# User Guide for TOPAS Version 1.3 (rev. 20151026a)

**Phase Space Sources** ................................................................. 68  
**Additional Control of Number of Histories** ......................................................... 70  
**Filtering Sources** .................................................................................. 71  

## Physics .......................................................................................... 72  
**Modular Physics Lists** ............................................................................... 72  
**Reference Physics Lists** .............................................................................. 76  
**User-Supplied Physics Lists** ....................................................................... 76  
**Multiple Physics Lists** ................................................................................ 76  
**Production Thresholds** ............................................................................... 76  
**Optical Photons** ......................................................................................... 77  
**Optical Surfaces** ........................................................................................ 78  

## Scoring ............................................................................................. 80  
**Volume Scorers** ......................................................................................... 80  
**Surface Scorers** .......................................................................................... 81  
**Phase Space Output** ..................................................................................... 83  
**Filtering Scorers** .......................................................................................... 85  
**Output Specification** ..................................................................................... 89  
**Binning in the Dividable Components, TsBox, TsCylinder and TsSphere** .......... 91  
**Binning by Energy** ....................................................................................... 91  
**Binning by Time** ............................................................................................ 91  
**Splitting a Scorer Under Control of a Time Feature** ...................................... 92  
**Statistical Information** .................................................................................. 93  
**Making Component's Color Change based on Scoring** ................................ 94  
**Toggling a Scorer Off and On** ...................................................................... 94  
**Restoring Results from Files** ....................................................................... 94  

## Graphics ............................................................................................ 95  

## Variance Reduction ........................................................................... 100  
**To Specify the Split Geometry** .................................................................. 100  
**To Define the Splitting Technique** .............................................................. 100  
**A Complete Example for GeometricalParticleSplit** .................................. 103  

## Time Features .................................................................................... 107  
**First Example** .............................................................................................. 107  
**Linear, Sine, Cosine and Sqrt Functions can Control Dimensioned Double or Unitless Values** ........................................................................................................ 107  
**Step Functions can Control Any Kind of Value - Dimensioned Double, Unitless, Integer, Boolean, or String** ........................................................................ 108  
**Combining Time Features for Complex Behaviors** ................................... 108  

## TOPAS Overall Control ................................................................... 110  
**Fixed Time Mode** ....................................................................................... 110  
**Sequential Time Mode** ................................................................................ 110  
**Random Time Mode** .................................................................................... 111  
**Random Seeds** ............................................................................................. 111  
**Additional Overall Control** ........................................................................ 112
Miscellaneous Notes ........................................................................................................114
  The Phase Space Format.................................................................................................114
  Changeable Parameters.................................................................................................117
  Transient Parameters.....................................................................................................117
  Quick Ways to Deactivate Parts of the Parameters Files .............................................117

Extending TOPAS ...........................................................................................................119
  Adding Geometry Components......................................................................................120
  Adding Scorers................................................................................................................123
  Adding Filters..................................................................................................................124
  Adding Physics Lists........................................................................................................125
  Adding Particle Sources..................................................................................................125
  Additional User Hooks...................................................................................................126

Authorship .......................................................................................................................126
Introduction to TOPAS

TOPAS wraps and extends the Geant4 Simulation Toolkit to provide an easier-to-use application for the medical physicist. TOPAS's unique parameter control system lets you assemble and control a rich library of simulation objects (geometry components, particle sources, scorers, etc.) with no need to write C++ code. Advanced users remain free to also code their own simulation objects to add to TOPAS (and while user-written objects benefit from underlying functionality of TOPAS base classes and the TOPAS parameter system, they can exploit the full flexibility of Geant4).

Users should read the first few sections of this Guide carefully (Paradigms and Parameters System, as well as the introductory parts of the sections Geometry, Particle Sources, Physics and Scoring). The rest of the guide provides a detailed reference that you may just want to skim initially.

Paradigms

TOPAS follows a consistent set of design paradigms. Understanding these paradigms will make the use of TOPAS more intuitive to you.

All control is through the TOPAS Parameters System.
Use of Geant4 macros or interactive commands is not supported as it does not give you the reliability and repeatability that comes from the parameters system.

- TOPAS Parameter files are not Geant4 macro files. TOPAS is specifically designed to avoid the kind of order-dependence risks that Geant4 macro files create.
- TOPAS Parameter files are not XML files. Those too involve the kind of order dependence that we explicitly avoid.

To keep OpenGL graphics from vanishing from the screen, you have the option to have TOPAS pause at the Geant4 command line by including the option:

- Ts/PauseBeforeQuit = "True" # defaults to "False"

To exit the Geant4 command line, and continue with the TOPAS session, type "exit"

For simulations that do not involve OpenGL graphics, just leave this option at "False".

All positions are set relative to Geometry Components.
If you want to place a particle source or a scorer, you place it relative to a particular Component. You may choose to do your placement relative to the center of the World component, in which case you have essentially used the overall coordinate system, but you will more likely choose some more directly relevant Component.
For example, a source that represents a particle beam might be placed at the beamline exit window. Doing so means that the source position will move appropriately with any nozzle movement.

All time dependent behaviors are controlled through the Time Feature system.

This guide covers how to run topas. For instructions on how to install TOPAS, see the TOPAS web site, http://topasmc.org or the README.txt included in the TOPAS release.
The TOPAS Parameters System

The TOPAS Parameter System is a control structure for applications in which a large number of complex inter-related parameters must be controllable by designers and end-users in a manner that is absolutely flexible but simultaneously easy to use. The system is designed with safety and repeatability as top priorities. A key error-checking strategy is strict type checking, in which every parameter must have a specific declared type (string, boolean, integer, etc.) and the provided values are checked to make sure they are appropriate to the given type.

The system takes a set of "Parameters Files," simple text files made up of lines of key/value pairs:
- Parameter_Type : Parameter_Name = Parameter_Value # Optional comment

When you edit parameter files, be careful to use a Plain Text editor. TOPAS will not understand the various hidden characters created by complex word processors (such as Word or Keynote). Whatever your editor, turn off advanced features such as "Smart quotes", "Smart dashes" and "Smart links".

Ten example parameter settings are given below:
- d:Ge/Phantom/HLX = 10. cm # Dimensioned Double
- u:Ge/Magnet/Dipole/DirectionX = 1.0 # Unitless Double
- i: Sc/DoseScorer/ZBins = 100 # Integer
- b:Sc/DoseScorer/Active = "True" # Boolean
- s:Ge/Phantom/Material = "Water" # String
- dv:Ge/RMW_Track1/Angles = 4 69.1 92.2 111.0 126.0 deg # Dimensioned Double Vector
- uv:Ma/Phantom_Plastic/Fractions=3 0.05549 0.75575 0.18875 # Unitless Double Vector
- iv:Gr/Color/yellow= 3 225 255 0 # Integer Vector
- bv:Tf/ScoringOnOff/Values=5 "true" "false" "true" "false" "true" # Boolean Vector
- sv:Ma/MyPlastic/Components=3 "Hydrogen" "Carbon" "Oxygen" # String Vector

The order of lines within a parameter file does not matter.

A Parameter_Name can be almost any string, but we have prefix conventions to keep things clear:
- Ma/ for Materials
- El/ for Elements
- Is/ for Isotopes
- Ge/ for Geometry Components
- So/ for Particle Sources
- Ph/ for Physics
- Vr/ for Variance Reduction
- Sc/ for Scoring
- Gr/ for Graphics
- Tf/ for Time Features
- Ts/ for TOPAS overall control

The Parameter_Type tells TOPAS what type of data will be in this parameter:
- d for Dimensioned Double
- u for Unitless Double
- i for Integer
- b for Boolean
- s for String
- dv for Dimensioned Double Vector
- etc. for uv, iv, bv and sv

The only forbidden characters in a parameter name are: = + - * " ' ` TAB NEWLINE and RETURN
The only forbidden characters in a parameter value are: = ' \\

TOPAS uses this Parameter_Type to perform "strict type checking," checking that the Parameter_Value is appropriate and complete for the given Parameter_Type.

A String parameter must be in quotes and may take any value.

A Boolean parameter must be in quotes and may be either:
  - "True", "t", or "1" (in any case) to mean true
  - "False", "f" or "0" (in any case) to mean false

An Integer parameter must be something that can be interpreted as an integer.
  - The value may not contain any decimal part, as this can lead to ambiguity as to the employed rounding strategy.
  - These are 32 bit integers, thus the values can range from 0 to 2147483647.

A Dimensioned Double parameter requires both a value and a unit.
  - We require the unit to avoid misunderstandings.
  - The value must be something that can be interpreted as a floating point number.

A Vector of Dimensioned Doubles parameter requires an integer (larger than zero) to indicate how many values are expected, then the values themselves, then a unit.
  - Vector of Dimensioned Doubles is useful when the definition of a single shape, motion, etc. requires multiple dimensioned double values.
  - Our usage of the term "vector" may be unfamiliar to some readers but is the standard term for such structures in modern programming languages.

Vectors of Unitless, Integer, Boolean and String again require an integer to indicate how many values are expected, then the values themselves. The individual strings in a Vector of Strings can not contain spaces (this requirement will be relaxed in a subsequent TOPAS release).

The comment character is #
Anything to the right of the comment character is taken as a comment.
Comments can span as many lines as desired, until a new line is found that contains the equals sign.

A given parameter name may not be defined more than once in a single file.
Blank lines are ignored.
Parameter names may use mixed case, but their interpretation is not case sensitive.
That is, "myParameter" is considered the same as "myparameter" or "myPaRaMeter", etc.
Relative Parameters

TOPAS supports "relative parameters", wherein one parameter may be set relative to another, as in:
- \( s_{Ge/Phantom/Material} = \text{SomeOtherParameterName} \)

The many uses of this relative parameter syntax become more clear once one understands the entirety of the TOPAS design, including hierarchical control files and time features.

With relative dimensioned double parameters, we must protect against a user setting a parameter relative to some other parameter that does not have appropriate units. The solution is to insist that a unit be included on the right side of the expression. In the example below, the unit of "cm" indicates that \( \text{SomeOtherParameter} \) must itself have units of length. If that other parameter’s unit is of the entirely wrong unit category (mass, angle, etc.), TOPAS will refuse to run. If the unit is of the right category but a different exact unit (m, mm, etc.), TOPAS will perform appropriate unit conversion:
- \( d_{Ge/Phantom/HLX} = \text{SomeOtherParameterName} \text{ cm} \)

TOPAS has a grammar for operations such as adding or multiplying parameters:
- \( \text{Ge/Compensator/ZTrans} = \text{Ge/Aperture/DistalEdge} + \text{Ge/Compensator/HLZ} \text{ mm} \)

Note that there must be a space before and after the plus sign.

Relative parameters allow only a limited number of functions, intentionally not a full math library, since other math functions may be ambiguous, requiring too much prior understanding of the mathematical syntax. The complete set of allowed syntax for any one parameter line is shown in the next section.

Hierarchical Control

Parameter files may pull in other parameters through "includeFile" statements, such as:
- \( \text{includeFile} = \text{someOtherParameterFile} \)

On most operating systems, the case of the file name matters - \( \text{MyIncludeFile.txt} \) is not considered the same file as \( \text{MYIncludeFile.txt} \) - so take care to match the exact case.

includeFile lines may appear anywhere in the parameter file.
includeFile can use either absolute or relative file positions, such as:
- includeFile = /Applications/topas/someDirectory/MyIncludeFile.txt
- includeFile = ../someDirectory/MyIncludeFile.txt

A file inherits all settings from its includeFiles, and can override any of those included settings by setting the same parameter name to a new value.

Type and kind of units has to match type and kind of units for the same parameter name in any includeFile.

Type can be omitted if the same parameter name has already been defined with a type in an includeFile.

When a parameter file includes another parameter file, and this in turn includes another parameter file, we refer to this as a "parameter file chain."

Parameter file chains fit nicely into research workflow. You can define most of your standard settings in one file, while a file higher on the chain overrides just those values that you want to change today.

To define a parameter in terms of the value of the same parameter in an includeFile, set the value to the same parameter name (always interpreted to mean this parameter value from an includeFile) or use the shortcut value, "inheritedValue". The following example would set a foil to be twenty percent thicker than in its included file:

- Ge/IonChamber/Layer2/Foil/HLZ = Ge/IonChamber/Layer2/Foil/HLZ mm * 1.2
- Ge/IonChamber/Layer2/Foil/HLZ = inheritedValue mm * 1.2

Note that there must be a space before and after the multiplication sign.

A basic set of TOPAS default parameters are built into the system (see Default Parameters below, after we discuss "parameter file graphs"). The defaults will set up a proton beam source at the edge of an empty world filled with air. You may override these defaults in your own parameter files if you wish.
Parameter File Graphs

TOPAS is designed to facilitate multiple independent workgroups focused on separate aspects such as treatment head design, patient handling and imaging devices. To this end, a parameter file may inherit settings from more than one other parameter file, forming a structure that is more complex than just a single chain. We refer to such a structure as a "parameter file graph." An example is shown below:

To implement such designs, a parameter file allows any number of includeFile statements, the statements may be located anywhere in the file, and you may specify one or more include files in a single includeFile statement, as in:

- includeFile = someFile someOtherFile someOtherOtherFile
- includeFile = stillAnotherFile

There is no significance to the left to right arrangement. That is:

- includeFile = fileA fileB fileC
  will behave the same as:
  - includeFile = fileC fileB fileA
  or even the same as if this was broken up into multiple includeFile statements:
  - includeFile = fileB
  - includeFile = fileC fileA

To keep this order from mattering, TOPAS does not allow you to set up a simulation object (a Geometry Component, a Scorer, etc.) in one chain but modify it in a different chain.

All parameters on a given Geometry Component need to be handled in the same chain.
All parameters on a given Scorer need to be handled in the same chain.
TOPAS also checks to make sure that no two parameter file chains modify the same parameter in a way that is ambiguous. If, for example, the material MySpecialTungstenAlloy has been defined in the imaging chain, it cannot also be defined differently in the treatment head chain (unless the top level file, the user file, itself defines this parameter in an absolute way, that is, not relative to any other parameter).

The basic paradigm controlling use of multiple parameter chains is that nothing you do in one chain should magically change the behavior of anything in another chain. Think of the simulation word as an actual physical room. We don't want the behavior of one thing in the room to magically change just because something else entered the room. So when you have multiple chains, TOPAS will check that neither chain modifies anything from the other chain.

This also means that no chain can redefine any of the Default Parameters, since all of the Default Parameters effectively belong at the base of every chain. The only place you can redefine the Default Parameters is in the top parameter file. This can feel like a heavy requirement, but it is essential. We want TOPAS to be a great tool for use in collaborative research environments where several people or teams may be contributing their own parts of the simulation setup.

It can take some experience to design complex parameter file chains. If you get stuck, feel free to ask for help on the TOPAS User Forum.

**Controlling Multiple Batch Jobs**

The hierarchial nature of parameter files makes it easy to control multiple batch jobs.

Make up a parameter file (or hierarchy of files) that has most of your settings:

- MostOfMySettings.txt

Then make small additional parameter files for each job you want to submit:

- Job1.txt
- Job2.txt
- Job3.txt

where each of these files has

- includeFile = MostOfMySettings.txt
- Ts/Seed = 1 # Set this differently for each of Job1, Job2, Job3, ...
- Sc/MyScorer/OutputFile = "Job1Output" # Set this differently for each of Job1, Job2, Job3, ...

Each job will thus have a unique starting random number seed (and hence produce a statistically distinct sample) and a unique output file specification, but all other aspects of the simulation will be identical from one job to the next.
**Default Parameters**

The following parameters are built-in by default. They are actually compiled into the code rather than set from a parameter file, so that all users will always have the same starting set of defaults. You can override any of these parameters in your own files.

### Overall program control
- `i:Ts/Seed = 1` # starting random seed
- `b:Ts/DumpParameters = "False"` # Set true to dump full set of parameters to html file TopasParameterDump_Run0.html
- `b:Ts/DumpNonDefaultParameters = "False"` # Like above but omits defaults
- `b:Ts/ListUnusedParameters = "False"` # Set true to list unused parameters on the console
- `i:Ts/ShowHistoryCountAtInterval = 1` # How often to print history count to the console
- `b:Ts/ShowHistoryCountOnSingleLine = "False"` # Set true to make history count reuse same line of console
- `b:Ts/PauseBeforeInit = "False"` # Pause for Geant4 commands before initialization
- `b:Ts/PauseBeforeSequence = "False"` # Pause for Geant4 commands before run sequence
- `b:Ts/PauseBeforeQuit = "False"` # Pause for Geant4 commands before quitting
- `i:Ts/RunVerbosity = 0` # Set to larger integer to see details of run. Maximum is 2
- `i:Ts/EventVerbosity = 0` # Set to larger integer to see details of event. Maximum is 5
- `i:Ts/TrackingVerbosity = 0` # Set to larger integer to see details of tracking
- `i:Ts/SequenceVerbosity = 0` # Set to larger integer to see details of TOPAS run sequence

### Overall timeline control
- `b:Tf/RandomizeTimeDistribution = "False"
- `d:Tf/TimelineStart = 0. s`
- `d:Tf/TimelineEnd = Tf/TimelineStart s`
- `i:Tf/NumberOfSequentialTimes = 1`
- `i:Tf/Verbosity = 0` # set to 1 to generate time log, set to 2 to get detailed update messages

### Optional checks on correctness of geometry
- `b:Ge/CheckForOverlaps = "true"
- `b:Ge/CheckForUnusedComponents = "true"

### Top level geometry component, the World Volume
- `s:Ge/World/Type="G4Box"
- `s:Ge/World/Material="Air"
- `d:Ge/World/HLX=5. m` # Half Length
- `d:Ge/World/HLY=5. m`
- `d:Ge/World/HLZ=5. m`
- `d:Ge/World/TransX=0. m`
- `d:Ge/World/TransY=0. m`
- `d:Ge/World/TransZ=0. m`
- `d:Ge/World/RotX=0. deg`
- `d:Ge/World/RotY=0. deg`
- `d:Ge/World/RotZ=0. deg`

### Default Beam position
- `s:Ge/BeamPosition/Parent="World"`
• s:Ge/BeamPosition/Type="Group"
• d:Ge/BeamPosition/TransX=0. m
• d:Ge/BeamPosition/TransY=0. m
• d:Ge/BeamPosition/TransZ= Ge/World/HLZ m
• d:Ge/BeamPosition/RotX=180. deg
• d:Ge/BeamPosition/RotY=0. deg
• d:Ge/BeamPosition/RotZ=0. deg

# Default Particle Source
• s:So/Default/Type = "Beam" # Beam, Twiss or PhaseSpace
• s:So/Default/Component = "BeamPosition"
• s:So/Default/BeamParticle = "proton"
• d:So/Default/BeamEnergy = 169.23 MeV
• u:So/Default/BeamEnergySpread = 0.757504
• s:So/Default/BeamShape = "Ellipse" # Point, Ellipse, Rectangle or Isotropic
• d:So/Default/BeamHWX = 10. cm
• d:So/Default/BeamHWY = 10. cm
• d:So/Default/BeamAngularSpreadX = 0.0032 rad
• d:So/Default/BeamAngularSpreadY = 0.0032 rad
• s:So/Default/BeamXYDistribution = "Gaussian" # Flat or Gaussian
• d:So/Default/BeamStandardDeviationX = 0.65 cm
• d:So/Default/BeamStandardDeviationY = 0.65 cm
• i:So/Default/NumberOfHistoriesInRun = 0

# Elements
• s:El/Hydrogen/Symbol="H"
• s:El/Helium/Symbol="He"
• s:El/Lithium/Symbol="Li"
• s:El/Beryllium/Symbol="Be"
• s:El/Boron/Symbol="B"
• s:El/Carbon/Symbol="C"
• s:El/Nitrogen/Symbol="N"
• s:El/Oxygen/Symbol="O"
• s:El/Flourine/Symbol="F"
• s:El/Neon/Symbol="Ne"
• s:El/Sodium/Symbol="Na"
• s:El/Magnesium/Symbol="Mg"
• s:El/Aluminum/Symbol="Al"
• s:El/Silicon/Symbol="Si"
• s:El/Phosphorus/Symbol="P"
• s:El/Sulfur/Symbol="S"
• s:El/Chlorine/Symbol="Cl"
• s:El/Argon/Symbol="Ar"
• s:El/Potassium/Symbol="K"
• s:El/Calcium/Symbol="Ca"
• s:El/Scandium/Symbol="Sc"
• s:El/Titanium/Symbol="Ti"
• s:El/Vanadium/Symbol="V"
• s:El/Chromium/Symbol="Cr"
• s:El/Tungsten/Symbol="W"
• s:El/Rhenium/Symbol="Re"
• s:El/Osmium/Symbol="Os"
• s:El/Iridium/Symbol="Ir"
• s:El/Platinum/Symbol="Pt"
• s:El/Gold/Symbol="Au"
• s:El/Mercury/Symbol="Hg"
• s:El/Thallium/Symbol="Tl"
• s:El/Lead/Symbol="Pb"
• s:El/Bismuth/Symbol="Bi"
• s:El/Polonium/Symbol="Po"
• s:El/Astatine/Symbol="At"
• s:El/Radium/Symbol="Ra"

# Materials
• s:Ma/DefaultColor="white"
• i:Ma/Verbosity = 0 # Set to 1 to report each time a material is defined

• sv:Ma/Vacuum/Components=4 "Carbon" "Nitrogen" "Oxygen" "Argon"
• uv:Ma/Vacuum/Fractions=4 0.000124 0.755268 0.231781 0.012827
• d:Ma/Vacuum/Density=1.0E-25 g/cm3
• s:Ma/Vacuum/State="Gas"
• d:Ma/Vacuum/Temperature=2.73 kelvin
• d:Ma/Vacuum/Pressure=3.0E-18 pascal
• s:Ma/Vacuum/DefaultColor="skyblue"

• sv:Ma/Carbon/Components=1 "Carbon"
• uv:Ma/Carbon/Fractions=1 1.0
• d:Ma/Carbon/Density=1.867 g/cm3
• d:Ma/Carbon/MeanExcitationEnergy=78 eV
• s:Ma/Carbon/DefaultColor="green"

• sv:Ma/Aluminum/Components=1 "Aluminum"
• uv:Ma/Aluminum/Fractions=1 1.0
• d:Ma/Aluminum/Density=2.6989 g/cm3
• i:Ma/Aluminum/AtomicNumber=13
• d:Ma/Aluminum/AtomicMass=26.98154 g/mole

• sv:Ma/Nickel/Components=1 "Nickel"
• uv:Ma/Nickel/Fractions=1 1.0
• d:Ma/Nickel/Density=8.902 g/cm3
• s:Ma/Nickel/DefaultColor="indigo"

• sv:Ma/Copper/Components=1 "Copper"
• uv:Ma/Copper/Fractions=1 1.0
• d:Ma/Copper/Density=8.96 g/cm3
• s:Ma/Copper/DefaultColor="orange"

• sv:Ma/Iron/Components=1 "Iron"
  uv:Ma/Iron/Fractions=1 1.0
  d:Ma/Iron/Density=7.87 g/cm³
  s:Ma/Iron/DefaultColor="skyblue"

• sv:Ma/Tantalum/Components=1 "Tantalum"
  uv:Ma/Tantalum/Fractions=1 1.0
  d:Ma/Tantalum/Density=16.654 g/cm³
  s:Ma/Tantalum/DefaultColor="indigo"

• sv:Ma/Lead/Components=1 "Lead"
  uv:Ma/Lead/Fractions=1 1.0
  d:Ma/Lead/Density=11.35 g/cm³
  i:Ma/Lead/AtomicNumber= 82
  d:Ma/Lead/AtomicMass=207.19 g/mole
  d:Ma/Lead/MeanExcitationEnergy=823 eV
  s:Ma/Lead/DefaultColor="brown"

• sv:Ma/Air/Components=4 "Carbon" "Nitrogen" "Oxygen" "Argon"
  uv:Ma/Air/Fractions=4 0.000124 0.755268 0.231781 0.012827
  d:Ma/Air/Density=1.20484 mg/cm³
  d:Ma/Air/MeanExcitationEnergy=85.7 eV
  s:Ma/Air/DefaultColor="lightblue"

• sv:Ma/Brass/Components=2 "Copper" "Zinc"
  uv:Ma/Brass/Fractions=2 0.7 0.3
  d:Ma/Brass/Density=8.550 g/cm³
  d:Ma/Brass/MeanExcitationEnergy=324.4 eV
  s:Ma/Brass/DefaultColor="grass"

• sv:Ma/Lexan/Components=3 "Hydrogen" "Carbon" "Oxygen"
  uv:Ma/Lexan/Fractions=3 0.055491 0.755751 0.188758
  d:Ma/Lexan/Density=1.2 g/cm³
  d:Ma/Lexan/MeanExcitationEnergy=73.1 eV
  s:Ma/Lexan/DefaultColor="grey"

• sv:Ma/Lucite/Components=3 "Hydrogen" "Carbon" "Oxygen"
  uv:Ma/Lucite/Fractions=3 0.080538 0.599848 0.319614
  d:Ma/Lucite/Density=1.190 g/cm³
  d:Ma/Lucite/MeanExcitationEnergy=74.0 eV
  s:Ma/Lucite/DefaultColor="grey"

• sv:Ma/Mylar/Components=3 "Hydrogen" "Carbon" "Oxygen"
  uv:Ma/Mylar/Fractions=3 0.041959 0.625017 0.333025
  d:Ma/Mylar/Density=1.40 g/cm³
  s:Ma/Mylar/DefaultColor="red"
sv:Ma/Mylon/Components=4 "Hydrogen" "Carbon" "Nitrogen" "Oxygen"
uv:Ma/Mylon/Fractions=4 0.097976 0.636856 0.123779 0.141389
d:Ma/Mylon/Density=1.140 g/cm^3
s:Ma/Mylon/DefaultColor="purple"

sv:Ma/Kapton/Components=4 "Hydrogen" "Carbon" "Nitrogen" "Oxygen"
uv:Ma/Kapton/Fractions=4 0.026362 0.691133 0.073270 0.209235
d:Ma/Kapton/Density=1.420 g/cm^3
s:Ma/Kapton/DefaultColor="purple"

sv:Ma/Water/Components=2 "Hydrogen" "Oxygen"
uv:Ma/Water/Fractions=2 0.111894 0.888106
d:Ma/Water/Density=1.0 g/cm^3
d:Ma/Water/MeanExcitationEnergy=75.0 eV
s:Ma/Water/DefaultColor="blue"

sv:Ma/Titanium/Components=1 "Titanium"
uv:Ma/Titanium/Fractions=1 1.0
d:Ma/Titanium/Density=4.54 g/cm^3
s:Ma/Titanium/DefaultColor="blue"

sv:Ma/Steel/Components=8 "Carbon" "Silicon" "Phosphorus" "Sulfur" "Chromium"
"Manganese" "Iron" "Nickel"
uv:Ma/Steel/Fractions=8 0.0015 0.01 0.00045 0.0003 0.19 0.02 0.67775 0.1
d:Ma/Steel/Density=8.027 g/cm^3
s:Ma/Steel/DefaultColor="lightblue"

# Physics
s:Ph/ListName = "Default"
s:Ph/Default/Type= "Geant4_Modular"
sv:Ph/Default/Modules = 7 "g4em-standard_opt3" "g4h-phy_QGSP_BIC_HP" "g4decay"
"g4ion-binarcascade" "g4h-elastic_HP" "g4q-stopping" "g4radioactivedecay"
d:Ph/Default/EMRangeMin = 100. eV
d:Ph/Default/EMRangeMax = 500. MeV
i:Ph/Default/dEdXBins = 220
i:Ph/Default/LamdaBins = 220

# Scoring
s:Sc/HistogramFileName = "topas" # name for root, hbook or xml output files

# Colors
iv:Gr/Color/lightblue= 3 175 255 255
iv:Gr/Color/skyblue= 3 175 124 255
iv:Gr/Color/red= 3 255 0 12
iv:Gr/Color/magenta= 3 255 0 255
iv:Gr/Color/violet= 3 224 0 255
iv:Gr/Color/pink= 3 255 0 222
iv:Gr/Color/indigo= 3 0 0 190
iv:Gr/Color/grass= 3 0 239 0
- `iv:Gr/Color/orange` = 3 241 224 0
- `iv:Gr/Color/purple` = 3 187 28 0
- `iv:Gr/Color/brown` = 3 225 126 66
- `iv:Gr/Color/grey` = 3 210 213 210
- `iv:Gr/Color/blue` = 3 0 0 255
- `iv:Gr/Color/green` = 3 0 126 126
- `iv:Gr/Color/yellow` = 3 225 255 0
- `iv:Gr/Color/white` = 3 255 255 255
- `iv:Gr/Color/black` = 3 0 0 0
- `iv:Gr/Color/grey020` = 3 20 20 20
- `iv:Gr/Color/grey040` = 3 40 40 40
- `iv:Gr/Color/grey060` = 3 60 60 60
- `iv:Gr/Color/grey080` = 3 80 80 80
- `iv:Gr/Color/grey100` = 3 100 100 100
- `iv:Gr/Color/grey120` = 3 120 120 120
- `iv:Gr/Color/grey140` = 3 140 140 140
- `iv:Gr/Color/grey160` = 3 160 160 160
- `iv:Gr/Color/grey180` = 3 180 180 180
- `iv:Gr/Color/grey200` = 3 200 200 200
- `iv:Gr/Color/grey220` = 3 220 220 220
- `iv:Gr/Color/grey240` = 3 240 240 240

# Graphics
- `b:Gr/Enable` = "True" # Set False to avoid instantiating any part of Geant4 visualization system (useful for running on batch machines that lack the OpenGL graphics library)
- `i:Gr/Verbosity` = 0 # Set to higher integer to increase verbosity of Geant4 visualization system
- `s:Gr/RefreshEvery` = "Run" # "History", "Run" or "Session"
- `i:Gr/ShowOnlyOutlineIfVoxelCountExceeds` = 8000 # Above this limit, only show outer box
- `i:Gr/SwitchOGLtoOGLIifVoxelCountExceeds` = 8000 # Above this limit, switch OpenGL Graphics to Immediate mode
Complete Set of Allowed Syntax for any one Parameter Line

Note that in all of the expressions below, there must be a space before and after any +, -, or *.

Dimensioned Double parameters:
- d: parameterName = number unit
- d: parameterName = number unit + name_of_dimensioned_double_parameter
- d: parameterName = number unit - name_of_dimensioned_double_parameter
- d: parameterName = number unit * name_of_unitless_or_integer_parameter
- d: parameterName = name_of_dimensioned_double_parameter unit
- d: parameterName = name_of_dimensioned_double_parameter unit * number
- d: parameterName = name_of_dimensioned_double_parameter unit *
  name_of_unitless_or_integer_parameter
- d: parameterName = name_of_unitless_or_integer_parameter * number unit
- d: parameterName = name_of_dimensioned_double_parameter + number unit
- d: parameterName = name_of_dimensioned_double_parameter - number unit
- d: parameterName = name_of_dimensioned_double_parameter +
  name_of_dimensioned_double_parameter unit
- d: parameterName = name_of_dimensioned_double_parameter -
  name_of_dimensioned_double_parameter unit
- d: parameterName = name_of_dimensioned_double_parameter +
  name_of_dimensioned_double_parameter unit
- d: parameterName = name_of_dimensioned_double_parameter -
  name_of_dimensioned_double_parameter unit

Unitless parameters:
- u: parameterName = number
- u: parameterName = number + name_of_unitless_or_integer_parameter
- u: parameterName = number - name_of_unitless_or_integer_parameter
- u: parameterName = number * name_of_unitless_or_integer_parameter
- u: parameterName = name_of_unitless_or_integer_parameter
- u: parameterName = name_of_unitless_or_integer_parameter + integer
- u: parameterName = name_of_unitless_or_integer_parameter -
  name_of_unitless_or_integer_parameter
- u: parameterName = name_of_unitless_or_integer_parameter *
  name_of_unitless_or_integer_parameter
- u: parameterName = name_of_unitless_or_integer_parameter +
  name_of_unitless_or_integer_parameter
- u: parameterName = name_of_unitless_or_integer_parameter -
  name_of_unitless_or_integer_parameter
- u: parameterName = name_of_unitless_or_integer_parameter *
  name_of_unitless_or_integer_parameter

Integer parameters:
- i: parameterName = integer
- i: parameterName = integer + name_of_integer_parameter
- i: parameterName = integer - name_of_integer_parameter
- i: parameterName = integer * name_of_integer_parameter
- i: parameterName = name_of_integer_parameter
- i: parameterName = name_of_integer_parameter + integer
- i: parameterName = name_of_integer_parameter - integer
- i: parameterName = name_of_integer_parameter *
  name_of_integer_parameter
- i: parameterName = name_of_integer_parameter + name_of_integer_parameter
- i: parameterName = name_of_integer_parameter - name_of_integer_parameter

i: parameterName = name_of_integer_parameter * name_of_integer_parameter

Boolean parameters:
- b: parameterName = value
- b: parameterName = name_of_boolean_parameter
- b: parameterName = name_of_boolean_parameter * name_of_boolean_parameter

String parameters:
- s: parameterName = string
- s: parameterName = string + name_of_integer_or_string_parameter
- s: parameterName = name_of_integer_or_string_parameter
- s: parameterName = name_of_integer_or_string_parameter +
  name_of_integer_or_string_parameter

Dimensioned Double Vector parameters:
- dv: parameterName = number_of_values value1 value2 ... valueN unit
- dv: parameterName = number_of_values value1 value2 ... valueN unit +
  name_of_dimensioned_double_or_double_vector_parameter
- dv: parameterName = number_of_values value1 value2 ... valueN unit -
  name_of_dimensioned_double_or_double_vector_parameter
- dv: parameterName = number_of_values value1 value2 ... valueN unit *
  name_of_dimensioned_double_or_double_vector_parameter or_integer_vector
- dv: parameterName = number_of_values value1 value2 ... valueN *
  name_of_unitless_or_integer_or_unitless_vector_or_integer_vector

Unitless Vector parameters:
- uv: parameterName = number_of_values value1 value2 ... valueN
- uv: parameterName = number_of_values value1 value2 ... valueN +
  name_of_unitless_or_integer_or_unitless_vector_or_integer_vector
- uv: parameterName = number_of_values value1 value2 ... valueN -
  name_of_unitless_or_integer_or_unitless_vector_or_integer_vector
- uv: parameterName = number_of_values value1 value2 ... valueN *
  name_of_unitless_or_integer_or_unitless_vector_or_integer_vector

Integer Vector parameters:
- iv: parameterName = number_of_values value1 value2 ... valueN
- iv: parameterName = number_of_values value1 value2 ... valueN +
  name_of_integer_or_integer_vector_parameter
- iv: parameterName = number_of_values value1 value2 ... valueN -
  name_of_integer_or_integer_vector_parameter
- iv: parameterName = number_of_values value1 value2 ... valueN *
  name_of_integer_or_integer_vector_parameter

Boolean Vector parameters:
- bv: parameterName = number_of_values value1 value2 ... valueN
value1, value2, etc. can be a numeric value or the name of a boolean parameter

String Vector parameters:
- sv: parameterName = number_of_values value1 value2 ... valueN
- sv: parameterName = number_of_values value1 value2 ... valueN +
  name_of_integer_or_string_or_integer_vector_or_string_vector

value1, value2, etc. can be a numeric value or the name of a string parameter

Other operations are intentionally not supported since their behavior might be unclear. Such things can be done in user C++ code, generating new parameters on the fly (see later discussion of "Transient Parameters"). d * d is forbidden because can create new units that we don't recognize. Division is forbidden because of divide by zero issues. etc.
Detailed Discussion and Syntax

Materials

We have pre-defined a few materials. You are free to define additional materials, as in:

- sv:Ma/Air/Components=4 "Carbon" "Nitrogen" "Oxygen" "Argon" # names of elements
- uv:Ma/Air/Fractions=4 0.000124 0.755268 0.231781 0.012827 # fractions of elements
- d:Ma/Air/Density=1.2048 mg/cm³
- d:Ma/Air/MeanExcitationEnergy=85.7 eV
- s:Ma/Air/DefaultColor="lightblue"

All Elements have been pre-defined with natural isotope abundance from the NIST database. You will only need to create your own Elements if you need something other than natural Isotope abundance. For that, see Elements and Isotopes below.

Fractions are by weight.

MeanExcitationEnergy is the I parameter in the Bethe equation, which not only includes ionization, but also inner-atomic excitations, etc.

In the section "Default Parameters" above, we show the complete list or pre-defined materials. This basically covers those materials that are used in our included examples.

You may also use any of the Materials and Compounds that are defined by default in Geant4. The names start with the prefix, "G4_", such as: "G4_Al", "G4_Ti", "G4_MUSCLE_SKELETAL_ICRP", etc. The complete list of these materials and compounds can be found at:

- NIST material names must be specified with exact case.
- As of this writing, the mean excitation energy listed in the above reference for G4_WATER is incorrect. It lists G4_WATER mean excitation energy as 75.0 eV but it is actually set to 78.0.

Special note on "Water" and "G4_WATER"

- "Water" is the water we define in TOPAS for our included examples. It has mean excitation energy set at 75.0 eV, consistent with the value used in commissioning tests at MGH. Like any TOPAS parameters, you can override this Water/MeanExcitationEnergy for your own work.
- "G4_WATER" is the water defined in NIST. It has a mean excitation energy of 78. You cannot override any of the parameters of this (or any) NIST material.
- In general, the only case where you will want to use "G4_WATER" rather than the TOPAS "Water" is if you intend to use Geant4’s G4-DNA physics modules, g4em-dna or g4em-dna-chemistry. These physics modules specifically work only for "G4_WATER".

It is up to you to define any additional materials that you want in your own parameter files. If you make your own material, make sure to pick a new material name (the string after the Ma/) and make sure that any other parameter file that uses this material includes the file where you defined this material (either directly or through a chain of includes).
Do not use the prefix "G4_" for the materials that you add. This prefix is reserved for materials and compounds from the pre-defined NIST database.

Where a pre-defined material definition exists, it is generally better to use that definition rather than making your own material. The pre-defined material may provide extra benefit by triggering specific corrections to ionization models.

If you redefine any of the default materials, the normal rules of parameter chains apply: the file with the new definition has to be at the base of all of the chains (otherwise you have an ambiguous situation). So, for example, if file A includes files B and C:
- You can not re-define a default material in file B or C
- You can re-define a default material in file D, that is included by both B and C

**Elements and Isotopes**

All Elements have been pre-defined with natural isotope abundance from the NIST database. You will only need to create your own Elements if you need something other than natural Isotope abundance. You can define additional elements as follows:

Define each Isotope that you will use, specifying Z, N and A:
- i:ls/U235/Z = 92
- i:ls/U235/N = 235
- d:ls/U235/A = 235.01 g/mole
- i:ls/U238/Z = 92
- i:ls/U238/N = 238
- d:ls/U238/A = 238.03 g/mole

Define your element with your desired proportion of these isotopes:
- s:El/MyEIU/Symbol = "MyEIU"
- sv:El/MyEIU/IsotopeNames = 2 "U235" "U238"
- uv:El/MyEIU/IsotopeAbundances = 2 90. 10.

See examples/Basic/Isotope.txt
Geometry Components

All Geometry Components must have at least the following parameters:

- \texttt{s:Ge/MyComponent/Parent = "World"}
- \texttt{s:Ge/MyComponent/Type = "TsBox"}
- \texttt{d:Ge/MyComponent/TransX=0.0 cm # defaults to 0}
- \texttt{d:Ge/MyComponent/TransY=0.0 cm # defaults to 0}
- \texttt{d:Ge/MyComponent/TransZ=0.0 cm # defaults to 0}
- \texttt{d:Ge/MyComponent/RotX=0.0 deg # defaults to 0}
- \texttt{d:Ge/MyComponent/RotY=0.0 deg # defaults to 0}
- \texttt{d:Ge/MyComponent/RotZ=0.0 deg # defaults to 0}

The Parent, Trans and Rot parameters place a component within its "mother" as described in the next section, Placement of Components.

Each Type has its own set of additional required parameters, discussed later when we show the specific component types.

The component name can include the forward slash character "/", and this is used in many examples to give some hints about component hierarchy, such as:

- \texttt{s:Ge/VBox2/Dipole/Parent = "Nozzle"}

This bit of hierarchy in the component name, such as "VBox2/Dipole", does NOT actually control how the components are assembled. The actual control is from the Parent parameter (discussed below). The forward slash is just another character here. You could just as well use "VBox2_Dipole" or "VBox2Dipole", as long as you use the same exact string whenever you refer to this component.

Components that are in the real world (as opposed to "Parallel Worlds", discussed later) must also have a material:

- \texttt{s:Ge/MyComponent/Material= "Air"}

To deactivate a Component (and all its children), you can either comment out the parameter that sets its Parent, or set its Include parameter to false, as in:

- \texttt{b:Ge/MyComponent/Include = "False" # defaults to "True"}

While it is not forbidden to have unused components (components that are never assigned a parent), this can often be a sign that you have not correctly assigned the parents in your geometry. Accordingly, we check for unused components on startup and given a warning message if any are found. You can disable this warning message by setting:

- \texttt{Ge/CheckForUnusedComponents = "False"}

In some cases you may want to keep unused components around. This can be like keeping extra pieces of unused laboratory equipment handy on a shelf. They will have no effect on your simulation, but remain available to quickly plug in when needed by assigning a parent and setting placement parameters.

Physics control for a specific component is done as part of the Ge/ parameters for that component rather than in the Ph/ parameters, such as:

- \texttt{d:Ge/MyComponent/MaxStepSize = 1. mm # sets maximum step size used in this component."}
**Placement of Components**

A component's "Parent" parameter tells which other component the current one is a child of. In this way, one can build a hierarchy of components.

- s:/Ge/MyComponent/Parent = SomeOtherComponent

The one component that is always provided automatically for you, into which you plug the rest of your hierarchy, is called "World".

Each component has three translation and rotation parameters. These give the position of the component in the coordinate system its parent component.

The following defines a box of air with half width of 5 m on each side placed at the center of the world:

- s:Ge/MyBox/Type="TsBox"
- s:Ge/MyBox/Parent = "World"
- s:Ge/MyBox/Material=Air
- d:Ge/MyBox/HLX=5. m # Half Length
- d:Ge/MyBox/HLY=5. m
- d:Ge/MyBox/HLZ=5. m
- d:Ge/MyBox/TransX=0. m
- d:Ge/MyBox/TransY=0. m
- d:Ge/MyBox/TransZ=0. m
- d:Ge/MyBox/RotX=0. deg
- d:Ge/MyBox/RotY=0. deg
- d:Ge/MyBox/RotZ=0. deg

If you set more than one rotation, note that rotation happens first around X, then the Y rotation is applied around the now-rotated axes, and then the Z rotation is applied around those rotated axes.

In general, a way to keep rotations more clear is to use intermediate Group components as follows:

- Place your component inside a Group component.
- Rotation your component around one axis.
- Rotate the group component around the other axis.

While the direction of rotation can be confusing, the correctness of rotations in TOPAS has been double, triple and quadruple checked and found consistent with the definitions in Geant4.

The TOPAS example:

- examples/Basic/Rotation.txt

shows an object being rotated first in the positive X direction, then in the positive Y direction, then in the positive Z direction.

For Geant4 experts, be advised that the rotation angles you provide to TOPAS are passed into:

- G4RotationMatrix()->rotateX, Y or Z.

Further discussion of Geant4 rotations can be found at:


The following overrides the size definition of the World box that was inherited from the built-in default parameters and then inserts a box into this world and another box into the first box:

- # Overrides the world size that was set in built-in defaults:
- Ge/World/HLX=10. m
- Ge/World/HLY=10. m
- Ge/World/HLZ=10. m
Overlap Checking

Because accidental overlaps of geometry volumes are a serious issue for all Monte Carlo simulations, Geant4 provides tools to automatically check for such overlaps. Overlap checking is not perfect, it works by testing a random set of points on each boundary, to see if they are inside any other boundary. Thus it will not necessarily find all overlaps. By default Geant4 checks 100 points on each volume (we will eventually provide a command to adjust this number of points). Overlap checking has a speed cost at initialization time, so once you are confident that your geometry has no overlaps, you may want to turn this feature off:

- Ge/CheckForOverlaps = "False"

Pre-defining Values

It may be useful to pre-define a range of named-values, such that you can easily access the values later. For example, we predefine the angles at which certain scatterers are stored on a scatterer selection wheel:

- d:Ge/Gantry1/Scatterer2/RotZForSS0 = 0. deg
- d:Ge/Gantry1/Scatterer2/RotZForSS8 = 270. deg
- d:Ge/Gantry1/Scatterer2/RotZForSS2 = 180. deg
- d:Ge/Gantry1/Scatterer2/RotZForSS3 = 90. deg

And then in our user file, the user doesn't have to know these actual angles, but can just rotate to one of the named scatterers:
Parallel Worlds

Components can be assigned to "parallel worlds" rather than the standard, mass world. Such components have no effect on physics (other than usually very minor step limitation effects) but can still be used for scoring. Such components can arbitrarily overlay the mass world. Their volumes can overlap any other volumes in other mass or parallel worlds.

To assign a component to a parallel world, include the line:

- b:Ge/MyComponent/IsParallel = "True"

Parallel world components may be children of other parallel world components.
Parallel world components may be children of mass world components.
Mass world components may not be children of parallel world components.

A new parallel world will be created each time you specify IsParallel, with an automatically generated parallel world name based on the component name. If you want to explicitly assign multiple components to the same parallel world, provide the additional parameter:

- s:Ge/MyComponent/ParallelWorldName = "SomeParallelWorldName"

This is useful if you will have many components in parallel worlds, as Geant4 only allows up to 8 total worlds.

In certain cases, TOPAS must represent a geometry by using a Geant4 technique called "parameterized volumes." However we have found that Geant4 behaves unreliably if parameterized volumes are placed in a parallel world. Accordingly, TOPAS applies safety restrictions:

- TsVPatient can not be in parallel world.
- TsCylinder can not be in parallel world if it has R or Phi divisions.
- TsSphere can not be in parallel world if it has any divisions.

Layered Mass Geometry

Components that are in a parallel world can have material or not. If they have material, and they are listed in the LayeredMassGeometryWorlds parameter, this material will take precedence over any real world material found in that location.

In Geant4 this is called Layered Mass Geometry. It is further described in:

Any time a component in a parallel world has material, that world must be listed in the LayeredMassGeometryWorlds parameter. The parameter is a string vector because any number of parallel worlds can have material. The order of the worlds in this parameter is significant. Material from worlds listed later in this list take precedence over material in worlds listed earlier. Thus, in the following example, material in the world Seed will take precedence over material in the world Tumor which will take precedence over material in the regular world.

- sv:Ph/Default/LayeredMassGeometryWorlds = 2 "Tumor" "Seed"

A simple example is provided in:
- examples/Basic/LayeredMassGeometry
Visualization Attributes

By default, Components are colored according to their materials, based on the default color that was assigned to the material, such as:

- s:Ma/Air/DefaultColor="lightblue"

Parameters of the Component let you override this color and set other visualization attributes:

- s:Ge/MyComponent/Color = "red"
- s:Ge/MyComponent/DrawingStyle = "Solid" # "Solid", "Wireframe" or "FullWireFrame". FullWireFrame includes drawing of additional edge lines that Geant4 calls "soft edges" - on many graphics devices WireFrame and FullWireFrame give the same result.
- i:Ge/MyComponent/VisSegsPerCircle = 100 # Number of line segments to use to approximate a circle, defaults to 24. Set to a larger number if you want a smoother curve
- b:Ge/MyComponent/Invisible = "True" # defaults to False meaning visible
Dividable Components ( type = "TsBox", "TsCylinder" and "TsSphere" )

TsBox Like G4Box, but can also specify divisions, XBins, YBins, ZBins
TsCylinder Like G4Tubs, but can also specify divisions, RBins, PhiBins, ZBins
TsSphere Like G4Sphere, but can also specify divisions, RBins, PhiBins, ThetaBins

Scorers associated with the dividable components may use the same or different divisions (thus one can do things like represent the patient with CT resolution but score with other resolutions). See scoring for details.

You should generally use these components rather than the G4Box, G4Tubs and G4Sphere, as they add more flexibility, but have no extra cost for cases in which you are not actually making divisions.

You can not place child components inside a divided component, but if the only reason for dividing this component is to have fine-grained scoring, you can easily work around this limitation. Use an undivided parent component. Place the children into this undivided parent component. Then when you specify that you want to score on this parent component, specify divided scoring (see the XBins, YBins and ZBins options in scoring). TOPAS will automatically create a parallel world version of your component to handle the divided scoring.

TsBox also allows you to specify different materials for each voxel.
**Generic Components** (type = "G4Box", "G4Tubs", "G4CutTubs", etc. below)

You can create a Geometry Component for any of the standard solids defined in the geometry section of the Geant4 Application Developers Guide:

The parameter file examples/Basic/ShapeTestWithAllParameters.txt demonstrates how to build each of the solids.

Below we list the parameters for each Geant4 solid. Further details about the parameters, along with helpful diagrams, can be found in the Geant4 Applications Developers Guide referenced above.

All parameters are type d: unless otherwise noted below.

Some of these parameters have default values. If so, the default value is shown in parentheses.

For most solids, sizes are specified in Half Lengths, denoted with an HL, such as HLX.

For a few solids, sizes are specified in full Lengths, denoted with just L, such as LX.

- **G4Box**
  - HLX, HLY, HLZ

- **G4Tubs**
  - RMin (0), RMax, HL, SPhi (0), DPhi (360 deg)

- **G4CutTubs**
  - RMin (0), RMax, HL, SPhi (0), DPhi (360 deg), LowNorm, HighNorm (these are both uv: with length of 3)

- **G4Cons**
  - RMin1 (0), RMax1, RMin2 (0), RMax2, HL, SPhi (0), DPhi (360 deg)

- **G4Para**
  - HLX, HLY, HLZ, Alpha, Theta, Phi

- **G4Trd**
  - HLX1, HLX2, HLY1, HLY2, HLZ

- **G4RTrap**
  - LZ, LY, LX, LTX

- **G4GTrap**
  - HLZ, Theta, Phi, HLY1, HLX1, HLX2, Alp1, HLY2, HLX3, HLX4, Alp2

- **G4Sphere**
  - RMin (0), RMax, SPhi (0), DPhi (360 deg), STheta (0), DTheta (180 deg)

- **G4Orb**
  - R

- **G4Torus**
  - RMin (0), RMax, RTor, SPhi (0), DPhi (360 deg)

- **G4HPolycone**
  - PhiStart, PhiTotal, Z, RInner, ROuter (these three are dv: with length of numZPlanes)

- **G4SPolycone**
  - PhiStart (0), PhiTotal (360 deg), R, Z (these two are dv: with length of numZPlanes)

- **G4HPolyhedra**
  - PhiStart (0), PhiTotal (360 deg), NSides (i:), Z, RInner, ROuter (these three are dv: with length of numZPlanes)

- **G4SPolyhedra**
  - PhiStart (0), PhiTotal (360 deg), NSides (i:), R, Z (these two are dv: with length of numRZ)

- **G4EllipticalTube**
  - HLX, HLY, HLZ

- **G4Ellipsoid**
  - HLX, HLY, HLZ, ZBottom (-HLZ), ZTop (HLZ)

- **G4EllipticalCone**
  - HLX, HLY, ZMax, ZTop (ZMax)

- **G4Paraboloid**
  - HLZ, R1, R2

- **G4Hype**
  - IR (0), OR, IS (0), OS, HLZ
- **G4Tet** Anchor, P2, P3, P4 (these four are dv: with length of 3)
- **G4Extruded** Polygons (dv: with length of 2 x number of polygons), HLZ,
  Off1 (dv: with length of 2),
  Scale1 (uv:),
  Off2 (dv: with length of 2),
  Scale2 (uv:)
- **G4TwistedBox** Twist, HLX, HLY, HLZ
- **G4RTwistedTrap** Twist, HLX1, HLX2, HLY, HLZ
- **G4GTwistedTrap** Twist, HLZ, Theta, Phi, HLY1, HLX1, HLX2, HLY2, HLX3, HLX4, Alpha
- **G4TwistedTrd** HLX1, HLX2, HLY1, HLY2, HLZ, Twist
- **G4GenericTrap** HLZ,
  Vertices (dv: with length of 2 x number of vertices
  Note: at least one user has seen bugs when placing several instances of G4GenericTraps in the same simulation. The shape looked correct in most visualizations but was not properly seen by particle tracking.
  Test by visualizing with RayTracer or RayTracerX, which itself uses Geant4 Tracking to create the image.
- **G4TwistedTubs** Twist, EndInnerRad, EndOuterRad, HLZ, Phi

Some examples of components that can be built just from TsGenericComponents:
- Scatterer
- Collimator
- Mirror
- Water Tank
- Rando Phantom (as constructive solid geometry rather than DICOM import)
- Pin Diode Chamber
- Flat Panel Imaging Device
- Standard Ion Chamber
- Segmented Ion Chamber
- Faraday Cup

We have built some complex things just from combinations of the above Generic Components (such as the STAR radiosurgery beamline at MGH).
**Group Component** ( type = "Group" )

Creates no actual solid, but still has a placement (Trans and Rot). Other components placed within this Group component are affected by this placement just as if the group were an enclosing box component.

The following defines a group component called "MyGroup":
- s:Ge/MyGroup/Type="Group"
- s:Ge/MyGroup/Parent = "World"
- d:Ge/MyGroup/TransX=2. m
- d:Ge/MyGroup/TransY=2. m
- d:Ge/MyGroup/TransZ=0. m
- d:Ge/MyGroup/RotX=0. deg
- d:Ge/MyGroup/RotY=0. deg
- d:Ge/MyGroup/RotZ=30. deg

The following example shows how a Group Component, "Jaws", placed in a nozzle, allows one to position two individual movable collimator blocks, "Jaw_Upper" and "Jaw_Lower", without the creation of an extraneous mother volume:
- s:Ge/Jaws/Type = "Group"
- s:Ge/Jaws/Parent = "Nozzle"
- d:Ge/Jaws/TransZ = 0. m
- ...
- s:Ge/Jaw_Upper/Type = "TsBox"
- s:Ge/Jaw_Upper/Parent = "Jaws"
- s:Ge/Jaw_Upper/Material = "Tungsten"
- d:Ge/Aperture/TransY = 2. cm
- ...
- s:Ge/Jaw_Lower/Type = "TsBox"
- s:Ge/Jaw_Lower/Parent = "Jaws"
- s:Ge/Jaw_Lower/Material = "Tungsten"
- d:Ge/Aperture/TransY = -2. cm
- ...
Specialized Components:

- Dipole Magnet
- Quadrupole Magnet
- Tabulated3DField Magnet
- Range Modulator Wheel
- Propeller
- Ridge Filter
- Multi Wire Chamber
- Multi Leaf Collimator
- CAD (Computer Aided Design)
- Aperture
- Compensator
- Patient in DICOM Patient
- Patient in XIO Format

Each of the specialized components has its own set of special parameters. Usage is best learned by studying the examples parameter files included in TOPAS (see following sections).

You may write your own additional components (see Extending TOPAS at the end of this user guide).

The following figure from Samsung Medical Center shows how their very specific quadrupole magnet system was coded as a TOPAS geometry.

An example of implementing user’s own geometry in TOPAS.
Dipole Magnet (type = "TsDipoleMagnet")

Simulation of proton beam steering by using magnet systems is efficiently implemented in TOPAS. TOPAS provides three types of magnets, i.e., dipole, quadrupole, and 3D fields map from external file. Due to the difficulties of providing a generic model for whole dipole system including yokes and coils, TOPAS offers simple magnetic fields model. However, users can write code to model their own yoke and coil system having a TOPAS field object so that the magnet fields can be controlled by using Time Features (See above SMC Quadrupole figure). Here is a complete set of the parameters for the Dipole magnet (examples/SpecialComponents/SimpleMagnet.txt):

Common parameters: type of geometry, position, and rotations: Ge/VBox2/Dipole/
  • s:Ge/VBox2/Dipole/Parent="Nozzle"
  • s:Ge/VBox2/Dipole/Material= "G4_AIR" #Material the field area is filled with.
  • d:Ge/VBox2/Dipole/HLX = 10 cm #Field dimensions
  • d:Ge/VBox2/Dipole/HLY = 10 cm
  • d:Ge/VBox2/Dipole/HLZ = 20 cm
  • d:Ge/VBox2/Dipole/TransX = 0.0 cm #Field position and rotation
  • d:Ge/VBox2/Dipole/TransY = 0.0 cm
  • d:Ge/VBox2/Dipole/TransZ = 90.0 cm
  • d:Ge/VBox2/Dipole/RotX = 0.0 deg
  • d:Ge/VBox2/Dipole/RotY = 0.0 deg
  • d:Ge/VBox2/Dipole/RotZ = 0.0 deg

Dipole specific parameters: Field direction and Strength
  • u:Ge/VBox2/Dipole/DirectionX = 1.0
  • u:Ge/VBox2/Dipole/DirectionY = 0.0
  • u:Ge/VBox2/Dipole/DirectionZ = 0.0
  • d:Ge/VBox2/Dipole/Strength= 0.3 tesla

You can vary the dipole field strength as a function of time by utilizing a Time Feature as following:
  • d:Ge/VBox2/Dipole/Strength= Tf/BField1st/Value tesla
Quadrupole Magnet ( type = "TsQuadrupoleMagnet" )

Here is a complete set of the parameters for the quadrupole magnet (examples/SpecialComponents/QuadAndDipoleMagnets.txt):

Common parameters: type of geometry, position, and rotation
- s:Ge/QBox1/Quad/Type="TsQuadrupoleMagnet"
- s:Ge/QBox1/Quad/Parent="Nozzle"
- s:Ge/QBox1/Quad/Material="G4_AIR"
- d:Ge/QBox1/Quad/HLX = 15 cm
- d:Ge/QBox1/Quad/HLY = 15 cm
- d:Ge/QBox1/Quad/HLZ = 10 cm
- d:Ge/QBox1/Quad/TransX = 0.0 cm
- d:Ge/QBox1/Quad/TransY = 0.0 cm
- d:Ge/QBox1/Quad/TransZ = 160 cm
- d:Ge/QBox1/Quad/RotX = 0.0 deg
- d:Ge/QBox1/Quad/RotY = 0.0 deg
- d:Ge/QBox1/Quad/RotZ = 0.0 deg

Quadrupole-specific parameter:
- d:Ge/QBox1/Quad/Gradient = 1.0 tesla/cm
**Tabulated3DField Magnet (** type = "TsTabulated3DField"")

In order to represent a magnetic field in cartesian coordination, TOPAS reused the code included in Geant4 package, (purging_magnet in advanced example) so that users can import a three dimensional fields values, which is in OPERA-3D format. However TOPAS offers more capabilities, e.g., changing fields files as a function of time, moving the field area, etc. The example below shows the latter. (examples/SpecialComponents/PurgingMagnet_move.txt):

Common parameters: type of geometry, position, and rotation
- s:Ge/V3DBox/Field3D/Type = "TsTabulated3DField"
- s:Ge/V3DBox/Field3D/Parent = "MagGroup"
- s:Ge/V3DBox/Field3D/Material = "Vacuum"
- d:Ge/V3DBox/Field3D/HLX = 5 cm #size of fields
- d:Ge/V3DBox/Field3D/HLY = 17 cm
- d:Ge/V3DBox/Field3D/HLZ = 26 cm
- d:Ge/V3DBox/Field3D/RotX = 0.0 deg
- d:Ge/V3DBox/Field3D/RotY = 0.0 deg
- d:Ge/V3DBox/Field3D/RotZ = 0.0 deg
- d:Ge/V3DBox/Field3D/TransX = 0.0 mm #Position of field but it moves backwardly and forwardly.
- d:Ge/V3DBox/Field3D/TransY = 0.0 mm
- d:Ge/V3DBox/Field3D/TransZ = Tf/BackForward/Value mm

Tabulated3DField-specific parameters: file name:
- s:Ge/V3DBox/Field3D/3DTable = "PurgMag3D.TABLE"

You can change input files as a function of time (see Time Feature section below).
**Range Modulator Wheel** (type = "TsRangeModulator")

TOPAS Range modulator is designed to accommodate various specifications from a vendor. We suggest to create or model your Range Modulator Wheel (RMW) by following procedure:

- Define the dimension of RMW drum, such as thickness and material of shell and hub (see figure below). Tracks will be placed in between the hub and the shell.
- This space (in between hub and shell) is vertically divided into three sections named, "Upper", "Middle", and "Bottom" so that each section can have its own tracks. You can adjust heights of these sections. The sum of these heights is the total height of your RMW.
- In order to reserve spaces for tracks, divide radially each section into as many as tracks you want by using the parameter, "RadialDivision."
- Using vector parameters, configure the tracks individually such as each block’s height, span angle, and material. Then assign vector parameter to the parameter, called "Pattern."

![Diagram of TOPAS RM dimensions](image)

Illustration for TOPAS RM dimensions. Tracks are placed in between Rout of Hub and Rin of Shell and this area is to be radially divided in case of placing multiple tracks. There are three vertical rooms, so it is possible to make double sided RMWs with an interface disk.
An example of RMW; (a) Perspective view. Upper section is divided into two but only inner radial division has a track pattern. In middle section, two track patterns are used to make a hole. (b) X-Y view from +z of RMW and (c) X-Y view from -z of RMW. Tracks are drawn in wireframe style, so more lines on the tracks are shown than the number of blocks.

Here is the complete set of the parameters for the above RMW (examples/SpecialComponents/RangeModulator.txt):

Common parameters: type of geometry, position, and rotation
- s:Ge/RangeModulatorA/Type = "TsRangeModulator"
- s:Ge/RangeModulatorA/Material = "Parent"
- s:Ge/RangeModulatorA/Parent = "World"
- d:Ge/RangeModulatorA/TransX = 10.0 cm
- d:Ge/RangeModulatorA/TransY = 0.0 cm
- d:Ge/RangeModulatorA/TransZ = 0.0 cm
- d:Ge/RangeModulatorA/RotX = 0.0 deg
- d:Ge/RangeModulatorA/RotY = 0.0 deg
- d:Ge/RangeModulatorA/RotZ = 0.0 deg
- b:Ge/RangeModulatorA/Invisible = "TRUE"

Set height of each sections and total height = 160.0 mm
- d:Ge/RangeModulatorA/HeightOfUpper = 150 mm
- d:Ge/RangeModulatorA/HeightOfMiddle = 1.0 mm
- d:Ge/RangeModulatorA/HeightOfLower = 9.0 mm

Shell dimensions, material, color, etc.
- d:Ge/RangeModulatorA/Shell/Rin = 15.0 cm
- d:Ge/RangeModulatorA/Shell/Rout = 15.5 cm
Hub dimensions, material, color, etc.
- d:Ge/RangeModulatorA/Hub/Rin = 6.0 cm
- d:Ge/RangeModulatorA/Hub/Rout = 7.0 cm
- s:Ge/RangeModulatorA/Hub/Material = "Aluminum"
- s:Ge/RangeModulatorA/Hub/Color = "grey"
- s:Ge/RangeModulatorA/Hub/DrawingStyle = "Solid"
- i:Ge/RangeModulatorA/Hub/VisSegsPerCircle = 360

Setting tracks on Upper area
- dv:Ge/RangeModulatorA/Upper/RadialDivisions=1 11.0 cm # Two tracks Track1 (7.0 cm ~ 11.0 cm) and Track2 (11.0 cm ~ 15.0 cm)

Assignment of track pattern of Upper area
- s:Ge/RangeModulatorA/Upper/Track1/Pattern = "LexanBlockT1" #Track1 refers the pattern named "LexanBlockT1" whose vector parameters are defined elsewhere (see below).
- s:Ge/RangeModulatorA/Upper/Track2/Pattern = "NULL" #NULL means empty track.

Track1 pattern: 14 blocks of Lexan.
Numbers of Angles, Heights, and Materials should be same.
  - d:Ge/LexanBlockT1/Offset=0.0 deg #means shift of zero-angle
Angle divisions. The first block’s spans from 5.0 deg to 115.0 deg. The last block starting at 324.0 deg spans to the first block’s boundary.
This case last block spans from 324.0 deg to 360.0 + 5.0 deg
  - dv:Ge/LexanBlockT1/Angles=14
    - 5.00 115.00 146.50 173.2 195.07
    - 216.15 230.14 243.00 255.5 270.60
    - 282.20 294.60 306.20 324.00 deg
Height of each block.
Note that zero height means that no block in that angle range.
  - dv:Ge/LexanBlockT1/Heights=14
    - 77.0 82.0 87.0 92.15 95.0
    - 100.4 106.0 110.2 115.3 119.5
    - 124.0 128.8 132.00 60.0 mm
Material of each block.
  - sv:Ge/LexanBlockT1/Materials=14
    - "Lexan" "Lexan" "Lexan" "Lexan" "Lexan"
    - "Lexan" "Lexan" "Lexan" "Lexan" "Lexan"
    - "Lexan" "Lexan" "Lexan" "Lexan" "Brass"

In same way, you can configure other tracks.
Then the track1 on upper area looks like following figure.
A track pattern from the parameter above; (left) a complete set of the track view. (right) blocks are constructed in counterclockwise.

- \texttt{b:Ge/RangeModulatorA/PrintInformation = "True"} #Print out specification, see below

When TOPAS builds the geometry, you will see the numbers are input properly from console output as:

```plaintext
---UpperTrack1 , # of Blocks: 14
  0 th Block
    Angle  : 5, 115 deg
    Height : 7.7 cm
    Material: Ts_Lexan
  1 th Block
    Angle  : 115, 146.5 deg
    Height : 8.2 cm
    Material: Ts_Lexan
  2 th Block
    Angle  : 146.5, 173.2 deg
    Height : 8.7 cm
    Material: Ts_Lexan
  3 th Block
    Angle  : 173.2, 195.07 deg
    Height : 9.215 cm
    Material: Ts_Lexan
  4 th Block
    Angle  : 195.07, 216.15 deg
    Height : 9.5 cm
    Material: Ts_Lexan
...
```

TOPAS RMW is a specialized geometry and so allows only the rotation around z-axis as well as the propeller rotation. Two examples demonstrate how to rotate RMW and modulate beam current. The
detail explanation for cooperating with Time Feature is followed later.
(examples/SpecialComponents/RangeModulator_ConstantBeam.txt)
(examples/SpecialComponents/RangeModulator_CurrentModulatedBeam.txt)
**Propeller** (type = "TsPropeller")

A propeller is a component widely used to modulate range of Bragg peaks. TOPAS currently supports a symmetrical propeller, i.e., each blade is in the same shape but in different placements. Users can specify number of blades with a spanning angle, thickness and materials of each layer. Here is an example of a single-layer propeller having 4 blades.

Each blade is constructed in the counterclockwise order. The figure shows its shape with coordination system.

Here is the complete set of the parameters for the above Propeller:

**Common parameters: type of geometry, position, and rotation**
- s:Ge/PropellerA/Type = "TsPropeller" #TsPropeller as type of geometry
- s:Ge/PropellerA/Parent = "World"
- s:Ge/PropellerA/Material = "Parent" #This is required to be set as "Parent"
- d:Ge/PropellerA/TransX = 0.0 cm
- d:Ge/PropellerA/TransY = 0.0 cm
- d:Ge/PropellerA/TransZ = 0.0 cm
- d:Ge/PropellerA/RotX = 0.0 deg
- d:Ge/PropellerA/RotY = 0.0 deg
- d:Ge/PropellerA/RotZ = 0.0 deg
- b:Ge/PropellerA/Invisible = "true" #To avoid visualize propeller’s mother volume.
- i:Ge/PropellerA/NbOfBlades = 4 #Number of blades
- d:Ge/PropellerA/Rin = 10.0 mm #Inner radius, Rin in the figure
- d:Ge/PropellerA/Rout = 127.5 mm #Outer radius, Rout in the figure
• dv:Ge/PropellerA/Thickness =1 0.356 mm #thickness.
• dv:Ge/PropellerA/Angles =1 63.15 deg #span angle of each blade.
• sv:Ge/PropellerA/Materials =1 "G4_POLYVINYL_ACETATE" #material.

#Visualization attributes setting
• s:Ge/PropellerA/Blade/Material = Ge/PropellerA/Material
• s:Ge/PropellerA/Blade/Color = "skyblue"
• s:Ge/PropellerA/Blade/DrawingStyle = "Solid"
• i:Ge/PropellerA/Blade/VisSegsPerCircle = 360

#you can apply a propeller specific step size .
• d:Ge/PropellerA/Blade/MaxStepSize = 10 mm
• b:Ge/PropellerA/PrintInformation = "True" #Print out specification, see below

While TOPAS starts to build geometries, you can confirm whether the numbers you put are input properly from console output as:
Layer: "0", Thickness: 0.0356 (cm), Angle: 63.15 (deg), Material: G4_POLYVINYL_ACETATE
Blade "0", Angle (-31.575 deg, 31.575deg)
Blade "1", Angle (88.425 deg, 151.575deg)
Blade "2", Angle (208.425 deg, 271.575deg)

With different numbers of blades, the angle of each blade will look like:

(Left) Ge/PropellerA/NbOfBlades = 2, (Right) Ge/PropellerA/NbOfBlades = 3.

You can model a multiple layered propeller just by extending the vector parameters, such as Thickness, Angles, and Materials (for more detail, see examples/SpecialComponents/Propeller.txt):
• dv:Ge/PropellerA/Thickness=10
  0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 mm
• dv:Ge/PropellerA/Angles=10
  63.15 54.15 46.45 40.65 35.85 31.8 28.1 24.725 21.8 19.1 deg
• sv:Ge/PropellerA/Materials=10
  "Lexan" "Water" "G4_POLYVINYL_ACETATE" "G4_POLYVINYL_ACETATE"
  "G4_POLYVINYL_ACETATE" "G4_POLYVINYL_ACETATE" "G4_POLYVINYL_ACETATE"
  "G4_POLYVINYL_ACETATE" "G4_POLYVINYL_ACETATE" "G4_POLYVINYL_ACETATE"

Layers are compiled in order of its places in the parameter vector, i.e., Lexan is the bottom layer in this case, Water layer is the next, and so on. It is possible to make each layer with different thickness, angles, and materials. Note that all these three vector parameter has same number of vector elements.
TOPAS propeller allows only the rotation around z-axis, which means that you can only assign rotation Time Feature to "RotZ", such as "Ge/PropellerA/RotZ = Tf/ContinuousRotation/Value." Two examples demonstrate how to handle propeller rotations.
(examples/SpecialComponents/Propeller_ContinuousRotation.txt)
(examples/SpecialComponents/Propeller_DiscreteRotation.txt)
Ridge Filter ( type = "TsRidgeFilter" )

A ridge filter is an energy modulation component used in proton therapy. TOPAS offers a generic way to model an arbitrary shape of a ridge and place the replica. Shape of a ridge is defined in x-z plane and then it become a volume by extending in y plane.

(left) A ridge shape in X-Y plane, represented by points-connection. Because the connection starts at the origin and ends at the last point, (width, 0), so users need to define the width of a ridge first. Depending on the topology of points, the arbitrary shape can be constructed. (right) A complete ridge by extending the shape along with y axis.

Here is a complete set of the parameters for the above ridge filter (examples/SpecialComponents/ RidgeFilter.txt):

Common parameters: type of geometry, position, and rotation
- s:Ge/RidgeFilterA/Type = "TsRidgeFilter"
- s:Ge/RidgeFilterA/Parent = "RidgeGroup"
- s:Ge/RidgeFilterA/Material = "Aluminum"
- d:Ge/RidgeFilterA/TransX = 0.0 cm
- d:Ge/RidgeFilterA/TransY = 0.0 cm
- d:Ge/RidgeFilterA/TransZ = 0.0 cm
- d:Ge/RidgeFilterA/RotX = 0.0 deg
- d:Ge/RidgeFilterA/RotY = 0.0 deg
- d:Ge/RidgeFilterA/RotZ = 0.0 deg
- s:Ge/RidgeFilterA/DrawingStyle = "Solid"

Ridge Filter-specific parameters;
Width definition
Note that points are sequentially connected.
Number of XPoints and YPoints should be same.

- \( \text{dv:Ge/RidgeFilterA/XPoints} = 8 \)
- \( 0.0 \ 0.8 \ 1.3 \ 1.8 \ 2.2 \ 2.7 \ 3.2 \ 4.0 \text{ mm} \)
- \( \text{dv:Ge/RidgeFilterA/ZPoints} = 8 \)
- \( 2.4 \ 4.0 \ 9.1 \ 14.0 \ 14.0 \ 9.1 \ 4.0 \ 2.4 \text{ mm} \)
- \( \text{d:Ge/RidgeFilterA/Width} = 4.0 \text{ mm} \)
- \( \text{d:Ge/RidgeFilterA/Length} = 1.0 \text{ cm} \)

To check that the numbers are input properly:

- \( \text{b:Ge/RidgeFilterA/PrintInformation} = \text{"True"} \)

will generate console output as:

```
Ridge points (x,z) --- :8
P initial : (0, 0) cm
P 0th     : (0, 0.24) cm
P 1th     : (0.08, 0.4) cm
P 2th     : (0.13, 0.91) cm
P 3th     : (0.18, 1.4) cm
P 4th     : (0.22, 1.4) cm
P 5th     : (0.27, 0.91) cm
P 6th     : (0.32, 0.4) cm
P 7th     : (0.4, 0.24) cm
P final   : (0.4, 0) cm
```

You can make replicas of the ridge and their positions in x axis.
A total of 3 replicas of the ridge and placed at -5.0, 0.0, 5.0. Each points represents x positions of the center of ridge width.

- \( \text{dv:Ge/RidgeFilterA/Displacement} = 3 \ -5.0 \ 0.0 \ 5.0 \text{ mm} \)

An example of replica set.
**Multi Wire Chamber** (type = "TsMultiWireChamber")

A multi wire chamber may be built from many of geometry primitives such as Box and Cylinder. However, it is quite cumbersome to place many wires individually and adjust their dimension on any request. So TOPAS multi wire chamber (TsMultiWireChamber) allows to instantiate many wires and to place them efficiently. TsMultiWireChamber is a box consisting of multiple sets of wires. Each set can have its own configuration such dimension and material of the wires, spaces between wires, alignment axis, Z-positions, and drawing-style.

Here is an example of TsMultiWireChamber (examples/SpecialComponents/MultiWire_Chamber.txt).

```
TOPAS Multi wire chamber consists of two wire sets aligned along X and Y axis. Three red wires are aligned to X axis while four gray wires are aligned to Y axis. These two sets of wires are placed at their mother box (gas filed).

The following parameters is full set for modeling the above multi wire chamber:
- s:Ge/WireChamberA/Parent = "World"
- s:Ge/WireChamberA/Type = "TsMultiWireChamber" #Type of geometry
- d:Ge/WireChamberA/HLX=30.0 cm #Chamber dimension
- d:Ge/WireChamberA/HLY=30.0 cm
- d:Ge/WireChamberA/HLZ=10.0 cm
- s:Ge/WireChamberA/Material="Air" #Chamber is filled with this material.
- d:Ge/WireChamberA/TransX=0.0 cm
- d:Ge/WireChamberA/TransY=0.0 cm
- d:Ge/WireChamberA/TransZ=0.0 cm
- d:Ge/WireChamberA/RotX=0.0 deg
- d:Ge/WireChamberA/RotY=0.0 deg
- d:Ge/WireChamberA/RotZ=0.0 deg
- i:Ge/WireChamberA/NbOfLayers=2 #Number of wire sets.
```

Parameters for specifying each wire set add ‘Layer#’ to geometry name. So Layer1 to Layer‘NumberofLayers’.
- d:Ge/WireChamberA/Layer1/RMin=0.0 cm #Wire’s inner radius
When TOPAS starts to build geometries, you will see the numbers are input properly from console output as:

Layer: "0", # of Wires: 3, Alignment: X, Wire (Rmin= 0 cm, Rmax= 1 cm, HL= 20 cm), Z Position in the Chamber: 5 (cm)
  Wire "0", Position (0 cm, -10 cm)
  Wire "1", Position (0 cm, 0 cm)
  Wire "2", Position (0 cm, 10 cm)
Layer: "1", # of Wires: 5, Alignment: Y, Wire (Rmin= 0.5 cm, Rmax= 1 cm, HL= 20 cm), Z Position in the Chamber: -5 (cm)
  Wire "0", Position (-20 cm, 0 cm)
  Wire "1", Position (-10 cm, 0 cm)
  Wire "2", Position (0 cm, 0 cm)
  Wire "3", Position (10 cm, 0 cm)
  Wire "4", Position (20 cm, 0 cm)
**Multi Leaf Collimator** (type = "TsMultiLeafCollimator")

Due to the design variations of Multi Leaf Collimator (MLC) from manufacturers, TOPAS provides a simplified MLC model instead of a generic design. With TOPAS MLC’s minimal set of parameters, users can define various width of each leaf and opening of each leaf.

Illustrations for TOPAS MLC dimensions. User can define arbitrary number of leaves with different width of each leaf. TOPAS detects the leaves collision when it is built and leaf is repositioned by Time Features operations.

Here is a complete set of the parameters for the above TOPAS MLC (examples/SpecialComponents/MultiLeafCollimator.txt):

**Common parameters:** type of geometry, position, and rotation

- s:Ge/MultiLeafCollimatorA/Type = "TsMultiLeafCollimator"
- s:Ge/MultiLeafCollimatorA/Parent = "World"
- s:Ge/MultiLeafCollimatorA/Material = "Aluminum"
- d:Ge/MultiLeafCollimatorA/TransX = 0.0 cm
- d:Ge/MultiLeafCollimatorA/TransY = 0.0 cm
- d:Ge/MultiLeafCollimatorA/TransZ = 0.0 cm
- d:Ge/MultiLeafCollimatorA/RotX = 0.0 deg
- d:Ge/MultiLeafCollimatorA/RotY = 0.0 deg
- d:Ge/MultiLeafCollimatorA/RotZ = 0.0 deg
- s:Ge/MultiLeafCollimatorA/DrawingStyle = "Solid"
- b:Ge/MultiLeafCollimatorA/PrintInformation = "True"

**MLC-specific parameters:**

Limits Leaf opening. Any of absolute values from X+- Leaf can’t exceed this value.

- d:Ge/MultiLeafCollimatorA/MaximumLeafOpen = 5.0 cm
- d:Ge/MultiLeafCollimatorA/Thickness = 5.0 cm #Leaf thickness (z)
- d:Ge/MultiLeafCollimatorA/Length = 6.0 cm #Leaf length (y)
- dv:Ge/MultiLeafCollimatorA/Widths = 5 1.5 0.5 0.5 0.5 1.5 cm #Leaves width

Each leaf’s opening distance from Y axis.

- XMinusLeavesOpen means the x position of X- leaf’s right edge.
- XPlusLeavesOpen means the x position of X+ leaf’s left edge.

- dv:Ge/MultiLeafCollimatorA/XMinusLeavesOpen = 5 0.0 -0.3 -0.2 -0.5 0.0 cm
• dv:Ge/MultiLeafCollimatorA/XPlusLeavesOpen = 5 0.0 0.3 0.2 0.5 0.0 cm

TOPAS MLC is a specialized geometry and so allows only the reposition of each leaf as a function of time. The detail explanation for cooperating with Time Feature is followed later. (examples/SpecialComponents/MultiLeafCollimator_sequence.txt)
**CAD (Computer Aided Design)** (type = "TsCAD")

The TsCAD component allows you to turn any geometry that has been designed in a CAD system into a TOPAS Component. This allows you to incorporate arbitrarily complex geometries.

Supported CAD formats supported are:
- STL - Stereolithography binary format
- PLY - Polygon ASCII format.

STL and PLY files describe a geometry as a tessellation, providing a set of vertices and faces of triangular or quadrangular surfaces to approximate the volume. While some STL and PLY files also contain additional information such as material and color, TOPAS does not currently accept such files. The STL and PLY files you provide to TOPAS must contain only the tessellation information. Internally, TOPAS represents this component as a G4TessellatedSolid.

Most CAD systems allows direct export of parts to the above formats. If your CAD system does not support one of those formats, you may be able to convert from some other CAD format by using a free conversion tool such as MeshLab (meshlab.sourceforge.net).

Common parameters: type of geometry, position, and rotation
- s:Ge/MyPartFromCAD/Type = "TsCAD"
- s:Ge/MyPartFromCAD/Parent = "World"
- s:Ge/MyPartFromCAD/Material = "G4_Water"
- d:Ge/MyPartFromCAD/TransX = 0.0 cm
- d:Ge/MyPartFromCAD/TransY = 0.0 cm
- d:Ge/MyPartFromCAD/TransZ = 0.0 cm
- d:Ge/MyPartFromCAD/RotX = 0.0 deg
- d:Ge/MyPartFromCAD/RotY = 0.0 deg
- d:Ge/MyPartFromCAD/RotZ = 0.0 deg
- s:Ge/MyPartFromCAD/DrawingStyle = "Wireframe"

CAD-specific parameters:
- s:Ge/MyPartFromCAD/InputFile = "Foot" # file name, without extensions. Match exact case
- s:Ge/MyPartFromCAD/FileFormat = "ply" # file extension
- d:Ge/MyPartFromCAD/Units = 1.0 cm # how to interpret dimension numbers in the file

Changing this value will re-scale the component

TOPAS does not automatically know where the center of your CAD component will be. This is affected by how your CAD system manages coordinates. For example, some CAD software exports the STL by relocating the volume to the first positive octant of its coordinate system. You may have to adjust the TransX,Y,Z of your component to center it as desired.
Above, A plastic scintillator with customized groove. Left: CAD, Right: TOPAS
Aperture ( type = "TsAperture" )

An aperture is a component used to shape the lateral penumbra of a (generally) double-scattered proton beam. It is basically a block of brass with a hole cut out in the middle in the shape of the treatment volume. The purpose is to block the beam outside the desired irradiation path. TOPAS models the aperture by connecting the aperture file points to create a polygon and then extruding this polygon in Z to cut out the aperture hole. A typical implementation of an apertures in TOPAS is given below followed by a more detailed description of each option (see examples/Nozzle/ScatteringNozzle.txt):

- s:Ge/Aperture/Type = "TsAperture"
- s:Ge/Aperture/Parent = "Snout"
- s:Ge/Aperture/Material= "Brass"
- d:Ge/Aperture/RMax = 4.5 cm
- d:Ge/Aperture/HL= 2.5 cm
- d:Ge/Aperture/TransX = 0.0 cm
- d:Ge/Aperture/TransY = 0.0 cm
- d:Ge/Aperture/TransZ = -13.0 cm
- d:Ge/Aperture/RotX = 0.0 deg
- d:Ge/Aperture/RotY = 0.0 deg
- d:Ge/Aperture/RotZ = 0.0 deg
- s:Ge/Aperture/InputFile = "ApertureFileIn.ap" # Match exact case
- s:Ge/Aperture/FileFormat = "XYCoordinates" # XYCoordinates or MGH
- b:Ge/Aperture/PrintPoints = "True" # Print points to console

FileFormat has two options:

- "XYCoordinates" takes is a simple list of points. The first line defines how many points there are in the file, each following line in the file is a comma separated x,y pair, such as:
  - numberOfPoints
  - x1, y1
  - x2, y2
  - ...
  - xN, yN
  - N = numberOfPoints is the number of data points (xi, yi). This is a required condition. The units of the coordinates are millimeter.

- "MGH" takes the milling data produced by the MGH machine shop. It consists of the same information as the XYCoordinates option, but with more overhead, such as:
  - patientName
  - Warning message about not fabricating this file
  - someDoubleValue
  - someIntValue
  - M (this is a number of dummy points, this amount of points will be skipped)
  - x1 y1 x2 y2 ... xM yM
  - N
  - x1 y1 x2 y2 ... xN yN "
  - Data pairs are listed in a simple space-separated list. The units are in centimeter.
Compensator ( type = "TsCompensator" )

A compensator is a component that is used to shape the distal edge of a proton beam by placing a varying amount of material in the beam path, usually behind the aperture. An example compensator is shown on the right in top and side views. The compensator consists of a material that is to be placed in the beam to attenuate the beam (usually lexan) and a number of drill holes that were drilled into the compensator.

A typical compensator has the following parameters (see examples/Nozzle/ScatteringNozzle.txt):

- s:Ge/Compensator/Type = "TsCompensator"
- s:Ge/Compensator/Parent = "Snout"
- s:Ge/Compensator/Material = "CompensatorLucite"
- d:Ge/Compensator/RMax = 5.5. cm
- d:Ge/Compensator/TransX = 0. cm
- d:Ge/Compensator/TransY = 0. cm
- dc:Ge/Compensator/Thickness = 0. cm # will be reset to actual thickness when compensator is read in. This allows other parameter files to access this variable thickness.
- d:Ge/Compensator/InvHL = -0.5 * Ge/Compensator/Thickness cm
- d:Ge/Compensator/TransZ = -15.5 cm + Ge/Compensator/InvHL # Allows centering regardless of thickness
- d:Ge/Compensator/RotX = 0. deg
- d:Ge/Compensator/RotY = 0. deg
- d:Ge/Compensator/RotZ = 0. deg
- s:Ge/Compensator/InputFile = "CompensatorFileInRowsDepths.rc" # match exact case
- s:Ge/Compensator/FileFormat = "RowsAndDepths" # RowsAndDepths or MGH
- s:Ge/Compensator/Method = "ExtrudedSolid" # Polyhedra, ExtrudedSolid, SubtractionCylinders or UnionCylinders
- b:Ge/Compensator/PrintPoints = "True"

Thickness has the special parameter type, "dc", where the "c" means this dimensioned double is changeable, that is, it can change on the fly based on what exact compensator is read in. Other parameters can then take this thickness into account when the perform placements.

FileFormat has two options:

- "RowsAndDepths": all sizes are in millimeters:
  - NumberOfRows
  - MainCylinderThickness
  - DrillHoleDiameter
  - n1 deltaX1 X1 Y1
  - D1 D2 ... Dn1
  - n2 deltaX2 X2 Y2
  - D1 D2 .... Dn2
  - ...
  - nN deltaXn Xn Yn
  - D1 D2 ... DnN
- `NumberOfRows = N` defines how many rows of drill holes there are (in Y), the `MainCylinderThickness`.
- The `DrillHoleDiameter` is the diameter of the drill hole, we approximate this by a hexagon.
- The values `ni` are the number of drill holes in X for each row of drill holes in Y, `deltaXi` defines the step size (and direction) and `Xi` and `Yi` are the starting position of the drilling for this row.

- "MGH": all sizes are in inches:
  - `Some line`
  - `numberOfRows`
  - `dummyDouble dummyDouble dummyDouble dummyDouble dummyDouble`
  - `MainCylinderThickness`
  - `dummyDouble dummyDouble dummyDouble dummyDouble dummyDouble dummyDouble`
  - `DrillHoleDiameter`
  - `n1 deltaX1 X1 Y1`
  - `D1 D2 … Dn1`
  - `n2 deltaX2 X2 Y2`
  - `D1 D2 …. Dn2`
  - `…`
  - `nN deltaXN Xn Yn`
  - `D1 D2 … DnN`

Method has four options:
- "ExtrudedSolid" builds the compensator from a set of extruded solids. This is the most reliable and efficient technique.
- "Polyhedra" carves hexagon shapes out of the compensator. This method has an extra check to adjust the position of each starting drill hole of each row to compensate for rounding inaccuracies produced by the drilling machine
  - `d:Ge/Compensator/XTolerance = 1. mm`
  - `d:Ge/Compensator/YTolerance = 1. mm`
- "SubtractionCylinders" builds the compensator by subtracting drill hole cylinders from the overall compensator cylinder. This technique gives the most perfect representation of the drilling process, however the added precision is insignificant, while particle navigation time is increased.
  Note that if you want to visualize this form of compensator, you should use RayTracer, as this is the only Geant4 visualization drivers that can correctly render boolean operations.
- "UnionCylinders" builds the compensator by first creating a union solid of all the holes, and then subtracting this union solid from the overall compensator cylinder. This technique is similar to SubtractionCylinders but slightly more efficient.
  Note that if you want to visualize this form of compensator, you should use RayTracer, as this is the only Geant4 visualization drivers that can correctly render boolean operations.
**Patient in DICOM Format** (type = "TsDicomPatient")

DICOM import is handled through the package GDCM, which is pre-built into TOPAS.
GDCM is further described at: [http://gdcm.sourceforge.net](http://gdcm.sourceforge.net)

See examples/DICOM/DoseToCT.txt and examples/DICOM/ViewAbdomen.txt for examples of how to use TsDicomPatient.
Note that before running this example, you must unzip the included DICOM files.

You specify the name of a directory containing one or more dcm files (one for each slice):
  - s:Ge/Patient/DicomDirectory = "DICOM_Box"

Files of other types in this directory will be ignored.
Exact titles of the dcm files are not important as TOPAS will re-order them based on the slice ordering information inside the DICOM headers.

Hounsfield Conversion refers to the process by which we map each of the Hounsfield numbers to a specific material that we will use in our simulation. For each material we need to determine density, material name and graphics color. We follow the technique described in the 2000 Schneider paper. The HU conversion parameters are typically stored in a separate parameter file.
  - includeFile = HUtoMaterialSchneider.txt

An example of such a HU conversion parameter file is examples/DICOM/HUtoMaterialSchneider.txt. Details on the HU conversion are given at the end of this section.

To dump your file's raw HU values to the console:
  - b:Ge/Patient/DumpHUValues = "True"

Set any parent you like, but it is often convenient to place patient into a group component which can then be rotated to represent couch setup:
  - s:Ge/Patient/Parent = "PatientGroup"

We will eventually get rid of the Ge/Patient/Material parameter, but for now, you need to provide one, though it doesn't really matter what value it has. The actual voxel materials will come from the HU conversion of the XiO file.
Specifying material as Water here is useful since it means you will have the option of scoring DoseToWater later even if there was no water in your actual CT image.
  - s:Ge/Patient/Material = "Water"

Patient positioning information from the DICOM file is not currently used. You must position as you would for any TOPAS component:
  - d:Ge/Patient/TransX=0. m
  - d:Ge/Patient/TransY=0. m
  - d:Ge/Patient/TransZ=0. m
  - d:Ge/Patient/RotX=0. deg
  - d:Ge/Patient/RotY=0. deg
  - d:Ge/Patient/RotZ=0. deg

To specify 4DCT, you can have DicomDirectory change under control of a Time Feature.
Even though a large number of materials are defined in your HU conversion file, TOPAS will only create those materials that are actually used in your CT image.

In the 4DCT case, if any image introduces new materials that were not in the first image, Geant4 will be unable to proceed (it cannot load new materials after physics has initialized). TOPAS will exit with a warning message advising you to set the parameter:

- **b:Ge/Patient/PreLoadAllMaterials = "True"**

Startup will then be slower, since TOPAS will preload the full set of materials defined in your HU conversion file, but your 4DCT will then work.

The DICOM component does not yet handle multiple slice thicknesses. This feature will be added later.

The built-in Geant4 visualization tools do not perform well when a complex voxel structure is loaded. To make visualization more successful, several additional parameters are provided.

There is generally little value in showing all pixels of the image at once. Each slice just covers up the last slice. To instead show only a specific set of slices in any dimension:

- **iv:Gr/Patient/ShowSpecificSlicesZ = 4 1 3 9 12 # will only show slices 1, 3, 9 and 12.**

Similar slicing is allowed in X and Y.

Three special values are also allowed:

- **iv:Gr/Patient/ShowSpecificSlicesZ = 1 0 # means show all slices**
- **iv:Gr/Patient/ShowSpecificSlicesZ = 1 -1 # means only show center slice**
- **iv:Gr/Patient/ShowSpecificSlicesZ = 1 -2 # means only first, center and last slice**

The following will result in a display that shows 27 pixels comprising the boundaries and center of the image. This allows you to see the overall placement of the image and see the individual voxel size:

- **iv:Gr/Patient/ShowSpecificSlicesX = 1 -2 # means only show center slice**
- **iv:Gr/Patient/ShowSpecificSlicesY = 1 -2 # means only show center slice**
- **iv:Gr/Patient/ShowSpecificSlicesZ = 1 -2 # means only show center slice**

Another option allows you to specify the maximum number of voxels to show. If the total number of voxels is greater than this limit, TOPAS will just draw the overall DICOM outline.

- **i:Gr/ShowOnlyOutlineIfVoxelCountExceeds = 10000**

By default, OpenGL graphics switches its fast "Stored" mode to its more memory efficient "Immediate" mode when the graphics scene gets very complicated. When this switch occurs, the current version of Geant4 has a bug such that part of the image is lost. To prevent this from impacting DICOM images, you can set a threshold at which Geant4 will use Immediate mode from the start:

- **i:Gr/SwitchOGLtoOGLIifVoxelCountExceeds = 10000 # Above this limit, switch OpenGL Graphics to Immediate mode**
TOPAS can read DICOM RT Structure Sets. A structure set is an extra file in the DICOM directory that provides information on structures such as organs, tumors, PTVs, etc. that have been outlined (contoured) in the planning process. The data is stored as a set of polygons, up to one per slice for each contoured structure. TOPAS can color code DICOM components according to this structure information and can filter scoring based on these structures (see the filter: OnlyIncludeIfInRTStructure).

To make TOPAS color code the DICOM by structure:

- **sv:Ge/Patient/ColorByRTStructNames = 2 "R_LUNG" "L_LUNG"**
- **sv:Ge/Patient/ColorByRTStructColors = 2 "yellow" "red"**
- If the structure name includes a space, substitute an underscore in the parameter. So, for example, if the structure name is "R LUNG", you should supply the parameter as "R_LUNG".
- If you don’t actually know what structures are included in your DICOM, just try providing in ColorByRTStructNames. TOPAS will give you an error message that includes a list of the known structure names.
- To allow easy testing of this feature in simple DICOM examples that don’t really have any structures, the following parameter will "fake" an RT structure set, assigning the given structure to all voxels in the lower XY quadrant.

**b:Ge/Patient/FakeStructures = "True"**
**Patient in XIO Format** (type = "TsXiOPatient")

The XiO patient is a specific implementation of a patient geometry. It requires a binary file containing a list of hounsfield units for each voxel of a patient in "short little endian" format.

See examples/DICOM/DoseToCT.txt for an example of how to use TsXiOPatient.

Hounsfield Conversion refers to the process by which we map each of the Hounsfield numbers to a specific material that we will use in our simulation. For each material we need to determine density, material name and graphics color. We follow the technique described in the 2000 Schneider paper. The HU conversion parameters are typically stored in a separate parameter file.

- includeFile = HUtoMaterialSchneider.txt

An example of such a HU conversion parameter file is examples/DICOM/HUtoMaterialSchneider.txt. Details on the HU conversion are given at the end of this section.

To dump your file's raw HU values to the console:

- b:Ge/Patient/DumpHUValues = "True"

Set any parent you like, but it is often convenient to place patient into a group component which can then be rotated to represent couch setup:

- s:Ge/Patient/Parent = "PatientGroup"

We will eventually get rid of the Ge/Patient/Material parameter, but for now, you need to provide one, though it doesn't really matter what value it has. The actual voxel materials will come from the HU conversion of the XiO file.

Specifying material as Water here is useful since it means you will have the option of scoring DoseToWater later even if there was no water in your actual CT image.

- s:Ge/Patient/Material = "Water"

You must position as you would for any TOPAS component::

- d:Ge/Patient/RotX = 0. deg
- d:Ge/Patient/RotY = 90. deg
- d:Ge/Patient/RotZ = 0. deg
- d:Ge/Patient/TransX = 1.5 mm
- d:Ge/Patient/TransY = 3.3 mm
- d:Ge/Patient/TransZ = 4.2 mm

XiO Format does not contain voxel number and size information. You must specify it as follows.

Number and size of voxels in X and Y:

- i:Ge/Patient/NumberOfVoxelsX = 25
- i:Ge/Patient/NumberOfVoxelsY = 25
- d:Ge/Patient/VoxelSizeX = 2.0 mm
- d:Ge/Patient/VoxelSizeY = 2.0 mm

Number and size of Voxels in Z are vectors rather than single values since XiO file may have multiple slice thickness sections.

If there is only one slice thickness in your image, just specify one element:

- iv:Ge/Patient/NumberOfVoxelsZ = 1 10
- dv:Ge/Patient/VoxelSizeZ = 1 2.5 mm
If there are multiple slice thicknesses in your image, specify number and thickness of voxels in each section. For example, a 30 slice image that has 10 slices of 2.5 mm and then 20 slices of 1.25 mm:

- iv:Ge/Patient/NumberOfVoxelsZ = 2 10 20
- dv:Ge/Patient/VoxelSizeZ = 2 2.5 1.25 mm

For single slice thickness images, scoring will use the same voxel divisions as your CT image. For multiple slice thicknesses, scoring will not know what divisions to use unless you explicitly specify these in your scoring parameters, such as:

- i:Sc/MyScorer/XBins = 512
- i:Sc/MyScorer/YBins = 512
- i:Sc/MyScorer/ZBins = 256

Specify file directory and file name:

- s:Ge/Patient/InputDirectory = "./"
- s:Ge/Patient/InputFile = "ctvolume.dat" # match exact case

To specify 4DCT, you can have InputDirectory or InputFile change under control of a Time Feature.

Even though a large number of materials are defined in your HU conversion file, TOPAS will only create those materials that are actually used in your CT image.

In the 4DCT case, if any image introduces new materials that were not in the first image, Geant4 will be unable to proceed (it cannot load new materials after physics has initialized). TOPAS will exit with a warning message advising you to set the parameter:

- b:Ge/Patient/PreLoadAllMaterials = "True"

Startup will then be slower, since TOPAS will preload the full set of materials defined in your HU conversion file, but your 4DCT will then work.

The built-in Geant4 visualization tools do not perform well when a complex voxel structure is loaded. To make visualization more successful, several additional parameters are provided.

There is generally little value in showing all pixels of the image at once. Each slice just covers up the last slice. To instead show only a specific set of slices in any dimension:

- iv:Gr/Patient/ShowSpecificSlicesZ = 4 1 3 9 12 # will only show slices 1, 3, 9 and 12.

Similar slicing is allowed in X and Y.

Three special values are also allowed:

- iv:Gr/Patient/ShowSpecificSlicesZ = 1 0 # means show all slices
- iv:Gr/Patient/ShowSpecificSlicesZ = 1 -1 # means only show center slice
- iv:Gr/Patient/ShowSpecificSlicesZ = 1 -2 # means only first, center and last slice

The following will result in a display that shows 27 pixels comprising the boundaries and center of the image. This allows you to see the overall placement of the image and see the individual voxel size:

- iv:Gr/Patient/ShowSpecificSlicesX = 1 -2 # means only show center slice
- iv:Gr/Patient/ShowSpecificSlicesY = 1 -2 # means only show center slice
- iv:Gr/Patient/ShowSpecificSlicesZ = 1 -2 # means only show center slice

Another option allows you to specify the maximum number of voxels to show. If the total number of voxels is greater than this limit, TOPAS will just draw the overall DICOM outline.

- i:Gr/ShowOnlyOutlineIfVoxelCountExceeds = 10000
By default, OpenGL graphics switches its fast "Stored" mode to its more memory efficient "Immediate" mode when the graphics scene gets very complicated. When this switch occurs, the current version of Geant4 has a bug such that part of the image is lost. To prevent this from impacting DICOM images, you can set a threshold at which Geant4 will use Immediate mode from the start:

- `i:Gr/SwitchOGLtoOGLIifVoxelCountExceeds = 10000` # Above this limit, switch OpenGL Graphics to Immediate mode
Hounsfield Conversion (for either of the Patient types = "TsDicomPatient" or "TsXioPatient")

Hounsfield Conversion refers to the process by which we map each of the Hounsfield numbers to a specific material that we will use in our simulation. For each material we need to determine density, material name and graphics color. We follow the technique described in the 2000 Schneider paper.

The HU conversion parameters are typically stored in a separate parameter file.

- `includeFile = HUtoMaterialSchneider.txt`

An example of such a HU conversion parameter file is examples/DICOM/HUtoMaterialSchneider.txt.

The first set of parameters in the HU file are used to calculate density:

- `dv:Ge/Patient/DensityCorrection = 3996 9.35212 5.5269 4.14652 ...1.06255 1.00275 g/cm^3`
- `iv:Ge/Patient/SchneiderHounsfieldUnitSections = 8 -1000 -98 15 23 101 2001 2995 2996`
- `uv:Ge/Patient/SchneiderDensityOffset = 7 0.00121 1.018 1.03 1.003 1.017 2.201 4.54`
- `uv:Ge/Patient/SchneiderDensityFactor = 7 0.00103 0.00089 0.0 0.00117 0.00059 0.0005 0.0`
- `uv:Ge/Patient/SchneiderDensityFactorOffset = 7 1000. 0. 1000. 0. 0. -2000. 0.`

DensityCorrection:

- One value for every possible HU value.
- Values start from `i:Ge/Patient/MinHUValue` which defaults to -1000

SchneiderHounsfieldUnitSections:

- Specifies how to break up the entire set of HU units into several density calculation sections. The HU conversion formula then uses different correction factors for each of these sections.
- The total range (last value minus first value) must equal the number of values in DensityCorrection.
- In the above example, the 8 values define 7 sections:
  - Section 1: -1000 to -99
  - Section 2: -98 to 14
  - ...
  - Section 7: 2995 to 2996

SchneiderDensityOffset, SchneiderDensityFactor and SchneiderDensityFactorOffset:

- Must have one value for each of the density calculation sections, so length must be one less than the length of SchneiderHounsfieldUnitSections

Thus, for any specific HU number, we can extract the appropriate:

- DensityCorrection
- SchneiderDensityOffset
- SchneiderDensityFactor
- SchneiderDensityFactorOffset

And use these in the Schneider formula:

- `Density = ( Offset + ( Factor * ( FactorOffset + HU[-1000,2995] ) ) ) * DensityCorrection`
The second set of parameters in the HU file are used to calculate material name and graphics color:

- iv:Ge/Patient/SchneiderHUToMaterialSections = 26 -1000 -950 -120 -83 ... 1500 2995 2996
- sv:Ge/Patient/SchneiderElements = 13 "Hydrogen" "Carbon" "Nitrogen" "Oxygen" ...
- uv:Ge/Patient/SchneiderMaterialsWeight1 = 13 0.0 0.0 0.755 0.232 ...
- uv:Ge/Patient/SchneiderMaterialsWeight2 = 13 0.103 0.105 0.031 0.749 ...

iv:SchneiderHUToMaterialSections:
- Specifies how to break up the entire set of HU units into several material name assignment sections.
- The total range (last value minus first value) must equal the number of values in DensityCorrection.
- In the above example, the 26 values define 7 material name assignment sections:
  - Section 1: -1000 to -949
  - Section 2: -50 to -119
  - ...
  - Section 26: 2995 to 2996

sv:SchneiderElements:
- Specifies all of the elements that will be used in the patient.
- All patient materials must be composed from combinations of this set of elements.

uv:SchneiderMaterialsWeight1 through SchneiderMaterialsWeight26:
- There should be one of these parameters for each of the material name assignment sections.
- The length of SchneiderMaterialsWeight must equal the length of SchneiderElements.
- Each value in SchneiderMaterialsWeight tells what proportion of the given element in SchneiderElements to use in this material.
- In our SchneiderMaterialsWeight2 example, the values: 0.103 0.105 0.031 0.749 mean:
  - 10.3 percent of the first element, Hydrogen
  - 10.5 percent of the second element, Carbon
  - 3.1 percent of the second element, Nitrogen
  - 74.9 percent of the second element, Oxygen

dv:SchneiderMaterialMeanExcitationEnergy:
- You may optionally provide this parameter to override the default mean excitation energies of some or all of the materials.
- There should be one value for each material name assignment section.
- To use the default mean excitation energy for a particular material, enter that value as 0.
- For example, the following just overrides defaults for two out of 26 assignment sections:
  - dv:Ge/Patient/SchneiderMaterialMeanExcitationEnergy = 26 88.8 0. 77.7. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. eV

iv:Gr/Color/PatientTissue1:
- Specifies what colors should be assigned to each of the materials.
- There should be one of these parameters for each of the SchneiderHUToMaterialSections.
- The three values specify the Red, Green and Blue components of the color.
Putting it all together, we have now specified density, material name, color and, optionally, mean excitation energy, for each of the Housfield numbers in the patient.

You can review the materials definitions that TOPAS created based on your patient file and the HU conversion settings. The following parameter tells TOPAS to dump parameters to a file:

- `Ts/DumpNonDefaultParameters = "True"`

For each HU number that was used in the patient file, you will see a set of parameters starting with `Ma/PatientTissueFromHU-` followed by an HU number.

For example, for HU number 295, you may see:

- `Ma/PatientTissueFromHU-295/Component = 9 Hydrogen Carbon Nitrogen Oxygen Phosphorus Sulfur Chlorine Sodium Potassium`
- `Ma/PatientTissueFromHU-295/Fractions = 9 0.103 0.105 0.031 0.749 0.002 0.003 0.003 0.002 0.002`
- `Ma/PatientTissueFromHU-295/Density = 0.707487 g/cm³`
- `Ma/PatientTissueFromHU-295/DefaultColor = PatientTissue2`

where you then follow the `DefaultColor` parameter named `PatientTissue2` to see that `Gr/Color/PatientTissue2` is `3 100 0 0`

which means a mixture of 100 percent Red, 0 percent green, 0 percent blue.
Particle Sources

We allow any number of particle sources, zero, one or many, with no limitation on how they can be mixed.

We provide three different types of particle sources, each with many options:

- Parameterized Beam
- TWISS Parameterization
- Phase Space

And you may also write your own entirely new particle source (see Extending TOPAS at the end of this user guide).

The position of the source always comes from an associated Geometry Component. This is in keeping with the general TOPAS paradigm that all geometrical information resides in the Components. We know that this may feel odd to experienced Geant4 users who are used to setting beam directions irrespective of any geometry volumes, but the TOPAS paradigm enables sources, components, scorers and even fields to all move together in an internally consistent manner.

Some examples place the source at a vacuum window at the entrance to a nozzle. The source then moves as the nozzle moves.

Particle Names

Throughout TOPAS, particle names can take the following forms (case does not matter):

- A simple string such as
  - "proton"
- A string describing an ion with arguments Z, A, and optionally Charge, such as:
  - "GenericIon(6,12,6)"
  - "GenericIon(6,12)" - Charge defaults to Z, that is, the ion is fully stripped
  - When used to filter sources, ions must be fully stripped (this is the only kind of ion that Geant4's primary particle generation supports).
  - When used to filter scoring, ions can have any Charge, and any of the arguments can have wildcard value "*"
    - so, for example, "GenericIon(6,*,*)" will score any Carbon ion (any A and any Charge).
- An integer PDG code, still contained in quotation marks, such as
  - "11"
  - PDG codes are as defined by the Particle Data Group and described at:
- When PDG code has 10 digits and starts with 100, this is passed to Geant4 either as the appropriate Geant4 light ion name ("alpha", "deuteron", "He3" or "triton") or as GenericIon(Z,A) where:
  - Characters 4-6 give Z
  - Characters 7-9 give A
  - Character 10 gives Isomer level (not used)

The full set of known particles depends on the physics you have defined.

Some common values, with associated PDG codes:

- "proton" = "2212"
- "neutron" = "2112"
- "e-" = "11"
- "e+" = "-11" (minus sign gets the antiparticle of any charged particle)
- "gamma" = "22"
- "He3" = "100002003"
- "alpha" = "100002004"
- "deuteron" = "100001002"
- "triton" = "100001003"
- "opticalphoton" = "0" (PDG group has no code for this particle)
- "geantino" = "0" (sees transportation processes but no physics, no PDG code)
- "chargedgeantino" = "0" (same as above but with charge, no PDG code)

**Beam Sources**

By default there is a single source named "Default" centered on a Component named "BeamPosition" that is placed at one end of the "World". The beam shape is an Ellipse.

- s:So/Default/Type = "Beam" # Beam, Twiss, PhaseSpace, GeneralParticleSource or Seed
- s:So/Default/Component = "BeamPosition"
- s:So/Default/BeamParticle = "proton"
- d:So/Default/BeamEnergy = 169.23 MeV
- u:So/Default/BeamEnergySpread = 0.757504
- s:So/Default/BeamShape = "Ellipse" # Point, Ellipse, Rectangle or Isotropic
- i:So/Default/NumberOfHistoriesInRun = 0

Where the default definition of "BeamPosition" is:

- s:Ge/BeamPosition/Parent="World"
- s:Ge/BeamPosition/Type="Group"
- d:Ge/BeamPosition/TransX=0. m
- d:Ge/BeamPosition/TransY=0. m
- d:Ge/BeamPosition/TransZ= Ge/World/HLZ m
- d:Ge/BeamPosition/RotX=180. deg
- d:Ge/BeamPosition/RotY=0. deg
- d:Ge/BeamPosition/RotZ=0. deg

**Details on BeamEnergySpread:**

- The number is unitless because we find it more convenient generally to speak of the spread in terms of percentage of the mean energy, rather than in an absolute number. We could have chosen either representation, but this one seemed most consistent with what we see from other beam modeling applications.
- This is a standard deviation. So the code we have is:
  - fEnergySpread = BeamEnergySpread * fEnergy / 100.;
  - p.kEnergy = CLHEP::RandGauss::shoot(fEnergy, fEnergySpread);
- So, for example, if you want a spread of 0.2 MeV, and your energy is 153 MeV, set BeamEnergySpread to:
  - 0.2 MeV / 153 MeV * 100 = 0.13

To run generate histories using this default source, set its number of histories to some value:

- So/Default/NumberOfHistoriesInRun = 10

You can redefine So/Default, or you can define some other source name, such as:

- s:So/MySource/BeamParticle = "proton"
- d:So/MySource/BeamEnergy = 200. MeV
• i:So/MySource/NumberOfHistoriesInRun = 100

You can provide an energy spectrum instead of a fixed energy by setting the following to "Discrete" or "Continuous":
  • s:So/Default/BeamSpectrum = "Continuous" # Either "None", "Discrete" or "Continuous" and providing energies and weights as:
    • dv:So/Default/BeamSpectrumEnergies = 3 50. 100. 150. MeV
    • uv:So/Default/BeamSpectrumWeights = 3 .20 .60 .20
An example is in:
  • examples/Basic/Spectrum.txt

Any source that has NumberOfHIstoriesInRun greater than zero will contribute primary particles.

The beam is emitted along the Z axis of the beam’s Component and may have some spread along the X and Y axes.

For Type = "Beam", the beam shape can be further described by:
  • d:So/Default/BeamHWX = 10. cm # extent of ellipse or rectangle shape in the X direction
  • d:So/Default/BeamHWY = 10. cm $ extent of ellipse or rectangle shape in the Y direction
  • d:So/Default/BeamAngularSpreadX = 0.0032 rad
  • d:So/Default/BeamAngularSpreadY = 0.0032 rad
  • s:So/Default/BeamXYDistribution = "Gaussian" # Flat or Gaussian
  • d:So/Default/BeamStandardDeviationX = 0.65 cm # distribution used for Gaussian
  • d:So/Default/BeamStandardDeviationY = 0.65 cm # distribution used for Gaussian

The relationship between BeamHWX and Y and BeamStandardDeviationX and Y is as follows:
Changing the BeamStandardDeviationX and Y changes the shape of the Gaussian.
However, even a very narrow Gaussian will still have infinite tails.
The BeamHWX and Y parameters allow you to cut off these tails.
Think of these half widths as forming an aperture through which the source is shining.
Then produce a hard cut off at the specified half widths.
Inside TOPAS, when the Gaussian formula generates a starting point outside of this cutoff, that starting point is rejected and instead the random function is thrown again until a value is found that is within the specified half widths.

Details on BeamAngularSpread:
  • These angles are not defining a limit but a standard deviation.
  • We take the sin of the numbers you provide to compute X and Y direction cosines.
    ○ fDCos1 = sin(BeamAngularSpreadX);
    ○ fDCos2 = sin(BeamAngularSpreadY);
  • For each individual source particle, we use random direction cosines, centered on zero, with a Standard Deviation of the above direction cosine.
    ○ p.dCos1 = CLHEP::RandGauss::shoot(0.,fDCos1);
    ○ p.dCos2 = CLHEP::RandGauss::shoot(0.,fDCos2);
    ○ p.dCos3 = sqrt(1. - p.dCos1*p.dCos1 - p.dCos2*p.dCos2);

Avoid putting a point source exactly on a voxel boundary (this can happen, for example, if you place an isotropic point source at the exact center of a TsBox or DICOM that has an even number of voxels
in each direction). This triggers a Geant4 bug that has not yet been fixed, resulting in a large number of "stuck track" warnings, of the form:
   - *** G4Exception : GeomNav1002
   - issued by : G4Navigator::ComputeStep()
   - Track stuck or not moving.

You can work around this by offsetting either the source or the voxel structure by just a tiny amount.

For Type = "Twiss", the beam shape can be further described by:
   - u:So/Default/AlphaOfX = 1.135
   - d:So/Default/BetaOfX = 7.40 cm
   - d:So/Default/EmittanceOfX = 1.2299 cm
   - d:So/Default/SigmaOfX = 0.2 cm
   - u:So/Default/SigmaOfXprime = 0.01
   - u:So/Default/AlphaOfY = 1.135
   - d:So/Default/BetaOfY = -7.407 cm
   - d:So/Default/EmittanceOfY = 1.2299 cm
   - d:So/Default/SigmaOfY = 0.2 cm
   - u:So/Default/SigmaOfYprime = 0.03

The Twiss parameters are used to sample beam positions and its divergence of each axes, so each axes (X and Y) requires 3 (for alpha, beta, and gamma) of twiss coefficients + 2 of standard deviations. The unit of Alpha and SigmaOfX(orY)prime is unitless while others (Beta, Emittance, and Sigma) expect the unit of length. For Alpha > 0, beam is converging. For Alpha < 0, beam is diverging. For Alpha = 0, beam size has minimum or maximum. SigmaOfX is used to sample X position from normal distribution N(0, SigmaX^2). The SigmaXprime is used to sample and calculate directional cosine of X as:
\[
dCos1 = \frac{\tan(T)}{\sqrt{1+\tan^2(T)}}
\]
where dCos1 is directional cosine and \(\tan(T)\) is the sampled value from N(0, SigmaXprime^2).

The following figures are from the phase space result of simulation.
For details, see examples/Basic/TwissPrimary.txt
Phase Space Sources

Phase Space refers to the technique of saving or replaying a set of particles crossing a given surface.

- When one saves a phase space, one defines a surface and then saves the position, particle type, energy and momentum of some or all particles crossing that surface.
- When one replays a phase space, one starts a set of particles from the saved positions, with the saved particle types, energy and momentum.

Phase Space enables separating two parts of a simulation or analysis job, and can be used to transfer sets of particles among different codes.

Each phase space must come as two related files (with same file name but different file types):

- A .header file tells the number of histories, the number of saved particles and the order of information in the .phsp file
- A .phsp file contains all the details of all the saved particles

We support three formats for Phase Space (and TOPAS automatically figures out the format of your .phsp file by studying the related .header file):

- ASCII provides particle information in an easy to read simple text file, which data encoded as a series of columns of text. The header file tells the contents and column order per particle.
- Binary provides the same information as ASCII, but in a much more compact format, with data encoded in a stream of bytes. The header file tells the contents and byte order per particle. Use Binary in cases where the ASCII format produces excessively large files.
- Limited is an alternate binary format compatible with some legacy codes. It has fewer options for what data can be expressed, but is compatible with codes such as that used by Varian for their TrueBeam phase space files. Use Limited format only when you need to exchange phase space with legacy codes.

For more detail on Phase Space formats, see Miscellaneous Notes at the end of this User Guide.

Phase Space sources ignore the parameters starting with "Beam" and instead use:

- s:So/MySource/Type = "PhaseSpace"
- s:So/MySource/Component = "World" # coordinate system of phase space. Usually "World"
- s:So/MySource/PhaseSpaceFileName = "ASCIIOutput" # match exact case

TOPAS will look for header and phsp files with the given PhaseSpaceFileName.

You can generate some sample data by running either of the examples:

- examples/PhaseSpace/WriteASCII.txt
- examples/PhaseSpace/WriteBinary.txt
- examples/PhaseSpace/WriteLimited.txt

When using phase space sources, it is important to decide how you want to handle a special case we call "Empty Histories." Recall that when a phase space is first recorded, for a given Original History, the set of resulting particles that cross the phase space surface:

- may include the primary particles, or
- may include a mix of primary and secondary particles, or
- may include only secondary particles, or
- may include no particles at all. We refer to this last case as an "Empty History."

When you subsequently use this file as a Phase Space Source, you need to decide how you want TOPAS to handle Empty Histories. If you're just calculating sums, it doesn't matter. The Empty Histories contribute nothing to the sum anyway. But if you're calculating statistical quantities, such as
Mean, then these Empty Histories matter. Imagine you want to know the mean dose per Original History. If half of the Original Histories never made it to the phase space file, the decision of whether or not to include these Empty Histories will give a factor of two difference in the calculated Mean Dose per History.

Depending on your use case you may or may not want to include these Empty Histories. It comes down to whether the statistics you want to calculate are:
- per Original History, or
- per Original Histories that Reached Phase Space

You control this with:
- s:So/MySource/PhaseSpaceIncludeEmptyHistories = "False" # defaults to false

TOPAS ASCII and Binary phase space format headers show all of the relevant information:
- Number of Original Histories
- Number of Original Histories that Reached Phase Space
- Number of Scored Particles

Limited phase space format header does not give:
- Number of Original Histories that Reached Phase Space
- so the only way to get that in Limited format is to first read through the entire phsp file and count how many histories contributed there.

TOPAS provides an option to check that the values in the header match what is in the file:
- s:So/MySource/PhaseSpacePreCheck = "True" # defaults to true

For TOPAS ASCII and Binary formats, this is a thorough safety check. It will catch any cases where the files have somehow become corrupted (as could happen, for example, if you are doing a very long phase space writing job and the output disk becomes full during some part of the job).

For Limited format, the check is still helpful but less thorough as the header file provides incomplete information. In Limited format, if you want to include Empty Histories, the check is required as it is the only way TOPAS can figure out how many Empty Histories there were.

If the phase space you are replaying came from a TOPAS job, the particle starting positions in that file will have been defined relative to the World Component. Set the Component parameter above to "World". If you want to offset these particles to some other center or orientation, choose a Component that has the new desired center and orientation (reuse some existing Component, or define a new Group Component just for this purpose). If the phase space you are replaying did not come from TOPAS, there is no automatic way to know what coordinate system was used. It will be up to you to choose a Component that has this appropriate coordinate system.

You can optionally tell the phase space source to ignore parts of its position information:
- b:So/MySource/PhaseSpaceIgnoreXPos = "True" # start all particles at Component’s X axis
- b:So/MySource/PhaseSpaceIgnoreYPos = "True" # start all particles at Component’s Y axis
- b:So/MySource/PhaseSpaceIgnoreZPos = "True" # start all particles at Component’s Z axis

That coordinate of the particle position then just exactly matches the Component center.

You can optionally invert any of the phase space axes by:
- b:So/MySource/PhaseSpaceInvertXAxis = "True"
- b:So/MySource/PhaseSpaceInvertYAxis = "True"
- b:So/MySource/PhaseSpaceInvertZAxis = "True"
In most cases you will instead want to just rotate the source component. However if the handedness of your source phase space is incorrect, one of these invert options will be necessary.

By default, a PhaseSpace source will run all of the histories in the file. To run all of the histories multiple times:

- `i:So/MySource/PhaseSpaceMultipleUse = 2 # reuse this phase space multiple times`

If you set `PhaseSpaceMultipleUse` to zero, the number of histories in the file will be ignored, and we will instead run the exact number from:

- `i:So/MySource/NumberOfHistoriesInRun`

This may mean only partial use of the phase space file, or partial reuse to get the right number of histories.

- If your data was generated with time dependence, partial reuse of phase space may not give valid results (you may be playing back only a part of the time sequence). Many more details on controlling number of histories are in the last section of this document, TOPAS Overall Control.
- Partial reuse of phase space can not Include Empty Histories. There is no statistically valid way to handle these empty histories when the phase space file is only partially used (since one does not know where in the phase space order these Empty Histories would have occurred).

For efficiency, the phase space file will be read in chunks of 10,000 particles at a time. Advanced users may find some reason to adjust this buffer size (though I can’t think of any):

- `i:So/MySource/PhaseSpaceBufferSize = 1000000`

Take care when mixing Phase Space Sources with Time Features (time features are discussed in detail later in this document).

While TOPAS can save the current TOPAS time to a phase space file, this time is not automatically applied when reading particles back in from phase space. Thus, if you want to correct replay source particles that were recorded with time features, it is your responsibility to apply the identical time features during the play back simulation. Some additional notes:

- Do not attempt to change the name of the phase space file over time. Save and replay all particles from a single phase space file.
- Do not use Random time mode. The randomly generated times during playback will not necessarily match the randomly generated times that were saved to the phase space. Only use Fixed or Sequential mode.
- If your intention is to play back with exactly the same sequence as you had when you generated the phase space file, make sure to set `s:So/MySource/PhaseSpaceIncludeEmptyHistories = "True"` otherwise empty histories will put the playback job out of synch with the original job

A future version of TOPAS will provide more tools to synchronize and check playback time features.

**Additional Control of Number of Histories**

Because TOPAS supports both sequential and random time, there are additional parameters that can control the number of histories in random mode. Read the last section of this document, TOPAS Overall Control, before using these parameters:

- `i:So/Default/NumberOfHistoriesInRandomJob = 100`
- `d:So/Default/ProbabilityOfUsingAGivenRandomTime = 1.`
Filtering Sources

Optionally filter what comes from the source. This is mainly intended for use with saved PhaseSpace, but is applied uniformly to all sources. Syntax is identical to that used for filtering in Scorers.

You may write your own additional filters (see Extending TOPAS at the end of this user guide).

Filter by Charge. Accepts one or more of "Positive", "Negative" or "Neutral":

- sv:So/MySource/OnlyIncludeParticlesCharged = 1 "Negative"
- sv:So/MySource/OnlyIncludeParticlesNotCharged = 1 "Negative"

Filter by Atomic Mass or Number:

- i:So/MySource/OnlyIncludeParticlesOfAtomicMass = 10  # allow all ions of atomic mass 10
- i:So/MySource/OnlyIncludeParticlesNotOfAtomicMass = 10
- i:So/MySource/OnlyIncludeParticlesOfAtomicNumber = 6  # allow all ions of Carbon
- i:So/MySource/OnlyIncludeParticlesNotOfAtomicNumber = 6

Filter by Particle’s Initial Kinetic Energy:

- d:So/MySource/OnlyIncludeParticlesWithInitialKEBelow = 1. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialKENotBelow = 1. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialKE = 1. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialKENot = 1. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialKEAbove = 10. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialKENotAbove = 10. MeV

Filter by Particle’s Initial Momentum:

- d:So/MySource/OnlyIncludeParticlesWithInitialMomentumBelow = 1. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialMomentumNotBelow = 1. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialMomentum = 1. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialMomentumNot = 1. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialMomentumAbove = 10. MeV
- d:So/MySource/OnlyIncludeParticlesWithInitialMomentumNotAbove = 10. MeV

Filter by Particle Name:

- sv:So/MySource/OnlyIncludeParticlesNamed = 2 "proton" "neutron"
- sv:So/MySource/OnlyIncludeParticlesNotNamed = 2 "proton" "neutron"

Particle names are as described above under Particle Sources...Particle Names.

You may specify more than one filter. For example, to score protons with initial KE over 100 MeV:

- sv:So/MySource/OnlyIncludeParticlesNamed = 1 "proton"
- d:So/MySource/OnlyIncludeParticlesWithInitialKEAbove = 100. MeV  # minimum energy

You can invert the results of all previous filters. The following would score only particles that are Not protons with initial KE over 100 MeV:

- sv:So/MySource/OnlyIncludeParticlesNamed = 2 "proton" "neutron"
- d:So/MySource/OnlyIncludeParticlesWithInitialKEAbove = 100. MeV  # minimum energy
- b:So/MySource/InvertFilter = "True"

Any filter property can be set by time features if you wish, to produce time-dependent filtering.
Physics

In Geant4, physics options are set in pieces of code called "Physics Lists". A physics list specifies what particles and physics processes are defined, plus various cuts and options.

By default, we set TOPAS physics to a list that has been shown to work well for proton therapy research at the Massachusetts General Hospital. This list includes models that handle not only protons but also all secondary particles (neutrons, helium ions, deuterons, tritons, photons, electrons, etc.). The default gives results that closely match a previous custom list that was described in:


but which can no longer be used since that list corresponded to a much earlier Geant4 release.

Advanced users can set their own parameters to override some of these default settings, or can specify entirely different physics lists.

You can choose from two general types of physics lists:

- "Reference" physics lists are pre-made, complete lists provided by Geant4.
- "Modular" physics lists are lists where you mix and match a set of modules to create a customized complete list.

You can also provide your own physics list (not recommended unless you have significant Geant4 expertise).

Modular Physics Lists

The default list we provide is a Modular physics list. It is specified by the following parameters that are set for you by default:

- s:Ph/ListName = "Default"
- s:Ph/Default/Type= "Geant4_Modular"
- sv:Ph/Default/Modules = 7 "g4em-standard_opt3" "g4h-phy_QGSP_BIC_HP" "g4decay"
  "g4ion-binarycascade" "g4h-elastic_HP" "g4q-stopping" "g4radioactivedecay"
- d:Ph/Default/EMRangeMin = 100. eV # See note below on recommended values
- d:Ph/Default/EMRangeMax = 500. MeV
- i:Ph/Default/dEdXBins = 220
- i:Ph/Default/LamdaBins = 220

The Geant4 EM physics group recommends against setting EMRangeMin too low:

- Set to 100. eV or greater when using standard Geant4 EM physics
- Set to 10. eV or greater when using Geant4-DNA physics

If you want to run with no physics, but only the transportation process (useful for some demos and tests), specify the modules in the following special way:

- sv:Ph/Default/Modules = 1 "Transportation_Only"

The full list of module names and the corresponding Geant4 class names are shown in the table at the end of this section.

The remaining options for Geant4_Modular are:

- d:Ph/Default/CutForAllParticles = 0.05 mm # single range cut to use for all particles
- d:Ph/Default/CutForGamma = 0.05 mm # overrides CutForAllParticles for Gamma
• `d:Ph/Default/CutForElectron = 0.05 mm` # overrides CutForAllParticles for Electron
• `d:Ph/Default/CutForPositron = 0.05 mm` # overrides CutForAllParticles for Positron
• `d:Ph/Default/CutForProton = 0.05 mm` # overrides CutForAllParticles for Proton
• `d:Ph/Default/CutForAlpha = 0.05 mm` # overrides CutForAllParticles for Alpha
• `d:Ph/Default/CutForDeuteron = 0.05 mm` # overrides CutForAllParticles for Deuteron
• `d:Ph/Default/CutForTriton = 0.05 mm` # overrides CutForAllParticles for Triton
• `d:Ph/Default/EMRangeMin = 100. eV` # minimum for EM tables
• `d:Ph/Default/EMRangeMax = 300. MeV` # maximum for EM tables
• `i:Ph/Default/dEdXBins = 220` # number of bins for dEdX tables
• `i:Ph/Default/LamdaBins = 220` # number of Lambda bins
• `b:Ph/Default/Fluorescence = "False"` # Set to true to turn on Flourescence
• `b:Ph/Default/Auger = "False"` # Set to true to turn on Auger
• `b:Ph/Default/PIXE = "False"` # Set to true to turn on PIXE

By default, cuts affect the entire world, but you can optionally divide the world into several regions and can specify different cuts in each region. First, specify which components belong to a given region:
• `s:Ge/MyComponent/AssignToRegionNamed = "MyRegion"`
  All children of this component will also be assigned to that region, unless the child has its own "AssignToRegionNamed" parameter.
  There is no requirement that all of the components in a given region be contiguous.

Then assign cuts per region by including the region name in the parameter name as in:
• `d:Ph/Default/ForRegion/MyRegion/CutForGamma = 0.05 mm`
• `d:Ph/Default/ForRegion/MyRegion/CutForElectron = 0.05 mm`
• `d:Ph/Default/ForRegion/MyRegion/CutForPositron = 0.05 mm`
• `d:Ph/Default/ForRegion/MyRegion/CutForProton = 0.05 mm`

Cuts do not affect all processes, but only those listed below:
• Energy thresholds for gamma are used in Bremsstrahlung
• Energy thresholds for electrons are used in ionisation and e+e- pair production processes
• Energy thresholds for positrons are used in e+e- pair production process
• Energy thresholds for gamma and electrons are used optionally in all discrete processes
  ○ Photoelectric effect
  ○ Compton
  ○ Gamma conversion
• Energy thresholds for protons are used in processes of elastic scattering for hadrons and ions defining the threshold for kinetic energy of nuclear recoil

<table>
<thead>
<tr>
<th>TOPAS Module Name</th>
<th>Geant4 Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>g4decay</td>
<td>G4DecayPhysics</td>
</tr>
<tr>
<td>g4em-standard_opt0</td>
<td>G4EmStandardPhysics</td>
</tr>
<tr>
<td>g4em-standard_opt1</td>
<td>G4EMStandardPhysics_option1</td>
</tr>
<tr>
<td>g4em-standard_opt2</td>
<td>G4EMStandardPhysics_option2</td>
</tr>
<tr>
<td>g4em-standard_opt3</td>
<td>G4EMStandardPhysics_option3</td>
</tr>
<tr>
<td>TOPAS Module Name</td>
<td>Geant4 Class Name</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>g4em-standard_opt4</td>
<td>G4EMStandardPhysics_option4</td>
</tr>
<tr>
<td>g4em-livermore</td>
<td>G4EmLivermorePhysics</td>
</tr>
<tr>
<td>g4em-polarized</td>
<td>G4EmLivermorePolarizedPhysics</td>
</tr>
<tr>
<td>g4em-penelope</td>
<td>G4EmPenelopePhysics</td>
</tr>
<tr>
<td>g4em-dna</td>
<td>G4EmDNAPhysics</td>
</tr>
<tr>
<td>ts-dna-extended</td>
<td>TsDNAPhysics</td>
</tr>
<tr>
<td>g4em-dna-chemistry</td>
<td>G4EmDNAPhysicsChemistry</td>
</tr>
<tr>
<td>g4em-extra</td>
<td>G4EmExtraPhysics</td>
</tr>
<tr>
<td>G4em-lowep</td>
<td>G4EmLowEPPhysics</td>
</tr>
<tr>
<td>G4emq-extra</td>
<td>G4EmQExtraPhysics</td>
</tr>
<tr>
<td>g4h-elastic</td>
<td>G4HadronElasticPhysics</td>
</tr>
<tr>
<td>g4h-elastic_D</td>
<td>G4HadronDElasticPhysics</td>
</tr>
<tr>
<td>g4h-elastic_H</td>
<td>G4HadronHElasticPhysics</td>
</tr>
<tr>
<td>g4h-elastic_Q</td>
<td>G4HadronQElasticPhysics</td>
</tr>
<tr>
<td>g4h-elastic_HP</td>
<td>G4HadronElasticPhysicsHP</td>
</tr>
<tr>
<td>g4h-elastic_LEND</td>
<td>G4HadronElasticPhysicsLEND</td>
</tr>
<tr>
<td>g4h-elastic_LHEP</td>
<td>G4HadronElasticPhysicsLHEP</td>
</tr>
<tr>
<td>g4h-elastic_XS</td>
<td>G4HadronElasticPhysicsXS</td>
</tr>
<tr>
<td>g4h-chargeexchange</td>
<td>G4ChargeExchangePhysics</td>
</tr>
<tr>
<td>g4ion</td>
<td>G4IonPhysics</td>
</tr>
<tr>
<td>g4ion-binarycascade</td>
<td>G4IonBinaryCascadePhysics</td>
</tr>
<tr>
<td>g4ion-inclxx</td>
<td>G4IonINCLXXPhysics</td>
</tr>
<tr>
<td>g4ion-LHEP</td>
<td>G4IonLHEPPhysics</td>
</tr>
<tr>
<td>g4ion-QMD</td>
<td>G4IonQMDPhysics</td>
</tr>
<tr>
<td>g4optical</td>
<td>G4OpticalPhysics</td>
</tr>
<tr>
<td>g4radioactivedecay</td>
<td>G4RadioactiveDecayPhysics</td>
</tr>
<tr>
<td>g4n-trackingcut</td>
<td>G4NeutronTrackingCut</td>
</tr>
<tr>
<td>g4stopping</td>
<td>G4StoppingPhysics</td>
</tr>
<tr>
<td>g4lhep-stopping</td>
<td>G4LHEPStoppingPhysics</td>
</tr>
<tr>
<td>TOPAS Module Name</td>
<td>Geant4 Class Name</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>g4q-atomic</td>
<td>G4QAtomicPhysics</td>
</tr>
<tr>
<td>g4q-capture</td>
<td>G4QCaptureAtRestPhysics</td>
</tr>
<tr>
<td>g4q-elastic</td>
<td>G4QERlasticPhysics</td>
</tr>
<tr>
<td>g4q-emextra</td>
<td>G4QEmExtraPhysics</td>
</tr>
<tr>
<td>g4q-ion</td>
<td>G4QIonPhysics</td>
</tr>
<tr>
<td>g4q-neutrino</td>
<td>G4QNeutrinoPhysics</td>
</tr>
<tr>
<td>g4q-photonuclear</td>
<td>G4QPhotoNuclearPhysics</td>
</tr>
<tr>
<td>g4q-stopping</td>
<td>G4QStoppingPhysics</td>
</tr>
<tr>
<td>g4h-inelastic_QBBC</td>
<td>G4HadronInelasticQBBC</td>
</tr>
<tr>
<td>g4h-phy_CHIPS</td>
<td>HadronPhysicsCHIPS</td>
</tr>
<tr>
<td>g4h-phy_CHIPS_HP</td>
<td>HadronPhysicsCHIPS_HP</td>
</tr>
<tr>
<td>g4h-phy_FTFP_BERT</td>
<td>HadronPhysicsFTFP_BERT</td>
</tr>
<tr>
<td>g4h-phy_FTFP_BERT_HP</td>
<td>HadronPhysicsFTFP_BERT_HP</td>
</tr>
<tr>
<td>g4h-phy_FTFP_BERT_TRV</td>
<td>HadronPhysicsFTFP_BERT_TRV</td>
</tr>
<tr>
<td>g4h-phy_FTF_BIC</td>
<td>HadronPhysicsFTF_BIC</td>
</tr>
<tr>
<td>g4h-phy_LHEP</td>
<td>HadronPhysicsLHEP</td>
</tr>
<tr>
<td>g4h-phy_LHEP_EMV</td>
<td>HadronPhysicsLHEP_EMV</td>
</tr>
<tr>
<td>g4h-phy_QGSC_BERT</td>
<td>HadronPhysicsQGSC_BERT</td>
</tr>
<tr>
<td>g4h-phy_QGSC_CHIPS</td>
<td>HadronPhysicsQGSC_CHIPS</td>
</tr>
<tr>
<td>g4h-phy_QGSP</td>
<td>HadronPhysicsQGSP</td>
</tr>
<tr>
<td>g4h-phy_QGSP_BERT</td>
<td>HadronPhysicsQGSP_BERT</td>
</tr>
<tr>
<td>g4h-phy_QGSP_BERT_95</td>
<td>HadronPhysicsQGSP_BERT_95</td>
</tr>
<tr>
<td>g4h-phy_QGSP_INCLXX</td>
<td>HadronPhysicsQGSP_INCLXX</td>
</tr>
<tr>
<td>g4h-phy_QGSP_CHIPS</td>
<td>HadronPhysicsQGSP_BERT_CHIPS</td>
</tr>
<tr>
<td>g4h-phy_QGSP_BERT_HP</td>
<td>HadronPhysicsQGSP_BERT_HP</td>
</tr>
<tr>
<td>g4h-phy_QGSP_BERT_NOLEP</td>
<td>HadronPhysicsQGSP_BERT_NOLEP</td>
</tr>
<tr>
<td>g4h-phy_QGSP_BERT_TRV</td>
<td>HadronPhysicsQGSP_BERT_TRV</td>
</tr>
<tr>
<td>g4h-phy_QGSP_BIC</td>
<td>HadronPhysicsQGSP_BIC</td>
</tr>
<tr>
<td>g4h-phy_QGSP_BIC_HP</td>
<td>HadronPhysicsQGSP_BIC_HP</td>
</tr>
<tr>
<td>TOPAS Module Name</td>
<td>Geant4 Class Name</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>g4h-phy_QGSP_FTFP_BERT</td>
<td>HadronPhysicsQGSP_FTFP_BERT</td>
</tr>
<tr>
<td>g4h-phy_QGS_BIC</td>
<td>HadronPhysicsQGS_BIC</td>
</tr>
<tr>
<td>g4h-phy_Shielding</td>
<td>HadronPhysicsShielding</td>
</tr>
</tbody>
</table>

**Reference Physics Lists**

Reference physics lists are pre-made, complete lists provided by Geant4.

One complication with reference lists is that they do not support use of Parallel Worlds. This means that you cannot place components into a parallel world, and, for the dividable components (TsBox, TsCylinder and TsSphere), you cannot score with a different set of divisions than you have set for the component itself (we handle such complex scoring by creating parallel worlds). TOPAS will give an error if you attempt to use a reference list in a situation where parallel worlds are needed. In such situations, use Geant4_Modular as described below.

The names of the reference physics lists, and their detailed descriptions, are at:


To use a reference physics list, specify the list name in the Type parameter, such as:

- s:Ph/Default/Type = "QGSP_BERT_HP"

Reference physics lists allow only one additional option:

- d:Ph/Default/CutForAllParticles = 0.05 mm # single range cut to use for all particles

**User-Supplied Physics Lists**

See Extending TOPAS at the end of this user guide for details on how to provide your own physics list. This option is not recommended unless you have significant Geant4 expertise.

**Multiple Physics Lists**

You can have more than one list defined at the same time, but only the one specified in Ph/ListName will actually be in effect:

- s:Ph/ListName = "MyList1"
- s:Ph/MyList1/Type= "QGSP_BERT_HP" # This list is in effect now
- d:Ph/MyList1/CutForAllParticles = 0.05 mm
- ...
- s:Ph/MyList2/Type = "Geant4_Modular" # This list goes into effect if Ph/ListName set to MyList2
- sv:Ph/MyList2/Modules = 1 "g4em-standard_opt3"
- d:Ph/MyList2/CutForGamma = 0.04 mm

**Production Thresholds**

Production Thresholds and range cuts are discussed in detail in the Geant4 Application Developers Guide. By default, appropriate limits are set by the physics list. You can override these defaults with:

- d:Ph/MyPhysics/SetProductionCutLowerEdge = 200 eV
- d:Ph/MyPhysics/SetProductionCutHighEdge = 30 MeV

For further discussion, see the Geant4 Application Developers Guide.
**Step Size**

The selection of step size is a complex issue in Monte Carlo tracking. Geant4 has its own complex logic for automatically selecting what it thinks will be an appropriate step size, based on local geometry and physics, and the user will not generally need to override this automatic behavior. However, your applications may be sensitive to this behavior, and you may therefore want to set a maximum step size in certain components. In general, larger step sizes give faster performance, but smaller step sizes may give better accuracy.

To limit Geant4’s maximum step size in a given component:

- \( d:Ge/MyComponent/MaxStepSize = 1. \text{mm} \) # sets maximum step size used in this component.

Step size settings do not affect other Components placed within this Component. You must explicitly set the step size for any subcomponents that you want to affect.

The choice of maximum step size is highly dependent on your exact simulation problem. If you think you need to set a maximum step size, try running with several values, and pick one for which a small variation up or down does not cause a significant change in results.

For cases where a particle is traveling through vacuum, if the MaxStepSize fits evenly into the distance from source to a scoring component, a known Geant4 scoring bug can cause the step to not score. For example, if you have a source exactly 10 cm from a scoring box, and the step size is 1 mm, and the medium is vacuum, then the step size limiter will limit the tenth step exactly at the scoring box. This will cause this step to not score (in Geant4 terms, we say that the "step limitation process" is triggered but not the "scoring" process). This bug tends to be seen more in simple test setups than in realistic applications (where complex geometry and various physics processes make it less likely that step limitation will fall exactly at a scoring component). We have reported this bug to Geant4 and hope for a fix soon. Until this bug is fixed, use caution when applying the MaxStepSize, perhaps adding a tiny bit to whatever size you want to make it an unusual value.

**Optical Photons**

A full description of the tracking of optical photons is available in the Geant4 Guide for Applications Developers.

Topas allows to include optical physics by means of the g4optical module in the physics list. The available optical processes included in the g4optical module are: scintillation, Cerenkov radiation, wavelength shifting, optical absorption, Rayleigh scattering and boundary processes. However, this module is not sufficient to set up the generation and tracking of optical photons. The optical properties of the material of the volumes must to be defined too (at the least the refractive index must to be defined). There exist two types of variables to define the optical properties: a vector based and constant based. The vector-based parameter allows to define a property (refractive index for example) in function of the photon’s energy. While the constant-based parameters allows to define an scalar (scintillation yield for example)

To activate the optical properties in a material one must to set:

- \( b:Ma/MyMaterial/EnableOpticalProperties = "True" \)

To set a property based on a vector, one must to define the energy of reference. For example to include the refractive index one must to define two parameters:

- \( dv:Ma/MyMaterial/RefractiveIndex/Energies = 3 \ 2.0 \ 2.5 \ 3.0 \ \text{eV} \)
- \texttt{uv:Ma/MyMaterial/RefractiveIndex/Values} = 3 \ 1.58 \ 1.58 \ 1.58

To set a property based on a scalar only one parameter is needed, for example:
- \texttt{u:Ma/MyMaterial/ScintillationYield} = 1120 \ # \text{ in ph/MeV}
- \texttt{d:Ma/MyMaterial/FastTimeConstant} = 2.1 \ \text{ns}

The full list of parameters available is listed in the next table.

<table>
<thead>
<tr>
<th>Parameter type</th>
<th>Parameter’s name</th>
</tr>
</thead>
<tbody>
<tr>
<td>uv</td>
<td>RefractiveIndex</td>
</tr>
<tr>
<td>uv</td>
<td>ImaginaryRefractiveIndex</td>
</tr>
<tr>
<td>uv</td>
<td>RealRefractiveIndex</td>
</tr>
<tr>
<td>dv</td>
<td>AbsLength</td>
</tr>
<tr>
<td>uv</td>
<td>FastComponent</td>
</tr>
<tr>
<td>uv</td>
<td>SlowComponent</td>
</tr>
<tr>
<td>uv</td>
<td>Miehg</td>
</tr>
<tr>
<td>uv</td>
<td>SpecularLobeConstant</td>
</tr>
<tr>
<td>uv</td>
<td>SpecularSpikeConstant</td>
</tr>
<tr>
<td>uv</td>
<td>BackScatterConstant</td>
</tr>
<tr>
<td>uv</td>
<td>WLSAbsLength</td>
</tr>
<tr>
<td>uv</td>
<td>WLSComponent</td>
</tr>
<tr>
<td>uv</td>
<td>Reflectivity</td>
</tr>
<tr>
<td>uv</td>
<td>Efficiency</td>
</tr>
<tr>
<td>uv</td>
<td>Transmittance</td>
</tr>
<tr>
<td>u</td>
<td>ScintillationYield (in photons/MeV)</td>
</tr>
<tr>
<td>u</td>
<td>ResolutionScale</td>
</tr>
<tr>
<td>d</td>
<td>FastTimeConstant</td>
</tr>
<tr>
<td>d</td>
<td>SlowTimeConstant</td>
</tr>
<tr>
<td>u</td>
<td>YieldRatio</td>
</tr>
<tr>
<td>u</td>
<td>MiehgForward</td>
</tr>
<tr>
<td>u</td>
<td>MiehgBackward</td>
</tr>
<tr>
<td>u</td>
<td>MiehgForwardRatio</td>
</tr>
<tr>
<td>u</td>
<td>WLSTimeConstant</td>
</tr>
<tr>
<td>u</td>
<td>BirksConstant (in mm/MeV)</td>
</tr>
</tbody>
</table>

### Optical Surfaces

If perfect smooth interface is between two dielectric materials, the users only needs to provide the refractive index. In all other cases, a surface or optical boundary needs to be defined. There exist two kinds of surfaces: the border surface that delimits the boundary between two components; and the skin surface which surrounds one single component.
Border surface is ordered in the sense that the order of the components matters, two border surfaces can exist between a pair of components. Thus, the following parameters define two surfaces for a pair of components:
- \( s:Ge/MyComponent1/OpticalBehaviorTo/MyComponent2 = "MySurface1" \)
- \( s:Ge/MyComponent2/OpticalBehaviorTo/MyComponent1 = "MySurface2" \)

For skin surface only one surface can be defined per component:
- \( s:Ge/MyComponent1/OpticalBehavior = "MySurface1" \)

Surfaces can be defined as follows (see next table for description):
- \( s:Su/MySurfaceName/Type = "\text{dielectric\_dielectric}" \) # or \( \text{dielectric\_metal} \)

Next, choose the model for optical surfaces:
- \( s:Su/MySurfaceName/Model = "\text{Glisur}" \) # Or Unified

Finally the finish:
- \( s:Su/MySurfaceName/Finish = "\text{Polished}" \)

In addition, more detailed properties can be added by parameters described in the table 1. In such a case, the way to define would be for example (with prefix Su instead of Ma):
- \( dv:Su/MySurfaceName/Energies = 2 \ 1.0 \ 4.0 \ \text{eV} \)
- \( uv:Su/MySurfaceName/Reflectivity = 2 \ 0.8 \ 0.8 \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Variable type</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
<td>( \text{dielectric_dielectric}, \text{dielectric_metal} )</td>
</tr>
<tr>
<td>Finish</td>
<td>string</td>
<td>polished: smooth perfectly polished surface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>polishedfrontpainted: smooth top-layer (front) paint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>polishedbackpainted: same as 'polished' but with a back-paint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ground: rough surface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>groundfrontpainted: rough top-layer (front) paint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>groundbackpainted: same as 'ground' but with a back-paint</td>
</tr>
<tr>
<td>SigmaAlpha</td>
<td>unitless</td>
<td>Between 0 and 1. By default 0</td>
</tr>
</tbody>
</table>
Scoring

There are two basic classes of scorers: one class score activity in a volume, the other class score activity at a surface. The Phase Space output system is a specific case of a Surface Scorer.

You can have any number of scorers. A scorer is defined when you have a line that ends with Quantity, such as:

- \texttt{s:Sc/MyScorer/Quantity = "DoseToMedium"}

You may write your own additional scorers (see Extending TOPAS at the end of this user guide).

Volume Scorers

Volume Scorer Quantities are:

- \texttt{DoseToMedium} - from sum of energy deposits divided by mass
- \texttt{DoseToWater} - from energy-dependent stopping power conversion (see below)
- \texttt{DoseToWaterBinned} - from energy-dependent stopping power conversion (see below)
- \texttt{DoseToMaterial} - from energy-dependent stopping power conversion (see below)
- \texttt{EnergyDeposit}
- \texttt{Fluence} - from sum of step lengths divided by volume
- \texttt{EnergyFluence} - from sum of step lengths times energy divided by volume
- \texttt{Charge}
- \texttt{EffectiveCharge} - from counting method described below
- \texttt{StepCount}
- \texttt{ProtonLET} - from various methods described below

Volume Scorers must indicate the relevant Component:

- \texttt{s:Sc/MyScorer/Component = "Phantom"}

For \texttt{DoseToMaterial}, you must also specify the material:

- \texttt{s:Sc/MyScorer/Material = "SomeMaterial"}

Note that in this case, the material name must exactly match the case defined in Geant4.

To check what materials have been defined, add the parameter:

- \texttt{i:Ma/Verbosity = 1}

For \texttt{DoseToWaterBinned}, \texttt{DoseToWater} and \texttt{DoseToMaterial}, we use energy-dependent stopping power conversion as in:

\[
dose\_to\_new\_material = dose\_to\_medium \times \left( \frac{\text{density\_of\_medium}}{\text{density\_of\_new\_material}} \right) \times \left( \frac{d\text{EdX\_in\_new\_material}}{d\text{EdX\_in\_medium}} \right)
\]

The \texttt{dEdX} comes from the Geant4 EmCalculator utility.

To make this work, at least a small amount of this material be present somewhere in the World. TOPAS will refuse to set up this scorer if you have not actually used this material.

The neutral particles, the stopping power is taken from the proton stopping power for a fixed energy:

- \texttt{Sc/MyScorer/SubstituteEnergyForNeutralScaling} (default = 100 MeV)

The two \texttt{DoseToWater} scorers provide you a choice of either best accuracy or best speed.

- \texttt{DoseToWater} gives best accuracy, calculating stopping power on-the-fly for the exact energy.
DoseToWaterBinned gives best speed, looking up stopping power from a pre-calculated table binned by energy. It is about 50% faster that DoseToWater for typical patient simulations. The difference in accuracy is not significant for most studies.

For DoseToWaterBinned, the pre-calculated table of stopping power ratios can be tuned by:
- Sc/MyScorer/ProtonEnergyBinSize (default = 1 MeV)
- Sc/MyScorer/MinProtonEnergyForStoppingPowerRatio (default = 1 MeV)
- Sc/MyScorer/MaxProtonEnergyForStoppingPowerRatio (default = 500 MeV)
- Sc/MyScorer/ElectronEnergyBinSize (default = 1 keV)
- Sc/MyScorer/MinElectronEnergyForStoppingPowerRatio (default = 1 keV)
- Sc/MyScorer/MaxElectronEnergyForStoppingPowerRatio (default = 1 MeV)

For Charge and EffectiveCharge:
- If a particle reaches zero kinetic energy in the scoring volume, its charge is accumulated
- If a particle is generated in the scoring volume, its charge is subtracted

The ProtonLET scorer gives the LET of primary and secondary protons, including the energy deposited by associated secondary electrons. It uses techniques discussed in two recent articles on best practices to score LET in Geant4:

In particular, we adopt the methods developed by Granville and Sawakuchi.

We compute dose-averaged LET, but you may instead request track-averaged:
- s:Sc/MyScorer/WeightBy = “Track” # defaults to "Dose"

By default, the LET is computed by dividing the energy deposited by the step length. Such distributions can feature spurious spikes, caused by events where the step length is severely constrained by a voxel boundary crossing. Three solutions to this issue are provided:
- By default, a step-by-step upper cut-off is set, such that steps contributing greater than this value are not be scored:
  - d:Sc/MyScorer/MaxScoredLET = 100 MeV/mm # defaults to 100 MeV/mm
- Alternately, you can set the LET computation to look up the electronic stopping power for the pre-step proton energy:
  - b:Sc/MyScorer/UsePreStepLookup = "True" # defaults to “False”
- Or you can increase the electron production threshold:
  - Ph/Default/CutForElectron = 1 mm # defaults to 0.05 mm

Surface Scorers

Surface Scorer Quantities are:
- SurfaceCurrent
- SurfaceTrackCount
- PhaseSpace

Surface Scorers must indicated the relevant Component and Surface name:
- s:Sc/MyScorer/Surface = "Phantom/ZMinusSurface"

Where the X, Y or Z refers to the coordinate system of the Component

The syntax to specify surface depends on which shape component is involved.
- For TsBox and G4Box:
  - XMinusSurface # in coordinate system of the Component
- XPlusSurface
- YMinusSurface
- YPlusSurface
- ZMinusSurface
- ZPlusSurface

- For TsCylinder and G4Tubs:
  - ZMinusSurface # in coordinate system of the Component
  - ZPlusSurface
  - InnerCurvedSurface
  - OuterCurvedSurface
  - MinusPhiSurface - relevant if cylinder or tubs has been cut or divided along Phi
  - PlusPhiSurface - relevant if cylinder or tubs has been cut or divided along Phi

- For TsSphere and G4Sphere:
  - InnerCurvedSurface
  - OuterCurvedSurface
  - MinusPhiSurface - relevant if cylinder or tubs has been cut or divided along Phi
  - PlusPhiSurface - relevant if cylinder or tubs has been cut or divided along Phi
  - MinusThetaSurface - relevant if cylinder or tubs has been cut or divided along Theta
  - PlusThetaSurface - relevant if cylinder or tubs has been cut or divided along Theta

If you are scoring on a divided component (TsBox, TsCylinder or TsSphere), all surfaces of the divided
component then become sensitive for scoring. So, for example, ZMinusSurface will mean to
accumulate hits on every ZMinusSurface of every voxel in the divided TsBox.

Creators of parameter files can pre-define more user-friendly synonyms through relative parameters,
such as:
- s:Ge/WaterTank/Water/UpstreamSurface = Ge/WaterTank/Water/ZMinusSurface
so that users can then score using these named Surfaces, as in:
- s:Sc/MyScorer/Surface = WaterTank/Water/UpstreamSurface

You may write your own additional scorers. A Programmer’s Guide will eventually be provided to help
with this. For now, just study some of the existing scorers. Once you have written a new scorer, link it
to the rest of TOPAS by adding your scorer to scoring/TsScoringtHub.cc
Phase Space Output

Phase Space refers to the technique of saving or replaying a set of particles crossing a given surface.

- When one saves a phase space, one defines a surface and then saves the position, particle type, energy and momentum of some or all particles crossing that surface.
- When one replays a phase space, one starts a set of particles from the saved positions, with the saved particle types, energy and momentum.

Phase Space enables separating two parts of a simulation or analysis job, and can be used to transfer sets of particles among different codes.

If your Surface Scorer has Quantity = "PhaseSpace", the output will be a pair of Phase Space files:
- A .header file tells the number of histories, the number of saved particles and the order of information in the .phsp file
- A .phsp file contains all the details of all the saved particles

We support three formats for Phase Space:
- ASCII provides particle information in an easy to read simple text file, which data encoded as a series of columns of text. The header file tells the contents and column order per particle.
- Binary provides the same information as ASCII, but in a much more compact format, with data encoded in a stream of bytes. The header file tells the contents and byte order per particle. Use Binary in cases where the ASCII format produces excessively large files.
- Limited is an alternate binary format compatible with some legacy codes. It has fewer options for what data can be expressed, but is compatible with codes such as that used by Varian for their TrueBeam phase space files. Use Limited format only when you need to exchange phase space with legacy codes.

You tell TOPAS what format to write out by setting:
- s:Sc/MyScorer/OutputType = "ASCII" # "Binary", "ASCII" or "Limited"

All formats provide at least ten quantities for each scored particle:
- X position
- Y position
- Z position
- U (direction cosine of momentum with respect to X)
- V (direction cosine of momentum with respect to Y)
- Energy in MeV
- Weight
- Particle ID
- Flag to tell if Third Direction Cosine is Negative (1 means true)
- Flag to tell if this is the First Scored Particle from this History (1 means true) (note that this may or may not be the primary, as the primary may or may not have made it all the way to the scoring plane).

The positions are relative to the center of the World.

For the ASCII and Binary formats, you can turn on additional columns of phase space output:
- b:Sc/MyScorer/IncludeTOPASTime = "True" # Time used by TimeFeatures for this history
- b:Sc/MyScorer/IncludeTimeOfFlight = "True" # Time of Flight of this particle from start of history to scoring plane
- b:Sc/MyScorer/IncludeRunID = "True"
- `b:Sc/MyScorer/IncludeEventID = "True"`
- `b:Sc/MyScorer/IncludeTrackID = "True"`
- `b:Sc/MyScorer/IncludeParentID = "True" # Track ID of parent particle`
- `b:Sc/MyScorer/IncludeCharge = "True"`
- `b:Sc/MyScorer/IncludeVertexInfo = "True" # Initial KE, Position and Momentum`
- `b:Sc/MyScorer/IncludeSeed = "True"

The last of these gives the four variable parts of a random seed. Replaying this random seed will get you the same event back later. The full random seed should be a file of the form:

```
Uvec
1878463799
3
1425618182
1466214412
```

To reuse a saved seed, create a file with the above five lines, replacing the four numeric parts with the four integers in the phase space file. Assuming you name that file "event1.rndm", you can then make TOPAS start from this random seed by having TOPAS wake up at the Geant4 command line, by using:

- `Ts/PauseBeforeSequence = "True"

And then typing:

- `/random/resetEngineFrom event1.rndm`
- `exit`

The phase space scorer buffers output to avoid excessive disk access. You will not generally need to adjust this buffering value, but can adjust if if you wish.

- `i:Sc/MyScorer/PhaseSpaceBufferSize = 1000 # Number of particles in phase space buffer`

For more detail on Phase Space formats, see Miscellaneous Notes at the end of this User Guide.
Filtering Scorers

You may add filters to limit what is scored.

You may write your own additional filters (see Extending TOPAS at the end of this user guide).

Filter by Generation. Accepts either "Primary" or "Secondary":

- s:Sc/MyScorer/OnlyIncludeParticlesOfGeneration = "Primary"

Filter by Charge. Accepts one or more of "Positive", "Negative" or "Neutral":

- sv:Sc/MyScorer/OnlyIncludeParticlesCharged = 1 "Negative"
- sv:Sc/MyScorer/OnlyIncludeParticlesNotCharged = 1 "Negative"

Filter by Atomic Mass or Number:

- i:Sc/MyScorer/OnlyIncludeParticlesOfAtomicMass = 10         # allow all ions of atomic mass 10
- i:Sc/MyScorer/OnlyIncludeParticlesNotOfAtomicMass = 10
- i:Sc/MyScorer/OnlyIncludeParticlesOfAtomicNumber = 6       # allow all ions of Carbon
- i:Sc/MyScorer/OnlyIncludeParticlesNotOfAtomicNumber = 6

Filter by Particle's Initial Kinetic Energy:

- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialKEBelow = 1. MeV
- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialKENotBelow = 1. MeV
- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialKE = 1. MeV
- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialKEAbove = 10. MeV
- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialKENotAbove = 10. MeV

Filter by Particle's Initial Momentum:

- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialMomentumBelow = 1. MeV
- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialMomentumNotBelow = 1. MeV
- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialMomentum = 1. MeV
- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialMomentumAbove = 10. MeV
- d:Sc/MyScorer/OnlyIncludeParticlesWithInitialMomentumNotAbove = 10. MeV

Filter by Kinetic Energy of Particle or its Ancestor when it hit the Scoring Component (excludes any particles descended from primaries that originated in the component):

- d:Sc/MyScorer/OnlyIncludeIncidentParticleKEBelow = 1. MeV
- d:Sc/MyScorer/OnlyIncludeIncidentParticleKENotBelow = 1. MeV
- d:Sc/MyScorer/OnlyIncludeIncidentParticleKE = 1. MeV
- d:Sc/MyScorer/OnlyIncludeIncidentParticleKEAbove = 10. MeV
- d:Sc/MyScorer/OnlyIncludeIncidentParticleKENotAbove = 10. MeV

Filter by Initial Momentum of Particle or its Ancestor when it hit the Scoring Component (excludes any particles descended from primaries that originated in the component):
Filter by Process that created the particle. Allows one or more process name:
- sv:Sc/MyScorer/OnlyIncludeParticlesFromProcess = 2 "hioni" "eBrem"
- sv:Sc/MyScorer/OnlyIncludeParticlesNotFromProcess = 2 "hioni" "eBrem"

Filter by Process that created the particle or any of its ancestors:
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorFromProcess = 2 "hioni" "eBrem"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotFromProcess = 2 "hioni" "eBrem"

Filter by Particle Name:
- sv:Sc/MyScorer/OnlyIncludeParticlesNamed = 2 "proton" "neutron"
- sv:Sc/MyScorer/OnlyIncludeParticlesNotNamed = 2 "proton" "neutron"

Filter by Particle Name or the name of any of the particle’s ancestors. Use this to, for example, score all charge that results from neutrons, even if the final particle is not a neutron.
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNamed = 1 "neutron"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotNamed = 1 "neutron"

Particle names are as described above under Particle Sources...Particle Names.

Filter by Particle’s Origin Volume, Component, or Component and Subcomponents:
- sv:Sc/MyScorer/OnlyIncludeParticlesFromVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeParticlesNotFromVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeParticlesFromComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeParticlesNotFromComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeParticlesFromComponentOrSubComponentsOf = 1 "Nozzle"
- sv:Sc/MyScorer/OnlyIncludeParticlesNotFromComponentOrSubComponentsOf = 1 "Nozzle"

Filter by Particle or its Ancestor’s Origin Volume, Component, or Component and Subcomponents:
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorFromVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotFromVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorFromComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotFromComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorFromComponentOrSubComponentsOf = 1 "Nozzle"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotFromComponentOrSubComponentsOf = 1 "Nozzle"

Filter by whether Particle Interacted in Volume, Component, or Component and Subcomponents:
- sv:Sc/MyScorer/OnlyIncludeIfParticleInteractedInVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleNotInteractedInVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleInteractedInComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleNotInteractedInComponent = 1 "Jaws"
Filter by Particle or its Ancestor Interacted in Volume, Component, or Component and Subcomponents:

- sv:Sc/MyScorer/OnlyIncludeIfParticleInteractedInComponentOrSubComponentsOf = 1 "Nozzle"
- sv:Sc/MyScorer/OnlyIncludeIfParticleNotInteractedInComponentOrSubComponentsOf = 1 "Nozzle"

Filter by Particle or its Ancestor Interacted in Volume, Component, or Component and Subcomponents:

- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorInteractedInVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotInteractedInVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorInteractedInComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotInteractedInComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorInteractedInComponentOrSubComponentsOf = 1 "Nozzle"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotInteractedInComponentOrSubComponentsOf = 1 "Nozzle"

Filter by whether Particle Traversed Volume, Component, or Component and Subcomponents:

- sv:Sc/MyScorer/OnlyIncludeIfParticleTraversedVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleNotTraversedVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleTraversedComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleNotTraversedComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleTraversedComponentOrSubComponentsOf = 1 "Nozzle"
- sv:Sc/MyScorer/OnlyIncludeIfParticleNotTraversedComponentOrSubComponentsOf = 1 "Nozzle"

Filter by Particle or its Ancestor Traversed Volume, Component, or Component and Subcomponents:

- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorTraversedVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotTraversedVolume = 1 "Propeller20/Leaf"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorTraversedComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotTraversedComponent = 1 "Jaws"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorTraversedComponentOrSubComponentsOf = 1 "Nozzle"
- sv:Sc/MyScorer/OnlyIncludeIfParticleOrAncestorNotTraversedComponentOrSubComponentsOf = 1 "Nozzle"

Filter by material:

- sv:Sc/MyScorer/OnlyIncludeIfInMaterial = 2 "Water" "Air"
- sv:Sc/MyScorer/OnlyIncludeIfNotInMaterial = 2 "Water" "Air"

Note that in this case, the material name must exactly match the case defined in Geant4.

To check what materials have been defined, add the parameter:

- i:Ma/Verbosity = 1

Filter on DICOM RT Structure Sets:

A structure set is an extra file in the DICOM directory that provides information on structures such as organs, tumors, PTVs, etc. that have been outlined (contoured) in the planning process. The data is stored as a set of polygons, up to one per slice for each contoured structure. TOPAS can color code DICOM components according to this structure information (see the TsDICOM Component above) and can filter scoring based on these structures:

- sv:Sc/MyScorer/OnlyIncludeIfInRTStructure = 2 "R_LUNG" "L_LUNG"
• If the structure name includes a space, substitute an underscore in the parameter. So, for example, if the structure name is "R LUNG", you should supply the parameter as "R_LUNG".

For Surface Scorers, you can also filter by whether particle is going "In" or "Out" of scoring surface. Omit this filter to allow either option.
  • s:Sc/MyScorer/OnlyIncludeParticlesGoing = "in"

You may specify more than one filter. For example, to score protons with initial KE over 100 MeV:
  • sv:Sc/MyScorer/OnlyIncludeParticlesNamed = 1 "proton"
  • d:Sc/MyScorer/OnlyIncludeParticlesWithInitialKEAbove = 100. MeV # minimum energy

You can invert the results of all previous filters. The following would score only particles that are Not protons with initial KE over 100 MeV:
  • sv:Sc/MyScorer/OnlyIncludeParticlesNamed = 2 "proton" "neutron"
  • d:Sc/MyScorer/OnlyIncludeParticlesWithInitialKEAbove = 100. MeV # minimum energy
  • b:Sc/MyScorer/InvertFilter = "True"

Any filter property can be set by time features if you wish, to produce time-dependent filtering.
Output Specification

Scored quantities can be output to simple files (csv or binary formats), data files for use in analysis systems (Root or Xml format) or to a DICOM file. There are also options to directly produce Volume Histograms (such as DVH).

To specify output file name:
- $s$:Sc/MyScorer/OutputFile = "myOutputFileName" # if null, use scorer name, e.g. "MyScorer"

Note that this can be more than just a file name - it can include a relative or absolute file path, as in:
- $s$:Sc/MyScorer/OutputFile = ".../myOutputFileName" # one directory above current directory
- $s$:Sc/MyScorer/OutputFile = "~/SomeSubdirectory/myOutputFileName"

To specify output file type for all except the phase space scorer:
- $s$:Sc/MyScorer/OutputType = "csv" # "csv", "binary", "Root", "Xml" or "DICOM"

For binary output of 3D data, such as from scoring in a water phantom or a patient, the following table shows the correspondence between TOPAS divisions and common 3D data viewing applications:

<table>
<thead>
<tr>
<th>TOPAS</th>
<th>fNi (X,R,R Bins)</th>
<th>fNj (Y,Phi,Phi Bins)</th>
<th>fNk (Z, Z, Theta) Bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageJ</td>
<td>Width</td>
<td>Height</td>
<td>Images</td>
</tr>
<tr>
<td>ParaView</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Amide</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Numpy</td>
<td>my3d = fromfile('topas_binary').reshape(fNk,fNj,fNi)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DICOM export is handled through the package GDCM, which is pre-built into TOPAS. GDCM is further described at: [http://gdcm.sourceforge.net](http://gdcm.sourceforge.net)

DICOM output is in the form of a single dcm file that contains DICOM header information (voxel size, spacing, etc.) and then a block of binary image data representing the 3D output.

We use the DICOM output format called "MONOCHROME2". Each pixel is represented by a 16 or 32 bit integer.

- 16 bit is the default.
- For 32 bit, specify:
  - $b$:Sc/MyScorer/DICOMOutput32BitsPerPixel = "True"

Because TOPAS accumulates all scores as doubles, you must tell TOPAS now to convert these double values to the DICOM 16bit or 32bit integers. You do so by providing a scaling factor:

- $u$:Sc/MyScorer/DICOMOutputScaleFactor = 1.0E12

The appropriate value depends on your specific setup. You need a factor that will make your expected double values fit into an appropriate range of 16bit or 32bit integers:

- 16bit integers are 0 through 65,536
- 32bit integers are 0 through 4,294,967,296
- For example, if you expect to score values that are only on the order of 1.0E-9, and you are using 16bit, an output scale factor of 1.0E12 will give you reasonable integers on order of 1000

"Root" and "Xml" will generate histogram files. Specify the binning of the scored quantity as follows:

- $i$:Sc/MyScorer/HistogramBins = 100 # number of bins
- $d$:Sc/MyScorer/HistogramMin = 0. MeV # with unit appropriate to scored quantity
- $d$:Sc/MyScorer/HistogramMax = 100. MeV # with unit appropriate to scored quantity
Histograms will be either 1 or 2D depending on how the scoring geometry is divided or the energy is binned (see below for energy binning).

- If the geometry is undivided and there is no energy binning, a 1D histogram is produced.
- If the geometry is undivided and there is energy binning, the second histogram axes will be energy.
- If the geometry is divided, it can only be divided in one dimension (such as either X, Y or Z for box geometries) and there can be no energy binning. The second histogram axes will be the axes of the geometry division.

All histogram output is combined into a single file, such as topas.root or topas.xml. The histogram file name can be adjusted by:

- s:Sc/HistogramFileName = "topas" # name for root, hbook or xml output files

Physicists often report the quality of a treatment plan by showing Dose Volume Histograms (DVH). Such histograms represent what percentage of a given structure has received a given Dose.

- In a differential DVH, the bin value indicates what percentage of the structure received the given dose.
- In a cumulative DVH, the bin value indicates what percentage of the structure received at least the given dose (the zeroth bin will always have a value of 1, since all bins receive at least zero dose).
- If you combine TOPAS DVH options with the filtering option OnlyIncludeIfInRTStructure (see details under Filtering Scorers), you can generate a DVH for a specific contoured structure (such as DVH to R_LUNG).

TOPAS can generate a Volume Histogram for any scored quantity, not just Dose. Just set the scorer's report parameter to include either "DifferentialVolumeHistogram" or "CumulativeVolumeHistogram", as in:

- sv:Sc/DoseAtPhantom/Report = 1 "CumulativeVolumeHistogram"

You cannot specify both types of volume histograms in a single scorer, but you can specify other reporting options, such as the following, which will give a basic histogram of Sum and Mean, plus a CumulativeVolumeHistogram:

- sv:Sc/DoseAtPhantom/Report = 3 "Sum" "Mean" "CumulativeVolumeHistogram"

As with any histogram, you also need to specify HistogramBins, HistogramMin and HistogramMax. For an example, see:

- examples/Scoring/DoseVolumeHistogram.txt

If your results has a 1 in the first bin and zero in the other bins, it probably means your HistogramMax was set too high, and thus none of the voxels had enough dose to get beyond the zeroth bin.

To specify output file type for the phase space scorer:

- s:Sc/MyScorer/OutputType = "ASCII" # "ASCII" or "Binary"

ASCII format has the advantage that it is human-readable text. Binary format has the advantage that it is much more compact, hence suitable for large files.

By default, output will occur just once, after the entire session. But if you are using Time Features and wish to have separate output from specific runs:

- b:Sc/MyScorer/OutputAfterRun = "True" # set True to trigger output of scorer after this run

If this is always set False, or not defined, we just output at the end of the session. If this is always set True, we output after every run. If this is set from time features, then whenever we find it set to True, we output after that run.
Additional output control options:

- \( b:Sc/MyScorer/OutputToConsole = "True" \) # control whether output is also dumped to console
- \( s:Sc/MyScorer/IfOutputFileAlreadyExists = "Increment" \) # "Exit", "Overwrite" or "Increment"

We keep name and type separate in the above so that one can do things like change all output from csv to AIDA by just changing a single parameter (by setting many OutputType params = to a common relative string param).

**Binning in the Dividable Components, TsBox, TsCylinder and TsSphere**

If the relevant component is one of the dividable ones, TsBox, TsCylinder or TsSphere, you have many options to bin the score.

By default, binning will match the divisions of the volume. So if you have divided the component, the score will be divided in the same manner.

You are free to specify some other binning.

- If solid is a TsBox, you can specify binning in x, y and z:
  - \( i:Sc/MyScorer/XBins = 512 \)
  - \( i:Sc/MyScorer/YBins = 512 \)
  - \( i:Sc/MyScorer/ZBins = 256 \)

- If solid is a TsCylinder, you can specify binning in r, phi and z:
  - \( i:Sc/MyScorer/ RBins = 100 \)
  - \( i:Sc/MyScorer/PhiBins = 20 \)
  - \( i:Sc/MyScorer/ZBins = 1 \)

- If solid is a TsSphere, you can specify binning in r, phi and theta:
  - \( i:Sc/MyScorer/ RBins = 20 \)
  - \( i:Sc/MyScorer/PhiBins = 20 \)
  - \( i:Sc/MyScorer/ThetaBins = 1 \)

Behind the scenes, TOPAS uses Geant4’s parallel worlds system to support this binning flexibility. When scoring binning is different from the component's natural binning, TOPAS actually scores in a parallel world copy of the component. This is all done automatically.

**Binning by Energy**

Any scorer can be binned by incident particle kinetic energy, the energy of the particle or its ancestor when it first hit the scoring component, by including the following:

- \( i:Sc/MyScorer/ EBins = 10 \) # defaults to 1, that is, un-binned
- \( d:Sc/MyScorer/EBinMin = 0. \text{ MeV} \) # defaults to zero
- \( d:Sc/MyScorer/EBinMax = 100. \text{ MeV} \) # must be specified if EBins is greater than 1

The output will include three extra bins, one for underflow (energy < EBinMin), one for overflow (energy > EBinMax) and one for the case where there is no incident track (the primary particle was created already inside the scoring component, so it was never incident upon the scoring component).

**Binning by Time**

Any scorer can be binned by time-of-flight, the elapsed time since the history was generated (in Geant4 this is called "global time"):

- \( i:Sc/MyScorer/ TimeBins = 10 \) # defaults to 0, that is, un-binned
- \( d:Sc/MyScorer/TimeBinMin = 0. \text{ ns} \) # defaults to zero
The output will include two extra bins, one for underflow (time < TimeBinMin) and one for overflow (time > TimeBinMax). Note that this time-of-flight is not the same as the TOPAS time feature time. To split results based on that TOPAS time, see the next section. When radioactive decay is present, some very large times can occur, as decay may be delayed for hours or days. Thus it is not unusual to have some times exceed the TimeBinMax. To get an interesting report on what particles and processes exceed TimeBinMax, set:

- Ts/TrackingVerbosity > 0

### Splitting a Scorer Under Control of a Time Feature

To split a scorer into separate scorers depending on the current value of any selected time feature:

- s:Sc/MyScorer/SplitByTimeFeature = some_time_feature_name

If the time feature is a Step function, one split scorer is made for each of the time feature’s values. If the time feature is a Continuous function, another parameter is expected to specify split values. This will be either a dimensioned double vector, unitless vector or integer vector, depending on the type of controlling time feature, such as:

- dv:Sc/DoseAtPhantom/SplitByTimeFeatureValues = 5 0. 90. 180. 270. 360. deg

#### Example 1 - Splitting under control of a Step Time Feature:

To split up a 4D CT simulation's dose output depending on the CT time slice, where the CT time slice is controlled by:

- s:Tf/ImageName/Function = "Step"
- sv:Tf/ImageName/Values = 3 "image1" "image2" "image3"

The following will make the scorer DoseAtPhantom split by current value of Tf/ImageName/Value:

- s:Sc/DoseAtPhantom/SplitByTimeFeature = "ImageName"

Creating one scorer for each ImageName:

- DoseAtPhantom-image1
- DoseAtPhantom-image2
- DoseAtPhantom-image3

#### Example 2 - Splitting under control of a Continuous Time Feature:

To split up a simulation's dose output depending on the position of a propeller, where the propeller position is controlled by:

- s:Tf/PropellerRotation/Function = "Linear deg"

The following will make DoseAtPhantom split by current value of Tf/PropellerRotation/Value:

- s:Sc/DoseAtPhantom/SplitByTimeFeature = "PropellerRotation"
- dv:Sc/DoseAtPhantom/SplitByTimeFeatureValues = 5 0. 90. 180. 270. 360. deg

Creating one scorer for each defined range of PropellerRotation:

- DoseAtPhantom-0.-90.deg
- DoseAtPhantom-90.-180.deg
- DoseAtPhantom-180.-270.deg
- DoseAtPhantom-270.-360.deg

See examples:

- examples/Scoring/SplitByTimeFeature.txt
- examples/DICOM/DoseTo4DCT.txt
Statistical Information

By default, scorers will report the sum of the scored quantity over all histories, but many additional reporting options are available:

- `sv:Sc/MyScorer/Report = 1 "Sum" # One or more of Sum, Mean, Histories, Count_In_Bin, Second_Moment, Variance, Standard_Deviation, Min, Max`

Output columns will be in the same order as the values in the Report parameter.

When there is binning by energy or time, and there is more than one Report option (such as Sum and Mean), the output will be ordered as:

- Sum (underflow), Mean (undeflow), Sum (bin 1), Mean (bin 1), Sum (bin 2), Mean (bin 2), etc.

Histories is the total number of histories that were simulated while this scorer was active (that is, excludes any histories that were produced when this scorer was gated to inactive). Count_In_Bin is the number of histories that contributed to this bin (that is, excludes any histories for which no particles hit this bin).

If only sum is requested, simple accumulation is used. If mean, second moment, variance or standard deviation is requested, accumulation uses the more numerically stable algorithm from:


- for x in data:
  - n = n + 1
  - delta = x - mean
  - mean = mean + delta/n
  - M2 = M2 + delta*(x - mean)
  - sum = n * mean
  - variance = M2/(n - 1)
  - standard deviation = sqrt(variance)

Note that if your geometry has many divisions (such as the 70M voxels of a 512 x 512 x 256 CT), and you ask for Mean, Second_Moment, Variance or Standard_Deviation, you will see a speed penalty. This occurs because any bin that has ever been hit will then have to recalculate its mean or second moment to account for the new history (even if the current history doesn't hit this bin).

TOPAS calculates the variance (and hence the standard deviation) associated with the distribution of the quantity of interest (dose, fluence, etc).

- For the standard deviation of the mean value, divide the standard deviation from TOPAS by the square root of the total number of histories.
- For the standard deviation of the sum, multiply the standard deviation from TOPAS by the square root of the total number of histories.
Making Component's Color Change based on Scoring

You can make TOPAS recolor a component during simulation to reflect a scored value. Using this technique, you can, for example, make a box become darker as it accumulates dose.

- See examples/TimeFeature/Darkening.txt

To activate this feature:

- s:Sc/EnergyInPhantom/ColorBy = "Sum" # sum, mean, histories, standard_deviation, min, max

You must then provide a list of colors, and cutoff values, such as:

- sv:Sc/EnergyInPhantom/ColorNames = 5 "white" "grey240" "grey220" "grey200" "grey180"

In the above example:

- if the total energy is from 0 to 1, the phantom will be colored "White".
- if the total energy is from 1 to 1000, the phantom will be colored "grey240".
- if the total energy is from 1000 to 2000, the phantom will be colored "grey220".
- etc.

This feature must be used in conjunction with Time Features, as the color will only update after each run. And your scorer must be set to output after each run:

- b:Sc/EnergyInPhantom/OutputAfterRun = "True"

This technique does not currently work in the divided components (TsBox, TsCylinder and TsSphere). We will add this capability in a future TOPAS release. For now it only works in simple components made of single Geant4 solids.

Toggling a Scorer Off and On

To turn off a scorer:

- b:Sc/MyScorer/Active = "False" # defaults to "True"

This feature can be combined with boolean time features to produce gated scoring.

If the scorer skipped any values due to being set inactive at any time, the total number of skipped values is written out at in the scoring summary.

Restoring Results from Files

TOPAS provides an option to read back scored values so that you can then redo the scoring output with different options. Set the parameter:

- Ts/RestoreResultsFromFile = "True" # defaults to "False"

With this set, simulation will not be run, but instead the scored values will be restored from the output of previous TOPAS simulations. For each scorer, there must be an appropriate file to read back, specified by name and type:

- s:Sc/MyScorer1/InputFile = "MySavedFileName" # match exact case
- s:Sc/MyScorer1/InputType = "csv"

The file to read back in must contain the appropriate scored quantity, the appropriate binning, and sufficient information to provide the new Report options. So, for example, if you previously scored Sum and Histories, you could now report Sum, Mean, Histories, and a DVH.

This option can also be used to read in binary output and write out csv, or vice versa.
Graphics

You may have zero, one or more graphics active at the same time.
- s:Gr/MyGraphic1/Type = "OpenGL" # OpenGL, HepRep, VRML, DAWN, RayTracer, RayTracerX

Note that the file-based graphics systems, HepRep, VRML and DAWN may not show any image until at least one history is run. We will revisit this issue when we move to the next Geant4 version.

HepRep files are designed to be viewed in a Java application called HepRApp. Details can be found here: http://geant4.slac.stanford.edu/Presentations/vis/G4HepRAppTutorial/G4HepRAppTutorial.html

Unfortunately, HepRApp has not kept up with the latest Java versions. Mac users may find that they can obtain Java 1.6 and then set:
- export JAVA_HOME=`/usr/libexec/java_home -v '1.6*'`
  - meaning set JAVA_HOME to whatever version of Java 1.6 you can find on this system.

Note that graphics can be one of the slowest parts of a simulation, so should be disabled if you are running a long simulation. To disable graphics, do one of the following three things:
- Comment out all of the Gr/*/Type parameters
- or
- Set all Gr/*/Active to "False"
  - or
- Disable graphics entirely, by setting: b:Gr/Enable = "False"

This last option is essential if you want to run on a batch system that does not contain any OpenGL graphics drivers.

File-based graphics systems will also expect a file name:
- s:Gr/MyGraphic1/FileName = "MyFileName" # Defaults to name of view (which for this example is MyGraphic1). Will use this filename plus an "_n" where n increments with each refresh.
  - Due to limitations in Geant4, only affects OpenGL and HepRep. For other cases, the file name is a fixed value, "g4_" followed by a file number.

This can be more than just a file name - it can include a relative or absolute file path, as in:
- s:Gr/MyGraphic1/FileName = ".../MyFileName" # one directory above current directory
- s:Gr/MyGraphic1/FileName = "~/SomeSubdirectory/MyFileName"

Basic options:
- b:Gr/MyGraphic1/IncludeGeometry = "True" # defaults to "True"
- b:Gr/MyGraphic1/IncludeTrajectories = "True" # defaults to "True"
- b:Gr/MyGraphic1/IncludeStepPoints = "True" # Show trajectory step points, defaults to "False"

Colors are defined by specifying their red, green, blue components, each on a scale of 0 to 255, as in:
- iv:Gr/Color/lightblue = 3 175 255 255

By default, trajectories will be drawn as what Geant4 calls "Smooth Trajectories", which means they include additional points to make them curve smoothly in a magnetic field. Geant4 does not actually use these "auxiliary points" in its simulation results, they are just present to make visualization in a field look better. In some cases, Geant4 has trouble handling these auxiliary points, and reports: "!!!!!!!! Filter: auxiliary points are being memory leaked !!!!!"
To work around this, turn off trajectory drawing or tell Geant4 not to making the trajectories smooth:
  • b:Gr/MyGraphic1/UseSmoothTrajectories = "False" # defaults to "True"

You can add axes to the display. Axes lines are colored red for X, green for Y, blue for Z:
  • b:Gr/MyGraphic1/IncludeAxes = "True" # defaults to "False"
  • s:Gr/MyGraphic1/AxesComponent = "World" # Component in which to center the axes. Defaults to World.
  • d:Gr/MyGraphic1/AxesSize = 3. m # size of axes

Note that on most OpenGL graphics systems, the shadowing on the arrowheads allows you to tell whether a given axis is coming towards or away from you.

By default, graphics views will refresh after every run. But you can change this to show each history individually or to accumulate all histories for the entire session (multiple runs) with:
  • s:Gr/RefreshEvery = "History" # "History", "Run" or "Session"

If parallel worlds are present, by default they will be visible. If you instead want to see only the main world, specify:
  • sv:Gr/MyGraphic1/VisibleWorlds = 1 "World" # "World", "All" or one or more specific world names

To turn off a graphic:
  • b:Gr/MyGraphic1/Active = "False" # defaults to "True"

Extra options used by OpenGL:
  • u:Gr/MyGraphic1/Zoom = 2. # increase to zoom in, decrease to zoom out
  • d:Gr/MyGraphic1/Theta = 45. deg # view angle as in /vis/viewer/set/viewpointThetaPhi
  • d:Gr/MyGraphic1/Phi = 45. deg # view angle as in /vis/viewer/set/viewpointThetaPhi
  • u:Gr/MyGraphic1/TransX = 0. # move left or right in the view window
  • d:Gr/MyGraphic1/TransY = 0. # move up or down in the view window
  • s:Gr/MyGraphic1/Projection = "Perspective" # Defaults to "Orthogonal"
  • d:Gr/MyGraphic1/PerspectiveAngle = 10. deg # Increase for stronger perspective effect
  • i:Gr/MyGraphic1/WindowSizeX = 600
  • i:Gr/MyGraphic1/WindowSizeY = 600
  • i:Gr/MyGraphic1/WindowPosX = 0
  • i:Gr/MyGraphic1/WindowPosY = 0
  • b:Gr/MyGraphic1/HiddenLineRemovalForGeometry = "False" # Remove hidden lines from wireframe geometries, like Geant4’s /vis/viewer/set/hiddenEdge
  • b:Gr/MyGraphic1/HiddenLineRemovalForTrajectories = "False" # Remove hidden trajectories lines from within geometries, like Geant4’s /vis/viewer/set/hiddenMarker
  • b:Gr/MyGraphic1/CopyOpenGLToEPS = "True" # saves each view to an eps file

Some views may result in one of the following warning messages from Geant4 Visualization. These messages are just informational and can be safely ignored.
  • "WARNING: Viewpoint direction is very close to the up vector direction. Consider setting the up vector to obtain definable behaviour."
  • "G4PhysicalVolumeSearchScene::FindVolume: Required volume "Phantom3_10x10x1", copy no. 0, found more than once. This function is not smart enough to distinguish identical physical volumes which have different
parentage. It is tricky to specify in general. This function gives you access to the first occurrence only.

To create movies, Zoom, Theta, Phi, TransX, TransY, Projection and PerspectiveAngle can be controlled by time features.

Trajectory Coloring:
- \text{Gr/MyGraphic1/ColorBy = "Charge" # "Charge", "ParticleType", "OriginComponent", "Energy", "Momentum", "Generation", "CreatorProcess"}

For ColorBy = Charge, trajectories default to red, greed, blue for negative, neutral and positive. You can override these defaults with:
- \text{sv:Gr/MyGraphic1/ColorByChargeColors = 3 "blue" "green" "red" # colors for neg, neutral, pos}

For ColorBy = ParticleType, colors are Geant4 defaults:
- gamma = green
- e- = red
- e+ = blue
- pi-+ = magenta
- proton = cyan
- neutron = yellow
- other = grey
You can override these settings with:
- \text{sv:Gr/MyGraphic1/ColorByParticleTypeNames = 4 "e-" "gamma" "proton" "neutron" # any number of particle names}
  - Particle names are described above under Particle Sources...Particle Names.
- \text{sv:Gr/MyGraphic1/ColorByParticleTypeColors = 4 "red" "green" "blue" "yellow" # for each particle type above. All other particles will be set to grey.}

For ColorBy = OriginVolume, trajectories are grey unless they come from a named volume in:
- \text{sv:Gr/MyGraphic1/ColorByOriginVolumeNames = 1 "Propeller20/Leaf" # one or more volume}
- \text{sv:Gr/MyGraphic1/ColorByOriginVolumeColors = 1 "red" # one color for each name above}

For ColorBy = OriginComponent, trajectories are grey unless they come from a named component in:
- \text{sv:Gr/MyGraphic1/ColorByOriginComponentNames = 1 "jaws" # one or more component names}
- \text{sv:Gr/MyGraphic1/ColorByOriginComponentColors = 1 "red" # one color for each name above}

For ColorBy = ColorByOriginComponentOrSubComponentOf, trajectories are grey unless they come from a named component or any of its subcomponents in:
- \text{sv:Gr/MyGraphic1/ColorByOriginComponentNames = 1 "Nozzle" # one or more components}
- \text{sv:Gr/MyGraphic1/ColorByOriginComponentColors = 1 "red" # one color for each name above}

For ColorBy = Energy:
- \text{dv:Gr/MyGraphic1/ColorByEnergyRanges = 3 1. 4. 8. MeV # limits of energy ranges}
- \text{sv:Gr/MyGraphic1/ColorByEnergyColors = 4 "red green blue yellow" # one for every energy interval that is defined by those ranges - one more value than number of ranges since includes less than first range value and greater than first range value}
For ColorBy = Momentum:
  * dv:Gr/MyGraphic1/ColorByMomentumRanges = 3 1. 4. 8. MeV # limits of momentum ranges
  * sv:Gr/MyGraphic1/ColorByMomentumColors = 4 "red" "green" "blue" "yellow" # one for every energy interval that is defined by those ranges - one more value than number of ranges since includes less than first range value and greater than first range value

For ColorBy = Generation:
  * sv:Gr/MyGraphic1/ColorByGenerationColors = 2 "red" "green" # colors for primary and secondaries

For ColorBy = CreatorProcess:
  * sv:Gr/MyGraphic1/ColorByCreatorProcessNames = 5 "eBrem" "annihil" "Decay" "eloni" "hloni" # one or more process name
  * sv:Gr/MyGraphic1/ColorByCreatorProcessColors = 5 "red" "green" "blue" "yellow" "magenta" # one for every process name

To control how often graphics refresh (applies globally to all graphics views):
  * s:Gr/RefreshEvery = "Run" # "History", "Run" or "Session"

To filter what trajectories will be in the graphics, use similar syntax to that used for filtering of scoring and particle source (apply globally to all graphics views):
  * sv:Gr/OnlyIncludeParticlesNamed = 2 "proton" "neutron" # one or more particle names
    o Particle names are as described above under Particle Sources...Particle Names.
  * sv:Gr/OnlyIncludeParticlesCharged = 1 "negative" # one or more "positive", "negative" or "neutral"
  * sv:Gr/OnlyIncludeParticlesFromVolume = 1 "Propeller20/Leaf" # one or more volume
  * sv:Gr/OnlyIncludeParticlesFromComponent = 1 "Jaws" # one or more component
  * sv:Gr/OnlyIncludeParticlesFromComponentOrSubComponentsOf = 1 "Nozzle" one or more
  * d:Gr/OnlyIncludeParticlesWithInitialKEBelow = 1. MeV # maximum energy
  * d:Gr/OnlyIncludeParticlesWithInitialKEAbove = 10. MeV # minimum energy
  * d:Gr/OnlyIncludeParticlesWithInitialMomentumBelow = 1. MeV # maximum momentum
  * d:Gr/OnlyIncludeParticlesWithInitialMomentumAbove = 10. MeV # minimum momentum
  * sv:Gr/OnlyIncludeParticlesFromProcess = 1 "hloni" # one or more process name

Note that the following three filters may cause a crash if the particle origin is at the world boundary:
  * sv:Gr/OnlyIncludeParticlesFromVolume
  * sv:Gr/OnlyIncludeParticlesFromComponent
  * sv:Gr/OnlyIncludeParticlesFromComponentOrSubComponentsOf

We will study this issue again when we move to the next Geant4 version.

Visualization control for a specific component is done as part of the Ge/ parameters for that component rather than in the Gr/ parameters:
  * s:Ge/MyComponent/Color = "red"
  * s:Ge/MyComponent/DrawingStyle = "Solid" # "Solid", "Wireframe" or "FullWireFrame". FullWireFrame includes drawing of additional edge lines that Geant4 calls "soft edges" - on many graphics devices WireFrame and FullWireFrame give the same result
  * i:Ge/MyComponent/VisSegsPerCircle = 100 # Number of line segments to use to approximate a circle, defaults to 24. Set to a larger number if you want a smoother curve
  * b:Ge/MyComponent/Invisible = "True" # defaults to False meaning visible
We sometimes see error messages from visualization of the following form:

- **G4PhysicalVolumeSearchScene::FindVolume:**
  Required volume "PhantomCentralDose_1x1x40", copy no. 0, found more than once…

Such messages can be ignored. They do not affect the simulation results. We will revisit how to solve these error messages once we move to the next Geant4 version.
Variance Reduction

Please note that Variance Reduction is highly dependent on your specific geometry. Approach these features with caution and test all variance reduced setups against an equivalent setup without variance reduction.

You should also review the Geant4 document that describes which cases are problematic: https://geant4.web.cern.ch/geant4/collaboration/working_groups/geometry/biasing/BiasScoreUseCases.html

To enable the particle split applied to protons:

- b:Vr/UseVarianceReduction = "true"
- b:Vr/ParticleSplit/Active = "true"
- sv:Vr/ParticleSplit/ParticleName = 1 "proton"

To Specify the Split Geometry

The geometry for variance reduction must be a parallel one. The type of component can be any standard solid (generic solid) defined in the geometry section. The geometry must consist of a geometrical component with a set of geometrical sub-components as daughters. The sub-components must be located in such a way that the borders in between coincide. The split process or Russian roulette will occur at these borders. In the next figure a simple scheme is shown.

Time features can be used to move or rotate the component or sub-components. But there is a restriction: the implementation of VRT does not allow to change the dimensions of the component and sub-components.

To set the geometry for VRT:

- s:Vr/ParticleSplit/Component = "MyComponent"
- sv:Vr/ParticleSplit/SubComponents = n "MySubComp_1" ... "MySubComp_n"

To Define the Splitting Technique

There is three variance reduction techniques available: GeometricalParticleSplit, ImportanceSampling and WeightWindow
To chose a technique, use for example:

- s:Vr/ParticleSplit/Type = "GeometricalParticleSplit"
**GeometricalParticleSplit**


This technique was designed for heavy charged particles. In this implementation, you must to specify if the beam entering into the sub-component has cylindrical symmetry or not, that is because the particles can be or not randomly redistributed around the SplitAxis. The Russian Roulette is applied in direction. That is, at the split plane and prior to be split, the particle is subject to Russian Roulette if its direction does not point towards a Region of Interest (ROI). Then the radius of the ROI and its position on the SplitAxis must to be defined too. Further, Russian Roulette can be turned on/off at specific surfaces between sub-components.

- `s:Vr/ParticleSplit/Type = "GeometricalParticleSplit"
- `s:Vr/ParticleSplit/SplitAxis = "zaxis"
- `d:Vr/ParticleSplit/RussianRoulette/ROIRadius = 7.8 cm`
- `d:Vr/ParticleSplit/RussianRoulette/ROITrans = 10 cm`
- `bv:Vr/ParticleSplit/RussianRoulette = 2 "false" "true"

To set if the region at each sub-component, is symmetric or not and to define the corresponding split number:
- `bv:Vr/ParticleSplit/Symmetric = 2 "false" "true"
- `uv:Vr/ParticleSplit/SplitNumber = 2 8 8`

In addition for this technique, geometrical Russian roulette will be played if a particle outs to the component or to the world in a scheme similar to the Geant4 importance sampling technique described below.

**ImportanceSampling**

In this technique, an importance value is assigned to each sub-component. If a particle goes from a sub-component with an importance value lower than importance value of the sub-component in which the particle enters, then the particle is split. If the opposite happens, then Russian roulette is played. By default an importance value of 1 is automatically assigned to the component that contains the sub-components and to the world.
A thickness of the sub-components of about the mean free path of the physical process to be biased is desirable. For the implementation, the importance values are defined by hand. For example, to split the particles by a factor of 2 between subsequent sub-components, one must to define:

- \( s:Vr/ParticleSplit/Type = \text{“ImportanceSampling”} \)
- \( uv:Vr/ParticleSplit/ImportanceValues = 5 \ 1 \ 2 \ 4 \ 8 \ 16 \)

**WeightWindow**

In this technique, the split process or Russian roulette will be applied depending on the statistical weight of the particle. Every time that a particle crosses from a sub-component to the next one, the statistical weight is evaluated. Particles with statistical weights larger than a lower weight bound and lower than an upper weight bound will be tracked normally. Particles with statistical weights lower than a lower weight bound will be subject to Russian roulette. If survives the particle is tracking normally but its statistical weight is increased. Particles with statistical weights larger than an upper weight bound will be split, and low statistical weights are assigned to the new particles. The split number and Russian roulette criteria are internally calculated from a energy map, a weight map, a upper limit factor and a survival factor. In simple geometries the maps can be input by hand.

The user must to provide a double vector with upper energy bounds and an unitless vector with lower weight bounds for every sub-component: WeightMap and EnergyMap. The inverse of a value named MaximumSplitNumber (100 as default) is used to specify the minimum survival probability to be used in Russian roulette. The parameter PlaceOfAction states if the split process (or Russian roulette) will occurs at the boundary of the sub-components, at physics interaction or at both.

The follow configuration is equivalent to importance sampling with importance generator of 2:

- \( s:Vr/ParticleSplit/Type = \text{“WeightWindow”} \)
- \( uv:Vr/ParticleSplit/WeightMap = 4 \ 1 \ 1 \ 0.125 \ 0.0615 \)
- \( dv:Vr/ParticleSplit/EnergyMap = 4 \ 1 \ 1 \ 1 \ 1 \ \text{GeV} \)
- \( u:Vr/ParticleSplit/UpperLimitFactor = 1 \)
• \( u:Vr/ParticleSplit/SurvivalFactor = 1 \)
• \( i:Vr/ParticleSplit/MaximumSplitNumber = 100 \)
• \( s:Vr/ParticleSplit/PlaceOfAction = \text{"onBoundary"} \)
• #Others options of PlaceOfAction: \text{"OnCollision" and "OnBoundaryAndCollision"}

Tracking only specific particles
In this option the particles are eliminated just after they were created. The user can choose which particles will be tracked in all components. Nevertheless, user can specifies if any component is going to be avoid. That is, all particles are tracked in such components. This option can be useful when the contribution of certain particles is negligible to the final scored quantity. But it must to be used with careful. This option is not a variance reduction.
To eliminate particles different than protons and electrons in all components but in component named water phantom:
• \( b:Vr/KillOtherParticles/Active = \text{"True"} \)
• \( sv:Vr/KillOtherParticles/HaveNoEffectInComponentsNamed = 1 \text{ "WaterPhantom"} \)
• \( sv:Vr/KillOtherParticles/OnlyTrackParticlesNamed = 2 \text{ "proton" } \text{ "e-"} \)

A Complete Example for GeometricalParticleSplit
Here three components are created as usually and set to parallel. The subComponent1 shares its distal boundary with the proximal boundary of subComponent2.

A time feature, see next section
• \( s:Tf/MoveOnZ/Function = \text{"Step"} \)
• \( dv:Tf/MoveOnZ/Times = 10 20 30 40 50 60 70 80 90 100 \text{ ms} \)
• \( dv:Tf/MoveOnZ/Values = 10 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 \text{ cm} \)

The geometry. The Component here is named VrtParallelWorld
• \( s:Ge/VrtParallelWorld/Type = \text{"G4Box"} \)
• \( s:Ge/VrtParallelWorld/Parent = \text{"World"} \)
• \( d:Ge/VrtParallelWorld/HLX = 10 \text{ cm} \)
• \( d:Ge/VrtParallelWorld/HLY = 10 \text{ cm} \)
• \( d:Ge/VrtParallelWorld/HLZ = 10 \text{ cm} \)
• \( d:Ge/VrtParallelWorld/TransX = 0 \text{ cm} \)
• \( d:Ge/VrtParallelWorld/TransY = 0 \text{ cm} \)
• \( d:Ge/VrtParallelWorld/TransZ = \text{Tf/MoveOnZ/Value cm} \)
• \( d:Ge/VrtParallelWorld/RotX = 0 \text{ deg} \)
• \( d:Ge/VrtParallelWorld/RotY = 0 \text{ deg} \)
• \( d:Ge/VrtParallelWorld/RotZ = 0 \text{ deg} \)
• \( b:Ge/VrtParallelWorld/IsParallel = \text{"True"} \)  # Must to be a parallel geometry
• # The first subComponent
• \( s:Ge/subComponent1/Type = \text{"G4Box"} \)
• \( s:Ge/subComponent1/Parent = \text{"VrtParallelWorld"} \)
• \( d:Ge/subComponent1/HLX = 3 \text{ cm} \)
• \( d:Ge/subComponent1/HLY = 3 \text{ cm} \)
• \( d:Ge/subComponent1/HLZ = 2 \text{ cm} \)
• \( d:Ge/subComponent1/TransX = 0 \text{ cm} \)
• \( d:Ge/subComponent1/TransY = 0 \text{ cm} \)
• \( d:Ge/subComponent1/TransZ = \text{Tf/MoveOnZ/Value cm} \)
• \( d:Ge/subComponent1/RotX = 0 \text{ deg} \)
Active the Variance Reduction for protons
- b:Vr/UseVarianceReduction = "True"
- b:Vr/ParticleSplit/Active = "True"
- sv:Vr/ParticleSplit/ParticleName = 1 "proton"

Assign the corresponding parallel geometry
- s:Vr/ParticleSplit/Component = "VrtParallelWorld"
- sv:Vr/ParticleSplit/SubComponents = 2 "subComponent1" "subComponent2"

Type of VRT: GeometricalParticleSplit
- s:Vr/ParticleSplit/Type = "GeometricalParticleSplit"

The split number per sub-component
- uv:Vr/ParticleSplit/SplitNumber = 2 8 8

The split axis with respect to VrtParallelWorld
- s:Vr/ParticleSplit/SplitAxis = "zaxis"

Let us assume that first sub-component is not symmetric but second is symmetric
- bv:Vr/ParticleSplit/Symmetric = 2 "False" "True"

In which sub-components to apply direction-based Russian roulette
- bv:Vr/ParticleSplit/RussianRoulette = 2 "True" "True"
- d:Vr/ParticleSplit/RussianRoulette/ROIRadius = 1.5 cm
- d:Vr/ParticleSplit/RussianRoulette/ROITrans = 5.0 cm

Only protons are tracked in components different than component named Phantom
- b:Vr/KillOtherParticles/Active = "True"
- sv:Vr/KillOtherParticles/HaveNoEffectInComponentsNamed = 1 "Phantom"
- sv:Vr/KillOtherParticles/OnlyTrackParticlesNamed = 1 "proton"
Secondary biasing
The split of secondary particles created after an electromagnetic interaction is also supported. A common example is the split of secondary photons created in bremsstrahlung process for conventional radiotherapy simulations. This variance reduction works per electromagnetic physical process per region. A region allows to multiple components to have specific production cuts. This is useful in complex geometry setups to improve the computational speed by assigning high production cuts in regions where detailed simulation is not important. To assign a region to a component, user can proceed as follows

- s:Ge/MyComponent/AssignToRegionNamed = “MyRegion”
The region MyRegion is automatically created if it not exists. The next step is to set secondary biasing option

- \texttt{s:Vr/ParticleSplit/Type} = “SecondaryBiasing”

After that three vectors must to be defined. One with the name of the electromagnetic processes, one with the split number for each process and one with the maximum energies for each processes. The biased particles with energies larger than this values are subject to Russian roulette.

- \texttt{sv:Vr/ParticleSplit/ForRegion/MyRegion/ProcessesNamed} = 2 “eBrem” “compt”
- \texttt{uv:Vr/ParticleSplit/ForRegion/MyRegion/SplitNumber} = 2 100 10
- \texttt{dv:Vr/ParticleSplit/ForRegion/MyRegion/MaximumEnergy} = 2 6.0 0.511 MeV

If suitable, further CPU time can be saved with a directional Russian roulette for secondary particles created with split (analogous to Geometrical particle split). For that, a reference component must to be chosen.

- \texttt{s:Vr/ParticleSplit/ReferenceComponent} = “Target”

And the directional filter is applied:

- \texttt{dv:Vr/ParticleSplit/ForRegion/MyRegion/DirectionalSplitLimits} = 2 1.0 1.0 m
- \texttt{dv:Vr/ParticleSplit/ForRegion/MyRegion/DirectionalSplitRadius} = 2 5.0 5.0 cm

Figure: Biasing particle of secondary photons after a bremsstrahlung process. On the left no directional Russian roulette, on the right a directional Russian roulette is applied.
Time Features

While the repeatability requirements of the TOPAS parameter system require that parameter definitions be well specified, there is still a need to define time-dependent behaviors (such as motion, beam current modulation, starting and stopping of scoring activities). The TOPAS Time Feature system allow such time-dependence to be specified in a manner that is both flexible and repeatable.

A Time Feature is a set of parameters that ultimately describes the change of a time feature Value. You provide parameters that define the time function, such as a linear change over time. TOPAS automatically creates a Value parameter for this function (a parameter you don’t define). TOPAS continually updates this Value parameter to the appropriate value for a given time.

(If you're doing complex things with parameter file chains, you may want to know where in chain this automatically added Tf/.../Value parameter goes: the answer is that it goes into the same virtual file as the Tf/.../Function parameter.)

In addition to specifying the time features, you need to specify the overall time sequence. For this, see the later section, "TOPAS Overall Control."

First Example

Here is an example, a Time Feature called ArmRot that describes a constant rotation:

• s:Tf/ArmRot/Function = "Linear deg"
• d:Tf/ArmRot/Rate = 2. deg/ms
• d:Tf/ArmRot/StartValue = 0.0 deg
• d:Tf/ArmRot/RepetitionInterval = 50. ms

TOPAS automatically creates another parameter:

• s:Tf/ArmRot/Value

and updates this parameter to the appropriate value for a given time.

You can then use this value to affect a component position through a statement such as:

• Ge/Arm/RotX = 0. deg + Tf/ArmRot/Value

Linear, Sine, Cosine and Sqrt Functions can Control Dimensioned Double or Unitless Values

For Dimensioned Double or Unitless values, the Function can be any one of:

• Linear: Value at any given time is StartValue + Rate * Time
• Sine: Value at any given time is Sine ( StartValue + Rate * Time )
• Cosine: Value at any given time is Cosine ( StartValue + Rate * Time )
• Sqrt: Value at any given time is Sqrt ( StartValue + Rate * Time )

If the value is Dimensioned Double, you must also provide a unit, such as the "deg" in:

• s:Tf/ArmRot/Function = "Linear deg"

You must provide appropriate StartValue and Rate parameters, such as:

• d:Tf/ArmRot/Rate = 2. deg/ms
• d:Tf/ArmRot/StartValue = 0.0 deg

You must also provide a RepetitionInterval, the time interval after which the function will return to the StartValue.
Step Functions can Control Any Kind of Value - Dimensioned Double, Unitless, Integer, Boolean or String

With a Step function, you define a set of times at which to change value, and a value for each of those times. The first value you provide specifies the starting value (the value at time zero).

Here is an example of a Step time feature that controls a String:

- s:Tf/ImageName/Function = "Step"
- dv:Tf/ImageName/Times = 3 10 20 30 ms
- sv:Tf/ImageName/Values = 3 "lung-1" "lung-2" "lung-3"
  - The first value is used for times 0 to 10 ms.
  - The second value is used for times 10 to 20 ms.
  - The third value is used for times 20 to 30 ms.
  - After 30 ms, the value cycles back to the first value.

Note that whereas continuous functions (Linear, Sine, Cosine and Sqrt) include a RepetitionInterval, Step Functions do not. They just cycle back to the first Value after the last of the Times is reached.

Here is an example of a Step time feature that controls a Boolean:

- s:Tf/ScoringOnOff/Function="Step"
- dv:Tf/ScoringOnOff/Times =10 10 20 30 40 50 60 70 80 90 100 ms
- bv:Tf/ScoringOnOff/Values=10 "true" "false" "true" "false" "true" "false" "true" "false" "true"

Note that:
- Tf/.../Times is always of type "dv:" and has unit of time.
- Tf/.../Values is a vector of whatever type the function controls.

Any individual member of the Values parameter vector can itself be a parameter, such as:

- bv:Tf/ScoringOnOff/Values=4 "true" "false" Some_Other_Boolean_Parameter_Name "false"

Combining Time Features for Complex Behaviors

You can add or multiply time feature Value parameters just as you can add or multiply any other kind of parameter. For example, here is how the number of histories in a run can be controlled by both a beam current and a beam weight:

- s:Tf/BeamCurrent/Function="Step"
- dv:Tf/BeamCurrent/Times = 1 10 ms
- iv:Tf/BeamCurrent/Values = 1 10

- s:Tf/BeamWeight/Function="Step"
- dv:Tf/BeamWeight/Times =10 1 2 3 4 5 6 7 8 9 10 ms
- iv:Tf/BeamWeight/Values =10 1 1 1 2 2 2 2 4 4 4

- i:Tf/BCM/Value = Tf/BeamWeight/Value * Tf/BeamCurrent/Value
- So/Default/NumberOfHistoriesInRun = Tf/BCMValue

By combining Step time features with other time features, you can control complex sequences. The following from examples/SpecialComponents/PurgingMagnet_move.txt moves a box first in one direction and then in the other:

- s:Tf/BackForward/Function = "Step"
- dv:Tf/BackForward/Times = 2 100.0 200.0 ms
Some complex examples of time features are in examples/Nozzle. While we have had examples of double scattering and pencil beam scanning for some time, those examples have included proprietary IBA information, so could not be generally shared. examples/Nozzle provides examples that have no vendor confidential information.

- RasterScanningPattern.txt: Time Features for controlling the dipole magnets are implemented. The time varying magnet will scan rectangle fields in a raster pattern.
- ScanningStationaryTarget.txt: In addition to RasterScanningPattern.txt, a water phantom including a plane target is added.
- ScanningTargetMovingHorizontal.txt: The perpendicularly moving target is defined. In order to make protons follow the moving target, compensated Time Features for the dipole magnets are implemented. The execution of this file will show the moving target in horizontal direction and the proton beams tracking the moving target.
- ScanningTargetMovingInDepth.txt: To trace the target moving along with the depth, the changes of proton’s incident energy should be synchronized with the motion.
- ConstantsForNozzles.txt: Common parameters used by non NDA TOPAS nozzles.
- ScanningNozzle.txt: All geometry for the scanning nozzle is defined. The nozzle consists of magnet systems, for example, two quadrupole magnets and two dipole magnets in helium gas filled beam pipe and various monitoring chambers. Magnet fields are set to zero in this parameter file.
- ScatteringNozzle.txt: All geometry for the scattering nozzle is defined.
- ScatteringNozzle_run.txt: Range Modulator Wheel rotates over time and scatterers move in and out of the beam.

Take care when mixing Phase Space Sources with Time Features. While TOPAS can save the current TOPAS time to a phase space file, this time is not automatically applied when reading particles back in from phase space. Thus, if you want to correct replay source particles that were recorded with time features, it is your responsibility to apply the identical time features during the play back simulation. Some additional notes:

- Do not attempt to change the name of the phase space file over time. Save and replay all particles from a single phase space file.
- Do not use Random time mode. The randomly generated times during playback will not necessarily match the randomly generated times that were saved to the phase space. Only use Fixed or Sequential mode.

A future version of TOPAS will provide more tools to synchronize and check playback time features.
TOPAS Overall Control

If you do nothing special, TOPAS will do a single run with no time variation. We call this "Fixed Time Mode". Other available modes are "Sequential" and "Random".

Fixed Time Mode

To run in Fixed Time Mode, just set your source’s NumberOfHistoriesInRun, as in:

- i:So/Default/NumberOfHistoriesInRun = 100

If your parameter files include time features, they will all be evaluated with time equals zero. To instead have them evaluated at a different fixed time, specify TimelineStart, as in:

- d:Tf/TimelineStart = 10. s # defaults to zero

If you have more than one source, the run will continue until all sources have run all of their histories. For each Geant4 "beamOn", each source will get called, but only those that have more histories left to produce will actually produce any.

Sequential Time Mode

To have TOPAS do several runs at fixed time intervals, specify the start time, end time and number of sequential times, as in:

- d:Tf/TimelineStart = 0. s # defaults to zero
- d:Tf/TimelineEnd = 10. s # must be larger than TimelineStart
- i:Tf/NumberOfSequentialTimes = 100 # defaults to 1

TOPAS will divide the overall time, TimelineEnd - TimelineStart, by NumberOfSequentialTimes and perform runs at each of these intervals.

- The first run will be at time = TimelineStart.
- The last run will be at time = TimelineEnd minus one interval. That is, TOPAS will stop Before it reaches TimelineEnd.

So, in the example above:

- Run 0 will have Time = 0. s
- Run 1 will have Time = 0.1 s
- ...
- Run 99 will have Time = 9.9 s

At each of these intervals, your source will generate your indicated NumberOfHistoriesInRun:

- i:So/Default/NumberOfHistoriesInRun = 10

So, for example, if you have 100 intervals, and NumberOfHistoriesInRun = 10, you will generate a total of 100 x 10 = 1000 histories.

To have TOPAS print time feature information to a log file and to the console:

- i:Tf/Verbosity = 2 # defaults to zero. set to 1 to get time log (NbParticlesInTime.txt), set to 2 to get detailed update messages

To implement beam current modulation, have your source’s NumberOfHistoriesInRun get its value from a time feature, as in:

- i:So/Default/NumberOfHistoriesInRun = Tf/MyBCMTimeFeature/Value
By default, scorers will output just once, after the entire session. But if you wish to have separate output from specific runs:

- `b:Sc/MyScorer/OutputAfterRun = "True"` # set True to trigger output of scorer after this run
- If this is always set False, or not defined, we just output at the end of the session.
- If this is always set True, we output after every run.
- If this is set from time features, then whenever we find it set to True, we output after that run.

**Random Time Mode**

Random Time Mode generates one history per run, with a randomly sampled time at each run. This has several uses.

- It allows one to sample time in a continuous fashion, so may show features that are obscured by Sequential Mode
- It provides a way to do a lower statistics run of what would have been a very long Sequential Mode job, yet still see aspects of the entire time interval, rather than just the first subset of the sequential times

To run in Random Time Mode, specify the start time and end time, turn on RandomizeTimeDistribution, and set your source’s NumberOfHistoriesInRandomJob, as in:

- `b:Tf/RandomizeTimeDistribution = "True"` # defaults to "False"
- `d:Tf/TimelineStart = 0. s` # defaults to zero
- `d:Tf/TimelineEnd = 10. s` # must be larger than TimelineStart
- `i:So/Default/NumberOfHistoriesInRandomJob = 1000` # defaults to 100

For each history, a random time will be sampled between TimelineStart and TimelineEnd.

We keep the parameters that control random mode (NumberOfHistoriesInRandomJob) separate from those that control sequential mode (NumberOfHistoriesInRun and NumberOfSequentialTimes) so that you can easily switch between the two modes (by just switching RandomizeTimeDistribution).

To implement beam current modulation, give your source a time-dependent ProbabilityOfUsingAGivenRandomTime, as in:

- `d:So/Default/ProbabilityOfUsingAGivenRandomTime = Tf/MyBCMTimeFeature/Value`

**Random Seeds**

To set the random seed:

- `i:Ts/Seed = 1` # starting random seed

To generate several statistically independent runs, give each run a different Ts/Seed. A typical solution to produce 10 independent runs would be to give starting seeds of 1 to 10. The allowed range is 0 to 2147483647 (the maximum 32 bit integer).

For more details see the discussion in the Geant4 Application Developer's Guide:


We use the random engine called:

- RanecuEngine

and the seed given to TOPAS is passed to the engine through

- CLHEP::HepRandom::setTheSeed
Additional Overall Control

To have TOPAS pause and wait for interactive Geant4 commands:

- b:Ts/PauseBeforeInit = "True"
- b:Ts/PauseBeforeSequence = "True"
- b:Ts/PauseBeforeQuit = "True"

After each pause, type the Geant4 command "exit" to return to TOPAS control.

- Most users will only use PauseBeforeQuit, typically to make a graphics window stay open at
  the end of the session (graphics windows close when Geant4 quits).
- The other two options, PauseBeforeInit and PauseBeforeSequence, provide the ability to enter
  Geant4 commands by hand, which may be useful in certain tests, but invalidates the basic
  TOPAS concept that the behavior of your simulation should be perfectly defined by TOPAS
  parameters.

Additional overall control parameters are:

- i:Ts/ShowHistoryCountAtInterval = 1 # how often to print history count to the console
  If set to 0, history count will never be printed
- b:Ts/ShowHistoryCountOnSingleLine = "False" # Make count reuse a single line of console
- b:Ts/ShowCPUTime = "True" # Show CPU time used in various phases of the simulation
- b:Ts/DumpParameters = "True" # dump full list of params to file
  TopasParameterDump_Run0.html
- b:Ts/DumpNonDefaultParameters = "False" # Like above but omits defaults
- sv:Ts/DumpParametersToSimpleFile = 2 "SomeParameter" "SomeOtherParameter" # Dumps
  the requested parameter types, names and values to a simple, human-readable file,
  TopasParameterDump_Run0.txt
- sv:Ts/DumpParametersToSemicolonSeparatedFile = 2 "SomeParameter"
  "SomeOtherParameter" # Dumps the requested parameter types, names and values to a
  semicolon separated file, TopasParameterDumpSSF_Run0.txt. This file is suitable for easy
  import into other applications
- i:Ts/TrackingVerbosity = 0 # Set to larger integer to see details of tracking
- i:Ts/RunIDPadding = 4 # Run numbers are padded in output files, such as
  MyScoringOutput_Run_0001.csv, so that they will sort naturally in various file viewers. This
  parameter sets how many places of padding are used.
To run in Fixed Time Mode, just set NumberOfHistoriesInRun (if you have more than one source, set desired number for each source).
To simulate at a time other than zero, adjust TimelineStart.
To run in Sequential Time Mode, provide TimelineStart and TimelineEnd, set the NumberOfSequentialTimes (how many different evenly spaced times Topas will use from this Timeline), and set NumberOfHistoriesInRun (how many histories should be run at each of those sequential times). Beam current can be modulated by having the NumberOfHistoriesInRun be a time feature.
To run in Random Time Mode, provide TimelineStart and TimelineEnd, turn on RandomizeTimeDistribution and set NumberOfHistoriesInRandomJob. Beam current can be modulated by having the ProbabilityOfUsingAGivenRandomTime be a time feature.

If source is phase space and contains no time data, it can only be used in Fixed Time Mode.
Number of histories is then controlled by i:\So/MySource\NumberOfHistoriesInRun, which defaults to 1.
If this value is zero, use So/MySource/NumberOfHistoriesInRun, possibly using partial phase space or partial reuse to get this number of histories.

In a related feature, the scoring system can tell Topas to end the run early, when some particular condition has been satisfied, such as:
   a) d:Sc/MyScorer/EndRunEarlyWhenValue = n
   b) d:Sc/MyScorer/EndRunEarlyWhenStandardDeviation = n

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Tf/...</th>
<th>So/MySource/...</th>
</tr>
</thead>
<tbody>
<tr>
<td>b:Tf/RandomizeTimeDistribution</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>d:Tf/TimelineStart default = 0. s</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>d:Tf/TimelineEnd default = TimelineStart</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>i:Tf/NumberOfSequentialTimes</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>i:Tf/NumberOfHistoriesInRun default = 1</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>i:So/MySource/NumberOfHistoriesInRun default = 10</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>i:So/MySource/NumberOfHistoriesInRandomJob default = 100</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>d:So/MySource/ProbabilityOfUsingAGivenRandomTime default = 1. unitless</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

| Fixed Time Mode                                        | ✓       | ✓             |
| Sequential Time Mode                                   | ✓       | ✓             |
| Random Time Mode                                       | ✓       | ✓             |

✗ means the parameter is ignored in the given mode.
Miscellaneous Notes

The Phase Space Format

Phase Space refers to the technique of saving or replaying a set of particles crossing a given surface.
- When one saves a phase space, one defines a surface and then saves the position, particle type, energy and momentum of some or all particles crossing that surface.
- When one replays a phase space, one starts a set of particles from the saved positions, with the saved particle types, energy and momentum.

Phase Space enables separating two parts of a simulation or analysis job, and can be used to transfer sets of particles among different codes.

A Phase Space is stored as a pair of related files:
- A .header file tells the number of histories, the number of saved particles and the order of information in the .phsp file
- A .phsp file contains all the details of all the saved particles

We support three formats for Phase Space:
- Binary is a compact format, with data encoded in a stream of bytes. The header file tells the contents and byte order per particle.
- ASCII provides the same information as Binary, but presents it as a much less compact, but easier to read simple text file, which data encoded as a series of columns of text. The header file tells the contents and column order per particle.
- Limited is an alternate binary format compatible with some legacy codes. It has fewer options for what data can be expressed, but is compatible with codes such as that used by Varian for their TrueBeam phase space files

For the Binary and ASCII formats, Particle ID is encoded using the large set of integer codes specified by the Particle Data Group (PDG):
- 22 = photon
- 11 = electron
- -11 = positron
- 2112 = neutron
- 2212 = proton
- Additional codes go all the way up to ten digit ion codes of the form ±10LZZZAAAI.
- See the PDG web site for a full explanation: http://pdg.lbl.gov/2012/mcdata/mc_particle_id_contents.html

For the Limited format, only a few particle codes are supported, while other particle types are not scored at all (and so this format is not recommended unless you need to interface with legacy codes):
- 1 = photon
- 2 = electron
- 3 = positron
- 4 = neutron
- 5 = proton
The Binary and ASCII formats are self-describing, with the complete column or byte order documented in the associated header file. The exact set of columns will depend on which options are used to create the phase space file. Run the examples to see these headers:

- examples/PhaseSpace/WriteASCII.txt
- examples/PhaseSpace/WriteBinary.txt

If you are attempting to create TOPAS Binary or ASCII phase space from some application other than TOPAS, be advised that the formatting requirements are very specific. It is best to compare your phase space header and phsp files to those produced by the TOPAS examples listed above. Some things to watch out for:

- First line of header has to be exactly as produced by TOPAS, with no extra spaces, tabs, etc.
- Integer values in the ASCII phase space must not contain decimal points.

The Limited format uses the following byte order (the format is not self-describing):

- Particle ID 1 byte
  Absolute value gives the particle code,
  Sign of this value encodes the direction of the 3rd direction cosine
- Energy 4 bytes
  Absolute value gives the energy in MeV
  Sign of this value is set to negative if this is the first scored particle from this history
- X position 4 bytes
- Y position 4 bytes
- Z position 4 bytes
- U (direction cosine of momentum with respect to X) 4 bytes
- V (direction cosine of momentum with respect to Y) 4 bytes
- Weight 4 bytes

Regardless of whether you use Binary, ASCII or Limited, the diagram on the following page explains the direction cosines.
Mathematical formula for cosine directions:

If \( \mathbf{v} \) is a vector

\[
\mathbf{v} = v_1 \hat{x} + v_2 \hat{y} + v_3 \hat{z}
\]

\[
\alpha = \cos \theta = \frac{\mathbf{v} \cdot \hat{x}}{\|\mathbf{v}\|} = \frac{v_1}{\sqrt{v_1^2 + v_2^2 + v_3^2}}.
\]

\[
\beta = \cos \phi = \frac{\mathbf{v} \cdot \hat{y}}{\|\mathbf{v}\|} = \frac{v_2}{\sqrt{v_1^2 + v_2^2 + v_3^2}}.
\]

\[
\gamma = \cos \varphi = \frac{\mathbf{v} \cdot \hat{z}}{\|\mathbf{v}\|} = \frac{v_3}{\sqrt{v_1^2 + v_2^2 + v_3^2}}.
\]

\[
\cos^2 \alpha + \cos^2 \beta + \cos^2 \gamma = 1
\]

\[
\theta = \sin^{-1} \left( \sqrt{\alpha^2 + \beta^2} \right)
\]

\[
\phi = \tan^{-1} \left( \frac{\beta}{\alpha} \right)
\]

\[
\gamma = \sqrt{1 - \alpha^2 - \beta^2}.
\]

\((\alpha, \beta, \gamma)\) are the Cartesian coordinates of the unit vector \( \hat{u} \). Here, \( \alpha, \beta \) and \( \gamma \) are the Direction cosines and \( a, b \) and \( c \) are the Direction angles of the vector \( \mathbf{v} \). The Direction angles \( a, b \) and \( c \) are acute, i.e., \( 0 \leq a \leq \pi, 0 \leq b \leq \pi \) and \( 0 \leq c \leq \pi \) and they denote the angles formed between \( \mathbf{v} \) and the unit coordinate vectors \( \hat{i}, \hat{j}, \hat{k} \).

http://mathworld.wolfram.com/DirectionCosine.html


Calculations of polar angles Theta and Phi, X, Y, Z directions and polar angles positions in TOPAS

\[
U = \frac{P_x}{|P|}
\]

\[
V = \frac{P_y}{|P|}
\]

\[
W = \frac{P_z}{|P|}
\]

Theta is the polar angle (with respect to the Z axis of the momentum)

\[
\theta = \sin^{-1} \sqrt{U^2 + W^2} \quad 0 < \theta < 180
\]

Phi is the azimuthal angle (which is defined within the XY plane)

\[
\varphi = \tan^{-1} \left( \frac{U}{W} \right) \quad 0 < \varphi < 360
\]

\[
W^2 + U^2 + V^2 = 1
\]
Changeable Parameters

In general, parameters cannot change once the TOPAS session has begun. Changes due to Time Features are fine (since the time feature's behavior itself is well defined), but any other change violates basic principles of repeatability.

C++ code that changes a parameter during the session, aside from time features, is allowed only for a special case in which a specialized geometry component needs to set a parameter value on the fly. An example is when TsCompensator reads in the compensator definition from a special file format. The resulting compensator thickness updates a parameter that affects positioning of other components.

Such a special case is allowed if the relevant parameter is defined from the start to be "Changeable". This is done by adding a "c" in front of the parameter type, for example:

```
• cd:Ge/Compensator/TransZ = 2. cm # the initial cd indicates that this is a changeable double
```

In a complex parameter file chain, if any level of the chain redefines this as just a "d" rather than a "cd", other parameter files will see this as a non-changeable parameter. Thus one parameter file may lock out others from making such changes.

TOPAS makes note of which parts of the system uses this changeable parameter (either directly or through a chain of parameters depending on other parameters) and takes care to explicitly update those parts of the system if this parameter ever changes.

Of course any parameter value can override the same parameter's value from a parent parameter file. This override at initial parameter read-in time is not what we mean by changeable. By Changeable we mean a value that changes during the TOPAS session.

The "c" syntax is not required when you are simply setting a parameter's value to a time feature. We allow:

```
• d:/Ge/Propeller/RotZ = Sq/TimeFeature/PropellerRot/Value.
```

It is true that this ...PropellerRot/Value is changeable, but that is handled internally by TOPAS.

Transient Parameters

When a parameter is changed during the session, either because it is a time feature value, or because some piece of C++ code changes the parameter, TOPAS does not actually overwrite the original parameter in memory, but instead adds it to a "Transient Parameter List". The Transient Parameter list always takes precedence over any other parameters file.

Transient parameters may be the first occurrence of a given parameter, as for the materials for a patient that are only instantiated as the patient is read in from DICOM, or transient parameters may override previously-defined parameters.

Quick Ways to Deactivate Parts of the Parameters Files

For most parameter categories, there is one key kind of parameter that triggers creation:

- TOPAS creates an Element when it sees a line like: El/.../Symbol =
- TOPAS creates a Material when it sees a line like: Ma/.../Components =
- TOPAS creates a Component when it sees a line like: Ge/.../Parent =
- TOPAS creates a Particle Source when it sees a line like: So/.../Type =
- TOPAS creates a Physics setup when it sees a line like: Ph/.../Type =
TOPAS creates a Scorer when it sees a line like: Sc/.../Quantity =
TOPAS creates a Graphic when it sees a line like: Gr/.../Type =
TOPAS creates a Variance Reduction setup when it sees a line like: Vr/.../Type =
TOPAS creates a Time Feature when it sees a line like: Tf/.../Function =

Thus you could effectively comment out the entire Component, Element, Material, Particle Source, etc. by just commenting out that line. But this way of turning something off can get you into trouble since you may then inherit behavior from a parent parameter file.

A better way to handle this is by setting a specific parameter designed for this purpose.
- Ge/MyComponent1/Include = "False"
- So/Default1/NumberOfHistoriesInRun = 0
- Sc/MyScorer1/Active = "False"
- Gr/MyGraphics1/Active = "False"

Such a parameter can then even be controlled by a time feature.
Extending TOPAS

While most TOPAS users will find that they can implement everything they want from parameter files, those who require additional functionality are free to extend TOPAS by writing new geometry components, scorers, filters, physics lists or particle sources. To do so, you write new classes in C++, taking advantage of the full syntax richness of C++. You may use almost any Geant4 class in your work. You link your extension into the rest of TOPAS by editing topas/extensions/TsExtensionsManager.cc and then relinking TOPAS as explained in the TOPAS README. Your extensions then coexist with the rest of the TOPAS code. The extension manager also provides places to hook in any other code that you want called at the start or end of the session, run or history.

Even when you have to write your own C++ code, TOPAS work is still easier than plain Geant4. You write your geometry components, scorers or filters as concrete implementations of TOPAS base components, scorers and filters which provide a wealth of helper functions to simplify your work. You may use the TOPAS parameter system to provide parameters to your classes, and those parameters can vary in time, like any other TOPAS parameters. All user extensions have a pointer to the parameter manager in their constructor. Thus, to access TOPAS parameters, call one of the following methods: fPm->someMethod

In all of the following forms, the parameterName argument can be either a G4String or a char*.

- // See if parameter exists
  G4bool ParameterExists(parameterName);

- // Get number of values in a vector parameter
  G4int GetVectorLength(parameterName);

- // Get dimensioned double value of parameter in Geant4’s internal units
  G4double GetDoubleParameter(parameterName, const char* unitCategory);

- // Get double value of a unitless parameter
  G4double GetUnitlessParameter(parameterName);

- // Get integer value of parameter
  G4int GetIntegerParameter(parameterName);

- // Get Boolean value of parameter
  G4bool GetBooleanParameter(parameterName);

- // Get string value of parameter (whether it is actually a string parameter or not)
  G4String GetStringParameter(parameterName);

- // Get vector of dimensioned double values of parameter in Geant4’s internal units
  G4double* GetDoubleVector(parameterName, const char* unitCategory);

- // Get vector of double values of a unitless parameter
  G4double* GetUnitlessVector(parameterName);
• // Get vector of integer values of parameter
  G4int* GetIntegerVector(parameterName);

• // Get vector of Boolean values of parameter
  G4bool* GetBooleanVector(parameterName);

• // Get vector of string values of parameter
  G4String* GetStringVector(parameterName);

• // Get TwoVector of double values of parameter in Geant4's internal units
  G4TwoVector GetTwoVectorParameter(parameterName, const char* unitCategory);

• // Get ThreeVector of double values of parameter in Geant4's internal units
  G4ThreeVector GetThreeVectorParameter(parameterName, const char* unitCategory);

A set of example components, scorers and filters are distributed as a zip file on the TOPAS users web site (see the file called extension_examples...). To create your own extension, start with the example that is the closest to what you want, then change the file name (and the class name throughout the file), then adjust the code as you wish. A few notes are given below.

**Adding Geometry Components**

TOPAS geometry components are like small pieces of what Geant4 users call their "detector construction" class. The Geometry sections of the Geant4 Application Developers Guide provide details on the full geometrical functionality of Geant4. In this section, we explain some details about how to write TOPAS components, but we assume that you are already comfortable with basic concepts of C++ and Geant4 geometry. The notes below are intended to discuss only those parts which may not be obvious.

Your geometry component class will be a concrete implementation of the base class: TsVGeometryComponent.

You can get any parameter name of the current component by using the GetFullParmName method. For example, if your parameter file specifies:

  d:Ge/MyComponent/Blatz = 42. mm

GetFullParmName("Blatz") will return "Ge/MyComponent/Blatz". You can then feed this resulting string into the parameter access methods such as:

  G4Double blatzLength = fPm->GetDoubleParameter(GetFullParmName("Blatz"), "length");

Your component may contain any of the following methods. Careful attention to what goes where will insure that your classes are robust under 4D and the base classes will do much of the work for you.

**Constructor:** Must exist and may be empty.
This method will only be called at the very beginning of the simulation. It will not be called after changes in 4D. Only put things here if you are absolutely certain you will not need to recompute them during the simulation.

**Destructor:** Must exist and may be empty.
Destroy any special objects you created with "new" statements. You may not destroy solids, logical volumes or physical volumes. These destructions are handled for you by the base class.
You do not need to do anything to handle the basic parameters, Parent, TransX, TransY, TransZ, RotX, RotY, RotZ, Material and Color. These are handled for you by the base class, including 4D capabilities.

If there are any other parameters that you may want to vary in 4D, provide a method

```
UpdateForSpecificParameterChange(G4String parameter).
```

- If the parameter name is one that you want to handle, do so.
  - Be sure to use GetFullParmNameLower rather than GetFullParmName in your check.
- If your handing of this parameter moves a volume relative to its mother volume, advise Geant4’s smart voxel system that it needs to reoptimize the mother volume by calling AddToReoptimizeList. The argument should be the mother’s logical volume.
- If the parameter name is not one that you want to handle, pass it on to the base class handler, TsVGeometryComponent::UpdateForSpecificParameterChange. This is essential to enable basic 4D behaviors such as overall component motion.

TsMultiLeafCollimator.cc is a good example of this kind of behavior. It allows leaf position to change over time.

For the rest of your work, provide a method Construct.

The first line of Construct MUST be:

```
BeginConstruction();
```

The rest of Construct is whatever you want to do to create Geant4 Solids, Logical Volumes and Physical Volumes. But you must follow some rules to insure that TOPAS will be able to properly manage your volumes in 4D.

- You create Geant4 Solids just as you would in any Geant4 geometry.
- You DO NOT create Geant4 Logical Volumes or Physical Volumes directly, but instead use helper methods from the base class. This allows TOPAS to manage your solids and volumes efficiently, even if they are moving.

To create the overall logical volume for your component, use:

```
fEnvelopeLog = CreateLogicalVolume(G4VSolid* solid);
```

The logical volume will automatically get the material and visualization properties specified in your parameter file for this component, such as Ge/MyComponent/Material and Ge/MyComponent/Color. Be sure that the value on the left side of the above is exactly "fEnvelopeLog". This insures that TOPAS knows the overall logical volume’s name and is essential for TOPAS to support your component in 4D.

If a component is made up of more than one volume, these additional volumes are called "SubComponents." A component may have more zero, one or more SubComponents. An example of SubComponents is the Blades in a Propeller, such as:

```
s:Ge/Propeller/Type = "TsPropeller"
...
i:Ge/Propeller/NbOfBlades = Ge/PropellerConstant/NbBlades
s:Ge/Propeller/Blade/Material = "World"
s:Ge/Propeller/Blade/Color = "skyblue"
```
In all of the following forms, the subComponentName argument can be either a G4String or a char*.

To create a logical volume for a subcomponent, use:

- G4LogicalVolume* CreateLogicalVolume(subComponentName, G4VSolid* solid);

TOPAS will look for material and visualization parameters such as:

- Ge/ComponentName/SubComponentName/Material = ...

To hard-code the material, rather than having it come from this parameter, use:

- G4LogicalVolume* CreateLogicalVolume(subComponentName, G4String& materialName, G4VSolid* solid);

This is particularly useful in cases where you want the material to be the same as the component’s mother, that is, the material surrounding your component. We do this, for example, when we want to make a void in a collimator. To get that mother volume’s material name, use:

- G4String envelopeMaterialName = fParentComponent->GetResolvedMaterialName();

The base class will take care of automatically setting your component’s visualization attributes based on the component’s parameters. But you can set different attributes for subComponents with code such as:

- G4VisAttributes* yokeColor = new G4VisAttributes(G4Colour(0.2, 1.0, 0.2)); // Sets RGB color
- RegisterVisAtt(yokeColor); // Necessary so that TOPAS can delete the attribute if the component is rebuild during 4D behavior
- yokeLogicalVolumePointer->SetVisAttributes(yokeColor);

To create the overall physical volume for your component, use:

- fEnvelopePhys = CreatePhysicalVolume(fEnvelopeLog);

Be sure that the value on the left side of the above is exactly "fEnvelopePhys". This insure that TOPAS knows the overall physical volume’s name and is essential for TOPAS to support your component in 4D.

Additional forms of CreatePhysicalVolume allow you to place subcomponents within your component.

To place a subcomponent in the center of your logical volume lVol:

- G4VPhysicalVolume* CreatePhysicalVolume(subComponentName, G4LogicalVolume* lVol, G4VPhysicalVolume* parent);

To place a subcomponent into your logical volume lVol, with an offset or rotation:

- G4VPhysicalVolume* CreatePhysicalVolume(subComponentName, G4LogicalVolume* lVol, G4RotationMatrix* rot, G4ThreeVector* trans, G4VPhysicalVolume* parent);

To place multiple copies of the same subcomponent name into your logical volume, call:

- G4VPhysicalVolume* CreatePhysicalVolume(subComponentName, G4int copy, G4bool reuseLogical, G4LogicalVolume* lVol, G4RotationMatrix* rot, G4ThreeVector* trans, G4VPhysicalVolume* parent);

copy should be a unique integer to differentiate the different copies of your subcomponet. This copy number is useful in some of the visualization commands when you want to control just one copy or another.

Set reuseLogical true if you are using the same logical volume in all of these placements. This is efficient if all of the copies of the subcomponent are identical except for their placement.
- Set `reuseLogical` false if you are using different logical volumes in each of these placements. This allows you to make each copy of the subcomponent different (different material, different shape, different size, etc.).

To place multiple copies of the same subcomponent using a Geant4 parameterization (creating Geant4 parameterized volumes), call:

```c
G4VPhysicalVolume* CreatePhysicalVolume(const char* subComponentName, G4LogicalVolume* lVol, G4VPhysicalVolume* parent, const EAxis pAxis, const G4int nReplicas, G4VPVParameterisation* pParam);
```

To place multiple copies of the same subcomponent using a Geant4 replica volume, call:

```c
G4VPhysicalVolume* CreatePhysicalVolume(const char* subComponentName, G4LogicalVolume* lVol, G4VPhysicalVolume* parent, const EAxis pAxis, const G4int nReplicas, G4double width);
```

The last line of `Construct` MUST be:

```c
return fEnvelopePhys;
```

Some helper functions you may want to use from the `TsParameterManager`:

- `G4VisAttributes* GetColor(G4String name);`
- `G4VisAttributes* GetColor(const char* name);`
- `G4VisAttributes* GetInvisible();`

Some helper functions you may want to use from the `TsVGeometryComponent`:

- `SetTooComplexForOGLS()`
  Call this to tell Graphics that this component has become too complex to efficiently render in OpenGL's Stored Mode. It will instead be rendered in OpenGL's Immediate Mode (can be less quick to update, but uses less memory)
- `GetMaterial`
  By default, the logical volumes you create will get their material from the material parameter you specified for this component. But you can use `GetMaterial` to obtain any other named material.

**Adding Scorers**

Scorers must be written carefully to avoid slowing down the simulation. The scorer’s `ProcessHits` method is called for every hit in the scoring component. Slow operations such as string comparisons should be avoided during this method. Try to write your code so that you perform these sorts of slow operations only during construction, save values and pointers in class variables and then use these precalculated values in the `ProcessHits` method.

At a minimum, your scorer should provide a constructor, a destructor and a `ProcessHits` method. The base class will take care of all the details of filtering, accumulating and outputting results.

If you want to take more complete control of the scoring process, you can provide optional methods:

- `void UserHookForEndOfTrack()`
  - called after the last hit of a given track
- `void UserHookForEndOfIncidentParticle()`
  - called after the last hit of all tracks resulting from a given particle incident on the scoring component
• void UserHookForEndOfEvent()
  ○ called at the end of the event
• void UserHookForEndOfRun()
  ○ called at the end of the run

Between the ProcessHits method and these other four methods, you have complete control over how you will accumulate and handle your scored values. Accumulate values in your own data structures that you provide in your scorer’s header file or in other classes that your scorer calls. Manipulate and output these values as you wish. It is all up to you. You can still choose to fill the fEvtMap just like a regular scorer, in which case TOPAS will accumulate and output those values, or you can fill nothing into that fEvtMap, in which case TOPAS will not take any further action for this scorer.

Some helper functions you may want to use from the TsParameterManager:
- SetNeedsTrackingAction
  Activates creation of the TsTrackInformation object
- SetNeedsSteppingAction
  Activates creation of the extra part of the TsTrackInformation object that contains information on what volumes were traversed

Some helper functions you may want to use from the TsVScorer:
- GetMaterial
  Get pointer to a material
- UsedMaterial
  Tell whether a given material is used in the geometry
- GetTime
  Get the current TOPAS time (for the time of flight, use fTimeOfFlight)
- GetRunID
- GetEventID
- GetRandomNumberStatusForThisEvent
- SuppressStandardOutputHandling
  Disable the automatic creation and filling of output, leaving this work entirely to your scorer

For divided components, the combined index one finds in scorers is formed from three bin indices (x,y,z or r, phi, z or r, phi, theta for TsBox, TsCylinder and TsSphere respectively). A helper function is now provided to return the individual bin indices given the combined index.
- GetBin(index, iBin) where iBin is 0, 1 or 2

Adding Filters
Filters must be written carefully to avoid slowing down the simulation. The filter’s Accept method is called for every hit in the scoring component. Slow operations such as string comparisons should be avoided during this method. Try to write your code so that you perform these sorts of slow operations only during the constructor, ResolveParameters method or CacheGeometryPointers method, save values and pointers in class variables and then use these precalculated values in the ProcessHits method.

Parameter lookups should be done in ResolveParameters. Call ResolveParameters directly from your constructor, and then you can also rely on TOPAS to re-call this method any time one of this filter’s parameters is changed.
4D behaviors may require TOPAS to destroy and rebuild components during the simulation. Accordingly, you can not rely on the pointer to a given component remaining the same throughout the simulation. Any lookup of a component pointer should be done in the filter’s CacheGeometryPointers method. TOPAS will re-call this method any time relevant components are rebuilt.

Some helper functions you may want to use from the TsParameterManager:
- SetNeedsTrackingAction
  Activates creation of the TsTrackInformation object
- SetNeedsSteppingAction
  Activates creation of the extra part of the TsTrackInformation object that contains information on what volumes were traversed

Some helper functions you may want to use from the TsVFilter:
- GetMaterial
  Get pointer to a material
- GetPhysicalVolume
  Get pointer to a named physics volume
- GetComponent
  Get pointer to a named component
- GetChildComponentsOf
  Get pointers to all children of a named component

Adding Physics Lists

You can supply your own physics list. Note however that this option is not recommended unless you have significant Geant4 expertise. Even most long-time Geant4 users get into difficulty writing their own physics lists. Wherever possible, you should try to use one of the Reference physics list or the Modular physics list.

The example physics list provided in topas/extensions/TsPhysicsList1 includes a pointer to the TOPAS parameter manager as its argument. This is not required, but allows you to use TOPAS parameters to adjust options within your physics list.

Adding Particle Sources

Your particle source defines the initial particles that are then transported by the simulation.

Parameter lookups should be done in ResolveParameters. Call ResolveParameters directly from your constructor, and then you can also rely on TOPAS to re-call this method any time one of this particle source’s parameters is changed.

TOPAS will call your GeneratePrimaries method once per history.
You should always start this method with this test:
- if (CurrentSourceHasGeneratedEnough()) return;
This allows your source to properly coexist with other sources that may have other numbers of histories.

The body of your GeneratePrimaries method should create and fill some number of TsPrimaryParticles (a single history may contain zero, one or more primary particles).

The TsPrimaryParticle structure is defined in the header file TsVParticleSource.hh.
For each TsPrimaryParticle that you define, call GenerateOnePrimary.

Once you have finished creating all of the TsPrimaryParticles for this history, call AddPrimariesToEvent.

**Additional User Hooks**

Six additional hooks are provided in TsExtensionManager for you to attach your own code.
- BeginSession
- BeginRun
- BeginHistory
- EndHistory
- EndRun
- EndSession

There are no particular constraints on what you can do in these methods. They are provided simply to give you more flexibility in the design of your extensions.

**Authorship**

Editor and Lead Author:
- Joseph Perl

Additional Authors:
- Jungwook Shin
- Jan Schümann
- José Ramos

User Contributions:
- Aleksandra Biegun
- Fada Guan

Contributions welcome from authorized TOPAS users via the User Forum at:
- http://topasmc.org