA file are ordered as *R, dummy, z, Field_R, Field_z, dummy*
2DBOperaXY – same as 2DBOpera except that the data in the field map OPERA file are ordered as *R, z, dummy, Field_R, Field_z, dummy*

- *fieldInputFileName* (string) – Name of the field map file.
- *fieldValue* (float) – the value of the field at some reference point (usually in the centre of the field). It serves as some multiplicative factor. The units are tesla for the magnetic field and kV/mm for the electric field.
- *[fieldValueFinal]* and *[fieldNrOfSteps]* (floats) – these optional parameters allow the user to ramp up (down) the field during a single run. The *fieldValue* serves as the initial field value, the *[fieldValueFinal]* is the final value and *[fieldNrOfSteps]* specifies number of steps, in which the rump up/down will happen.

/musr/command globalfield *fieldName X Y Z* quadrupole *halfLength fieldRadius fringeFactor logicalVolume gradientValue [gradientValueFinal] [gradientNrOfSteps]*
Set up the field of a quadrupole magnet including the Enge function approximation of the fringe fields. The unit of *gradientValue* is T/m. The description is similar to the uniform field and to the tabulated fields. See "musrDetectorConstruction.cc" and "BLEngeFunction.hh" for the details.

/musr/command globalfield setparameter *parameterName parameterValue*
Set up some parameters used internally by Geant4 when calculating the motion of charged particles in the magnetic field.
parameterName (string) – one of the following parameters: "SetDeltaIntersection" "SetDeltaOneStep", "SetMinimumEpsilonStep", "SetMaximumEpsilonStep", "SetLargestAcceptableStep" and "SetMaxLoopCount". The exact meaning of these parameters can be found in Geant4 manual.

/musr/command globalfield printparameters
Print out the accuracy parameters (see "/musr/command globalfield setparameter").

/musr/command globalfield printFieldValueAtPoint *x y z*
Print out the field value at the point (x, y, z) (given in the global coordinate system).

/musr/command globalfield printFieldDerivativeAtPoint *x y z*
Print out the values of the derivative of the magnetic field at the point (x, y, z) (given in the global coordinate system).

6 Physics processes

/musr/command process addDiscreteProcess *particle process*
/musr/command process addProcess *particle process ordAtRestDoIt ordAlongSteptDoIt ordPostStepDoIt*
Adds processes for particles.
particle (string) – name of the particle to which a process is applied.
process (string) – name of the process to be assigned.
ordAtRestDoIt, ordAlongSteptDoIt, ordPostStepDoIt (int) – priority switches.
See the file musrPhysicsList.cc for the list of defined processes (e.g. G4MultipleScattering, G4eIonisation, ...) and Geant4 manual for the detail description of the processes.

There is one special process, combined from G4MultipleScattering and G4CoulombScattering, defined by the following command:
/musr/command process addProcess *particle* MultipleAndCoulombScattering *ordAtRestDoIt ordAlongSteptDoIt ordPostStepDoIt G4Region1* [*G4Region2*] [*G4Region3*]
The G4MultipleScattering (rough but very fast approximation of scattering) will be applied elsewhere in the detector, except for the *G4Region1* (and eventually *G4Region2* and *G4Region3*), where more precise but very slow process G4CoulombScattering will be applied instead of G4MultipleScattering. Note that up to three G4Regions are supported at the moment, but this limitation is not intrinsic to GEANT4 and it can be therefore changed in musrPhysicsList.cc, if needed. The G4Regions have to be defined in the detector construction phase by the command "/musr/command region define ...".

7 Initial (muon) beam parameters

/gun/primaryparticle *primaryParticleName*
(default: /gun/primaryparticle mu+)
Set the primary particle type, if it is not positive muon. For example, the negative muons are specified by "/gun/primaryparticle mu-".

/gun/meanarrivaltime *meanArrivalTime*
(default: /gun/meanarrivaltime 33.33333 microsecond)
Set mean arrival time difference between two subsequent muons (at continuos beam). The output variable "timeToNextEvent" is subsequently generated using the value of *meanArrivalTime* and filled into the Root tree.

/gun/vertex *x0 y0 z0 unit*
(default: /gun/vertex 0 0 -100 mm)
Set mean values of the x, y and z coordinates of the generated particles (muons). The smearing around these mean values instead is set by /gun/vertexsigma and restricted by /gun/vertexboundary (see below).
(Applicable also to TURTLE input).

/gun/vertexsigma *xSigma ySigma zSigma unit*
(default: /gun/vertexsigma 0 0 0 mm)
If $xSigma > 0$ ... set σ, i.e. the standard deviation (RMS), of the x coordinate of the generated particles (muons) to $\sigma = xSigma$. The x coordinate of the initial muon is then generated according to the Gaussian distribution with the mean value of $x0$ and the standard deviation of $xSigma$.
If $xSigma < 0$ ... the x coordinate of the initial muon is generated **uniformly** in the interval of ($x0-$ $xSigma$, $x0+$ $xSigma$).
If $xSigma = 0$ ... no smearing on the x coordinate is applied.
Similar is true for $ySigma$ and $zSigma$.
(Ignored by the TURTLE input).

/gun/starttime *t0 unit*
By default, muons are generated at time = 0. The time of generation of muons can be set randomly according to the Gaussian or uniform distribution using variables "/gun/starttime" and "/gun/starttimesigma". See the description of the "/gun/vertexsigma" to understand how the choice is done.

/gun/starttimesigma *t0 unit*
See the description of "/gun/starttime" command.

/gun/vertexboundary *R_max z_min z_max unit*
Set maximum allowed radius, and minimum and maximum z coordinate of the generated particles (muons). This command might be useful especially if the user wants to restrict the area, in which initial particles are created, e.g. to force the initial muons to be created inside the beam pipe.

If the particles (muons) are read in from an external TURTLE file, only the restriction on the maximum radius R_max is applied on the initial particles, while z_min and z_max are ignored.

/gun/vertexrelativer *relativeRMaxAllowed unit*
Set maximum allowed radius of the beam relative to x0 and y0, i.e. relative to the centre of the beam. This command might be useful when the beam centre is not created at x0=y0=0, but the user wishes to restrict the beam within a symmetric cone around x0 and y0.
(Ignored by the TURTLE input).

/gun/boxboundarycentre *xMaxSource0 yMaxSource0 xMaxSource0 unit*
See below "/gun/boxboundary".

/gun/boxboundary *xMaxSource yMaxSource xMaxSource unit*
Define a box, within which the generated particles (muons) are created. The variables *xMaxSource0*, *yMaxSource0* and *xMaxSource0* define the centre of the box, the variables *xMaxSource*, *yMaxSource* and *xMaxSource* define the halfwidth, halfheight and halflength of the box.
This command can be useful for muonium excitations by laser (low energy muon beam induced by

laser).
(Ignored by the TURTLE input).

/gun/kenergy *kineticEnergy unit*
Set the mean kinetic energy of the initial particles (muons).
(Ignored by the TURTLE input).

/gun/momentum *momentum unit*
Set the mean momentum of the initial particles (muons).
(Ignored by the TURTLE input).

/gun/momentumsmearing *momentumSigma unit*
Set σ, i.e. the standard deviation (RMS), of the momentum spread, which is applied randomly to each generated initial particle (muon). It is the magnitude of the momentum, which is smeared.
(Ignored by the TURTLE input. However, a similar command "/gun/turtleMomentumBite" can be used for the TURTLE input file.)

/gun/momentumboundary *p_min p_max dummy unit*
Set a boundary for the minimum and maximum momentum of the initial particles (muons). The third argument *dummy* is ignored.
(Presently ignored by the TURTLE input).

/gun/tilt *xangle0 yangle0 dummy unit*
The "beam tilt" is understood as a constant angle tilt that is applied to all initial particles (muons) regardless on their distance from the centre of the beam.
(Applicable also to TURTLE input).

/gun/tiltsigma *xangleSigma yangleSigma dummy unit*
Gaussian smearing of the tilt angle.
(Presently ignored by the TURTLE input).

/gun/pitch *pitch unit*
The "beam pitch" is understood as a variable angle applied to the initial particles (muons), which is directly proportional to the distance from the beam axis. The particles closer to the beam axis become smaller pitch than particles further away from the beam axis. The angle given as *pitch* will be applied to a particle generated 1 mm away from the beam centre, i.e. the particle generated 7 mm away from the beam axis will be assigned the angle of $7 \cdot pitch$. The pitch allows the user to focus or defocus the initial particle beam. Particles will be focused for positive pitch and defocused for the negative pitch.
(Applicable also to TURTLE input).

/gun/muonPolarizVector *xpolaris ypolaris zpolaris*
Set polarisation of the initial particle (muon) in a form of a vector $P = (xpolaris, ypolaris, zpolaris)$. The polarisation vector does not have to be normalised, the normalisation will be done internally by the program. However note that if the magnitude of P is set to less than 1e-8, the user can achieve an unpolarised muon beam. See the source code of musrPrimaryGeneratorAction.cc if you need to use unpolarised beam by this parameter, because there is some trick based on the magnitude of P.
(Applicable also to TURTLE input).

/gun/muonPolarizFraction *polarisFraction*
Set the fraction of the muon polarisation. The variable *polarisFraction* has to be set in the range of -1 to 1.
If *polarisFraction* is set to 1, all muons are polarised in the direction of polarisation vector defined by "/gun/muonPolarizVector".
If *polarisFraction* is set to 0, half of the muons are polarised in the direction of polarisation vector, the second half is polarised in the opposite direction, so in the end the muon beam should act as unpolarised.
If *polarisFraction* is set to -1, all muons are polarised in the direction opposite to the polarisation vector.
If *polarisFraction* is set to 0.9, then 95% of the muons is polarised in in the direction of polarisation vector, and 5% of them is polarised in the opposite direction!.

This command is ignored if magnitude of polarisation vector defined by "/gun/muonPolarizVector" is smaller than 1e-8!
(Applicable also to TURTLE input).

/gun/decaytimelimits *muDecayTimeMin muDecayTimeMax muMeanLife unit*
(default: /gun/decaytimelimits −1 −1 2197.03 ns)
If *muDecayTimeMax* is less or equal to zero, this command is ignored, and the muon decay time is set internally by GEANT4. Otherwise the muon will be forced to decay within a time interval given by *muDecayTimeMin* and *muDecayTimeMax*, and the mean muon lifetime will be set to *muMeanLife*. In such case *muDecayTimeMin* has to be equal or larger than 0 and *muDecayTimeMax* has to be larger **or equal** to *muDecayTimeMin*.
(Applicable also to TURTLE input).

/gun/turtlefilename *turtleFileName*
Set the filename of the TURTLE input file. If this variable is set, TURTLE file will be used to initiate muons. Otherwise the muons would be generated randomly. If the end of the TURTLE file is reached (because the user requested to simulate more events than saved in the TURTLE file), the TURTLE file be be rewind to its beginning. Note that this does not mean that the same events will be simulated after the rewind, because the random seed will be set differently than at the beginning of the simulation. Note that the muons initialised at the same position and with the same momentum will have completely different (random) multiple scattering, penetration depths, decay times, decay positron energies and angles, ..., and therefore will be (almost completely) different events not affecting the statistical quality of the sample.

/gun/turtleZ0position *z0_InitialTurtle unit*
Set the z-position which has been used to generate the TURTLE file.
If this value differs from the $z0$ value of the "/gun/vertex" command, than the particle initial position is extrapolated from $z0\_InitialTurtle$ to the point corresponding to $z0$, using the direction of its momenta.
MORE DETAILS:
When running TURTLE (e.g. when generating the TURTLE file using the TURTLE program), the user has to specify the z position, at which the TURTLE particles (muons) would be exported. Sometimes this z position does not correspond to the point of origin of the musrSim geometry. In such case, the variable *z0_InitialTurtle* should be set to the value, that in musrSim coordinate system corresponds to the point, at which the TURTLE file was exported. For example – if the TURTLE file was exported just after the last quadrupole of a beam-pipe, and in the simulation the edge of the last quadrupole corresponds to -100 cm, than the *z0_InitialTurtle* should be also set to -100 cm.

/gun/turtleInterpretAxes *axesWithSign*
Normally it is expected that the coordinates in TURTLE are x, xprime, y and yprime. One can specify whether the x and y axes of the position in TURTLE should be interpretted differently. The following options are supported for *axesWithSign*: x-y, -xy, -x-y, yx, y-x, -yx, -y-x .
Example: the option y-x means that first four coordinates in the TURTLE input file are interpreded as y, yprime, -x, -xprime.

/gun/turtleMomentumBite *turtleMomentumP0 turtleSmearingFactor dummy*
Modify the smearing of the momentum bite specified in the TURTLE input file. Normally the muon momentum is defined already in the TURTLE input file. This command allows the user to modify the momentum smearing (momentum bite) of the muon beam. The variable *turtleMomentumP0* will be taken as the mean momentum (in MeV/c), around which the momentum will be increased/decreased. It does not have to be the real mean value of the initial muon momentum distribution. The variable *turtleSmearingFactor* is the smearing factor in per cent, by which the momentum bite will be increased/decreased around the *turtleMomentumP0*. The following equation is used to change the muon momentum: p_{new} = *turtleMomentumP0* - (*turtleMomentumP0*-p_{TURTLE})·0.01·*turtleSmearingFactor*.
This means that:
turtleSmearingFactor = 100 ... the muon beam momentum will not be modified.
turtleSmearingFactor = 0 ... the muon beam momentum will be set to the constant value of *turtleMomentumP0*.

turtleSmearingFactor = 200 ... the muon beam will have two times broader distribution compared to the original TURTLE file.

/gun/turtleFirstEventNr *lineNumberOfTurtleFile*
Set the line number that should be taken as the first event from the TURTLE input file. This option is needed when the user wants to reproduce the simulation of an event using the same random number generator and TURTLE initial particle as in some previous run, however he wants to skip some (uninteresting) events at the beginning of the simulation.

/gps/*
In most cases, musrSim uses the so called "G4ParticleGun" to generate the primary particles (muons). The commands for G4ParticleGun were summarised previously, they start with /gun/ keyword. However, there is an alternative particle generator called "GPS (General Particle Source)", which is useful when simulating the decays of radioactive atoms and for other purposes. Whenever the /gps/ keyword is used, the "G4ParticleGun" is not initiated (and all /gun/* commands are ignored). The description of GPS can be found on the web, some of the useful commands are:
/gps/particle ion
/gps/ion 38 90 0 0
/gps/position 0 0 0
/gps/energy 0 keV
/gps/ang/maxtheta 2 deg
/gps/ang/maxphi 2 deg

8 Optical photons

Normaly the simulation of "detected signal" stops at the level of the deposited energy in a sensitive volume (e.g. in a scintilator tile). However, in some special cases, one would like to know how the light is propagated through the scintillators. In such case simulation of optical photons is possible. Note that the optical photon in Geant are treaded as a class of particles distinct from higher energy gamma particles – and there is no smooth transition between the two. Some additional material properties of an active detector and of the detector surface have to be defined for optical photons. We strongly recommend the users willing to simulate optical photons to read the chapter "Optical Photon Processes" in [5] to understand the rest of this chapter.

The simulation of optical photons will significantly (in some cases dramatically) reduce the speed of the program. The output of the optical photon simulation is stored in variables starting with the string "odet_". The user has to specify several parameters in order to simulate optical photons:

/musr/command G4OpticalPhotons *true*
Specifies to musrSim whether optical photons should be simulated or not. If this parameter is set to *false*, anything connected with optical photons will be ignored internally in musrSim, and the user therefore does not have to comment out other parameters connected with optical photons in the macro file.

/musr/command materialPropertiesTable *MPT_name property n E(1) ... E(n) val(1) ... val(n)*
Defines some optical *property* of a given material (e.g. absorption lenght, refractive index, scintillation yield, ...) to a material property table. *MPT_name* stands for the material property table name. The table has *n* different values *val(1) ... val(n)* for the same number of optical photon energies *E(1) ... E(n)* expressed in MeV. If *n*=0, the *property* is called "constant property". Some of the possible *property* keywords are: ABSLENGTH, RINDEX, FASTCOMPONENT, SLOWCOMPONENT, SCINTILLATIONYIELD, and some of the constant *properties* are SCINTILLATIONYIELD, RESOLUTIONSCALE, FASTTIMECONSTANT, SLOWTIMECONSTANT, and YIELDRATIO. See other *property* keywords in chapter "Optical Photon Processes" in [5].

SCINTILLATIONYIELD ... nr. of photons per 1 MeV of deposited energy.

RESOLUTIONSCALE ... intrinsic resolution – normally 1, larger than 1 for crystals with impurities, smaller than 1 when Fano factor plays a role.

YIELDRATIO ... relative strength of the fast component as a fraction of total scintillation yeald.

/musr/command setMaterialPropertiesTable *MPT_name material_name*
Assigns a material property table defined by "/musr/command materialPropertiesTable" to a given material.

/musr/command opticalSurface *surface_name phys_volName1 phys_volName2 surfaceType surfaceFinish surfaceModel MPT_name*
Defines "border surface" (G4LogicalBorderSurface, see subsection "Boundary Process" of chapter "Optical Photon Processes" in [5]).

surface_name name of the newly defined surface.

phys_volName1, phys_volName2 names of the physics volume in question (it is an ordered pair!).

surfaceType one of the following: dielectric_dielectric, dielectric_metal, dielectric_LUT, firsov, x_ray.

surfaceFinish surface finish properties, such as: polished, ground, etchedair, ...

surfaceModel one of the following: glisur, unified, LUT.

MPT_name optional "material properties table name" which should be assigned to the surface. E.g. REFLECTIVITY or EFFICIENCY for a given surface may be assigned this way.

One has to assign a non-zero EFFICIENCY and a REFLECTIVITY smaller than 1 to a boundary surface between the scintillator and sensitive device (e.g. an APD).

9 Some other parameters

/run/beamOn *nrOfEvents*
Specify how many events will be simulated in this run/job.
nrOfEvents (int) – number of events to be simulated.
(This is a default Geant4 command, which has to be specified in any simulation run).

/musr/command logicalVolumeToBeReweighted mu *logicalVolume weight*
(default: not defined; no reweighting is done unless explicitly requested by this command.)
Events can be reweighted by this command. If muon **stops and decays** in the volume *logicalVolume*, the event will be reweighted using the requested *weight*. Namely, only each n^{th} event will be stored in the output Root tree (n =*weight*) with the Root tree output variable "weight" set to *weight*, while other (non-n^{th}) events will be aborted. (The decision which event is to be stored and which to be aborted is done at random). This reweighting might be useful in the cases when the user wants to speed-up the simulation (respectively to reduce the number of fully simulated events), while keeping the high number of events interesting for the analysis. For example, one can set the reweighting of events in which muons stop in the collimator. The user should then use the *weight* stored in the Root tree when analysing the simulated data (i.e. when filling histograms). Compared to the simulation with no weighting applied, the histograms with weighted events will have larger errors, but the distributions should not differ more then within the statistical errors.
Note that the *weight* parameter is integer, and "mu" stands for "muons" (at the moment reweighting based on electrons or positrons is not supported).

/musr/command SetUserLimits *logicalVolume ustepMax utrakMax utimeMax uekinMin urangMin*
Set the so-called user limits (G4UserLimits) in a volume *logicalVolume*. The five last parameters correspond to the Geant4 methods "SetMaxAllowedStep", "SetUserMaxTrackLength", "SetUserMaxTime", "SetUserMinEkine" and "SetUserMinRange". **NOTE THAT G4StepLimiter AND/OR G4UserSpecialCuts HAS TO BE DEFINED FOR THE REQUIRED PARTICLE TYPES BEFORE CALLING THIS COMMAND!**
(E.g.: "/musr/command process addProcess mu+ G4StepLimiter -1 -1 5")
See chapter "5.7. User Limits" (namely "5.7.2. Processes co-working with G4UserLimits") of the Geant User Manual for more details about this issue.

The user can set G4UserLimits to logical volume and/or to a region. **At the moment, "/musr/command SetUserLimits" in musrSim supports the G4UserLimits in logical**

volumes only, not in the G4Regions. User limits assigned to logical volume do not propagate to daughter volumes, while User limits assigned to region propagate to daughter volumes unless daughters belong to another region. If both logical volume and associated region have user limits, those of logical volume win.

/musr/command storeOnlyEventsWithHits false
By default, only the events in which at least one hit in an active volume (detector) has been recorded are saved to the output Root tree, because the events with no hit in any detector will anyway not contribute to the real measurement (even not to the pileup background). However, the user has a possibility to use this command to store all events for some technical study, e.g. to learn where the muons stop in collimators, etc.

/musr/command storeOnlyEventsWithHitInDetID *volumeID*
This command is similar to the previous one. Only the events, in which there was at least one hit in the volume with the *volumeID* will be saved into the output Root tree. This command might be useful in some technical studies, it might introduce some bias in a physics study.

/musr/command storeOnlyTheFirstTimeHit true
This command specifies that only the hit that happens first will be saved, while all the other hits will be ignored. This command might be useful in some technical studies, it would be harmful in most physics studies.

/musr/command killAllPositrons true
It might be useful in some technical studies to abandon all positron tracks (to ignore all positrons). For example if the user wants to study where the muon hit detectors and where do they stop and decay, this command might help him to get rid of all hits caused by the decay positron. This command would be harmful in most physics studies.

/musr/command killAllGammas true
See "/musr/command killAllPositrons true" for the explanation.

/musr/command killAllNeutrinos false
By default the neutrino tracks are "killed" in the musrSim to speed up the simulation, because the neutrinos anyway do not interact with the detectors. (This "killing" of neutrinos does not affect the muon decay in any way). However, it might be useful not to kill the neutrinos when the user wants to display the complete muon decay event. This command allows one not to kill the neutrinos.

/musr/command getDetectorMass *logicalVolume*
This command prints out the mass of a given volume (detector) including all its daughter volumes (components).

/musr/command signalSeparationTime *timeSeparation*
There is some time for each detectors, during which it can not distinguish two subsequent hits. The command mimics such feature. If there are two energy deposits that happen in the same active volume (detector) within the time *timeSeparation* (in ns), then these two energy deposits are summed up into a single hit. Otherwise they will form two different hits. This is true regardless on whether the two energy deposits were induced by the same particle or by different particles. Presently the parameter *timeSeparation* is common to all scintillator detectors in the system, which means it is not possible to set different *timeSeparation* for a slow and fast scintillator detectors of the instrument.

/musr/command maximumRunTimeAllowed *timeMax*
If a musrSim job is run on a pc farm with a time limit on the job execution, and the job exceeds the time limit, the simulation will be killed. The output Root tree will be not closed properly, and the information stored in the Root vector "geantParametersD" will be not saved. To avoid the hard abort, the job will be terminated gently if its physical execution time exceeds *timeMax*. Note that the units of *timeMax* are seconds, and the default value is set to 85000 s (23 hours, 37 minutes).
(Note that the simulation can also be terminated gently by creating a file "RUNNUMBER.stop" in the working directory, where RUNNUMBER matches the run number specified in the name of the macro file.)

10 Output root tree variables

The value of -999 or -1000 indicates that the given variable could not be filled (was undefined in a given event). For example if the variable "muTargetTime" is set to -1000 it means that the initial muon missed the sample, and therefore no time can be assigned to the sample hit.

The user can choose which variables should not be stored in the output file using the command

/musr/command rootOutput *variableName* off
The *variableName* is identical with the variable names stored in the Root tree (see below). Presently the exception is "save" volume, for which all variables will be stored in the Root tree, if such a "save" volume is requested. Another exception are the Root tree variables "nFieldNomVal" and "fieldNomVal[nFieldNomVal]", which are both suppressed using the keyword "fieldNomVal". The last exceptions are the variables "fieldIntegralBx", "fieldIntegralBy", "fieldIntegralBz", "fieldIntegralBz1", "fieldIntegralBz2", "fieldIntegralBz3", which are usually not required in an analysis program, and they are therefore not written out to the Root tree by default. This can be changed using the command "/musr/command rootOutput *variableName* on".

The list of variables that can be stored in the Root tree:

runID (Int_t) – run ID number.

eventID (Int_t) – event ID number.

timeToNextEvent (Double_t) – time difference between two subsequent events (at continuous beam facility) (in μs). This time difference is generated randomly using the "meanArrivalTime" defined in "/gun/meanarrivaltime meanArrivalTime".

weight (Double_t) – event weight.

BFieldAtDecay_Bx, BFieldAtDecay_By, BFieldAtDecay_Bz, BFieldAtDecay_B3, BFieldAtDecay_B4, BFieldAtDecay_B5 (Double_t) – value of the 6 coordinates of the electromagnetic field at the position and time where and when the muon decayed. The first three coordinates correspond to the magnetic field (in tesla), the last three to the electric field (in kV/mm).

muIniTime (Double_t) – time when the initial muon was generated (in μs).

muIniPosX, muIniPosY, muIniPosZ (Double_t) – initial position where muon was generated (in mm).

muIniMomX, muIniMomY, muIniMomZ (Double_t) – initial momentum of the muon when it was generated (in MeV/c).

muIniPolX, muIniPolY, muIniPolZ (Double_t) – initial polarisation of the muon when it was generated.

muDecayDetID (Int_t) – ID number of the detector in which the muon stopped and decayed.

muDecayTime (Double_t) – the time at which the muon decayed (in μs).

muDecayPosX, muDecayPosY, muDecayPosZ (Double_t) – the position where the muon stopped and decayed (in mm).

muDecayPolX, muDecayPolY, muDecayPolZ (Double_t) – polarisation of the muon when it stopped and decayed.

muTargetTime (Double_t) – time at which the muon entered the volume whose name starts by "target" – usually the sample (in μs).

muTargetPolX, muTargetPolY, muTargetPolZ (Double_t) – polarisation of the muon when it entered the volume whose name starts with "target" – usually the sample.

muM0Time (Double_t) – time at which the muon entered the detector called "M0" or "m0" (in μs).

muM0PolX, muM0PolY, muM0PolZ (Double_t) – polarisation of the muon when it entered the detector called "M0" or "m0".

muM1Time (Double_t) – time at which the muon entered the detector called "M1" or "m1" (in μs).

muM1PolX, muM1PolY, muM1PolZ (Double_t) – polarisation of the muon when it entered the detector called "M1" or "m1".

muM2Time (Double_t) – time at which the muon entered the detector called "M2" or "m2" (in μs).

muM2PolX, muM2PolY, muM2PolZ (Double_t) – polarisation of the muon when it entered the detector called "M2" or "m2".

posIniMomX, posIniMomY, posIniMomZ (Double_t) – Initial momentum of the decay positron (in MeV/c).

nFieldNomVal (Int_t) – number of the elementary fields that make together the global field.

fieldNomVal[nFieldNomVal] (array of Double_t) – nominal values of all elementary fields. (They are usually constant, but sometimes they may vary from event to event).

BxIntegral, ByIntegral, BzIntegral, BzIntegral1, BzIntegral2, BzIntegral3 (Double_t) – calculates the field integrals along the muon path and path lengths defined as

$$\text{BxIntegral} = \int_{\mu \text{ path}} B_x(s)\, ds \tag{1}$$

$$\text{ByIntegral} = \int_{\mu \text{ path}} B_y(s)\, ds \tag{2}$$

$$\text{BzIntegral} = \int_{\mu \text{ path}} B_z(s)\, ds \tag{3}$$

$$\text{BzIntegral1} = \int_{Z_0}^{Z_{decay}} B_z(z)\, dz \tag{4}$$

$$\text{BzIntegral2} = \int_{\mu \text{ path}} ds \tag{5}$$

$$\text{BzIntegral3} = \int_{Z_0}^{Z_{decay}} dz \tag{6}$$

The units are tesla·m (for the first four variables) and mm (for the last two variables). To calculate the integrals properly, the user must force Geant to use very small step size (e.g. by using something like "/musr/command globalfield setparameter SetLargestAcceptableStep 2"), and probably also to generate the muons well outside the magnetic field and put target such that muons stop at $z = 0$.

Note that these variables are by default not calculated (and not stored) and therefore the user has to switch the calculation on by "/musr/command rootOutput fieldIntegralBx on" in the macro file.

nOptPhot (Int_t) – total number of optical photons generated in the current event.

det_n (Int_t) – number of "detector hits" in this event. Note that more then 1 detector might be hit, and even the same detector might be hit more than once. The hit might be induced by just one particle, by more then one particle originating from the same particle initially hitting the detector, or from more "independent" particles. For example, the decay positron can emit an Bremsstrahlung photon in the sample and then both the Bremsstrahlung photon and positron hit the same positron counter at approximately the same time.

det_ID[det_n] (array of Int_t) – ID number of the detector where the given hit occurred.

det_edep[det_n] (array of Double_t) – energy deposited in the given hit (in MeV).

det_edep_el[det_n] (array of Double_t) – energy deposited in the given hit due to electron-based interactions (in MeV).

det_edep_pos[det_n] (array of Double_t) – energy deposited in the given hit due to positron-based interactions (in MeV).

det_edep_gam[det_n] (array of Double_t) – energy deposited in the given hit due to photon-based interactions (in MeV).

det_edep_mup[det_n] (array of Double_t) – energy deposited in the given hit due to muon-based interactions (in MeV).

det_nsteps[det_n] (array of Int_t) – number of "steps" (in GEANT4 terminology) that were integrated together in the given hit. (The det_edep[] energy is the sum of the energy deposits during all these steps).

det_length[det_n] (array of Double_t) – the length of the trajectory of the particle (particles) that contributed to the given hit (in mm).

det_time_start[det_n], det_time_end[det_n] (array of Double_t) – the initial and final time belonging of the hit. It should be the "global time" of the track when the first and last hit occurred (in μs).

det_x[det_n], det_y[det_n], det_z[det_n] (array of Double_t) – the coordinates of the first step of the given hit.

det_kine[det_n] (array of Double_t) – should be kinetic energy of the first particle contributing to the hit, but it is not clear how to interpret this variable, so check the code for the exact meaning (in MeV).

det_Vrtx***[det_n]** – All the variables starting with "det_Vrtx" refer to the particle with the first (in time) energy deposit belonging to the given hit. (Note that the hit might be induced by more than one particle.) The vertex, at which the particle was created, may or may not be positioned within the sensitive volume, in which the hit is observed.

det_VrtxKine[det_n] (array of Double_t) – the kinetic energy of the first (in time) particle belonging to the hit.

det_VrtxX[det_n], det_VrtxY[det_n], det_VrtxZ[det_n] (array of Double_t) – the position of the vertex of the first particle that belongs to the given hit (in mm).

det_VrtxVolID[det_n] (array of Int_t) – ID of the detector in which the vertex (see above) was created.

det_VrtxProcID[det_n] (array of Int_t) – ID of the physics process in which the vertex (see above) was created.

det_VrtxTrackID[det_n] (array of Int_t) – track ID of the first particle that belongs to the given hit. If the track ID is negative, there were more than just one track contributing to this hit. The absolute value of det_VrtxTrackID[det_n] corresponds to the first (in time) track.

det_VrtxParticleID[det_n] (array of Int_t) – particle ID of the first particle that belongs to the given hit.

det_Vvv***[det_n]** – similar to the variables det_Vrtx*****[det_n] above, but if the first particle belonging to the hit was created inside of the logical volume where the hit occurs, then it's track is followed to its mother track (even several times) until the track (particle) is found that has been created outside the given volume. This way one can better investigate which (hopefully) single particle caused the hit. Even though even in this case it is not guaranteed that only a single particle gave origin to the hit, it is quite likely, though, that it was in fact just a single particle. If the

odet_n (Int_t) – number of "optical photon signals" in this event. Note that there can be several optical photon signals per detector. In principle, the optical photons contributing to the same signal may originate from more than one charged particle. Ideally, there should be the same number of optical photon signals as there are "detector hits", i.e. odet_n = det_n. However, detector hits are treated independently of the optical photon signals, and they do not have to match perfectly.

odet_ID[odet_n] (array of Int_t) – ID number of the detector where the given optical photon signal occurred. The detection of optical photons happens at the boundary of surfaces, and it is assigned to the volume ID *to* which the photon enters (and not *from* which it comes). This e.g. allows one to attach more APDs to one scintillator, in which case odet_ID[] will correspond to the volume IDs of individual APDs.

odet_nPhot[odet_n] (array of Int_t) – number of photons detected in the given optical photon signal.

odet_timeFirst[odet_n], odet_timeLast[odet_n] (array of Int_t) – times, when the the first and last photons contributing to the given signal were detected (odet_timeFirst and odet_timeLast).

odet_timeA[odet_n],
odet_timeB[odet_n], odet_timeC[odet_n], odet_timeD[odet_n], odet_timeE[odet_n] (array of Int_t) – time, when the n^{th} photon was detected, where n for *odet_timeA[i]* is given as *OPSA_fracA* multiplied by *odet_nPhot[i]*. Similary for *odet_timeB[i]*, ..., *odet_timeE[i]*. The variables *OPSA_fracA*, ..., *OPSA_fracE*, are defined by "/musr/command OPSA photonFractions" command (see chapter 8).

save_n (Int_t) – number of special kind of "save" volume that were hit in this event. The "save volume" is any volume whose name starts with letters "save". Their purpose in the simulation is usually to check positions and momenta of particles at some position of the detector, even if the particle does not deposit any energy in the given "save" volume. Save volumes can therefore be made of vacuum.

save_detID[save_n] (array of Int_t) – ID number of the save volume.

save_particleID[save_n] (array of Int_t) – particle ID of the particle that entered the save volume.

save_time[save_n] (array of Double_t) – time when the particle enetered in the volume (in μs).

save_x[save_n], save_y[save_n], save_z[save_n] (array of Double_t) – position of the particle where it entered the save volume ("GetPreStepPoint()") (in mm).

save_px[save_n], save_py[save_n], save_pz[save_n] (array of Double_t) – momentum of the particle when it entered the save volume (in MeV/c).

save_polx[save_n], save_poly[save_n], save_polz[save_n] (array of Double_t) – polarisation of the particle when it entered the save volume.

save_ke[save_n] (array of Double_t) – kinetic energy of the particle when it entered the save volume (in MeV).

11 Other outputs of the simulation

The output of the simulation is stored in the file "data/musr_RUNNUMBER.root". There are four kind of objects stored in this file:

Tree "t1" – tree containing simulated information for all events.

Vector "geantParametersD" – vector containing information valid for the whole run, e.g. the run number of the given run, number of generated events, ...

Histogram "hGeantParameters" – contains the same information as "geantParametersD", but in the form of histogram. The (only) reason for storing the same information in two different ways (TVector and histogram TH1D) is the feature of the root "hadd" command, used for merging two or more different simulated results into one file. The command "hadd" will sum-up variables stored in "hGeantParameters", while it will do nothing for variables stored in "geantParametersD". Both ways are useful for different type of information.

Other histograms – some other histograms can be filled during the simulation for debugging purposes. These are typically not interesting for the analysis of the results of the simulation.

12 Visualisation

/musr/command visattributes *volumeName colour*
/musr/command visattributes *materialName colour*
In case of visualisation, one can set the colour of a logical volume *volumeName* or of all volumes made of the material with the name *materialName*. The distinction between the two options is by the first four letters of the *volumeName* – if it contains the string "log_", it is considered as *volumeName*, otherwise it is considered to be a material with *materialName*.

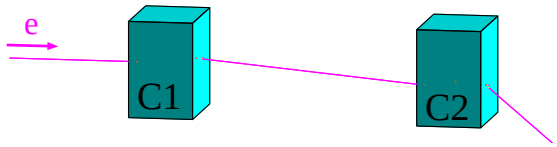

Figure 1. A simple simulation of an electron passing through two scintillator tiles.

Presently the following colours are predefined: "invisible", "white", "black", "red", "darkred", "green", "blue", "lightblue", "darkblue", "blue_style", "fblue_style", "yellow", "gray", "cyan", "magenta", "oxsteel", "MCP_style", "MACOR_style", "SCINT_style", "dSCINT_style", "VTBB_style", "Grid_style" and "RA_style".

New colours can be easily added, if needed, in the member function "musrDetectorConstruction::SetColourOfLogicalVolume".

13 Upgrading Geant4 version

Geant 4.9.1 and older - copy the file "src/G4DecayWithSpin.cc_for_Geant4.9.1_and_older" to the file "src/G4DecayWithSpin.cc". Also copy "src/G4EqEMFieldWithSpin.cc_for_Geant4.9.1_and_older" to "src/G4EqEMFieldWithSpin.cc".

Geant 4.9.2.p02 – (only this particular Geant version!) copy the file "src/G4EqEMFieldWithSpin.cc_for_Geant4.9.2p02_only" to the file "src/G4EqEMFieldWithSpin.cc".

Geant 4.9.2 and older - copy the file "src/musrPhysicsList.cc_Geant4.9.2p02_and_older" to the file "src/musrPhysicsList.cc".

Geant 4.9.3 From the point of view of musrSim, the largest change in version 4.9.3 with respect to 4.9.2 is the different treatment of the physics processes, therefore the musrPhysicsList.cc had to be modified, and also the physics processes listed in the *.mac file have to be changed! Note that using Geant 4.9.3 with old *.mac files will formally run OK, but the results of the simulation (slightly) differ from the same simulation based on Geant 4.9.2. However after updating the *.mac file, the simulation done with Geant 4.9.3 more or less reproduces results obtained with Geant 4.9.2.

14 Example 1 – Electrons passing through two scintillator tiles (101.mac)

One of the easiest example to illustrate the basic features of the musrSim (and/or Geant4) is to shoot electrons into a scintillator block, and to observe the energy deposited inside it. Figure 1 illustrates a simple geometry made of an electron source and two blocks of scintilator tiles with the dimensions of $3 \times 3 \times 2$ mm, which is defined in the macro file "101.mac".

15 Conclusions

The ... in [8].

16 Appendix A: Steering file for the simulation

```
# Macro file for seg06.cc
# set detector parameters
# This line fills some space
# This line fills some space
/run/beamOn 2
```

References

1. S.J. Blundel, Contemp. Phys. 40 (1999) 175.
2. A. Stoykov *et al.*, "First experience with G-APDs in μSR instrumentation", NDIP08, to be published in Nucl. Instrum. Meth. A.
3. S. Agostinelli, et al., Nucl. Instr. and Meth. A 506 (2003) 250.
   J. Allison, et al., IEEE Trans. Nucl. Sci. 53 (2006) 270.
4. R. Brun, F. Rademakers "ROOT - An Object Oriented Data Analysis Framework", Nucl. Inst. and Meth. in Phys. Res. A 389 (1997) 81.See also http://root.cern.ch/.
5. Geant4 User Manual
6. K.L. Brown, Ch. Iselin, D.C. Carey, *"Decay Turtle"*, CERN 74-2 (1974).
   U. Rohrer, *"Compendium of Decay Turtle Enhancements"*, http://pc532.psi.ch/turtcomp.htm
7. T. Shiroka *et al.* "GEANT4 as a simulation framework in muSR", Physica **B 404**, (2009) 966-969
8. A. Aktas *et al.* [H1 Collaboration], Submitted to Eur. Phys. J. **C**, [hep-ex/0401010].