
pyg4ometry

S. Boogert, A. Abramov, A. Butcher, L. Nevay, W. Shields, S. Walk

Jan 08, 2024

CONTENTS

1	User Manual	3
1.1	Installation	3
1.2	Introduction	6
1.3	Loading Geometry	7
1.4	Viewing Geometry	12
1.5	Validating Geometry	17
1.6	Exporting Geometry	19
1.7	Creating Geometry	21
1.8	Converting Geometry	29
1.9	Combining Geometry	34
1.10	Comparing Geometry	35
1.11	Analysing Geometry	39
2	Indices and tables	41
3	Referencing & Citation	43
4	Changelog (legacy)	45
4.1	V1.0.2 - 2022 / 04 / 11	45
4.2	v1.0.1 - 2022 / 02 / 10	46
4.3	v0.9.0 - 2021 / 07 / 01	47
4.4	Pre-History	47
5	API Reference	49
5.1	pyg4ometry	49
6	Indices and tables	427
	Python Module Index	429
	Index	431

pyg4ometry is a package to create, load, write and visualise solid geometry for particle tracking simulations.

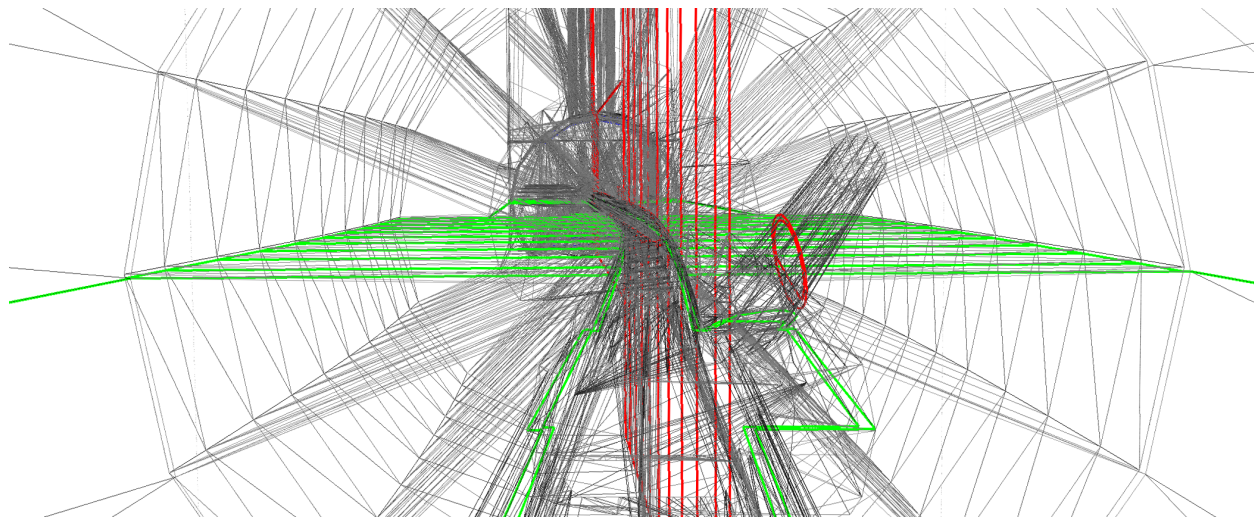


Fig. 1: Example accelerator tunnel complex at CERN.

USER MANUAL

1.1 Installation

1.1.1 Requirements

pyg4ometry is developed exclusively for Python 3 (Python2 is deprecated). It is developed on Python 3.9 and 3.10.

- [VTK \(Visualisation toolkit\)](#) (including Python bindings)
- [antlr4](#)
- [cython](#)
- [CGAL](#)
- [pybind11](#)

Packages that are required but will be found through PIP automatically:

- [matplotlib](#)
- [GitPython](#)
- [pandas](#)
- [py pandoc](#)
- [networkx](#)
- [numpy](#)
- [sympy](#)

Optional

- [Freecad](#) for CAD conversion.

Note: A full list can be found in `pyg4ometry/setup.py`.

Note: if you are choosing a python version, it is worth choosing according to which version VTK provides a python build of through PIP if you use that. See <https://pypi.org/project/vtk/#files> For example, there are limited builds for M1 Mac (ARM64).

1.1.2 Installation

To install pyg4ometry, simply run `make install` from the root pyg4ometry directory:

```
cd /my/path/to/repositories/  
git clone http://bitbucket.org/jairhul/pyg4ometry  
git checkout develop  
cd pyg4ometry  
  
make install
```

Note: To build using the git directory and not installing into `/usr/local` use `make develop` instead of `make install`

To build pycsg with cpython:

```
make build_ext
```

Or install from pypi:

```
pip install pyg4ometry
```

or alternatively, run `make develop` from the same directory to ensure that any local changes are picked up.

1.1.3 Docker image

1. Download and install [Docker desktop](#)
2. open a terminal (linux) or cmd (windows)
3. (windows) Start [Xming](#) or [Vxsrv](#)
4. Download the [pyg4ometry docker file](#)
5. `docker build -t ubuntu-pyg4ometry -f Dockerfile-ubuntu-pyg4ometry .`

If you need to update increment the variable ARG `PYG4OMETRY_VER=1`

To start the container

1. open a terminal (linux/mac) or cmd (windows)
2. get your IP address `ifconfig` (linux/mac) or `ipconfig /all` (windows)
3. Start XQuartz (mac) or Xming/Vxsrv (windows). For Xming/Vxsrv (might need to play with the settings when launching)
4. `docker run -ti -v /tmp/.X11-unix:/tmp/.X11-unix -v YOURWORKDIR:/tmp/Physics -e DISPLAY=YOUR_IP ubuntu-pyg4ometry` (the `-v /tmp/.X11-unix:/tmp/.X11-unix` is only required for mac/linux)

Test the installation

1. `docker> cd pyg4ometry/pyg4ometry/test/pythonGeant4/`
2. `docker> ipython`
3. `python> import pyg4ometry`
4. `python> import T001_Box`

5. `python> T001_Box.Test(True,True)`

1.1.4 Linux installation

There are docker files for Centos 7 and Ubuntu 20. The docker files can be used as list of instructions for installation for each of these OSes.

- [Ubuntu 20.02](#)
- [Centos 7](#)

1.1.5 FreeCAD support for CAD to GDML conversion

For FreeCAD support and you already have it installed you need to add library to PYTHONPATH, for example

```
export PYTHONPATH=/opt/local/libexec/freecad/lib/
```

Building FreeCAD can be a pain for MAC so

```
mkdir FreeCAD
cd FreeCAD
set FCROOT=$pwd
wget https://github.com/FreeCAD/FreeCAD/archive/refs/tags/0.19.4.tar.gz
tar xzf 0.19.4.tar.gz
mkdir build
mkdir install
cd build
cmake ../FreeCAD-0.19.4 -DCMAKE_INSTALL_PREFIX=../install \
-DCOIN3D_LIBRARIES=/opt/local/Library/Frameworks/Inventor.framework/Libraries/libCoin.
↳dylib -DBUILD_FEM=0 \
-DBUILD_MATERIAL=0 -DBUILD_SHIP=0 -DBUILD_DRAFT=0 -DBUILD_TUX=0 -DBUILD_ARCH=0 -DBUILD_
↳PLOT=0 \
-DBUILD_OPENSCAD=0
make -jN
make install
export PYTHONPATH=$PYTHONPATH:$FCROOT/install
```

1.1.6 Python 3.9

At the time of writing, there are limited VTK distributions for Python 3.9 on pypi (what PIP uses when finding packages). However, you can have VTK with Python 3.9 through say MacPorts or by compiling it yourself. In this case, you can comment out the VTK requirement from the setup.py around line 86, as long as you know you can `import vtk` ok in your Python installation.

Warning: ANTLR will create an unbelievable amount of warnings when using a different ANRLR version that the one the parser was generated with. It should work though. We are trying to include multiple versions of the ANTLR parser to avoid this in future.

1.2 Introduction

This package started as an internal tool for the BDSIM and machine backgrounds group at Royal Holloway. BDSIM is a tool to rapidly create Geant4 models of accelerator systems. Creation of geometry is a time consuming activity and pyg4ometry hopefully will improve the time taken to create accurate reliable geometry files.

1.2.1 Need for programmatic geometry generation

- Non-expert user creation and maintenance of geometry
- Reduce time spent creating geometry
- Reproducibility
- Lower number of errors
- Parameterisation of geometry
- Visualisation of geometry
- Overlap checking
- Import from other geometry packages

1.2.2 Geant4 key concepts

- **solid** - describes shape only.
- **logical volume** - a solid (shape) plus a material. Practically, in Geant4 it can include fields, regions, visualisation attributes and user limits.
- **physical volume** - a placement of a logical volume. A ‘stamp’ out of the logical volume. It is uniquely identified by an associated integer called “copy number”.
- **placement** - the term placement is used often to describe a physical volume. They are one and the same.
- **geometry reuse** - individual solids and logical volumes are encouraged to be reused. For example a row of copper boxes all the same would require only 1x solid and 1x logical volume with N placements (also known as physical volumes).

1.2.3 Geometry key concepts

- Constructive Solid Geometry (CSG)
- Boolean operations
- Boundary representation (B-REP)
- Boundary mesh

1.2.4 Implementation concepts

Registry

In pyg4ometry and in GDML we must uniquely identify objects by their associated name. However, in Geant4 (in C++) objects are uniquely identified by their memory address (pointer) and even for objects that have a name parameter, there is no requirement for these to be unique.

To resolve this we have the concept of a registry. This is a holder for all the definitions for a given set of geometry. It can be thought of as a namespace. It will protect against duplicate names that would prevent writing the geometry to GDML.

A `pyg4ometry.geant4.Registry` instance holds dictionaries to all solids, logical volumes and other objects, but also a nominated 'top volume' - the world volume. This needn't be the "world" as such, but is identified as the topmost part of the geometry.

When loading geometry, the typical result is a registry that contains all definitions and a top volume.

It is possible to merge two registries and all the name conflicts will be explicitly resolved. See [Combining Geometry](#).

- Parameter
- ParameterVector
- Pycsg

1.3 Loading Geometry

Generally, a reader class is provided for each format. The reader is created, then told to load a file, and it creates a Registry object (see [Registry](#)) containing the model. The registry is the final object from a reader, and its top-most volume can be used for visualisation or other operations.

In directory `pyg4ometry/test/gdmlG4examples/ChargeExchangeMC/`

```

1 import pyg4ometry
2
3 r = pyg4ometry.gdml.Reader("lht.gdml")
4 l = r.getRegistry().getWorldVolume()
5 v = pyg4ometry.visualisation.VtkViewerColouredMaterial()
6 v.addLogicalVolume(l)
7 v.view()
```

In directory `pyg4ometry/test/flukaCompoundExamples`

Note FLUKA geometry can be loaded but cannot be visualised directly without conversion to a Geant4 model. This is not necessary for loading, but shown here.

```

1 import pyg4ometry
2
3 r = pyg4ometry.fluka.Reader("corrector-dipole2.inp")
4 flukaRegistry = r.flukaregistry
5
6 geantRegistry = pyg4ometry.convert.fluka2Geant4(flukaRegistry)
7
8 l = geantRegistry.getWorldVolume()
9 v = pyg4ometry.visualisation.VtkViewerColouredMaterial()
```

(continues on next page)

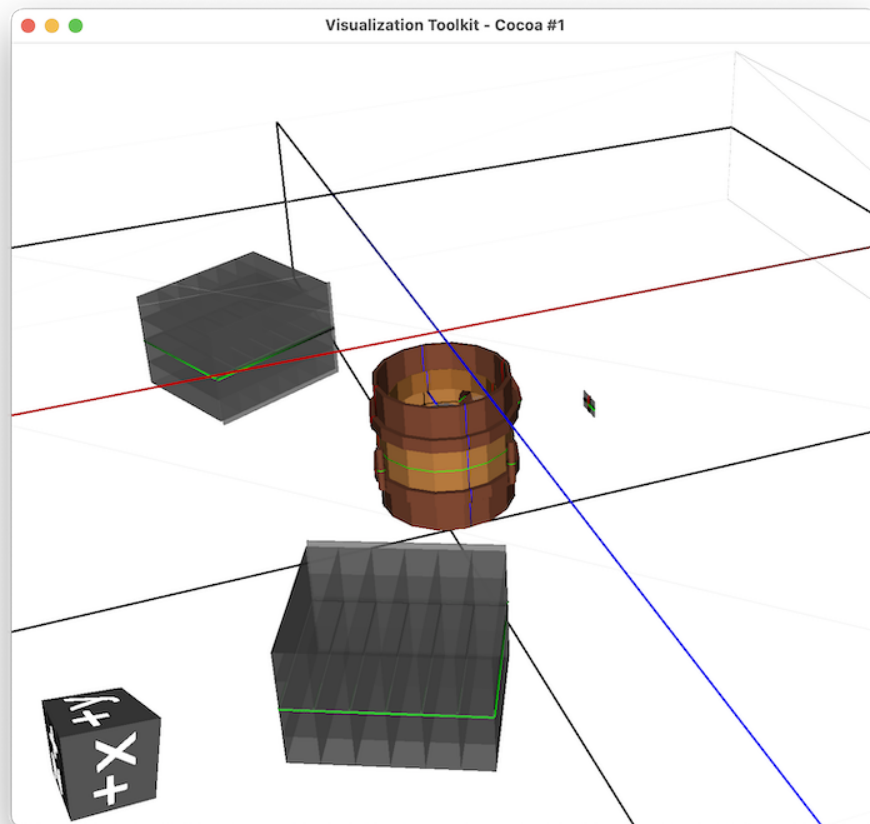


Fig. 1: Geant4 example GDML from ChargeExchangeMC example.

(continued from previous page)

```

10 v.addLogicalVolume(l)
11 v.view()

```

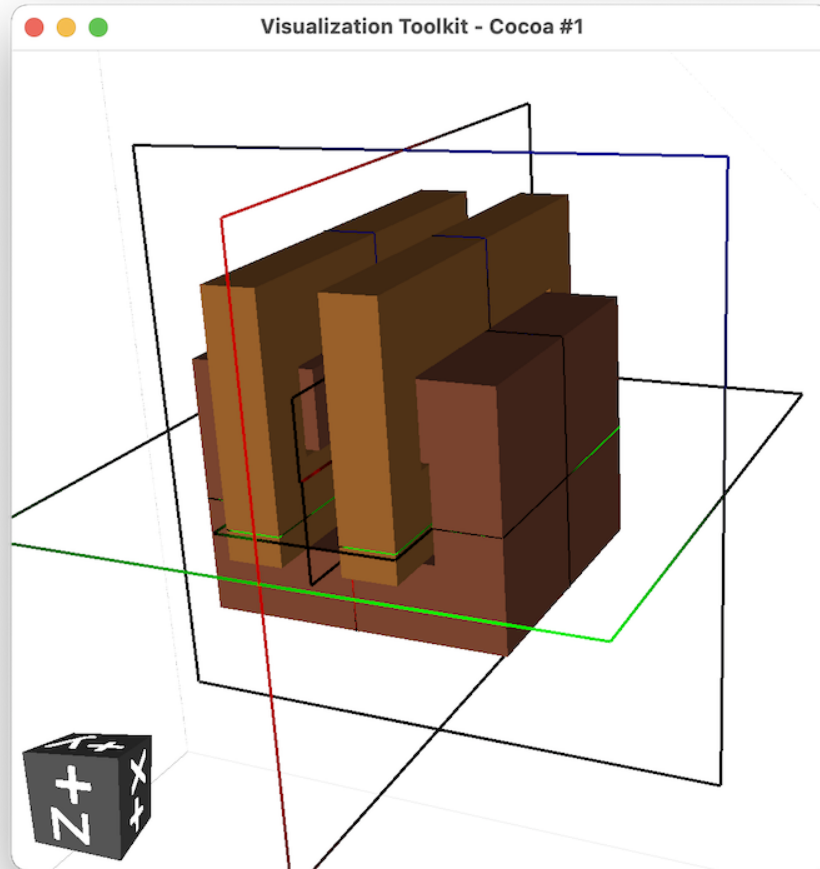


Fig. 2: FLUKA example of a dipole magnet.

In directory `pyg4ometry/test/root2Gdml`

```

1 import pyg4ometry
2
3 r = pyg4ometry.io.R00TTGeo.Reader("example.root")
4 l = r.getRegistry().getWorldVolume()
5 v = pyg4ometry.visualisation.VtkViewerColouredMaterial()
6 v.addLogicalVolume(l)
7 v.view()

```

In directory `pyg4ometry/test/stl`

STL files are typically used for a single watertight solid mesh. This mesh is converted to a `TessellatedSolid` and then a logical volume which can be placed in a geometry. In directory `pyg4ometry/test/stl`.

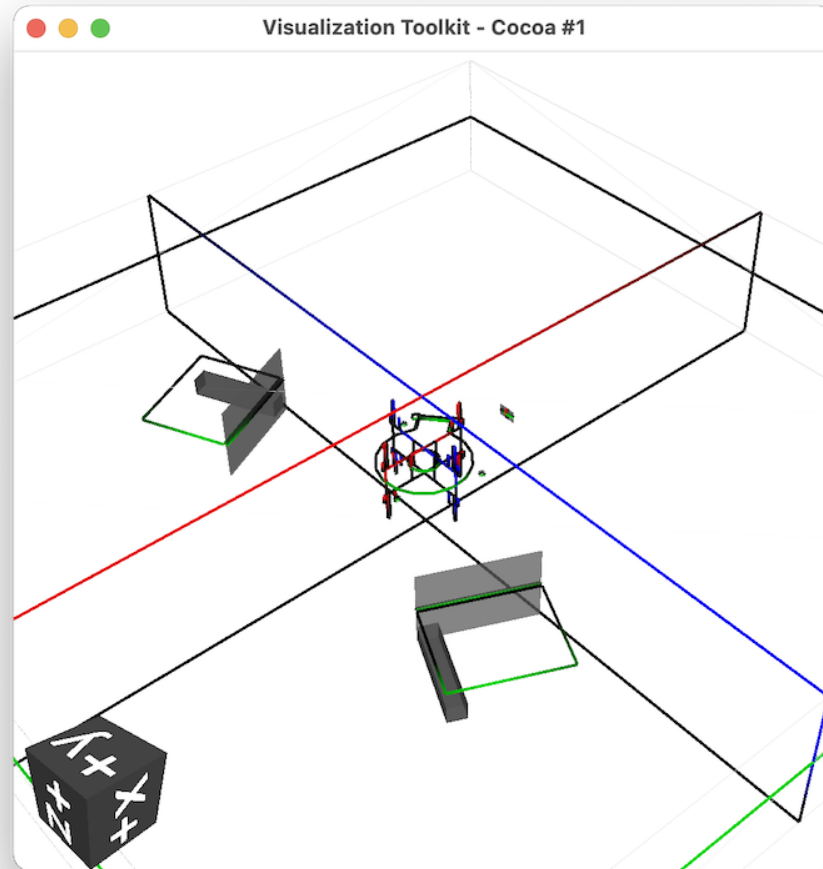


Fig. 3: ROOT example of Geant4's LHT geometry.

```
import pyg4ometry

reg = pyg4ometry.geant4.Registry()
r = pyg4ometry.stl.Reader("utahteapot.stl", reg)
s = r.getSolid()
copper = pyg4ometry.geant4.MaterialPredefined("G4_Cu", reg)
l = pyg4ometry.geant4.LogicalVolume(s, copper, "utahteapot_lv", reg)
v = pyg4ometry.visualisation.VtkViewerColouredMaterial()
v.addLogicalVolume(l)
v.view()
```



alt

Example of STL loading in pyg4ometry

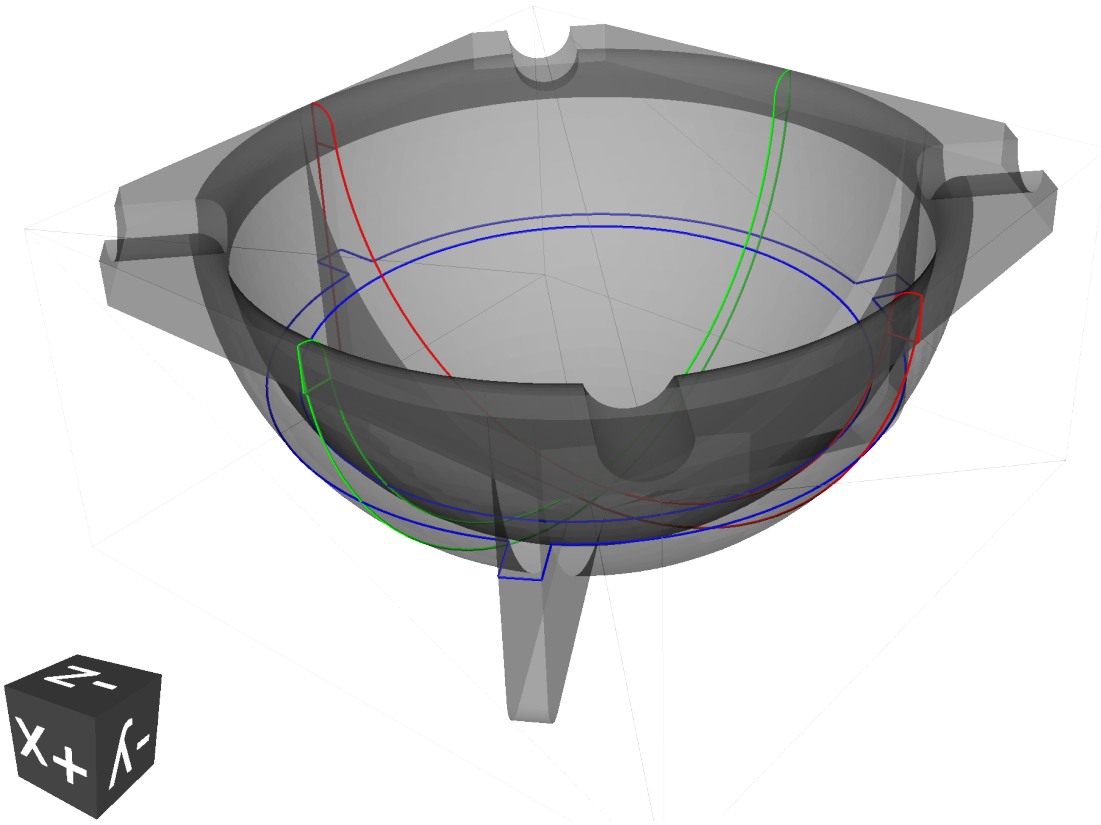
In directory pyg4ometry/test/freecad

```
1 import pyg4ometry
2
3 r = pyg4ometry.freecad.Reader("08_AshTray.step")
4 r.relabelModel()
5 r.convertFlat()
6 l = r.getRegistry().getWorldVolume()
7 v = pyg4ometry.visualisation.VtkViewer()
```

(continues on next page)

(continued from previous page)

```
8 v.addLogicalVolume(1)
9 v.view()
```



1.4 Viewing Geometry

pyg4ometry uses the Visualisation Toolkit - VTK - to view geometry. This provides a window with a 3D view of a model that can be manipulated with a cursor (rotate, pan, zoom, click). Most manipulation can be done by calling methods on the instance of the visualiser class in the terminal.

Basic loading and viewing is described in [Loading Geometry](#). Here, more detailed individual features are described.

1.4.1 Visualiser Classes

Several visualiser classes are provided but ultimately, the only difference is the default colouring of volumes.

- `pyg4ometry.visualisation.VtkViewer` - all in grey
- `pyg4ometry.visualisation.VtkViewerColoured` - user-provided dictionary of materials to colours
- `pyg4ometry.visualisation.VtkViewerColouredMaterial` - default dictionary of material colours included

Both `VtkViewerColoured` and `VtkViewerColouredMaterial` inherit `VtkViewer` and have the same functionality and differ only in colouring of volumes.

Generally:

```
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(lv)
v.view() # or
v.view(interactive=False) # to not block the terminal
```

For FLUKA, ROOT, STL, STEP visualisation, see [Loading Geometry](#) where each example has visualisation.

Exact documentation can be found in `module-docs-visualisation`.

If not adding a whole geometry tree, then individual solids can be added and overlaid with transparency for comparison purposes.

1.4.2 Using the Visualiser

Action	Outcome
click and drag	rotate the geometry
shift key + click and drag	pan (move) the geometry
right click	'pick' a volume and print out its name if found
scroll	zoom in and out

Rotating

Rotate by clicking and dragging, then release.

Zooming

Scroll in and out on a mouse or trackpad whilst pointing at the visualiser.

Panning And Rotating

Click and drag to rotate. Hold the shift key on the keyboard, then click and drag to pan.

When we rotate the geometry it may twist in multiple angles. To rotate in a specific way we can click and drag and draw it small circles where the geometry will precess.

Picking

If you right click on a volume and look at the terminal, if pyg4ometry can find a volume behind the point clicked it will print out the name.

1.4.3 Solid or Wireframe

When using the visualiser window, the same geometry can be viewed as solid surfaces or as a wireframe by pressing s key or the w key respectively.

Note, the original visualisation has the outermost volume as wireframe and the contents as solid. Once, the wireframe or solid option has been chosen, all volumes will have the same style.

1.4.4 Logical Volume

A `pyg4ometry.geant4.LogicalVolume` instance can be added to the visualiser. A logical volume has no concept of translation or rotation on its own, so it is placed in the centre of the visualiser coordinate system, i.e. in its own frame.

```
lv # pyg4ometry.geant4.LogicalVolume instance
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(lv)
v.view()
```

It is possible to view the logical volume with an offset (i.e. translation) and rotation. This is purely for adding the scene of the viewer and does not affect the logical volume itself or anything it is used in. We can see the docstring:

```
>>> v = pyg4ometry.visualisation.VtkViewer()
>>> v.addLogicalVolume?
Signature:
v.addLogicalVolume(
logical,
mtra=matrix([[1, 0, 0],
[0, 1, 0],
[0, 0, 1]]),
tra=array([0, 0, 0]),
recursive=True,
)
```

If we start from a rotation as a series of Tait-Bryan angles, we can turn this into a matrix with:

```
import numpy as np
rotation = [0, np.pi/2, 0] # for example
matrix = np.linalg.inv(pyg4ometry.transformation.tbxyz2matrix(rotation))
l # a pyg4ometry.geant4.LogicalVolume instance
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(l, mtra=rotation, tra=[0,0,500])
```

Note: When directly using rotations and translations, the units are radians and mm.

If overlap checking has been used, this produces overlap meshes (if any) and these will be visualised automatically when visualising a LogicalVolume instance as they are associated with that instance.

1.4.5 Solid

It is possible to view an individual solid, i.e. any instance of a class in `pyg4ometry.geant4.solid` module.

```
s # e.g. a pyg4ometry.geant4.solid.Box instance
v = pyg4ometry.visualisation.VtkViewer()
v.addSolid(s)
v.view()
```

Similarly to a logical volume, an individual solid has no concept of placement position and will by default be placed at the centre of the scene. It is also possible to add it to the scene with a rotation and translation.

```
>>> v.addSolid?
Signature:
v.addSolid(
solid,
rotation=[0, 0, 0],
position=[0, 0, 0],
representation='surface',
colour=[0.5, 0.5, 0.5],
opacity=0.2,
)
```

This uses Tait-Bryan angles for the rotation.

1.4.6 Boolean Solid

When creating geometry, it is common to use Boolean operations. Sometimes, we make mistakes in these and it is useful to understand the individual constituents even if the result is not a valid solid or mesh (i.e. completely disconnected solids). To do this we can visualise just a Boolean solid on its own.

```
s # e.g. a pyg4ometry.geant4.solid.Subtraction instance
v = pyg4ometry.visualisation.VtkViewer()
v.addBooleanSolidRecursive(s)
v.view()
```

This will work recursively for each solid that makes up the Boolean even if they are Booleans themselves. It will tolerate shapes that cannot form a valid mesh such as the resultant Boolean solid.

1.4.7 Default Colour Coding

With the `VtkViewer` class all volumes are visualised as semi-transparent grey.

1.4.8 Custom Colour Coding

With the `VtkViewerColoured` class, we can provide a default general colour and also a dictionary of specific colours for materials by name.

1.4.9 Random Colours

With the `VtkViewerColoured` class, we can supply the default colour as "random", which will result in every volume being visualised with a random colour to be different.

```
v = pyg4ometry.visualisation.VtkViewerColoured(defaultColour="random")
```

1.4.10 Overlaying Two Geometries

In the visualiser we add “meshes” to the scene that are displayed. We are not restricted to make a physically accurate model and we can draw multiple meshes on top of each other by successively adding them to the scene.

Logical Volume Difference

The function `pyg4ometry.visualisation.viewLogicalVolumeDifference()` is provided that will view two `pyg4ometry.geant4.LogicalVolume` instances. It will also calculate the difference mesh between the two and visualise that also on top of the two with a different colour to highlight it.

1.4.11 Plotting Cutter Outlines From Files

Cutters are a feature in the visualiser to generate the usually red, green, blue lines that intercept the edges of the geometry in a given plane. By default, these are along each axis aligned with 0,0,0. These can be plotted or added to an existing plot as follows:

:: code-block:

```
pyg4ometry.visualisation.Plot.AddCutterDataToPlot("crosssection-ZX.dat",
                                                  "zx",
                                                  unitsFactor=0.001)
```

An optional Matplotlib axes instance can be given if there is one from an existing plot.

:: code-block:

```
f = matplotlib.pyplot.figure()
ax = f.add_subplot(111)
ax.plot([0,1],[0,1])
pyg4ometry.visualisation.Plot.AddCutterDataToPlot("crosssection-ZX.dat",
                                                  "zx", ax,
                                                  unitsFactor=0.001)
```

The full documentation can be found for `pyg4ometry.visualisation.Plot.AddCutterDataToPlot()`.

1.5 Validating Geometry

1.5.1 Overlap Checking

“Overlaps” is a general term used to describe malformed geometry. Such geometry is unphysical and may causing particle tracking problems in simulations such as stuck particles, or particles completely missing certain volumes entirely. Such errors are rarely easy to spot from results or running the simulation.

Given all the PVs (daughters) of a LV (mother) should be bounded by the LV/mother solid. It is possible to check between all daughter solid meshes and between daughters and the mother solid mesh. Given an `pyg4ometry.geant4.LogicalVolume` instance (“lv”), this check can be performed by calling the following code:

```
lv.checkOverlaps()
```

This will check only the immediate daughters of this logical volume. To descend further into a geometry, the recursive flag can be used:

```
lv.checkOverlaps(recursive=True)
```

See `g4-module : LogicalVolume.checkOverlaps()` for full details. A more complete example is:

```
1 # cd pyg4ometry/test/pythonGeant4
2 import pyg4ometry
3 r = pyg4ometry.freecad.Reader("./T103_overlap_copl.gdml")
4 l = r.getRegistry().getWorldVolume()
5 l.checkOverlaps(recursive=False, coplanar=True, debugIO=False)
6 v = pyg4ometry.visualisation.VtkViewer()
7 v.addLogicalVolume(l)
8 v.view()
```

Text is by default only printed out when an overlap is found. Any overlaps will be prepared for visualisation in a `VtkViewer` (must be constructed and given the LV after this).

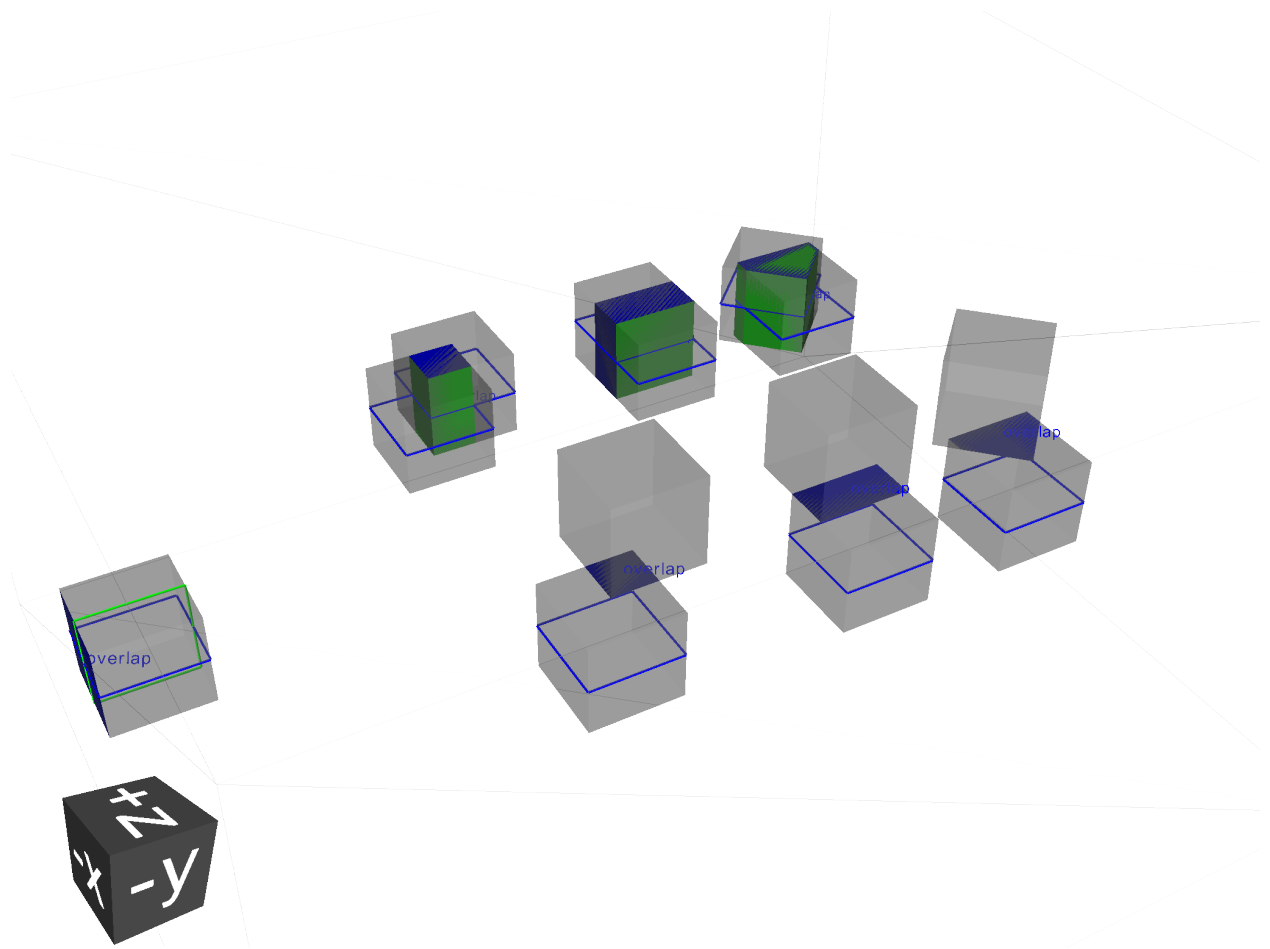
The following overlap checks are performed:

1. daughter with other daughter overlap
2. co-planar daughter with other daughter overlap
3. protrusion of a daughter from the mother volume
4. co-planar daughter with mother volume

Colour Coding

In the visualiser, text will be overlaid saying “overlap” where some kind of overlap is detected. Additionally, the actual overlap itself will be visualised and colour coded according to:

- red: protrusion overlap
- green: daughter-daughter overlap
- blue: co-planar overlap



Limitations

1. The overlap detection is performed by checking for overlaps in the visualisation meshes generated for each volume. In the case of curved solids (e.g. a cylinder), the mesh is not truly curved but a polygon. Very closely spaced curved surfaces may produce false overlaps. By default, all curved solids will use the same number of points around a circle, so usually we can “get away” with this if the curved solids aren’t rotated about their axis.
2. Currently, division and parameterised volumes are not handled explicitly.

Assemblies

In the case of assembly volumes, and if an overlap is detected, a unique name is built up based on the parent `PhysicalVolume`, the assembly and the `PhysicalVolume` inside it. Furthermore, this is done recursively as assemblies of assemblies (etc) are used. The name is built up with an underscore “_” for padding and the user should decode this from their input.

As there is no ‘mother’ of an assembly, there is no mother protrusion directly. The contents of an assembly are compared to all other daughters and the mother at the higher level in which they are placed.

1.6 Exporting Geometry

1.6.1 Registry and GDML Output

Strictly speaking a registry class to store all of the GDML is not required. As with normal Geant4 given a `lv` pointer it should be possible to form an aggregation hierarchy that contains all necessary objects. Now GDML breaks this as the structure is built up using name tags. For example a placement requires a position. In Geant4 this would just be a pointer to a transformation object, but GDML has two mechanisms to represent this, firstly child nodes of a `PhysicalVolume` tag or secondly a position define, see below

The registry class is a storage class for a complete GDML file. At the construction stage of almost all objects a registry is required. If the object is added to the registry then it will appear explicitly in the GDML output

1.6.2 GDML Output

To write an GDML file, a `pyg4ometry.geant4.registry` instance (`reg` here), must be supplied.

```

1 import pyg4ometry
2 w = pyg4ometry.gdml.Writer()
3 w.addDetector(reg)
4 w.write('file.gdml')
5 # make a quick bdsim job for the one component in a beam line
6 w.writeGmadTester('file.gmad', 'file.gdml')
```

1.6.3 Export scene to paraview/vtk

```
1 import pyg4ometry
```

1.6.4 Export scene to unity/unreal

The quickest way to get geometry to Unity/Unreal is to use a standard asset format. This takes a `VtkRenderer` and creates a OBJ file. The `VtkRenderer` managed within `pyg4ometry` from the `VtkViewer` class, once a geometry is created (either from any source) then an OBJ file can be created. Taking the example in `pyg4ometry/test/pythonCompoundExamples/`

```
1 import pyg4ometry
2 r = pyg4ometry.gdml.Reader("./Chamber.gdml")
3 l = r.getRegistry().getWorldVolume()
4 v = pyg4ometry.visualisation.VtkViewer()
5 v.addLogicalVolume(l)
6 v.exportOBJScene("Chamber")
```

obj files are written `Chamber.obj` and `Chamber.mtl`.

For a FLUKA file, first it must be converted to Geant4 and then the same process should be followed.

```
1 import pyg4ometry
2 r = pyg4ometry.fluka.Reader("./Chamber.inp")
3 greg = pyg4ometry.convert.fluka2Geant4(r.getRegistry())
4 l = greg.getWorldVolume()
5 v = pyg4ometry.visualisation.VtkViewer()
6 v.addLogicalVolume(l)
7 v.exportOBJScene("Chamber")
```

As the meshing might need to be changed for the visualisation application, the parameters for the meshing for each solid might need to be changed.

An obj file for an entire experiment does not help with work flows where meshes have to be UV-ed and textured. Tools like Blender and Gaffer can be used for this workload but require meshes for each object and their placement. To enable there is a special writer

```
1 import pyg4ometry
2
3 r = pyg4ometry.gdml.Reader("./Chamber.gdml")
4 l = r.getRegistry().getWorldVolume()
5 w = pyg4ometry.visualisation.RenderWriter()
6 w.addLogicalVolumeRecursive(l)
7 w.write("./SphericalChamber")
```

The directory `SphericalChamber` contains all the meshes in OBJ format along with an instance file `0_instances.dat` which contains a row for each instance of a mesh.

1.7 Creating Geometry

1.7.1 Precepts

- Units may be specified but default to Geant4 ones (e.g. mm, rad).
- Rotations are made using Tait-Bryan angles (rotation about reference x,y,z axes).
- A `Registry` object should be used to hold all things related in a model and passed into the constructors of most objects.
- GDML-like **full lengths** are used instead of typically half lengths

1.7.2 Units

- The default units are mm and rad for length and angle.
- Most constructors will take units as an optional key word argument ('kwarg').
- The kwargs are typically `lunit` for length unit and `aunit` for angle unit.
- Units are defined in `pyg4ometry.gdml.Units.py`.

The following units (as strings) are accepted:

Unit	Value
nm	1e-6
um	1e-3
mm	1
cm	10
m	1e3
km	1e6
deg	$\pi/180$
degree	$\pi/180$
rad	1
radian	1
mrad	1e-3
urad	1e-6
eV	1e-3
keV	1
MeV	1e+3
none	1
ns	1e-9
us	1e-6
ms	1e-3
s	1

Examples:

```
reg = pyg4ometry.geant4.Registry()
boxSolid = pyg4ometry.genat4.solid.Box("aBox", 10, 20, 30, reg)
```

This defines a box with the default units (none specified), so mm. We can specify them:

```
boxSolid = pyg4ometry.geant4.solid.Box("aBox", 10, 20, 30, reg, "cm")
```

1.7.3 Geant4 Python Scripting

Making use of pyg4ometry requires the following modules

```
import pyg4ometry
```

To make a simple geometry of a box located at the origin

```

1  # load pyg4ometry
2  import pyg4ometry
3
4  # registry to store gdml data
5  reg = pyg4ometry.geant4.Registry()
6
7  # world solid and logical
8  ws = pyg4ometry.geant4.solid.Box("ws", 50, 50, 50, reg)
9  wl = pyg4ometry.geant4.LogicalVolume(ws, "G4_Galactic", "wl", reg)
10 reg.setWorld(wl.name)
11
12 # box placed at origin
13 b1 = pyg4ometry.geant4.solid.Box("b1", 10, 10, 10, reg)
14 b1_l = pyg4ometry.geant4.LogicalVolume(b1, "G4_Fe", "b1_l", reg)
15 b1_p = pyg4ometry.geant4.PhysicalVolume([0, 0, 0], [0, 0, 0], b1_l, "b1_p", wl, reg)
16
17 # visualise geometry
18 v = pyg4ometry.visualisation.VtkViewer()
19 v.addLogicalVolume(wl)
20 v.addAxes(20)
21 v.view()
```

Here is the vtk visualiser output of the above example

1.7.4 GDML Defines

In GDML there are multiple define objects that can be used parameterise geometry, materials etc. These can be used as variables or definitions and mean that any equations used will be retained in GDML output. For example a GDML constant can be created in the following way

```

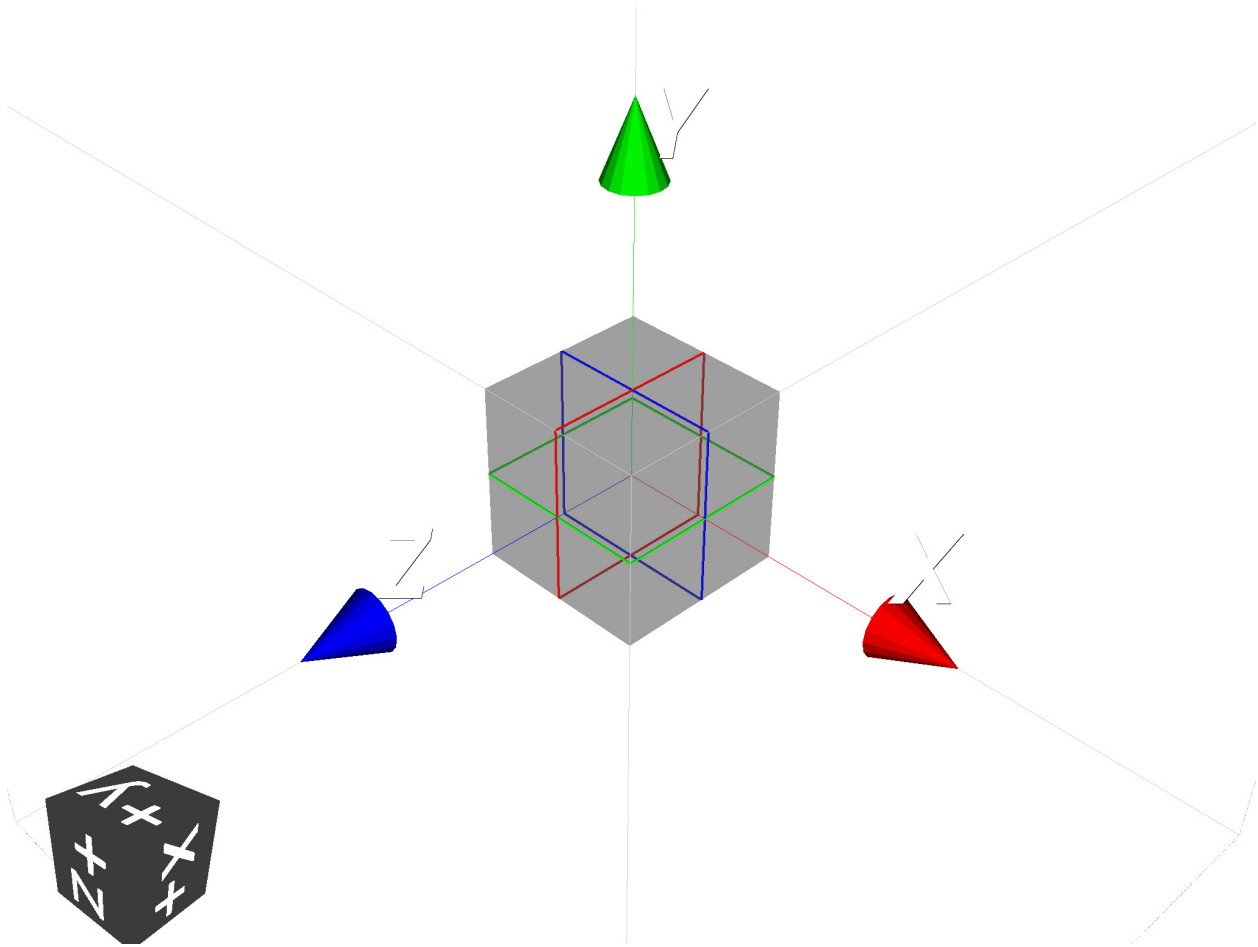
# registry to store gdml data
reg = pyg4ometry.geant4.Registry()

# constant called x
x = pyg4ometry.gdml.Constant("x", 10, reg)
```

The normal set of mathematical operations in python can be performed and evaluated

```

y = 2*x + 10
y.eval()
```



```
>> 30
```

The constant `x` can of course be changed and `y` re-evaluated

```
x.setExpression(5)
y.eval()
```

```
>> 20
```

Note: Standard mathematical functions can be used with GDML defines (Constant, Variable, etc). So `sin`, `cos`, `tan`, `exp` and so on, but `pyg4ometry` functions have to be used

```
1 x = pyg4ometry.gdml.Constant("x",10,reg)
2 cx = pyg4ometry.gdml.cos(x)
```

So the box example above can be rewritten using constants

```
1 # load pyg4ometry
2 import pyg4ometry
3
4 # registry to store gdml data
5 reg = pyg4ometry.geant4.Registry()
6
7 bx = pyg4ometry.gdml.Constant("bx","10",reg,True)
8 by = pyg4ometry.gdml.Constant("by",2*bx,reg,True)
9 bz = pyg4ometry.gdml.Constant("bz",2*by,reg,True)
10
11 # world solid and logical
12 ws = pyg4ometry.geant4.solid.Box("ws",50,50,50,reg)
13 wl = pyg4ometry.geant4.LogicalVolume(ws,"G4_Galactic","wl",reg)
14
15 # box placed at origin
16 b1 = pyg4ometry.geant4.solid.Box("b1",bx,by,bz,reg)
17 b1_l = pyg4ometry.geant4.LogicalVolume(b1,"G4_Fe","b1_l",reg)
18 b1_p = pyg4ometry.geant4.PhysicalVolume([0,0,0],[0,0,0],b1_l,"b1_p",wl,reg)
19
20 # visualise geometry
21 v = pyg4ometry.visualisation.VtkViewer()
22 v.addLogicalVolume(wl)
23 v.addAxes(20)
24 v.view()
```

Note: All GDML defines (Constant, Variable, etc) can be used in the construction of other `pyg4ometry` classes interchangeably instead of floats or strings (where strings are either numbers or a GDML expression)

Warning: Avoid reassigning variables used as defines, this can have unexpected consequences so for example

```
1 b1 = pyg4ometry.geant4.solid.Box("b1", bx, by, bz, reg)
2 b1.pX = 20 # do not do this
```

```
3 b1.pX.setExpression(20) # rather do this
```

1.7.5 Solids

The python geant4 solids match the Geant4 constructors as much possible (different constructor signatures are not supported in python). For example looking at the G4Box class

```
pyg4ometry.geant4.solid.Box(name, pX, pY, pZ, registry, lunit)
```

```
G4Box(const G4String& pName, G4double pX, G4double pY, G4double pZ)
```

A full list of solids can be found in all-solids.

Warning: The parameters stick to the GDML convention of **full** lengths opposed to half lengths.

1.7.6 Materials

As with solids materials are defined in a similar way to Geant4 C++. Python does not have overloaded constructors, so unique signatures are needed, in contrast to Geant4.

To define a material from the Geant4 predefined (e.g. NIST) materials

```
1 import pyg4ometry.geant4 as _g4
2 wm = _g4.MaterialPredefined("G4_Galactic")
3 bm = _g4.MaterialPredefined("G4_Fe")
```

To define a single element in terms of atomic number, atomic mass and density.

```
1 import pyg4ometry.geant4 as _g4
2 wm = _g4.MaterialSingleElement("galactic",1,1.008,1e-25,reg) # low density hydrogen
3 bm = _g4.MaterialSingleElement("iron",26,55.8452,7.874,reg) # iron at near room temp
```

To define a compound two elements using the mass fraction

```
1 import pyg4ometry.geant4 as _g4
2 wm = _g4.MaterialCompound("air",1.290e-3,2,reg)
3 ne = _g4.ElementSimple("nitrogen","N",7,14.01)
4 oe = _g4.ElementSimple("oxygen","O",8,16.0)
5 wm.add_element_massfraction(ne,0.7)
6 wm.add_element_massfraction(oe,0.3)
7 bm = _g4.MaterialSingleElement("iron",26,55.8452,7.874,reg) # iron at near room temp
```

To define a compound using number of atoms

```
1 import pyg4ometry.geant4 as _g4
2 bm = _g4.MaterialCompound("plastic",1.38,3,reg) # Generic PET C_10 H_8 O_4
3 he = _g4.ElementSimple("hydrogen","H",1,1.008)
4 ce = _g4.ElementSimple("carbon","C",6,12.0096)
5 oe = _g4.ElementSimple("oxygen","O",8,16.0)
```

(continues on next page)

(continued from previous page)

```

6 bm.add_element_natoms(he,8)
7 bm.add_element_natoms(ce,10)
8 bm.add_element_natoms(oe,4)

```

Material as a mixture of materials

```

1 import pyg4ometry.geant4 as _g4
2 bm = _g4.MaterialCompound("YellowBrass_C26800", 8.14, 2, reg)
3 copper = _g4.MaterialPredefined("G4_Cu")
4 zinc = _g4.MaterialPredefined("G4_Zn")
5 bm.add_material(copper, 0.67)
6 bm.add_material(zinc, 0.33)

```

Example of elements formed by isotopes

```

1 import pyg4ometry.geant4 as _g4
2 u235 = _g4.Isotope("U235", 92, 235, 235.044)
3 u238 = _g4.Isotope("U238", 92, 238, 238.051)
4 uranium = _g4.ElementIsotopeMixture("uranium", "U", 2)
5 uranium.add_isotope(u235, 0.00716)
6 uranium.add_isotope(u238, 0.99284)
7 bm = _g4.MaterialCompound("natural_uranium", 19.1, 1, reg)
8 bm.add_element_massfraction(uranium, 1)

```

NIST Materials

Geant4 has many predefined materials according to the NIST database. Their name typically starts with G4_. These typically can be used with `MaterialPredefined` and we **do not need** to specify the full composition - Geant4 will find them at run time.

However, in the case of conversion to FLUKA, these are fully expanded according to their definition in Geant4 based on a cache in pyg4ometry of the material compositions generated using BDSIM from Geant4 (10.7.p01 as of writing). Should the user wish to use these, they can be accessed from the functions in the geant4 module.

```

1 import pyg4ometry
2 nistHydrogenElement = pyg4ometry.geant4.nist_element_2geant4Element('G4_H')

```

Note, an ‘element’ cannot be used as a ‘material’ in a logical volume. We must upgrade it to a material for that. The NIST elements contain the appropriate mixture of natural isotopes and can be used in `MaterialCompound` as demonstrated above.

Alternatively, we can access the NIST materials and materials of elements.

```

1 import pyg4ometry
2 nistHydrogenMaterial = pyg4ometry.geant4.nist_material_2geant4Material('G4_H')
3 nistConcreteMaterial = pyg4ometry.geant4.nist_material_2geant4Material('G4_CONCRETE')

```

1.7.7 Detector Construction

This largely proceeds in exactly the same way as in G4 or GDML. Hierarchy of solids, booleans, logical, physical (replica, division, param) volumes.

0. Create registry to hold everything
1. Create solids
2. Create logical volumes
3. Place logical volumes (construct physical volumes)
4. Visualise
5. Check
6. Export

1.7.8 Transformations & Physical Volumes

Transformations in 3D are essential for the easy placement of solids in a CSG tree or LV placement. There is not a specific transformation class like in Geant4. The matrices and vectors used for placements are here typically Numpy arrays or matrices.

Geant4 has two possible constructors for a physical volume. These provide active and passive transformations. In pyg4ometry, only one is provided.

- The transform in a physical volume first translates the placed logical volume with respect to the mother logical, then rotates it.

The physical volume class is documented here: [g4-module](#), but an example is shown here.

```

1 import pyg4ometry
2
3 r = pyg4ometry.geant4.Registry()
4 vacuum = _g4.MaterialPredefined("G4_Galactic")
5 water = _g4.MaterialPredefined("G4_WATER")
6 worldSolid = pyg4ometry.geant4.solid.Box("world_solid", 100, 100, 100, reg)
7 boxSolid = pyg4ometry.geant4.solid.Box("box_solid", 10, 20, 40, reg)
8 worldLV = pyg4ometry.geant4.LogicalVolume(worldSolid, vacuum, "world_lv", reg)
9 boxLV = pyg4ometry.geant4.LogicalVolume(boxSolid, water, "box_lv", reg)
10
11 pyg4ometry.geant4.PhysicalVolume([0, 0, 0], [0, 0, 0], boxLV, "box_pv", worldLV, reg)

```

This creates a box of water inside a box of vacuum. The box of water is 10 x 20 x 50 mm long (note mm are the default length units), and it is placed with no offset and no rotation (i.e. at the centre) of the world volume. Alternatively:

```

1 import numpy as np
2
3 pyg4ometry.geant4.PhysicalVolume(
4     [0, np.pi / 3.0, 0], [0, 0, 0], boxLV, "box_pv", worldLV, reg
5 )

```

In this case, the box is placed with no offset but with a rotation of $\pi/3$ radians about the y axis of the world box.

Note: The rotations are Tait-Bryan angles, which are rotations about the reference x,y,z axes. i.e. if there is a rotation about both x and y, these are independent and it is **not** a compound frame that is rotated. These are commonly thought

of like an aircraft and called pitch, yaw and tilt.

There are utility functions for translation between different transformations in `pyg4ometry.transformation`. See `transformation-module`.

1.7.9 Optical Surfaces

Optical surfaces can be created in a similar way as in Geant4 C++. A `pyg4ometry.geant4.solid.OpticalSurface` instance holds all the needed properties of the surface (including extra properties, e.g. for optical processes). This is then assigned to the surface between either

- two physical volumes: `pyg4ometry.geant4.BorderSurface`, or
- a logical volume and all its neighbouring volumes: `pyg4ometry.geant4.SkinSurface`.

```
1 opa = _g4.solid.OpticalSurface(  
2     "AirSurface",  
3     finish="polished",  
4     model="glisur",  
5     surf_type="dielectric_dielectric",  
6     value="1",  
7     registry=reg,  
8 )  
9 opw = _g4.solid.OpticalSurface(  
10     "WaterSurface",  
11     finish="ground",  
12     model="unified",  
13     surf_type="dielectric_dielectric",  
14     value="0",  
15     registry=reg,  
16 )  
17  
18 _g4.SkinSurface("AirSurface", air_lv, opa, reg)  
19 _g4.BorderSurface("WaterSurface", water_phys, world_phys, opw, reg)
```

1.7.10 Properties of Materials and Optical Surfaces

Materials and optical surfaces support adding properties that can be used by Geant4 to influence processes, e.g. for scintillation, refraction or other optical processes.

In the GDML, a matrix is used to hold the value(s) of the property.

- `addProperty(name, matrix)` - Add a property based on an existing `pyg4ometry.gdml.Matrix` object.
- `addVecProperty(name, e, v, eunit='eV', vunit='')` - Add a property based on a energy vector and a value vector.
- `addConstProperty(name, value, vunit='')` - Add a property that has only one constant value.

Units can be specified by setting the parameters `eunit` for the energy vector and `vunit` for the values. The given vectors are expected to be homogeneous in their units.

Note: Optical properties can only use units (or combinations of units) that are also defined in `pyg4ometry`. If needed, additional units can be added: `pyg4ometry.gdml.Units.units['ps'] = 1e-12`.

```

1 scint = _g4.Material(...)
2 scint.addConstProperty("SCINTILLATIONTIMECONSTANT1", 2.5, vunit="ns")
3 scint.addConstProperty("SCINTILLATIONYIELD", 8000, vunit="/MeV")
4 scint.addVecProperty("RINDEX", [1, 10], [1.3, 1.05])

```

1.7.11 FLUKA Geometry Creation

In a very similar way to geant4 geometry authoring it is possible to use pyg4ometry to create fluka output. To create a simple region consisting of a single body

```

1 import pyg4ometry.convert as convert
2 import pyg4ometry.visualisation as vi
3 from pyg4ometry.fluka import RPP, Region, Zone, FlukaRegistry
4
5 freg = FlukaRegistry()
6
7 rpp = RPP("RPP_BODY", 0, 10, 0, 10, 0, 10, flukaregistry=freg)
8 z = Zone()
9 z.addIntersection(rpp)
10 region = Region("RPP_REG", material="COPPER")
11 region.addZone(z)
12 freg.addRegion(region)
13
14 greg = convert.fluka2Geant4(freg)
15 greg.getWorldVolume().clipSolid()
16
17 v = vi.VtkViewer()
18 v.addAxes(length=20)
19 v.addLogicalVolume(greg.getWorldVolume())
20 v.view()

```

1.8 Converting Geometry

1.8.1 Assembly Conversion

It is possible transform a logical volume into an assembly volume. This is useful in the case of two sources of geometry: placement of top level world logical volume solids will likely result in an overlap. This effectively removes the outermost logical volume but keeps the daughters. The produced assembly volume can then be placed or imprinted somewhere in the geometry.

An assembly **cannot** be used as an outermost ‘world’ volume for a geometry hierarchy.

Assuming lv is a pyg4ometry.geant4.LogicalVolume instance:

```
av = lv.assemblyVolume()
```

1.8.2 GDML to FLUKA

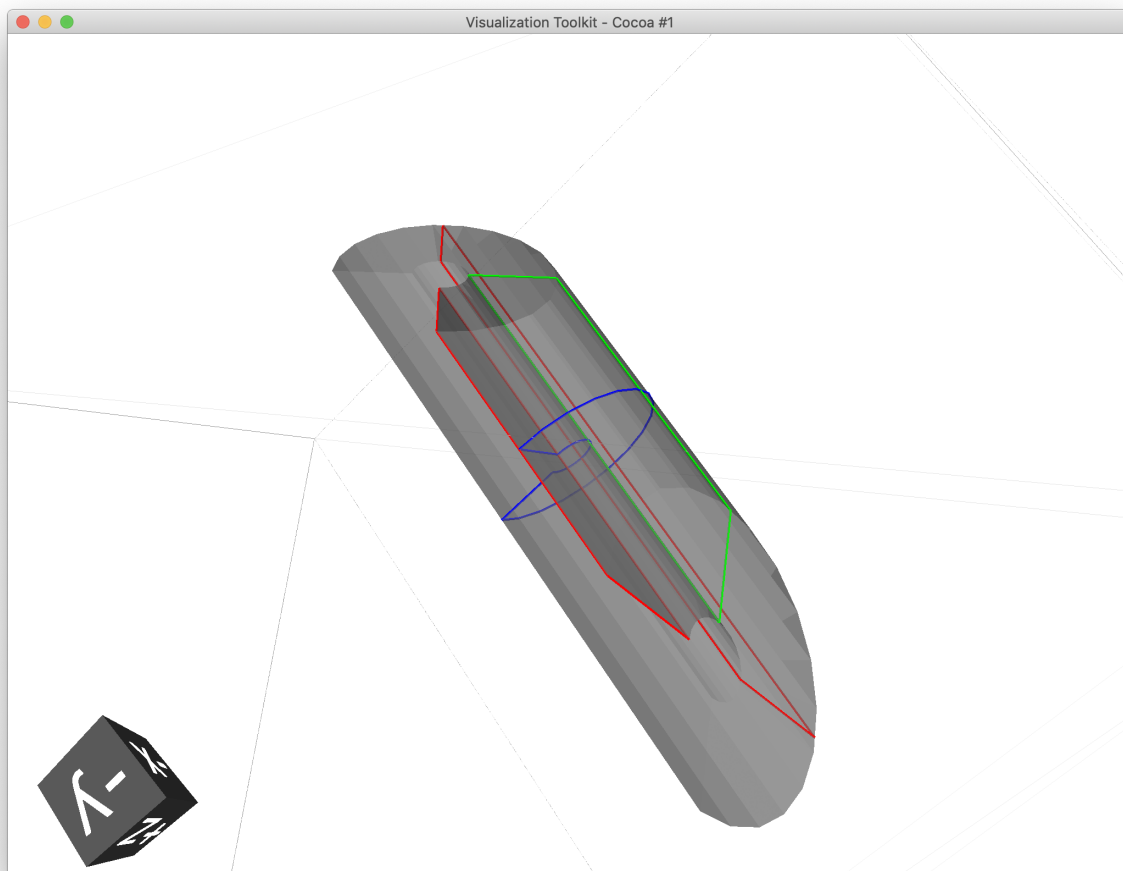
It is possible to convert a pyg4ometry geometry to FLUKA. This is currently a work in progress and not all Geant4-GDML constructions are implemented, although they can be quickly added. Given a LV variable named `logical`

```
1 import pyg4ometry
2 reader = pyg4ometry.gdml.Reader("input.gdml")
3 logical = reader.getRegistry().getWorldVolume()
4 freg = pyg4ometry.convert.geant4Logical2Fluka(logical)
5 w = pyg4ometry.fluka.Writer()
6 w.addDetector(freg)
7 w.write("FileName.inp")
```

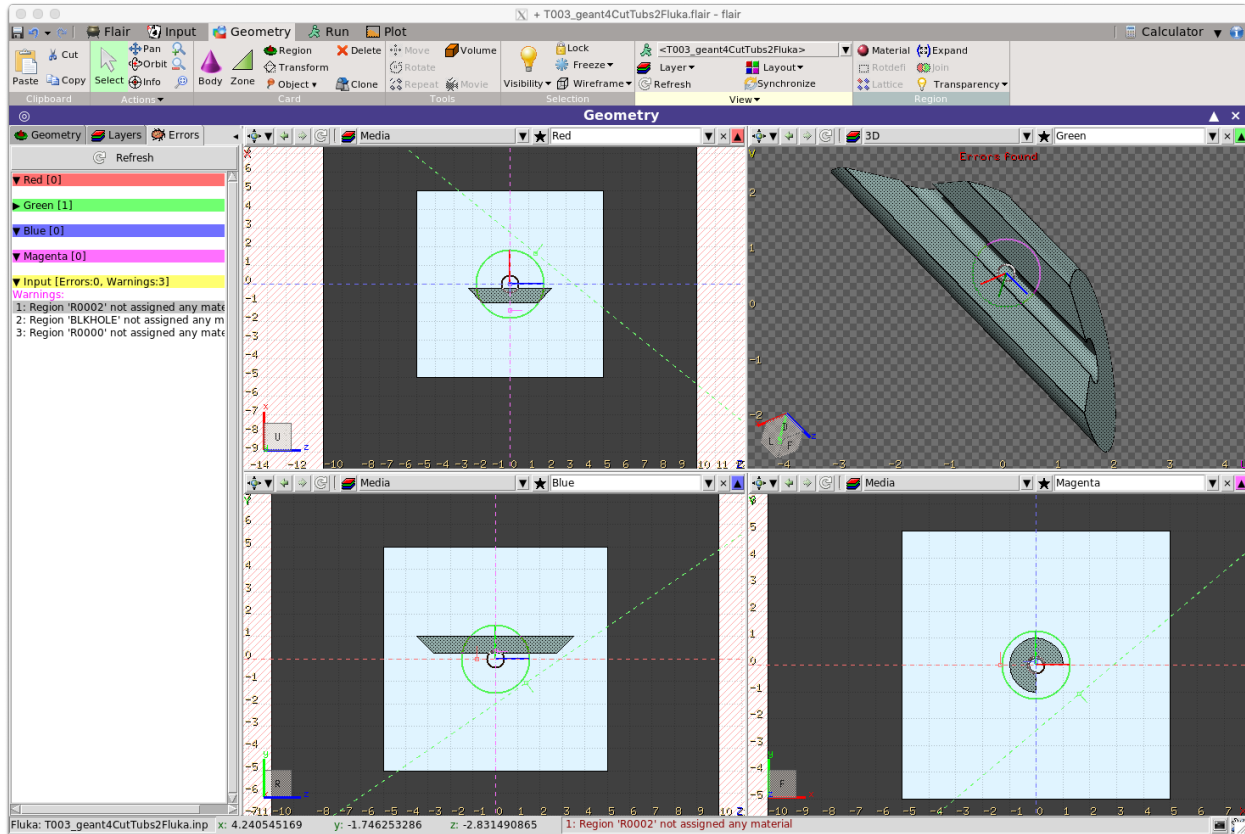
If you want to load a file into Flair then a flair file can be written based on `FileName.inp` using the following

```
1 extent = logical.extent(includeBoundingSolid=True)
2 f = pyg4ometry.fluka.Flair("FileName.inp", extent)
3 f.write("FileName.flair")
```

Here is an example (viewed in Flair) of a simple Geant G4 solid that has been converted to FLUKA using this method



Note: All GDML placements are respected in the conversion from GDML to FLUKA, for both Placements and



Boolean Solids. So for example a tree of LV-PV placements are reduced into a single transformation of a LV into a global coordinate space for FLUKA. A similar process is used for a tree of CSG operations.

Warning: Currently there are some things which are not implemented in the conversion. 1) Materials, 2) Scaled solids, 3) Reflections in placements, 4) Division, replica and parameterised placements. Some of these are straight forward to implement, like Materials and the non-Placement physical volumes can be done quickly if a user requires it.

1.8.3 FLUKA To GDML

FLUKA geometry can be converted to GDML using `pyg4ometry.convert.fluka2geant4`. The conversion process is robust and supports all FLUKA geometry constructs. Given a FLUKA file `model.inp`, the following code can be used to translate it to a GDML file.

```
1 import pyg4ometry.fluka as fluka
2 import pyg4ometry.gdml as gdml
3 from pyg4ometry.convert import fluka2Geant4
4
5 # Read the FLUKA file, get the FlukaRegistry, convert the registry to a
6 # Geant4 Registry
7 reader = fluka.Reader("model.inp")
8 flukaregistry = reader.flukaregistry
```

(continues on next page)

(continued from previous page)

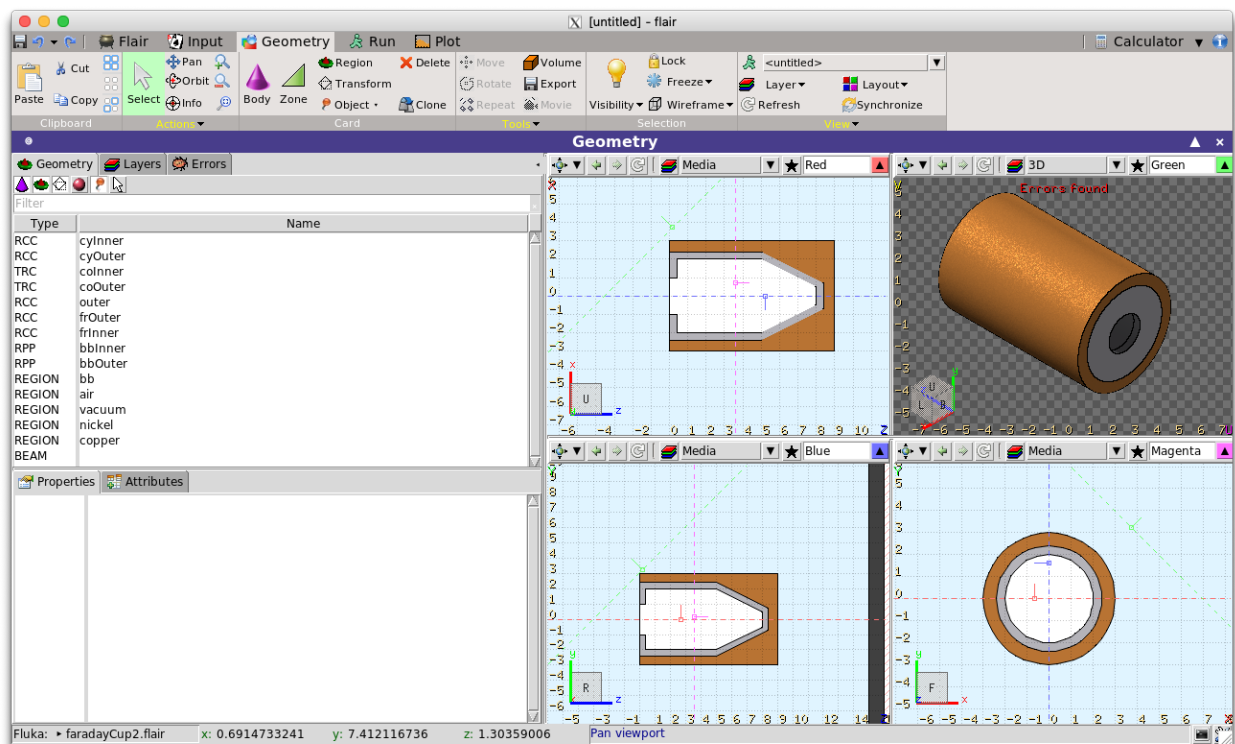
```

9 geant4Registry = fluka2Geant4(flukaRegistry)
10
11 worldLogicalVolume = geant4Registry.getWorldVolume()
12 worldLogicalVolume.clipSolid()
13
14 writer = gdml.Writer()
15 writer.addDetector(geant4Registry)
16 writer.write("model.gdml")

```

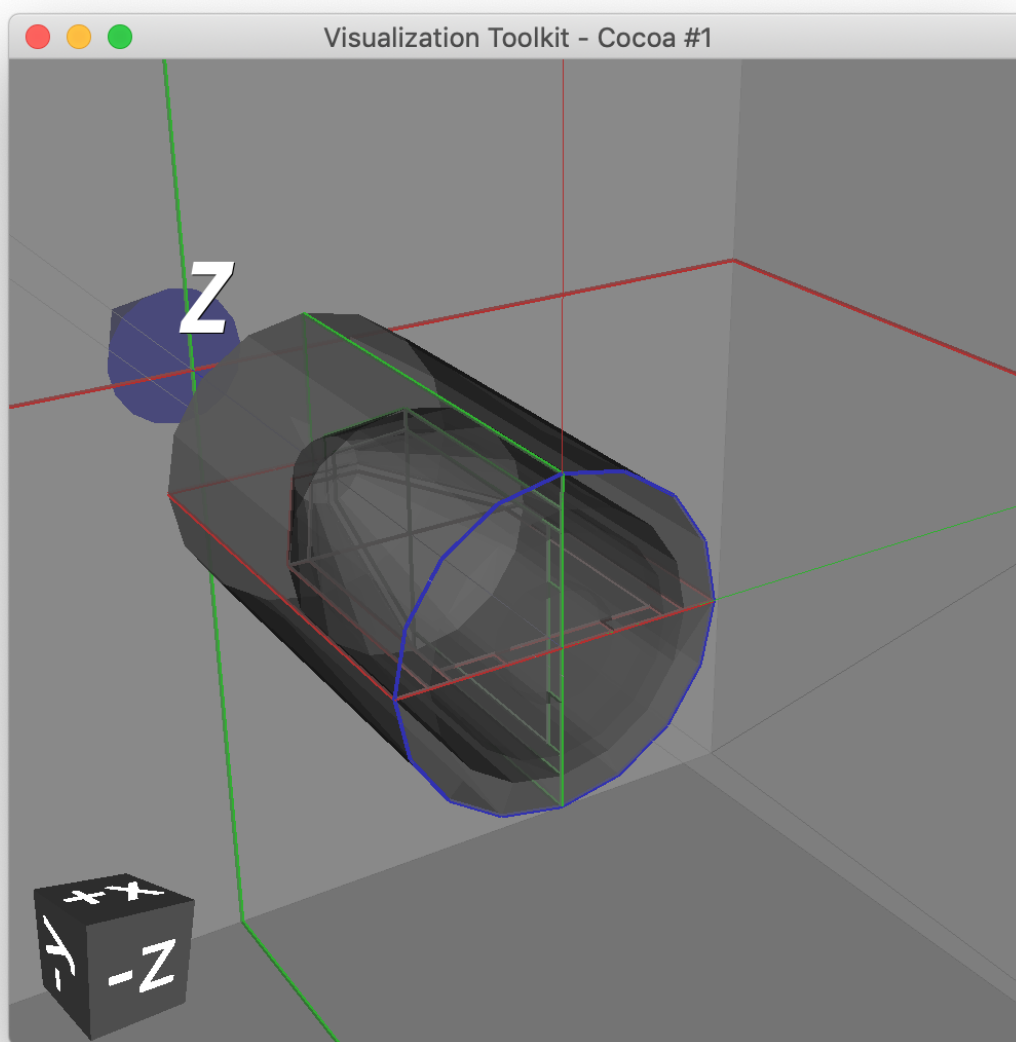
The core of this functionality is the translation of the *FlukaRegistry* instance into the equivalent *Registry* (i.e. *Geant4*) instance.

Here is an example of a model viewed in flair and the resulting visualisation in VTK of the Geant4 model



A number of keyword arguments are available to further modify the conversion. The *fluka2Geant4* keyword arguments *region* and *omitRegions* allow the user to select a subset of the named regions to be translated.

The conversion of QUA bodies (*fluka2geant4* kwarg *quadricRegionAABBs*) is complex and requires further explanation. In Pyg4ometry the mesh and GDML representations of FLUKA infinite circular cylinders, elliptical cylinders and half-spaces are all finite (but very large) cylinders, elliptical cylinders and boxes. This is robust as increasing the length of cylinders and depth/breadth of boxes does not increase the number of polygons used in the underlying mesh representation for that solid. However, this is not true of the quadric surface. A quadric surface cannot simply be generated to be “very large”, as the number of polygons will grow quickly, along with the memory consumption and facets in the resulting GDML TesselatedSolid, which will also slowing down tracking time in Geant4. For this reason the user must provide axis-aligned bounding boxes of the regions where any QUA bodies are present. It is recommended that these boxes be a centimetre larger than formally necessary to ensure a correct conversion. Providing the bounding box ensures that an efficient and accurate mesh of the QUA bodies can be generated meaning that the conversion to be performed in a tractable amount of time as well giving more performant tracking in Geant4.



1.9 Combining Geometry

There are ways to incorporate geometry from multiple sources in GDML. This has potentially lots of problems as each file needs to be a well formed GDML file and care has to be taken with degenerate names from the different sources. For example a volume can be extracted from GdmlFile1 and added to GdmlFile2, clearly all solids, materials and variables need to also transferred. For this example we need two GDML files, so `pyg4ometry/test/pythonGeant4/T001_Box.py` and `pyg4ometry/test/pythonGeant4/T002_Tubs.py`, so run them

```

1 import T001_Box
2 T001_Box.Test(True, True)
3
4 import T002_Tubs
5 T002_Tubs(True, True)

```

This will create two GDML files `T001_Box.gdml` and `T002_Tubs.gdml`. It is possible to find the volumes contained in each file (using the `tubs.gdml` file as the example) by performing the following

```

1 import pyg4ometry
2 r = pyg4ometry.gdml.Reader("T002_Tubs.gdml")
3 reg = r.getRegistry()
4
5 # printing the names of the logical volumes
6 print(reg.logicalVolumeDict.keys())
7
8 # printing the names of the physical volumes
9 print(reg.physicalVolumeDict.keys())
10
11 lv = reg.logicalVolume["t1"]

```

Now merging the `t1` logicalVolume (which is a simple `tubs`) with the `box.gdml` file

```

1 import pyg4ometry
2 r1 = pyg4ometry.gdml.Reader("T001_Box.gdml")
3 reg1 = r1.getRegistry()
4
5 r2 = pyg4ometry.gdml.Reader("T002_Tubs.gdml")
6 reg2 = r2.getRegistry()
7
8 lv = reg2.logicalVolumeDict["t1"]
9
10 # create physical volume with placement
11 pv = pyg4ometry.geant4.PhysicalVolume([0,0,0],[50,0,0], lv, "t1_pv", reg1.
    ↪ getWorldVolume(), reg1)
12
13 reg1.addVolumeRecursive(pv)
14
15 # gdml output
16 w = pyg4ometry.gdml.Writer()
17 w.addDetector(reg1)
18 w.write("MergeRegistry.gdml")

```

Note: In the example two registry objects are created and objects from `reg2` are merged into `reg1`. Of course one

registry might be formed by pyg4ometry commands opposed created from a file.

Warning: The pv needs to added with addVolumeRecursive otherwise it is possible that GDML definitions which lv depends on are not transferred over.

1.10 Comparing Geometry

When converting or preparing geometry, it is useful to compare to other geometry to validate work. Examples of use include:

- validating changes to existing pyg4ometry scripts
- validating geometry created in pyg4ometry versus another source
- comparing different representations
- validating conversion between formats

pyg4ometry provides a set of tests for comparing two geometry “trees” that are loaded into memory. Each hierarchy of geometry is navigated and a variety of user-selectable tests are carried out. A result class with details of tests carried out and their outcome is produced and can be easily printed or inspected programmatically.

When we compare 2 things, we use the terminology **reference** and **other** for each. When we need to identify a pair, we use the reference object name. Reference is the object that is the control, or assumed to be the ‘right’ one.

1.10.1 Tests

Which tests to conduct are defined by an instance of `pyg4ometry.compare.Tests`, which has a set of Boolean options for each test.

The available tests are described below.

Test Name	Description
names	Test whether objects have the exact same name.
namesIgnorePointers	Test whether objects have the exact same name, but whilst ignoring pointer suffixes such as 0x1234567. Note, the geometry will load ok, the name stripping is only for comparison.
nDaughters	Test for a matching number of daughter volumes in a LogicalVolume or AssemblyVolume.
solidExact	Compare the class type of every solid.
shapeExtent	Numerically compare the size of the bounding box (i.e. extent) of each solid.
shapeVolume	Compare the volume as per the visualisation mesh of each shape.
shapeArea	Compare the surface area as per the visualisation mesh of each shape.
placement	Numerically compare the rotation and translations in each placement.
scale	Compare the 'scale' for a physical volume that can be used for a reflection. No scale is equal to unit scale [1,1,1].
copyNumber	Compare the copy number for a physical volume.
materials	Whether to do a series of tests of materials in a LogicalVolume.
materialClassType	If doing materials tests, compare the class name of each material.
materialComposition-Type	If doing materials tests, compare the class name of each component of a material.
testDaughtersByName	Whether to check daughter volumes by name or by index. If true, then a set of overlapping names is used and the matching ones that exist in both are compared. If false, the daughters are strictly compared in numerical order.

Note: Volumes and areas are based on the polygonal meshes generated for visualisation purposes. These may not be the exact area of a perfect shape, such as a sphere, but an approximation. They will however, be consistently generated. Consider the tolerances also.

Tolerances

When making numerical comparisons (e.g. position of a volume), we must consider the finite numerical precision in a computer. Also, this may be useful as geometry from another source may be practically equivalent, but not exactly at the last decimal place.

Therefore, we include a set of tolerances for numerical comparisons that are included in the Tests class. These are typically fractional.

Note: The fraction is calculated with respect to the reference value. So $(value_{other} - value_{reference}) / value_{reference}$.

Variable Name	Default Value	Description
toleranceSolidParameterFraction	1e-3	Maximum fractional difference for any solid parameter.
toleranceSolidExtentFraction	1e-6	Maximum fractional difference for the calculated extent of any solid.
toleranceVolumeFraction	1e-2	Maximum fractional difference in volume.
toleranceAreaFraction	1e-2	Maximum fractional difference in area.
toleranceTranslationFraction	1e-6	Maximum fractional difference for translations in placements.
toleranceScaleFraction	1e-3	Maximum fractional difference in scale parameters in placements.
toleranceRotationFraction	1e-6	Maximum fractional difference in rotation angles in placements.
toleranceMaterialDensityFraction	1e-4	Maximum fractional difference in the density of a material.
toleranceMaterialMassFraction	1e-4	Maximum fractional difference in the mass fraction of a material component.

1.10.2 Running a Comparison

See also `geometry-compare-module` for all functions.

A comparison is run by using a function in `pyg4ometry.compare`, giving it two objects and a Tests instance. The comparison result returned is a class with a list of test results that can be inspected and also printed.

GDML Files

```
>>> comparison = pyg4ometry.compare.gdmlFiles("file_reference.gdml", "file_other.gdml")
>>> comparison.print()
```

Logical Volumes

Compare two logical volumes, such as the top volume of a registry from a loaded file.

```
>>> comparison = pyg4ometry.compare.geometry(refLV, otherLV)
>>> comparison.print()
```

There are functions to compare individual objects too. These follow the same pattern. A few are:

- `pyg4ometry.compare.physicalVolumes`
- `pyg4ometry.compare.assemblyVolumes`
- `pyg4ometry.compare.replicaVolumes`
- `pyg4ometry.compare.divisionVolumes` - not implemented yet
- `pyg4ometry.compare.parameterisedVolumes` - not implemented yet
- `pyg4ometry.compare.materials`
- `pyg4ometry.compare.solids`

Example Output

For a test that failed the following is an example of output.

```
Overall result> TestResult.Failed
Test> position
(av): a_assembly: (pv): a_a_pv1: a_a_pv1_pos: TestResult.Failed: z: (reference): 100.0,
↪(other): -100.0

Test> shapeExtentBoundingBoxMin
a_assembly_a_a_pv1: TestResult.Failed: axis-aligned bounding box lower edge: dimension:
↪z, (reference): 85.0, (other): -115.0

Test> shapeExtentBoundingBoxMax
a_assembly_a_a_pv1: TestResult.Failed: axis-aligned bounding box upper edge: dimension:
↪z, (reference): 115.0, (other): -85.0
```

1.10.3 Seeing the Results

The return of a comparison is a `pyg4ometry.compare.ComparisonResult` instance. The most useful function here is the print function, which will print all results but also you can print select sets of tests or only certain results.

- See `geometry-compare-module` and `ComparisonResult` for details.

1.10.4 Examples

See `pyg4ometry/test/pythonGeant4/T7*.py` for examples that form the tests of this code.

Only Volume

Here, two ways are shown for creating the set of tests.

```
>>> import pyg4ometry
>>> t = pyg4ometry.compare.Tests()
>>> pyg4ometry.compare.Tests.printAllTestNames()
"names"
"namesIgnorePointer"
"nDaughters"
"solidExact"
"solidExtent"
"shapeExtent"
"shapeVolume"
"shapeArea"
"placement"
"scale"
"copyNumber"
"materials"
"materialClassType"
"materialCompositionType"
"testDaughtersByName"
>>> t.setAllFalse()
```

(continues on next page)

(continued from previous page)

```
>>> t.shapeVolume = True
>>> comparison = pyg4ometry.compare.gdmlFiles("file1.gdml", "file2.gdml", t)
>>> comparison.print()
```

or

```
>>> import pyg4ometry
>>> t2 = pyg4ometry.compare.Tests("shapeVolume")
>>> comparison = pyg4ometry.compare.gdmlFiles("file1.gdml", "file2.gdml", t2)
>>> comparison.print()
```

Removing a Test

A test can be turned off by name:

```
>>> import pyg4ometry
>>> t = pyg4ometry.compare.Tests()
>>> t.setFalse("names")
```

1.11 Analysing Geometry

1.11.1 Finding volumes

Before editing geometry it is useful to find a logical volume. The registry contains all the logical volumes using the GDML in `pyg4ometry/test/gdmlG4examples/ChargeExchangeMC/`

```
1 import pyg4ometry
2 r = pyg4ometry.gdml.Reader("lht.gdml")
3 reg = r.getRegistry()
```

The registry instance `reg` has a member variable called `logicalVolumeDict` so calling

```
reg.logicalVolumeDict.keys()
```

should print

```
In [4]: reg.logicalVolumeDict.keys()
Out[4]: odict_keys(['vMonitor', 'vMonitorBack', 'vTarget', 'vTargetInnerCover',
→ 'vTargetColumn', 'vTargetInnerColumn',
→ 'vTargetVacuumSpace', 'vTargetOuterCover', 'vCrystal', 'vCrystalRow',
→ 'vCalorimeter', 'vVetoCounter',
→ 'vOuterFerrumRing', 'vInnerFerrumRing', 'vInnerCuprumRing',
→ 'vTargetWindow', 'vTargetWindowCap',
→ 'vTargetWindowMylarCover', 'vTargetWindowAluminiumCover',
→ 'vWorldVisible', 'World'])
```

then the LogicalVolume can be obtained simply from the dictionary

```
lv = reg.logicalVolumeDict['vTargetInnerColumn']
```

This `lv` can be used for manipulating geometry, passing to visualisers etc.

1.11.2 Navigating the LV-PV hierarchy

There is a hierarchy of LV-PVs to describe a GDML/Geant4 geometry. An LV in terms of geometry consists of an outer solid `lv.solid` and `lv.daughterVolumes`. `lv.solid` is one of the `pyg4ometry.geant4.solid` types which match the GDML/Geant4 solids. `lv.daughterVolumes` is a list of `pyg4ometry.geant4.PhysicalVolumes`.

The best way to explore the methods and data members of `pyg4ometry.geant4.LogicalVolume` and `pyg4ometry.geant4.PhysicalVolume` is to explore in iPython. See cute video based on the `lht.gdml` example above.

1.11.3 Geometry Complexity Analysis

For a given logical volume we can get some statistics on the complexity of the geometry. A simple class called *GeometryComplexityInformation* is returned that has a series of dictionaries with information.

```
cd pyg4ometry/test/gdmlCompoundExamples/bdsim_2
ipython
>>> import pyg4ometry
>>> r = pyg4ometry.gdml.Reader("22-size-variation-facetcrop-quad.gdml")
>>> info = pyg4ometry.geant4.AnalyseGeometryComplexity(r.getRegistry().getWorldVolume())
>>> info.printSummary()
Types of solids
ExtrudedSolid      : 96
Tubs               : 51
Intersection       : 24
Polyhedra         : 12
Subtraction        : 6
Box               : 1

# of daughters      count
0                  : 152
2                  : 19
4                  : 12
13                 : 6
25                 : 1

Depth of booleans   count
1                   : 30

Booleans width depth over 3
Solid name          : n Booleans

>>> info. <tab>
comp.booleanDepth    comp.nDaughtersPerLV
comp.booleanDepthCount comp.printSummary
comp.nDaughters      comp.solids
```

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

REFERENCING & CITATION

Important: Citation information can be obtained on GitHub by pointing your browser to github.com/g4edge/pyg4ometry and selecting “Cite this repository” in the sidebar on the right. pyg4ometry uses a [CITATION.cff](#) file.

Citations allow the authors to demonstrably show the code usage and therefore to continue to support the development and maintenance of pyg4ometry.

Any publications including simulations made using pyg4ometry **must** cite:

S.D. Walker, A. Abramov, L.J. Nevay, W. Shields, S.T. Boogert, “*pyg4ometry: A Python library for the creation of Monte Carlo radiation transport physical geometries*”, Computer Physics Communications **272** 108228 (2022). DOI: [10.1016/j.cpc.2021.108228](https://doi.org/10.1016/j.cpc.2021.108228)

Additionally, we automatically generate a DOI for each pyg4ometry code release through Zenodo at [this page](#), if you wish to cite a specific pyg4ometry version.

Other publications on pyg4ometry:

- S.T. Boogert, A. Abramov, J. Albrecht, G. D’Alessandro, L.J. Nevay, W. Shields, S.D. Walker, *Pyg4ometry : A Tool To Create Geometries For Geant4, Bdsim, G4Beamline and Fluka For Particle Loss and Energy Deposit Studies*, IPAC2019, Melbourne, Australia, 2019. DOI: [10.18429/JACoW-IPAC2019-WEPTS054](https://doi.org/10.18429/JACoW-IPAC2019-WEPTS054)

CHANGELOG (LEGACY)

V1.0.3 - 2022 / 08 / XX

New features

- Feature extraction from tessellated meshes (with some accelerator applications)
- New opencascade based step/iges loading
- Command line interface
- Clipping of geometry when top level solid is moved (rotated/translated) or changed
- New more efficient visualisation

4.1 V1.0.2 - 2022 / 04 / 11

4.1.1 New Features

- Function to view the difference between two logical volumes.
- Comparison test for names that ignores pointers as part of the name.
- New `addSolid` function for visualiser to view a solid in the current scene without having a logical volume.
- Every solid now has the function `convert2Tessellated` to allow it to be easily converted to a tessellated solid.
- Ability to 'collapse' assemblies (T. Latham)

4.1.2 General

- All solids now have a list of units to accompany their list of variables for inspection.
- Checking angles for solids (e.g. start and sweep angle) are within 2π now includes a numerical precision tolerance that is by default floating point precision. See `pyg4ometry.config.twoPiComparisonTolerance`.

4.1.3 Bug Fixes

- Fix deployment of nist_materials.txt and nist_elements.txt package files.
- Fix loading of ROOT geometry for material and various solids. Put tests in for parameters that ROOT allows to be 0 but ultimately mean we cannot construct a valid shape from.
- Reduce length of names in Geant4 to FLUKA conversion to fix rejected names by FLUKA.
- Fix import of MutableMappings for Python 3.10
- Fix use of units throughout all solids for the geometry comparison tests.
- Fix units for define vectors.
- Fix copy number of physical volumes sometimes not being an integer for loaded GDML geometry.
- Fix reading and writing of abundance if it is an expression in a GDML material.
- Fix missing length unit for GDML writing of Generic Trap.
- Fix units in GenericTrap and Extruded solid classes.
- Fix renaming of materials (in a recursion chain) when transferring from one registry to another (T. Latham).
- Reduce verbosity of ROOT tests.
- Reduce verbosity of comparison tests.
- Fix zero division errors in various comparison tests.

4.2 v1.0.1 - 2022 / 02 / 10

4.2.1 New Features

- Geometry comparison tests for comparing two different geometry trees.
- Copy number for physical volumes.
- Overlap checking for assembly and replica volumes.
- Ability to analyse a registry structure and usage of objects throughout.
- ROOT geometry loader.

4.2.2 General

- Improved documentation and docstrings.
- Refactored registry for transfer and object versus simply adding an object.
- Optimised imports.
- Cleaned up private imports throughout code to make tab complete cleaner.
- Simplified code in solids by using base class.

4.2.3 Bug Fixes

- Improved search paths for libraries in setup.py.
- Fix merging of registries - object names won't be altered if they don't have to be. Fixed a clash with scales.
- Fixed conversion of FLUKA materials to Geant4 - many fixes.
- Fixed some material issues when converting Geant4 to FLUKA.

4.3 v0.9.0 - 2021 / 07 / 01

- Working version regularly used, submitted to CPC Journal for review.
- Based on CGAL for Boolean mesh operations, using pybind11, whereas previously was based on pycgal.
- FLUKA conversion to pyg4ometry and GDML has been reimplemented from the pyfluka package.
- Extensive code testing has been introduced and basic functionality documented.
- Given the strictness of CGAL, many bugs in meshing algorithms were fixed for all solids in *pyg4ometry.geant4.solid*.

4.4 Pre-History

- v0.2.0 - 2018 / 06 / 23
- v0.1.4 - 2018 / 06 / 04
- v0.1.2 - 2018 / 06 / 03
- v0.1.1 - 2018 / 06 / 03
- v0.1.0 - 2017 / 06 / 05
- v0.4.0 - 2017 / 10 / 17
- v0.3.0 - 2017 / 07 / 06

API REFERENCE

This page contains auto-generated API reference documentation¹.

5.1 pyg4ometry

5.1.1 Subpackages

`pyg4ometry.analysis`

Submodules

`pyg4ometry.analysis.Data`

Module Contents

Classes

TH1

TH2

TH3

```
class pyg4ometry.analysis.Data.TH1(data, nPrimary=1, flukaBdxVariable='energy')
```

```
    __getitem__(slice)
```

```
    plot(ms='.')
```

```
class pyg4ometry.analysis.Data.TH2(data, nPrimary=1)
```

```
    __getitem__(slice)
```

```
    plot()
```

¹ Created with `sphinx-autoapi`

```
class pyg4ometry.analysis.Data.TH3(data, nPrimary=1)
    __getitem__(slice)
    plot()
    plot3Projections()
```

pyg4ometry.analysis.Plot

Module Contents

Functions

```
plotEventSections()
plotEventSection(geometryFile[, flukaFile, bdsim-
File])
plotScoringMeshSection(geometryFile[,
planeAxis, ...])
flukaEventDisplay(gdmlFile, dumpFile)
```

```
pyg4ometry.analysis.Plot.plotEventSections()
pyg4ometry.analysis.Plot.plotEventSection(geometryFile, flukaFile=None, bdsimFile=None)
pyg4ometry.analysis.Plot.plotScoringMeshSection(geometryFile, planeAxis=1, flukaFile=None,
flukaBin=None, bdsimFile=None, bdsimHisto=None)
pyg4ometry.analysis.Plot.flukaEventDisplay(gdmlFile, dumpFile)
```

pyg4ometry.analysis.bdsimData

pyg4ometry.analysis.flukaData

Module Contents

Classes

_FlukaDataFile

FlukaBinData

FlukaBdxData

Usrbin

Usrbdx

Usrdump

Functions

fortran_skip(f)

fortran_read(f)

debugDumpFile(fd[, limit])

```

pyg4ometry.analysis.flukaData.fortran_skip(f)
pyg4ometry.analysis.flukaData.fortran_read(f)
pyg4ometry.analysis.flukaData.debugDumpFile(fd, limit=10000000)
class pyg4ometry.analysis.flukaData._FlukaDataFile(fd)
    read_header(fd)
    read_data(fd)
    print_header()
class pyg4ometry.analysis.flukaData.FlukaBinData(index, name, type)
    binToVtkGrid()
class pyg4ometry.analysis.flukaData.FlukaBdxData(index, name, type)
class pyg4ometry.analysis.flukaData.Usrbin(fd, read_data=False)
    Bases: _FlukaDataFile
    read_file(fd)
    read_data(fd)
    read_stats(fd)

```

```
read_header(fd)
```

```
print_header()
```

```
class pyg4ometry.analysis.flukaData.Usrbdx(file)
```

```
    Bases: _FlukaDataFile
```

```
    read_file(fd)
```

```
    read_data(fd)
```

```
    read_stats(fd)
```

```
    read_header(fd)
```

```
class pyg4ometry.analysis.flukaData.Usrdump(fd, iEventHeaderToRead=10000)
```

```
    Bases: _FlukaDataFile
```

```
    read_structure(fd, iEventHeaderToRead=10000)
```

```
    read_event(fd, ievent=0)
```

```
    trackDataToPolydata()
```

```
pyg4ometry.bdsim
```

Submodules

```
pyg4ometry.bdsim.beam
```

Module Contents

Classes

Beam

```
class pyg4ometry.bdsim.beam.Beam(particleName='e-', energy='16.5*GeV', pos=[0, 0, 0], dir=[0, 0, 1])
```

```
    write(fd)
```

```
pyg4ometry.bdsim.beamline
```

Module Contents

Classes

Beamline

```
class pyg4ometry.bdsim.beamline.Beamline(name)
    addElement(element)
    write(fd)
```

pyg4ometry.bdsim.element

Module Contents

Classes

Element

```
class pyg4ometry.bdsim.element.Element(name, type, **kwargs)
    write(fd)
```

pyg4ometry.bdsim.gmad

Module Contents

Classes

Gmad

```
class pyg4ometry.bdsim.gmad.Gmad(beamline=None, beam=None, options=None, samplers=None,
                                  scorers=[], scorer_meshes=[])
    write(fd='test.gmad')
```

pyg4ometry.bdsim.options

Module Contents

Classes

Options

```
class pyg4ometry.bdsim.options.Options
```

```
addOption(key, value)
write(fd)
```

pyg4ometry.bdsim.sampler

Module Contents

Classes

Sampler

```
class pyg4ometry.bdsim.sampler.Sampler(range=None, type=None)
    write(fd)
```

pyg4ometry.bdsim.scoring

Module Contents

Classes

Scorer

ScorerMesh

```
class pyg4ometry.bdsim.scoring.Scorer(name='score1', type='depositedenergy', parameters=None,
                                       conversionFactorFile=None, conversionFactorPath=None)
    write(fd)
class pyg4ometry.bdsim.scoring.ScorerMesh(name, scorer, mesh='box', position=[0, 0, 0], rotation=[0, 0,
0], nbins=[10, 10, 10], size=[100, 100, 100])
    write(fd)
```

Package Contents

Classes

*Sampler**Scorer**ScorerMesh**Beamline**Options**Gmad**Element**Beam*

```

class pyg4ometry.bdsim.Sampler(range=None, type=None)
    write(fd)

class pyg4ometry.bdsim.Scorer(name='score1', type='depositedenergy', parameters=None,
                               conversionFactorFile=None, conversionFactorPath=None)
    write(fd)

class pyg4ometry.bdsim.ScorerMesh(name, scorer, mesh='box', position=[0, 0, 0], rotation=[0, 0, 0],
                                   nbins=[10, 10, 10], size=[100, 100, 100])
    write(fd)

class pyg4ometry.bdsim.Beamline(name)
    addElement(element)
    write(fd)

class pyg4ometry.bdsim.Options
    addOption(key, value)
    write(fd)

class pyg4ometry.bdsim.Gmad(beamline=None, beam=None, options=None, samplers=None, scorers=[],
                             scorer_meshes=[])
    write(fd='test.gmad')

class pyg4ometry.bdsim.Element(name, type, **kwargs)
    write(fd)

class pyg4ometry.bdsim.Beam(particleName='e-', energy='16.5*GeV', pos=[0, 0, 0], dir=[0, 0, 1])
    write(fd)

```

pyg4ometry.compare

Submodules

pyg4ometry.compare._Compare

Module Contents

Classes

<i>Tests</i>	Set of options of which tests to perform and potentially with what tolerance.
<i>TestResult</i>	A test result can be either pass, fail or not conducted.
<i>TestResultNamed</i>	
<i>ComparisonResult</i>	Holder for a test result. Roughly a dict[testname] = list(TestResultNamed)

Functions

<code>gdmlFiles(referenceFile, otherFile[, tests, ...])</code>	param referenceFile GDML file to use as a reference.
<code>geometry(referenceLV, otherLV[, tests, ...])</code>	param referenceLV LogicalVolume instance to compare against.
<code>logicalVolumes(referenceLV, otherLV, tests[, ...])</code>	Compare two LogicalVolume instances with a set of tests.
<code>physicalVolumes(referencePV, otherPV, tests[, ...])</code>	lvName is an optional parent object name to help in print out details decode where the placement is.
<code>assemblyVolumes(referenceAV, otherAV, tests[, ...])</code>	
<code>_checkPVLikeDaughters(referencePVLikeObject, ...[, ...])</code>	
<code>_testDaughterNameSets(referenceDaughterNameSet, ...)</code>	
<code>replicaVolumes(referenceRV, otherRV, tests[, ...])</code>	
<code>divisionVolumes(referenceRV, otherRV, tests[, ...])</code>	Compare two DivisionVolume instances with a set of tests.
<code>parameterisedVolumes(referenceRV, otherRV, tests[, ...])</code>	Compare two ParameterisedVolume instances with a set of tests.
<code>materials(referenceMaterial, otherMaterial, tests[, ...])</code>	Compare two materials with a set of tests.
<code>_elements(referenceElement, otherElement, tests[, ...])</code>	
<code>solids(referenceSolid, otherSolid, tests[, lvName, ...])</code>	Compare any two solids with a set of tests.
<code>_names(testName, str1, str2[, parentName, ...])</code>	
<code>_namesIgnorePointer(testName, str1, str2[, ...])</code>	
<code>_vector(vectortype, r1, r2, tests[, parentName, ...])</code>	
<code>_copyNumber(pvname, c1, c2, tests[, includeAll-TestResults])</code>	
<code>_getBoundingBox(obj)</code>	
<code>_getRawMesh(obj)</code>	
<code>_meshes(lvname, referenceMesh, otherMesh, tests[, ...])</code>	

class pyg4ometry.compare._Compare.Tests(*testsByNameToTurnOn)

Set of options of which tests to perform and potentially with what tolerance.

Parameters

testsByNameToTurnOn (*str*, *list of*) – optional strings of tests to turn on. If any, all will be turned off first.

```
>>> t = Tests()
>>> t = Tests("nDaughters") # only nDaughters will be tested
>>> t = Tests("nDaughters", "shapeArea") # only nDaughters and shapeArea will be
↳ tested
```

```
_testNames = ['names', 'namesIgnorePointer', 'nDaughters', 'solidExact',
'solidExtent', 'shapeExtent',...]
```

setAllFalse()

Utility to turn off all tests at once. Then can just turn on the one we want.

setFalse(testName)

Safely set a test by name to False.

__len__()

classmethod printAllTestNames()

Print all tests names - the exact strings that can be used to turn them off or on.

__repr__()

Return repr(self).

class pyg4ometry.compare._Compare.TestResult

Bases: `enum.Enum`

A test result can be either pass, fail or not conducted.

Use 0,1 so we can also implicitly construct this with a Boolean.

Use the bitwise or operator `|` and not the keyword *or*. This bitwise or operator returns Failed if either have failed. Only returns NotTested if both are not tested. Cannot use bitwise `|=` as we cannot update an Enum internally.

Use `TestResult.All()` for a list of all possible results - useful as an argument for printing.

Failed = 0

Passed = 1

NotTested = 2

__or__(other)

__ior__(other)

static All()

Utility function to get a list of all test types in one.

class pyg4ometry.compare._Compare.TestResultNamed(nameIn, testResultIn=TestResult.Failed, detailsIn="")

__str__()

Return str(self).

class pyg4ometry.compare._Compare.ComparisonResult

Holder for a test result. Roughly a dict[testname] = list(TestResultNamed)

Use `+` and `+=` to append to this object. Uses a default dictionary so no need to initialise any key names. Should always append a list even if only 1 item.

```
>>> cr = ComparisonResult()
>>> cr['nDaughtersTest'] += [TestResultNamed('volume_1', TestResult.Failed,
↳ 'different number')]
>>> cr.print()
```

print() can take a list of test result outcomes to print. e.g. TestResult.All()

__getitem__(key)

__setitem__(key, value)

__add__(other)

__iadd__(other)

__len__()

testNames()

print(testName=None, testResultsToPrint=[TestResult.Failed, TestResult.NotTested], allTests=False)

Parameters

- **testName** (str) – (optional) name of specific single test to print - see testNames()
- **testResultsToPrint** (list(TestResult)) – (optional) list of result outcomes to print
- **allTests** (bool) – (optional) print all tests irrespective of the result

Print failed tests and ones relevant but not implemented yet by default. Control level of print out with optional argument of list of test outcomes to print, or allTests=True.

```
>>> cr.print()
>>> cr.print('solidName')
>>> cr.print(testResultsToPrint=TestResult.All())
>>> cr.print(allTests=True)
```

pyg4ometry.compare._Compare.gdmlFiles(referenceFile, otherFile, tests=Tests(), includeAllTestResults=False)

Parameters

- **referenceFile** (str.) – GDML file to use as a reference.
- **otherFile** (str.) – GDML file to compare.
- **tests** (pyg4ometry.compare._Compare.Tests.) – Tests instance to use.
- **includeAllTestResults** (bool.) – document all tests attempted in result.

pyg4ometry.compare._Compare.geometry(referenceLV, otherLV, tests=Tests(), includeAllTestResults=False)

Parameters

- **referenceLV** (LogicalVolume) – LogicalVolume instance to compare against.
- **otherLV** (LogicalVolume) – LogicalVolume instance to compare with referenceLV.
- **tests** (pyg4ometry.compare._Compare.Tests.) – Tests instance to use.
- **includeAllTestResults** (bool.) – document all tests attempted in result.

```
pyg4ometry.compare._Compare.logicalVolumes(referenceLV, otherLV, tests, recursive=False,  
                                              includeAllTestResults=False, testsAlreadyDone=[])
```

Compare two LogicalVolume instances with a set of tests.

testsAlreadyDone should be purposely given as an empty list if using more than once. This is an artefact of the only way we can nicely pass by reference a set of tests while we use these functions recursively. However, this is defined on import and must be updated if reused.

```
pyg4ometry.compare._Compare.physicalVolumes(referencePV, otherPV, tests, recursive=False, lvName="",  
                                              includeAllTestResults=False, testsAlreadyDone=[])
```

lvName is an optional parent object name to help in print out details decode where the placement is.

```
pyg4ometry.compare._Compare.assemblyVolumes(referenceAV, otherAV, tests, recursive=False,  
                                              includeAllTestResults=False, testsAlreadyDone=[])
```

```
pyg4ometry.compare._Compare._checkPVLikeDaughters(referencePVLikeObject, otherPVLikeObject, tests,  
                                                    parentName, testName, result, recursive=True,  
                                                    includeAllTestResults=True, testsAlreadyDone=[])
```

```
pyg4ometry.compare._Compare._testDaughterNameSets(referenceDaughterNameSet,  
                                                    otherDaughterNameSet, result, testName,  
                                                    includeAllTestResults)
```

```
pyg4ometry.compare._Compare.replicaVolumes(referenceRV, otherRV, tests, recursive=True,  
                                              includeAllTestResults=False, testsAlreadyDone=[])
```

```
pyg4ometry.compare._Compare.divisionVolumes(referenceRV, otherRV, tests, includeAllTestResults=False,  
                                              testsAlreadyDone=[])
```

Compare two DivisionVolume instances with a set of tests.

```
pyg4ometry.compare._Compare.parameterisedVolumes(referenceRV, otherRV, tests,  
                                                    includeAllTestResults=False, testsAlreadyDone=[])
```

Compare two ParameterisedVolume instances with a set of tests.

```
pyg4ometry.compare._Compare.materials(referenceMaterial, otherMaterial, tests, lvName="",  
                                       includeAllTestResults=False, testsAlreadyDone=[])
```

Compare two materials with a set of tests.

This tests assumes both referenceMaterial and otherMaterial are derived from the type `pyg4ometry.geant4._Material.Material`.

Compares, name, classname, density, n components

```
pyg4ometry.compare._Compare._elements(referenceElement, otherElement, tests, lvName="",  
                                       includeAllTestResults=False)
```

```
pyg4ometry.compare._Compare.solids(referenceSolid, otherSolid, tests, lvName="",  
                                     includeAllTestResults=False)
```

Compare any two solids with a set of tests.

```
pyg4ometry.compare._Compare._names(testName, str1, str2, parentName="", includeAllTestResults=False)
```

```
pyg4ometry.compare._Compare._namesIgnorePointer(testName, str1, str2, parentName="",  
                                                  includeAllTestResults=False)
```

```
pyg4ometry.compare._Compare._vector(vectortype, r1, r2, tests, parentName="", includeAllTestResults=False)
```

```
pyg4ometry.compare._Compare._copyNumber(pvname, c1, c2, tests, includeAllTestResults=False)
```

```
pyg4ometry.compare._Compare._getBoundingBox(obj)
```

```
pyg4ometry.compare._Compare._getRawMesh(obj)
```

```
pyg4ometry.compare._Compare._meshes(lvname, referenceMesh, otherMesh, tests,
                                     includeAllTestResults=False)
```

Package Contents

Classes

<code>_Material</code>	This class provides an interface to GDML material definitions.
<code>_Element</code>	This class provides an interface to GDML material definitions. Because of the different options
<code>Tests</code>	Set of options of which tests to perform and potentially with what tolerance.
<code>TestResult</code>	A test result can be either pass, fail or not conducted.
<code>TestResultNamed</code>	
<code>ComparisonResult</code>	Holder for a test result. Roughly a dict[testname] = list(TestResultNamed)

Functions

<code>_evaluateToFloat</code> (reg, obj)	
<code>gdmlFiles</code> (referenceFile, otherFile[, tests, ...])	param referenceFile GDML file to use as a reference.
<code>geometry</code> (referenceLV, otherLV[, tests, ...])	param referenceLV LogicalVolume instance to compare against.
<code>logicalVolumes</code> (referenceLV, otherLV, tests[, ...])	Compare two LogicalVolume instances with a set of tests.
<code>physicalVolumes</code> (referencePV, otherPV, tests[, ...])	lvName is an optional parent object name to help in print out details decode where the placement is.
<code>assemblyVolumes</code> (referenceAV, otherAV, tests[, ...])	
<code>_checkPVLikeDaughters</code> (referencePVLikeObject, ...[, ...])	
<code>_testDaughterNameSets</code> (referenceDaughterNameSet, ...)	
<code>replicaVolumes</code> (referenceRV, otherRV, tests[, ...])	
<code>divisionVolumes</code> (referenceRV, otherRV, tests[, ...])	Compare two DivisionVolume instances with a set of tests.
<code>parameterisedVolumes</code> (referenceRV, otherRV, tests[, ...])	Compare two ParameterisedVolume instances with a set of tests.
<code>materials</code> (referenceMaterial, otherMaterial, tests[, ...])	Compare two materials with a set of tests.
<code>_elements</code> (referenceElement, otherElement, tests[, ...])	
<code>solids</code> (referenceSolid, otherSolid, tests[, lvName, ...])	Compare any two solids with a set of tests.
<code>_names</code> (testName, str1, str2[, parentName, ...])	
<code>_namesIgnorePointer</code> (testName, str1, str2[, ...])	
<code>_vector</code> (vectortype, r1, r2, tests[, parentName, ...])	
<code>_copyNumber</code> (pvname, c1, c2, tests[, includeAll-TestResults])	
<code>_getBoundingBox</code> (obj)	
<code>_getRawMesh</code> (obj)	
<code>_meshes</code> (lvname, referenceMesh, otherMesh, tests[, ...])	

```
pyg4ometry.compare._evaluateToFloat(reg, obj)
```

```
class pyg4ometry.compare._Material(**kwargs)
```

Bases: `MaterialBase`

This class provides an interface to GDML material definitions.

Because of the different options for constructing a material instance the constructor is kwarg only. Proxy methods are provided to instantiate particular types of material. Those proxy methods are:

`MaterialSingleElement` `MaterialCompound` `MaterialPredefined`

It is possible to instantiate a material directly through kwargs. The possible kwargs are (but note some are mutually exclusive): `name` - string `density` - float `atomic_number` - int `atomic_weight` - float `number_of_components` - int `state` - string `pressure` - float `pressure_unit` - string `temperature` - float `temperature_unit` - string

property state_variables

`add_element_massfraction(element, massfraction)`

Add an element as a component to a material as a fraction of the material mass. Can only add elements to materials defined as composite.

Inputs:

`element` - `pyg4ometry.geant4.Material.Element` instance `massfraction` - float, $0.0 < \text{massfraction} \leq 1.0$

`add_element_natoms(element, natoms)`

Add an element as a component to a material as a number of atoms in the material molecule. Can only add elements to materials defined as composite.

Inputs:

`element` - `pyg4ometry.geant4.Material.Element` instance `natoms` - int, number of atoms in the compound molecule

`add_material(material, fractionmass)`

Add a material as a component to another material (mixture) as a fraction of the mixture mass. Can only add new materials to materials defined as composite.

Inputs:

`material` - `pyg4ometry.geant4.Material.Material` instance `massfraction` - float, $0.0 < \text{massfraction} \leq 1.0$

`set_pressure(value, unit='pascal')`

`set_temperature(value, unit='K')`

`__str__()`

Return `str(self)`.

`addProperty(name, matrix)`

Add a material property from a matrix.

Parameters

- **`name`** (`str`) – key of the material property
- **`matrix`** (`Matrix`) – matrix defining the value(s) of the property

`addVecProperty(name, e, v, eunit='eV', vunit='')`

Add a property from an energy and a value vector to this object.

Parameters

- **`name`** (`str`) – key of property
- **`e`** (`list` or `numpy.array` - `shape (1,)`) – energy list/vector in units of `eunit`

- **v** (*list* or *numpy.array* - *shape* (1,)) – value list/vector in units of vunit
- **eunit** (*str*) – unit for the energy vector (default: eV)
- **vunit** (*str*) – unit for the value vector (default: unitless)

addConstProperty(*name*, *value*, *vunit*="")

Add a constant scalar property to this object.

Parameters

- **name** (*str*) – key of property
- **value** (*str*, *float*, *int*) – constant value for this property
- **vunit** (*str*) – unit for the value vector (default: unitless)

class pyg4ometry.compare._Element(**kwargs)

Bases: MaterialBase

This class provides an interface to GDML material definitions. Because of the different options for constructing a material instance the constructor is kwarg only. Proxy methods are provided to instantiate particular types of material. Those proxy methods are:

ElementSimple ElementIsotopeMixture

It is possible to instantiate a material directly through kwargs. The possible kwargs are (but note some are mutually exclusive): name - string symbol - string Z - int A - int n_comp - int

add_isotope(*isotope*, *abundance*)

Add an isotope as a component to an element as an abundance fraction in the element.

Inputs:

element - pyg4ometry.geant4.Material.Isotope instance abundance - float, 0.0 < abundance <= 1.0

class pyg4ometry.compare.Tests(*testsByNameToTurnOn)

Set of options of which tests to perform and potentially with what tolerance.

Parameters

testsByNameToTurnOn (*str*, *list* of) – optional strings of tests to turn on. If any, all will be turned off first.

```
>>> t = Tests()
>>> t = Tests("nDaughters") # only nDaughters will be tested
>>> t = Tests("nDaughters", "shapeArea") # only nDaughters and shapeArea will be tested
```

_testNames = ['names', 'namesIgnorePointer', 'nDaughters', 'solidExact', 'solidExtent', 'shapeExtent', ...]

setAllFalse()

Utility to turn off all tests at once. Then can just turn on the one we want.

setFalse(*testName*)

Safely set a test by name to False.

__len__()

classmethod printAllTestNames()

Print all tests names - the exact strings that can be used to turn them off or on.

`__repr__()`

Return repr(self).

class pyg4ometry.compare.TestResult

Bases: `enum.Enum`

A test result can be either pass, fail or not conducted.

Use 0,1 so we can also implicitly construct this with a Boolean.

Use the bitwise or operator `|` and not the keyword *or*. This bitwise or operator returns Failed if either have failed. Only returns NotTested if both are not tested. Cannot use bitwise `|=` as we cannot update an Enum internally.

Use TestResult.All() for a list of all possible results - useful as an argument for printing.

Failed = 0

Passed = 1

NotTested = 2

`__or__(other)`

`__ior__(other)`

static All()

Utility function to get a list of all test types in one.

class pyg4ometry.compare.TestResultNamed(nameIn, testResultIn=TestResult.Failed, detailsIn='')

`__str__()`

Return str(self).

class pyg4ometry.compare.ComparisonResult

Holder for a test result. Roughly a dict[testname] = list(TestResultNamed)

Use `+` and `+=` to append to this object. Uses a default dictionary so no need to initialise any key names. Should always append a list even if only 1 item.

```
>>> cr = ComparisonResult()
>>> cr['nDaughtersTest'] += [TestResultNamed('volume_1', TestResult.Failed,
↪ 'different number')]
>>> cr.print()
```

print() can take a list of test result outcomes to print. e.g. TestResult.All()

`__getitem__(key)`

`__setitem__(key, value)`

`__add__(other)`

`__iadd__(other)`

`__len__()`

testNames()

```
print(testName=None, testResultsToPrint=[TestResult.Failed, TestResult.NotTested], allTests=False)
```

Parameters

- **testName** (*str*) – (optional) name of specific single test to print - see testNames()
- **testResultsToPrint** (*list(TestResult)*) – (optional) list of result outcomes to print
- **allTests** (*bool*) – (optional) print all tests irrespective of the result

Print failed tests and ones relevant but not implemented yet by default. Control level of print out with optional argument of list of test outcomes to print, or allTests=True.

```
>>> cr.print()
>>> cr.print('solidName')
>>> cr.print(testResultsToPrint=TestResult.All())
>>> cr.print(allTests=True)
```

```
pyg4ometry.compare.gdmlFiles(referenceFile, otherFile, tests=Tests(), includeAllTestResults=False)
```

Parameters

- **referenceFile** (*str.*) – GDML file to use as a reference.
- **otherFile** (*str.*) – GDML file to compare.
- **tests** (*pyg4ometry.compare._Compare.Tests.*) – Tests instance to use.
- **includeAllTestResults** (*bool.*) – document all tests attempted in result.

```
pyg4ometry.compare.geometry(referenceLV, otherLV, tests=Tests(), includeAllTestResults=False)
```

Parameters

- **referenceLV** (*LogicalVolume*) – LogicalVolume instance to compare against.
- **otherLV** (*LogicalVolume*) – LogicalVolume instance to compare with referenceLV.
- **tests** (*pyg4ometry.compare._Compare.Tests.*) – Tests instance to use.
- **includeAllTestResults** (*bool.*) – document all tests attempted in result.

```
pyg4ometry.compare.logicalVolumes(referenceLV, otherLV, tests, recursive=False,
                                   includeAllTestResults=False, testsAlreadyDone=[])
```

Compare two LogicalVolume instances with a set of tests.

testsAlreadyDone should be purposively given as an empty list if using more than once. This is an artefact of the only way we can nicely pass by reference a set of tests while we use these functions recursively. However, this is defined on import and must be updated if reused.

```
pyg4ometry.compare.physicalVolumes(referencePV, otherPV, tests, recursive=False, lvName="",
                                   includeAllTestResults=False, testsAlreadyDone=[])
```

lvName is an optional parent object name to help in print out details decode where the placement is.

```
pyg4ometry.compare.assemblyVolumes(referenceAV, otherAV, tests, recursive=False,
                                   includeAllTestResults=False, testsAlreadyDone=[])
```

```
pyg4ometry.compare._checkPVLikeDaughters(referencePVLikeObject, otherPVLikeObject, tests,
                                           parentName, testName, result, recursive=True,
                                           includeAllTestResults=True, testsAlreadyDone=[])
```

```
pyg4ometry.compare._testDaughterNameSets(referenceDaughterNameSet, otherDaughterNameSet, result,
                                          testName, includeAllTestResults)
```

```
pyg4ometry.compare.replicaVolumes(referenceRV, otherRV, tests, recursive=True,
                                  includeAllTestResults=False, testsAlreadyDone=[])
```

```
pyg4ometry.compare.divisionVolumes(referenceRV, otherRV, tests, includeAllTestResults=False,
                                   testsAlreadyDone=[])
```

Compare two DivisionVolume instances with a set of tests.

```
pyg4ometry.compare.parameterisedVolumes(referenceRV, otherRV, tests, includeAllTestResults=False,
                                         testsAlreadyDone=[])
```

Compare two ParameterisedVolume instances with a set of tests.

```
pyg4ometry.compare.materials(referenceMaterial, otherMaterial, tests, lvName="",
                             includeAllTestResults=False, testsAlreadyDone=[])
```

Compare two materials with a set of tests.

This tests assumes both `referenceMaterial` and `otherMaterial` are derived from the type `pyg4ometry.geant4._Material.Material`.

Compares, name, classname, density, n components

```
pyg4ometry.compare._elements(referenceElement, otherElement, tests, lvName="",
                             includeAllTestResults=False)
```

```
pyg4ometry.compare.solids(referenceSolid, otherSolid, tests, lvName="", includeAllTestResults=False)
```

Compare any two solids with a set of tests.

```
pyg4ometry.compare._names(testName, str1, str2, parentName="", includeAllTestResults=False)
```

```
pyg4ometry.compare._namesIgnorePointer(testName, str1, str2, parentName="",
                                       includeAllTestResults=False)
```

```
pyg4ometry.compare._vector(vectortype, r1, r2, tests, parentName="", includeAllTestResults=False)
```

```
pyg4ometry.compare._copyNumber(pvname, c1, c2, tests, includeAllTestResults=False)
```

```
pyg4ometry.compare._getBoundingBox(obj)
```

```
pyg4ometry.compare._getRawMesh(obj)
```

```
pyg4ometry.compare._meshes(lvname, referenceMesh, otherMesh, tests, includeAllTestResults=False)
```

pyg4ometry.convert

Submodules

pyg4ometry.convert.fluka2Geant4

Module Contents

Functions

<code>fluka2Geant4(flukareg[, regions, omitRegions, ...])</code>	Convert a FLUKA registry to a Geant4 Registry.
<code>_flukaRegistryToG4Registry(flukareg, regions, ...)</code>	Convert a transformed fluka registry to a geant4 registry.
<code>_filteredRegions(flukareg, regions)</code>	
<code>_makeWorldVolume(dimensions, material, g4registry)</code>	Make a world solid and logical volume with the given dimensions,
<code>_makeLengthSafetyRegistry(flukareg, regions)</code>	Make a new registry from a registry with length safety applied
<code>_getRegionZoneAABBs(flukareg, regions, quadricRegionAABBs)</code>	Loop over the regions, and for each region, get all the aabbs
<code>_filterRegistryNullZones(flukareg, regionZoneAABBs)</code>	
<code>_filterRegionNullZones(region, aabbs)</code>	
<code>_filterNullAABBs(regionZoneAABBs)</code>	
<code>_makeBodyMinimumAABBMap(flukareg, regionZoneAABBs, regions)</code>	
<code>_getMaximalOfTwoAABBs(aabb1, aabb2)</code>	Given two aabbs, returns the total aabb that tightly bounds
<code>_filterBlackHoleRegions(flukareg, regions)</code>	Returns a new FlukaRegistry instance with all regions with
<code>_getOverlappingAABBs(aabb, aabbs)</code>	
<code>_getContentsOfLatticeCells(flukaregistry, regionAABBs)</code>	
<code>_getTransformedCellRegionAABB(lattice)</code>	
<code>_isTransformedCellRegionIntersectingWithReg(...)</code>	
<code>_checkQuadricRegionAABBs(flukareg, quadricRegionAABBs)</code>	Loop over the regions looking for quadrics and for any quadrics we
<code>_getWorldDimensions(worldDimensions)</code>	Get world dimensions and if None then return the global constant
<code>_getSelectedRegions(flukareg, regions, omitRegions)</code>	
<code>_filterHalfSpaces(flukareg, regionZoneAABBs)</code>	Filter redundant half spaces from the regions of the
<code>_distanceFromPointToPlane(normal, pointOnPlane, point)</code>	
<code>_convertLatticeCells(greg, flukareg, wlv, ...)</code>	
<code>_makeUniqueQuadricRegions(flukareg, quadricRegionAABBs)</code>	
<code>_makeQuadricRegionBodyAABBMap(flukareg, quadricRegionAABBs)</code>	Given a map of regions featuring quadrics to their aabbs, we
<code>_getMaximalQuadricRegionAABBs(freg, quadricRegionAABBs)</code>	
<code>_regionZoneAABBsToRegionAABBs(regionZoneAABB)</code>	Given a map of region names to zone aabbs, return a map of
<code>_copyStructureToNewFlukaRegistry(freg, fregtarget)</code>	

Attributes

logger

WORLD_DIMENSIONS

`pyg4ometry.convert.fluka2Geant4.logger`

`pyg4ometry.convert.fluka2Geant4.WORLD_DIMENSIONS = [10000, 10000, 10000]`

exception `pyg4ometry.convert.fluka2Geant4.NullModel`

Bases: `Exception`

Common base class for all non-exit exceptions.

`pyg4ometry.convert.fluka2Geant4.fluka2Geant4(flukareg, regions=None, omitRegions=None, worldMaterial='G4_Galactic', worldDimensions=None, omitBlackholeRegions=True, quadricRegionAABBs=None, **kwargs)`

Convert a FLUKA registry to a Geant4 Registry.

Parameters

- **flukareg** (`FlukaRegistry`) – FlukaRegistry instance to be converted.
- **regions** (`list`) – Names of regions to be converted, by default all are converted. Mutually exclusive with `omitRegions`.
- **omitRegions** (`list`) – Names of regions to be omitted from the conversion. This option is mutually exclusive with the `kwargs` regions.
- **worldMaterial** (`string`) – name of world material to be used.
- **worldDimensions** (`list`) – dimensions of world logical volume in converted Geant4. By default this is equal to `WORLD_DIMENSIONS`.
- **omitBlackholeRegions** (`bool`) – whether or not to omit regions with the FLUKA material `BLACKHOLE` from the conversion. By default, true.
- **quadricRegionAABBs** (`dict`) – The axis-aligned aabbs of any regions featuring QUA bodies, mapping region names to `fluka.AABB` instances.

Developer options (to `kwargs`) with `LengthSafety`: Whether or not to apply automatic length safety.

`minimiseSolids`: Whether or not to minimise the boxes and tubes of Geant4 used to represent infinite solids in FLUKA.

`pyg4ometry.convert.fluka2Geant4._flukaRegistryToG4Registry(flukareg, regions, worldinfo, aabbinfo)`

Convert a transformed fluka registry to a geant4 registry.

`pyg4ometry.convert.fluka2Geant4._filteredRegions(flukareg, regions)`

`pyg4ometry.convert.fluka2Geant4._makeWorldVolume(dimensions, material, g4registry)`

Make a world solid and logical volume with the given dimensions, material, and size, and add it to the geant4 Registry provided

Parameters

dimensions – list of 3 elements providing the dimensions in [x, y, z]

of the world box. :type dimension: list :param material: The name of the material to be used for the world volume. :type material: str :param g4registry: The geant4 Registry instance this world solid and logical volume is to be added to.

`pyg4ometry.convert.fluka2Geant4._makeLengthSafetyRegistry(flukareg, regions)`

Make a new registry from a registry with length safety applied to the zones and regions within.

Parameters

flukareg – The FlukaRegistry from which the new registry

with length safety applied should be built. :type flukareg: FlukaRegistry :param regions: The names of the regions that are to be converted.

`pyg4ometry.convert.fluka2Geant4._getRegionZoneAABBs(flukareg, regions, quadricRegionAABBs)`

Loop over the regions, and for each region, get all the aabbs of the zones belonging to that region. Don't do this for quadricRegionAABBs, instead, just continue to use the aabb provided by the user.

`pyg4ometry.convert.fluka2Geant4._filterRegistryNullZones(flukareg, regionZoneAABBs)`

`pyg4ometry.convert.fluka2Geant4._filterRegionNullZones(region, aabbs)`

`pyg4ometry.convert.fluka2Geant4._filterNullAABBs(regionZoneAABBs)`

`pyg4ometry.convert.fluka2Geant4._makeBodyMinimumAABBMap(flukareg, regionZoneAABBs, regions)`

`pyg4ometry.convert.fluka2Geant4._getMaximalOfTwoAABBs(aabb1, aabb2)`

Given two aabbs, returns the total aabb that tightly bounds the two given aabbs.

Parameters

- **aabb1** ([AABB](#)) – The first aabb.
- **aabb2** ([AABB](#)) – The second aabb

`pyg4ometry.convert.fluka2Geant4._filterBlackHoleRegions(flukareg, regions)`

Returns a new FlukaRegistry instance with all regions with BLKCHOLE material removed.

Parameters

flukareg – The FlukaRegistry instance from which the new

FlukaRegistry instance sans BLCKHOLE regions is built. :type flukareg: FlukaRegistry :param regions: The names of the regions to be copied to the new instance. :type regions: list

`pyg4ometry.convert.fluka2Geant4._getOverlappingAABBs(aabb, aabbs)`

`pyg4ometry.convert.fluka2Geant4._getContentsOfLatticeCells(flukaregistry, regionAABBs)`

`pyg4ometry.convert.fluka2Geant4._getTransformedCellRegionAABB(lattice)`

`pyg4ometry.convert.fluka2Geant4._isTransformedCellRegionIntersectingWithRegion(region, lattice)`

`pyg4ometry.convert.fluka2Geant4._checkQuadricRegionAABBs(flukareg, quadricRegionAABBs)`

Loop over the regions looking for quadrics and for any quadrics we find, make sure that the region has a defined region aabb in quadricRegionAABBs.

`pyg4ometry.convert.fluka2Geant4._getWorldDimensions(worldDimensions)`

Get world dimensinos and if None then return the global constant WORLD_DIMENSIONS.

`pyg4ometry.convert.fluka2Geant4._getSelectedRegions(flukareg, regions, omitRegions)`

`pyg4ometry.convert.fluka2Geant4._filterHalfSpaces(flukareg, regionZoneAABBs)`

Filter redundant half spaces from the regions of the FlukaRegistry instance. AABBs is a dictionary of region names to region aabbs.

`pyg4ometry.convert.fluka2Geant4._distanceFromPointToPlane(normal, pointOnPlane, point)`

`pyg4ometry.convert.fluka2Geant4._convertLatticeCells(greg, flukareg, wlv, regionZoneAABBs, regionNamesToLVs)`

`pyg4ometry.convert.fluka2Geant4._makeUniqueQuadricRegions(flukareg, quadricRegionAABBs)`

`pyg4ometry.convert.fluka2Geant4._makeQuadricRegionBodyAABBMap(flukareg, quadricRegionAABBs)`

Given a map of regions featuring quadrics to their aabbs, we loop over the bodies of the region and set their aabbs equal to the region aabb.

`pyg4ometry.convert.fluka2Geant4._getMaximalQuadricRegionAABBs(freg, quadricRegionAABBs)`

`pyg4ometry.convert.fluka2Geant4._regionZoneAABBsToRegionAABBs(regionZoneAABBs)`

Given a map of region names to zone aabbs, return a map of region names to a total region aabb.

`pyg4ometry.convert.fluka2Geant4._copyStructureToNewFlukaRegistry(freg, fregtarget)`

`pyg4ometry.convert.fluka2g4materials`

Module Contents

Classes

`_FlukaToG4MaterialConverter`

`_PeriodicTable`

Functions

`_getPeriodicTable()`

`makeFlukaToG4MaterialsMap(freg, greg)`

Convert the materials defined in a FlukaRegistry, and populate

Attributes

*_periodicTable**FLUKA_BUILTIN_TO_G4_MATERIAL_MAP**_FLUKA_ELEMENT_SYMBOLS*

`pyg4ometry.convert.fluka2g4materials._periodicTable``pyg4ometry.convert.fluka2g4materials.FLUKA_BUILTIN_TO_G4_MATERIAL_MAP``pyg4ometry.convert.fluka2g4materials._FLUKA_ELEMENT_SYMBOLS``pyg4ometry.convert.fluka2g4materials._getPeriodicTable()``class pyg4ometry.convert.fluka2g4materials._FlukaToG4MaterialConverter(freg, greg)``addPredefinedElements()``convertAll()``convertBuiltin(name, flukaMaterial)``convertElement(name, flukaElement)``convertCompound(flukaCompound)``_makeBaseCompoundMaterial(name, flukaCompound)``convertMassFractionCompound(name, flukaCompound)``convertVolumeFractionCompound(name, flukaCompound)``convertAtomicFractionCompound(name, flukaCompound)``_totalMassWeightedFractionOfCompound(compound)``static _mangleElementName(name)``pyg4ometry.convert.fluka2g4materials.makeFlukaToG4MaterialsMap(freg, greg)`

Convert the materials defined in a FlukaRegistry, and populate the provided geant4 registry with said materials.

`class pyg4ometry.convert.fluka2g4materials._PeriodicTable``massNumberFromZ(z)``atomicMassFromZ(z)``atomicNumberAndMassFromSymbol(symbol)`

`pyg4ometry.convert.freecad2Fluka`

Module Contents

Functions

<code><i>freecadDoc2Fluka</i>(fcd)</code>

<code><i>part2Region</i>(obj, trfm, fgreg[, meshDeviation])</code>
--

`pyg4ometry.convert.freecad2Fluka.freecadDoc2Fluka(fcd)`

`pyg4ometry.convert.freecad2Fluka.part2Region(obj, trfm, fgreg, meshDeviation=0.05)`

`pyg4ometry.convert.gdml2stl`

Module Contents

Functions

<code><i>gdml2stl</i>(gdmlFileName[, stlFileName, solidName])</code>
--

`pyg4ometry.convert.gdml2stl.gdml2stl(gdmlFileName, stlFileName='output.gdml', solidName='ws')`

`pyg4ometry.convert.geant42Fluka`

Module Contents

Functions

<code>geant4Reg2FlukaReg(greg[, logicalVolumeName])</code>	Convert a Geant4 model to a FLUKA one. This is done by handing over a complete
<code>geant4Logical2Fluka(logicalVolume[, flukaRegistry])</code>	Convert a single logical volume - not the main entry point for the conversion.
<code>geant4PhysicalVolume2Fluka(physicalVolume[, mtra, ...])</code>	
<code>geant4Solid2FlukaRegion(flukaNameCount, solid[, mtra, ...])</code>	
<code>geant4MaterialDict2Fluka(matr, freg)</code>	
<code>geant4Material2Fluka(material, freg[, ...])</code>	
<code>pycsgmesh2FlukaRegion(mesh, name, transform, ...)</code>	
<code>makeStripName(mn)</code>	
<code>makeShortName(mn)</code>	

`pyg4ometry.convert.geant42Fluka.geant4Reg2FlukaReg(greg, logicalVolumeName="")`

Convert a Geant4 model to a FLUKA one. This is done by handing over a complete `pyg4ometry.geant4.Registry` instance.

Parameters

greg (`pyg4ometry.geant4.Registry`) – geant4 registry

returns: `pyg4ometry.fluka.FlukaRegistry`

`pyg4ometry.convert.geant42Fluka.geant4Logical2Fluka(logicalVolume, flukaRegistry=None)`

Convert a single logical volume - not the main entry point for the conversion.

`pyg4ometry.convert.geant42Fluka.geant4PhysicalVolume2Fluka(physicalVolume, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]), flukaRegistry=None, flukaNameCount=0)`

`pyg4ometry.convert.geant42Fluka.geant4Solid2FlukaRegion(flukaNameCount, solid, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]), flukaRegistry=None, addRegistry=True, commentName="")`

`pyg4ometry.convert.geant42Fluka.geant4MaterialDict2Fluka(matr, freg)`

`pyg4ometry.convert.geant42Fluka.geant4Material2Fluka(material, freg, suggestedDensity=None, elementSuffix=False, materialNameShort=None)`

```
pyg4ometry.convert.geant42Fluka.pycsgmesh2FlukaRegion(mesh, name, transform, flukaRegistry,
                                                         commentName)
```

```
pyg4ometry.convert.geant42Fluka.makeStripName(mn)
```

```
pyg4ometry.convert.geant42Fluka.makeShortName(mn)
```

pyg4ometry.convert.geant42FlukaBake

Module Contents

Functions

<code>geant4Reg2FlukaReg(greg[, logicalVolumeName])</code>	Convert a Geant4 model to a FLUKA one. This is done by handing over a complete
<code>geant4Logical2Fluka(logicalVolume[, flukaRegistry])</code>	Convert a single logical volume - not the main entry point for the conversion.
<code>geant4PhysicalVolume2Fluka(physicalVolume[, mtra, ...])</code>	
<code>geant4Solid2FlukaRegion(flukaNameCount, solid[, mtra, ...])</code>	
<code>geant4MaterialDict2Fluka(matr, freg)</code>	
<code>geant4Material2Fluka(material, freg[, ...])</code>	
<code>pycsgmesh2FlukaRegion(mesh, name, transform, ...)</code>	
<code>makeStripName(mn)</code>	
<code>makeShortName(mn)</code>	

```
pyg4ometry.convert.geant42FlukaBake.geant4Reg2FlukaReg(greg, logicalVolumeName="")
```

Convert a Geant4 model to a FLUKA one. This is done by handing over a complete `pyg4ometry.geant4.Registry` instance.

Parameters

greg (`pyg4ometry.geant4.Registry`) – geant4 registry

returns: `pyg4ometry.fluka.FlukaRegistry`

```
pyg4ometry.convert.geant42FlukaBake.geant4Logical2Fluka(logicalVolume, flukaRegistry=None)
```

Convert a single logical volume - not the main entry point for the conversion.

```
pyg4ometry.convert.geant42FlukaBake.geant4PhysicalVolume2Fluka(physicalVolume,
                                                                mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]), flukaRegistry=None,
                                                                flukaNameCount=0)
```

```
pyg4ometry.convert.geant42FlukaBake.geant4Solid2FlukaRegion(flukaNameCount, solid,  
                                                             mtra=_np.array([[1, 0, 0], [0, 1, 0],  
                                                             [0, 0, 1]]), tra=_np.array([0, 0, 0]),  
                                                             flukaRegistry=None,  
                                                             addRegistry=True, commentName="")
```

```
pyg4ometry.convert.geant42FlukaBake.geant4MaterialDict2Fluka(matr, freg)
```

```
pyg4ometry.convert.geant42FlukaBake.geant4Material2Fluka(material, freg, suggestedDensity=None,  
                                                           elementSuffix=False,  
                                                           materialNameShort=None)
```

```
pyg4ometry.convert.geant42FlukaBake.pycsgmesh2FlukaRegion(mesh, name, transform, flukaRegistry,  
                                                           commentName)
```

```
pyg4ometry.convert.geant42FlukaBake.makeStripName(mn)
```

```
pyg4ometry.convert.geant42FlukaBake.makeShortName(mn)
```

pyg4ometry.convert.geant42Geant4

Module Contents

Functions

<code>geant4Solid2Geant4Tessellated(solid)</code>

<code>geant4Solid2Geant4Tessellated_NoVTK(solid)</code>

```
pyg4ometry.convert.geant42Geant4.geant4Solid2Geant4Tessellated(solid)
```

```
pyg4ometry.convert.geant42Geant4.geant4Solid2Geant4Tessellated_NoVTK(solid)
```

pyg4ometry.convert.geant42Vtk

pyg4ometry.convert.oce2Geant4

Module Contents

Functions

<code>oceShape_Geant4_LogicalVolume</code> (name, solid, material, greg)	Make a logical volume from input or get from registry
<code>oceShape_Geant4_Assembly</code> (name, greg)	Make a assembly volume from input or get from registry
<code>oceShape_Geant4_Tessellated</code> (name, shape, greg[, ...])	Make a tessellated solid from a OpenCascade shape
<code>_oce2Geant4_traverse</code> (shapeTool, label, greg, ...[, ...])	
<code>oce2Geant4</code> (shapeTool, shapeName[, materialMap, ...])	Convert CAD geometry starting from shapeName

Attributes

<code>defaultLinDef</code>
<code>deftaulAngDef</code>

`pyg4ometry.convert.oce2Geant4.defaultLinDef = 0.5`

`pyg4ometry.convert.oce2Geant4.deftaulAngDef = 0.5`

`pyg4ometry.convert.oce2Geant4.oceShape_Geant4_LogicalVolume`(name, solid, material, greg)

Make a logical volume from input or get from registry

Parameters

- **name** (*str*) – Name of logical volume
- **solid** (*SolidBase*) – Geant4 solid
- **material** (*str* or `pyg4ometry.geant4.Material`) – Material for logical volume
- **greg** (`geant4.Registry`) – Geant4 registry

`pyg4ometry.convert.oce2Geant4.oceShape_Geant4_Assembly`(name, greg)

Make a assembly volume from input or get from registry

Parameters

- **name** (*str*) – Name of logical volume
- **greg** (`geant4.Registry`) – Geant4 registry

`pyg4ometry.convert.oce2Geant4.oceShape_Geant4_Tessellated`(name, shape, greg, linDef=0.5, angDef=0.5)

Make a tessellated solid from a OpenCascade shape

Parameters

- **name** (*str*) – Name of logical volume
- **shape** (*TopoDS_Shape*) – OpenCascade shape
- **greg** (`geant4.Registry`) – Geant4 registry

```
pyg4ometry.convert.oce2Geant4._oce2Geant4_traverse(shapeTool, label, greg, materialMap,  
                                                    labelToSkipList, meshQualityMap,  
                                                    badCADLabels, addBoundingSolids=False,  
                                                    oceName=False)
```

```
pyg4ometry.convert.oce2Geant4.oce2Geant4(shapeTool, shapeName, materialMap={}, labelToSkipList=[],  
                                          meshQualityMap={}, oceName=False)
```

Convert CAD geometry starting from shapeName

Parameters

- **shapeTool** (*pyoce.TopoDS_Shape*) – OpenCascade TopoDS_Shape
- **shapeName** (*str*) – Name of the shape in the CAD file
- **materialMap** (*dict*) – dictionary to map shape name to material shapeName:materialName or shapeName:Material
- **meshQualityMap** (*dict*) – dictionary to map shape name to meshing quality str:[LinDef,AngDef]

```
pyg4ometry.convert.stl2gdml
```

Module Contents

Functions

```
stl2gdml(stlFileName[, gdmlFileName, worldMaterial, ...])
```

```
pyg4ometry.convert.stl2gdml.stl2gdml(stlFileName, gdmlFileName='output.gdml',  
                                     worldMaterial='G4_Galactic', solidMaterial='G4_Au')
```

```
pyg4ometry.convert.vis2oce
```

Module Contents

Functions

```
convertMeshToPolyTriangulation(m)
```

```
convertMeshToShapeUsingMakeShapeOnMesh(m)
```

```
convertMeshToShape(m)
```

```
vis2oce(vis[, stepFileName])
```

```
pyg4ometry.convert.vis2oce.convertMeshToPolyTriangulation(m)
```

```
pyg4ometry.convert.vis2oce.convertMeshToShapeUsingMakeShapeOnMesh(m)
```

```
pyg4ometry.convert.vis2oce.convertMeshToShape(m)
```

```
pyg4ometry.convert.vis2oce.vis2oce(vis, stepFileName='output.step')
```

Package Contents

Classes

<i>_PolygonProcessing</i>	
<i>RPP</i>	Rectangular Parallelepiped
<i>BOX</i>	General Rectangular Parallelepiped
<i>SPH</i>	Sphere
<i>RCC</i>	Right Circular Cylinder
<i>REC</i>	Right Elliptical Cylinder
<i>TRC</i>	Truncated Right-angled Cone
<i>ELL</i>	Ellipsoid of Revolution
<i>WED</i>	Right Angle Wedge
<i>RAW</i>	Base class representing a body as defined in FLUKA
<i>ARB</i>	Arbitrary Convex Polyhedron
<i>XYP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the z-axis)
<i>XZP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the y-axis)
<i>YZP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the x-axis)
<i>PLA</i>	Infinite half-space delimited by the x-y plane (perpendicular to the z-axis) Generic infinite half-space.
<i>XCC</i>	Infinite Circular Cylinder parallel to the x-axis
<i>YCC</i>	Infinite Circular Cylinder parallel to the y-axis
<i>ZCC</i>	Infinite Circular Cylinder parallel to the z-axis
<i>XEC</i>	Infinite Elliptical Cylinder parallel to the x-axis
<i>YEC</i>	Infinite Elliptical Cylinder parallel to the y-axis
<i>ZEC</i>	Infinite Elliptical Cylinder parallel to the z-axis
<i>QUA</i>	Generic quadric

Functions

<i>fluka2Geant4</i> (<i>flukareg</i> [, <i>regions</i> , <i>omitRegions</i> , ...])	Convert a FLUKA registry to a Geant4 Registry.
<i>_rotoTranslationFromTra2</i> (<i>name</i> , <i>tra2</i> [, <i>flukaregistry</i>])	
<i>geant4Reg2FlukaReg</i> (<i>greg</i> [, <i>logicalVolumeName</i>])	Convert a Geant4 model to a FLUKA one. This is done by handing over a complete
<i>geant4Logical2Fluka</i> (<i>logicalVolume</i> [, <i>flukaRegistry</i>])	Convert a single logical volume - not the main entry point for the conversion.

continues on next page

Table 1 – continued from previous page

<code>geant4PhysicalVolume2Fluka</code> (physicalVolume[, mtra, ...])
<code>geant4Solid2FlukaRegion</code> (flukaNameCount, solid[, mtra, ...])
<code>geant4MaterialDict2Fluka</code> (matr, freg)
<code>geant4Material2Fluka</code> (material, freg[, ...])
<code>pycsgmesh2FlukaRegion</code> (mesh, name, transform, ...)
<code>makeStripName</code> (mn)
<code>makeShortName</code> (mn)
<code>freecadDoc2Fluka</code> (fcd)
<code>part2Region</code> (obj, trfm, fgreg[, meshDeviation])
<code>stl2gdml</code> (stlFileName[, gdmlFileName, worldMaterial, ...])
<code>geant4Solid2Geant4Tessellated</code> (solid)
<code>geant4Solid2Geant4Tessellated_NoVTK</code> (solid)
<code>gdml2stl</code> (gdmlFileName[, stlFileName, solidName])
<code>mkVtkIdList</code> (it)
<code>pycsgMeshToVtkPolyData</code> (mesh)
<code>vtkPolyDataToNumpy</code> (data)
<code>pycsgMeshToObj</code> (mesh, fileName)
<code>pyg42VtkTransformation</code> (mtra, tra)
<code>vtkTransformation2PyG4</code> (vt)
<code>pycsgMeshToStl</code> (mesh, fileName)
<code>oceShape_Geant4_LogicalVolume</code> (name, solid, material, greg) Make a logical volume from input or get from registry
<code>oceShape_Geant4_Assembly</code> (name, greg) Make a assembly volume from input or get from registry
<code>oceShape_Geant4_Tessellated</code> (name, shape, greg[, ...]) Make a tessellated solid from a OpenCascade shape
<code>_oce2Geant4_traverse</code> (shapeTool, label, greg, ...[, ...])
<code>oce2Geant4</code>
<code>convertMeshToPolyTriangulation</code> (m)

continues on next page

Table 1 – continued from previous page

`convertMeshToShapeUsingMakeShapeOnMesh(m)`
`convertMeshToShape(m)`
`vis2oce`

Attributes

`defaultLinDef`
`defaultAngDef`

```
pyg4ometry.convert.fluka2Geant4(flukareg, regions=None, omitRegions=None,
                                worldMaterial='G4_Galactic', worldDimensions=None,
                                omitBlackholeRegions=True, quadricRegionAABBs=None, **kwargs)
```

Convert a FLUKA registry to a Geant4 Registry.

Parameters

- **flukareg** (*FlukaRegistry*) – FlukaRegistry instance to be converted.
- **regions** (*list*) – Names of regions to be converted, by default all are converted. Mutually exclusive with omitRegions.
- **omitRegions** (*list*) – Names of regions to be omitted from the conversion. This option is mutually exclusive with the kwarg regions.
- **worldMaterial** (*string*) – name of world material to be used.
- **worldDimensions** (*list*) – dimensions of world logical volume in converted Geant4. By default this is equal to WORLD_DIMENSIONS.
- **omitBlackholeRegions** (*bool*) – whether or not to omit regions with the FLUKA material BLCKHOLE from the conversion. By default, true.
- **quadricRegionAABBs** (*dict*) – The axis-aligned aabbs of any regions featuring QUA bodies, mapping region names to fluka.AABB instances.

Developer options (to kwargs) withLengthSafety: Whether or not to apply automatic length safety.

minimiseSolids: Whether or not to minimise the boxes and tubes of Geant4 used to represent infinite solids in FLUKA.

class pyg4ometry.convert._PolygonProcessing

classmethod windingNumber(*pgon*)

return the winding number of pgon :param pgon: list of points [[x1,y1], [x2,y2], ...] :type pgon: List[List[x1,y1], ...] returns: Integer winding number

classmethod reversePolygon(*pgon*)

return reversed polygon :param pgon: list of points [[x1,y1], [x2,y2], ...] :type pgon: List[List[x1,y1], ...] returns: List[List[x1,y1], ...]

classmethod **makePolygonFromList**(*pgon*, *type*="")

Convert list of points `[[x1,y1], [x2,y2], ...]` to cgal `Polygon_2`

Parameters

- **pgon** (*List[List[x,y], ...]*) – list of points `[[x1,y1], [x2,y2], ...]`
- **type** – Class of polygon (`Polygon_2_EPICK`, `Polygon_2_EPECK`, `Partition_traits_2_Polygon_2_EPECK`)
- **type** – str

returns: `Polygon_2`

classmethod **makeListFromPolygon**(*pgon*)

Convert 2D polygon to list of points `[[x1,y1], [x2,y2], ...]`

Parameters

pgon (*Polygon_2_EPECK* or *Polygon_2_EPICK*) – cgal `Polygon_2` input

returns: `[[x1,y1], [x2,y2], ...]`

classmethod **decomposePolygon2d**(*pgon*)

Decompose general 2D polygon (*pgon*) to convex 2D polygons

Parameters

pgon (*List(List[2])*) – list of pgon points (which are lists) `[[x1,y1], [x2,y2], ...]`

returns: List of polygons `[pgon1, pgon2, ...]`

classmethod **decomposePolygon2dWithHoles**(*pgonOuter*, *pgonHoles*)

Decompose general 2D polygon with holes (*pgon*) to convex 2D polygons

Parameters

- **pgonOuter** – list of pgon points (which are lists) `[[x1,y1], [x2,y2], ...]`
- **pgonHoles** – List of polygons `[pgon1, pgon2, ...]`

returns: List of polygons `[pgon1, pgon2, ...]`

classmethod **triangulatePolygon2d**(*pgon*)

Triangulate general 2D polygon

Parameters

pgonOuter – list of pgon points (which are lists) `[[x1,y1], [x2,y2], ...]`

returns: List of triangles `[[[x1,y1], [x2,y2], [x3,y3]], [[x1,y1], [x2,y2], [x3,y3]], ...]`

`pyg4ometry.convert._rotoTranslationFromTra2`(*name*, *tra2*, *flukaRegistry=None*)

`pyg4ometry.convert.geant4Reg2FlukaReg`(*greg*, *logicalVolumeName=""*)

Convert a Geant4 model to a FLUKA one. This is done by handing over a complete `pyg4ometry.geant4.Registry` instance.

Parameters

greg (`pyg4ometry.geant4.Registry`) – geant4 registry

returns: `pyg4ometry.fluka.FlukaRegistry`

`pyg4ometry.convert.geant4Logical2Fluka`(*logicalVolume*, *flukaRegistry=None*)

Convert a single logical volume - not the main entry point for the conversion.

```
pyg4ometry.convert.geant4PhysicalVolume2Fluka(physicalVolume, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]), flukaRegistry=None, flukaNameCount=0)
```

```
pyg4ometry.convert.geant4Solid2FlukaRegion(flukaNameCount, solid, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]), flukaRegistry=None, addRegistry=True, commentName="")
```

```
pyg4ometry.convert.geant4MaterialDict2Fluka(matr, freg)
```

```
pyg4ometry.convert.geant4Material2Fluka(material, freg, suggestedDensity=None, elementSuffix=False, materialNameShort=None)
```

```
pyg4ometry.convert.pycsgmesh2FlukaRegion(mesh, name, transform, flukaRegistry, commentName)
```

```
pyg4ometry.convert.makeStripName(mn)
```

```
pyg4ometry.convert.makeShortName(mn)
```

```
class pyg4ometry.convert.RPP(name, xmin, xmax, ymin, ymax, zmin, zmax, transform=None, flukaregistry=None, addRegistry=True, comment="")
```

Bases: BodyMixin

Rectangular Parallelepiped

Parameters

- **name** (*str*) – of body
- **xmin** (*float*) – lower x coordinate of RPP
- **xmax** (*float*) – upper x coordinate of RPP
- **ymin** (*float*) – lower y coordinate of RPP
- **ymax** (*float*) – upper y coordinate of RPP
- **zmin** (*float*) – lower z coordinate of RPP
- **zmax** (*float*) – upper z coordinate of RPP

centre(*aabb=None*)

rotation()

lengths()

geant4Solid(*reg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

```
class pyg4ometry.convert.BOX(name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None, comment="")
```

Bases: BodyMixin

General Rectangular Parallelepiped

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the corners.
- **edge1** (*list*) – vector [x, y, z] denoting the first side of the box.
- **edge2** (*list*) – vector [x, y, z] denoting the second side of the box.
- **edge3** (*list*) – vector [x, y, z] denoting the second side of the box.

centre(*aabb=None*)

rotation()

lengths()

geant4Solid(*greg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.convert.**SPH**(*name, point, radius, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin

Sphere

Parameters

- **name** (*str*) – of body
- **point** (*list*) – position [x, y, z] of the centre of the sphere.
- **radius** (*float*) – radius of the sphere.

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.convert.**RCC**(*name, face, direction, radius, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin

Right Circular Cylinder

Parameters

- **name** (*str*) – of body

- **vertex** (*list*) – position [x, y, z] of one of the faces of the cylinder.
- **edge1** (*list*) – vector [x, y, z] denoting the direction along the length of the cylinder.
- **edge2** (*float*) – radius of the cylinder face.

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

```
class pyg4ometry.convert.REC(name, face, direction, semiminor, semimajor, transform=None,
                             flukaregistry=None, comment="")
```

Bases: BodyMixin

Right Elliptical Cylinder

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the faces of the cylinder.
- **semiminor** (*list*) – vector [x, y, z] denoting the direction along the semiminor axis of the ellipse.
- **semimajor** (*list*) – vector [x, y, z] denoting the direction along the semimajor axis of the ellipse.

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

```
class pyg4ometry.convert.TRC(name, major_centre, direction, major_radius, minor_radius, transform=None,
                             flukaregistry=None, comment="")
```

Bases: BodyMixin

Truncated Right-angled Cone

Parameters

- **name** (*str*) – of body
- **major_centre** (*list*) – vector [x, y, z] position of the centre of the larger face.
- **direction** (*list*) – vector [x, y, z] pointing from the larger face to the smaller face.

- **major_radius** (*float*) – radius of the larger face.
- **minor_radius** (*float*) – radius of the smaller face.

centre(*aabb=None*)

rotation()

geant4Solid(*registry, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

```
class pyg4ometry.convert.ELL(name, focus1, focus2, length, transform=None, flukaregistry=None,
                             comment="")
```

Bases: BodyMixin

Ellipsoid of Revolution

Parameters

- **name** (*str*) – of body
- **focus1** (*list*) – position [x, y, z] denoting of one of the foci.
- **focus2** (*list*) – position [x, y, z] denoting the other focus.
- **length** (*float*) – length of the ellipse axis which the foci lie on.

centre(*aabb=None*)

rotation()

_linearEccentricity()

_semiminor()

geant4Solid(*greg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

```
class pyg4ometry.convert.WED(name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None,
                             comment="")
```

Bases: _WED_RAW

Right Angle Wedge

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the the rectangular corners.
- **edge1** (*list*) – vector [x, y, z] denoting height of the wedge.

- **edge2** (*list*) – vector [x, y, z] denoting width of the wedge.
- **edge3** (*list*) – vector [x, y, z] denoting length of the wedge.

```
class pyg4ometry.convert.RAW(name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None,
                             comment="")
```

Bases: `_WED_RAW`

Base class representing a body as defined in FLUKA

`__doc__`

```
class pyg4ometry.convert.ARB(name, vertices, facenumbers, transform=None, flukaregistry=None,
                             comment="")
```

Bases: `BodyMixin`

Arbitrary Convex Polyhedron

Parameters

- **name** (*str*) – of body
- **vertices** (*list*) – Eight vertices which make up the polyhedron as [[x1, y1, z1], [x2, y2, z2], ...]. There must be eight even if only six or seven vertices are needed to make up the polyhedron.
- **facenumbers** (*float*) – The faces of the polyhedron expressed as floats where each digit of the float refers to one of the vertices which makes up that face. Six must always be provided as [1234,8765, ...], even if only four or five faces are needed. Any unneeded faces must be set to 0 (no less than 4 sides). Note that the references to the vertices are not zero-counting. The order of the vertices denoted in the facenumbers must be either all clockwise or all anticlockwise, which if not obeyed will result in erroneous output without warning.

`centre(aabb=None)`

`rotation()`

`_faceNumbersToZeroCountingIndices()`

`_extent()`

`geant4Solid(greg, aabb=None)`

`_toTessellatedSolid(verticesAndPolygons, greg, addRegistry)`

`_getVerticesAndPolygons()`

`_toVerticesAndPolygons(reverse)`

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

```
class pyg4ometry.convert.XYP(name, z, transform=None, flukaregistry=None, comment="")
```

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the z-axis)

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the x-y plane on the z-axis. All points less than z are considered to be part of this body.

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`toPlane()`

`class pyg4ometry.convert.XZP(name, y, transform=None, flukaregistry=None, comment="")`

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the y-axis)

Parameters

- **name** (*str*) – of body
- **y** (*float*) – position of the x-y plane on the y-axis. All points less than y are considered to be part of this body.

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`toPlane()`

`class pyg4ometry.convert.YZP(name, x, transform=None, flukaregistry=None, comment="")`

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the x-axis)

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the x-y plane on the x-axis. All points less than x are considered to be part of this body.

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`toPlane()`

```
class pyg4ometry.convert.PLA(name, normal, point, transform=None, flukaregistry=None, comment="")
```

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the z-axis) Generic infinite half-space.

Parameters

- **name** (*str*) – of body
- **normal** (*list*) – position of a point on the plane
- **normal** – vector perpendicular to the face of the plane, pointing away from the contents of the half space.

rotation()

__repr__()

_withLengthSafety(safety, reg=None)

flukaFreeString()

hash()

toPlane()

```
class pyg4ometry.convert.XCC(name, y, z, radius, transform=None, flukaregistry=None, comment="")
```

Bases: `_InfiniteCylinderMixin`, `_ShiftableCylinderMixin`

Infinite Circular Cylinder parallel to the x-axis

Parameters

- **name** (*str*) – of body
- **y** (*float*) – position of the centre on the
- **z** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

centre(aabb=None)

rotation()

__repr__()

Return repr(self).

_withLengthSafety(safety, reg=None)

flukaFreeString()

hash()

point()

direction()

```
class pyg4ometry.convert.YCC(name, z, x, radius, transform=None, flukaregistry=None, comment="")
```

Bases: `_InfiniteCylinderMixin`, `_ShiftableCylinderMixin`

Infinite Circular Cylinder parallel to the y-axis

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the centre on the
- **x** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

point()

direction()

class pyg4ometry.convert.**ZCC**(*name, x, y, radius, transform=None, flukaregistry=None, comment=""*)

Bases: **_InfiniteCylinderMixin**, **_ShiftableCylinderMixin**

Infinite Circular Cylinder parallel to the z-axis

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the centre on the
- **y** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

point()

direction()

class pyg4ometry.convert.**XEC**(*name, y, z, ysemi, zsemi, transform=None, flukaregistry=None, comment=""*)

Bases: **BodyMixin**, **_ShiftableCylinderMixin**

Infinite Elliptical Cylinder parallel to the x-axis

Parameters

- **name** (*str*) – of body

- **y** (*float*) – position of the centre on the
- **z** (*float*) – position of the centre on the
- **ysemi** (*float*) – position of the centre on the
- **zsemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

class pyg4ometry.convert.**YEC**(*name, z, x, zsemi, xsemi, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin, _ShiftableCylinderMixin

Infinite Elliptical Cylinder parallel to the y-axis

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the centre on the
- **x** (*float*) – position of the centre on the
- **zsemi** (*float*) – position of the centre on the
- **xsemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

class pyg4ometry.convert.**ZEC**(*name, x, y, xsemi, ysemi, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin, _ShiftableCylinderMixin

Infinite Elliptical Cylinder parallel to the z-axis

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the centre on the

- **y** (*float*) – position of the centre on the
- **xsemi** (*float*) – position of the centre on the
- **ysemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

```
class pyg4ometry.convert.QUA(name, cxx, cyy, czz, cxy, cxz, cyz, cx, cy, cz, c, transform=None,
                             flukaregistry=None, comment="", **kwargs)
```

Bases: BodyMixin

Generic quadric

Parameters

- **name** (*str*) – of body
- **cxx** (*float*) – x^2 coefficient
- **cyy** (*float*) – y^2 coefficient
- **czz** (*float*) – z^2 coefficient
- **cxy** (*float*) – xy coefficient
- **cxz** (*float*) – xz coefficient
- **cyz** (*float*) – yz coefficient
- **cx** (*float*) – x coefficient
- **cy** (*float*) – y coefficient
- **cz** (*float*) – z coefficient
- **c** (*constant*) – constant

centre(*aabb=None*)

rotation()

coefficientsMatrix()

static _quadricMatrixToCoefficients(*matrix*)

mesh(*lower, upper, capping=True*)

geant4Solid(*reg, aabb=None*)

_withLengthSafety(*safety, reg=None*)


```

__repr__()
flukaFreeString()
hash()

pyg4ometry.convert.freecadDoc2Fluka(fcd)
pyg4ometry.convert.part2Region(obj, trfm, fgreg, meshDeviation=0.05)
pyg4ometry.convert.stl2gdml(stlFileName, gdmlFileName='output.gdml', worldMaterial='G4_Galactic',
                           solidMaterial='G4_Au')
pyg4ometry.convert.geant4Solid2Geant4Tessellated(solid)
pyg4ometry.convert.geant4Solid2Geant4Tessellated_NoVTK(solid)
pyg4ometry.convert.gdml2stl(gdmlFileName, stlFileName='output.gdml', solidName='ws')
pyg4ometry.convert.mkVtkIdList(it)
pyg4ometry.convert.pycsgMeshToVtkPolyData(mesh)
pyg4ometry.convert.vtkPolyDataToNumpy(data)
pyg4ometry.convert.pycsgMeshToObj(mesh, fileName)
pyg4ometry.convert.pyg42VtkTransformation(mtra, tra)
pyg4ometry.convert.vtkTransformation2PyG4(vt)
pyg4ometry.convert.pycsgMeshToStl(mesh, fileName)
pyg4ometry.convert.defaultLinDef = 0.5
pyg4ometry.convert.deftaulAngDef = 0.5
pyg4ometry.convert.oceShape_Geant4_LogicalVolume(name, solid, material, greg)
    Make a logical volume from input or get from registry

```

Parameters

- **name** (*str*) – Name of logical volume
- **solid** (*SolidBase*) – Geant4 solid
- **material** (*str* or *pyg4ometry.geant4.Material*) – Material for logical volume
- **greg** (*geant4.Registry*) – Geant4 registry

```
pyg4ometry.convert.oceShape_Geant4_Assembly(name, greg)
```

Make a assembly volume from input or get from registry

Parameters

- **name** (*str*) – Name of logical volume
- **greg** (*geant4.Registry*) – Geant4 registry

```
pyg4ometry.convert.oceShape_Geant4_Tessellated(name, shape, greg, linDef=0.5, angDef=0.5)
```

Make a tessellated solid from a OpenCascade shape

Parameters

- **name** (*str*) – Name of logical volume
- **shape** (*TopoDS_Shape*) – OpenCascade shape
- **greg** (*geant4.Registry*) – Geant4 registry

```
pyg4ometry.convert._oce2Geant4_traverse(shapeTool, label, greg, materialMap, labelToSkipList,  
                                         meshQualityMap, badCADLabels, addBoundingSolids=False,  
                                         oceName=False)
```

```
pyg4ometry.convert.oce2Geant4(shapeTool, shapeName, materialMap={}, labelToSkipList=[],  
                               meshQualityMap={}, oceName=False)
```

Convert CAD geometry starting from shapeName

Parameters

- **shapeTool** (*pyoce.TopoDS_Shape*) – OpenCascade TopoDS_Shape
- **shapeName** (*str*) – Name of the shape in the CAD file
- **materialMap** (*dict*) – dictionary to map shape name to material shapeName:materialName
or shapeName:Material
- **meshQualityMap** (*dict*) – dictionary to map shape name to meshing quality
str:[LinDef,AngDef]

```
pyg4ometry.convert.convertMeshToPolyTriangulation(m)
```

```
pyg4ometry.convert.convertMeshToShapeUsingMakeShapeOnMesh(m)
```

```
pyg4ometry.convert.convertMeshToShape(m)
```

```
pyg4ometry.convert.vis2oce(vis, stepFileName='output.step')
```

pyg4ometry.features

Submodules

pyg4ometry.features._accelerator

Module Contents

Functions

```
beamPipeCADFeature(shape)
```

```
beamPipe(stlFileName[, feature1, feature2, ...])
```

```
pyg4ometry.features._accelerator.beamPipeCADFeature(shape)
```

```
pyg4ometry.features._accelerator.beamPipe(stlFileName, feature1=-1, feature2=-1, planeAngles=[[0, 0, 0]], vis=True, interactive=True)
```

`pyg4ometry.features.algos`

Module Contents

Classes

<i>Line</i>	Line class taking a point and dir(ection)
<i>Plane</i>	Plane class taking a point on plane and normal
<i>CoordinateSystem</i>	
<i>vtkViewer</i>	Simple visualiser for feature extraction only
<i>FeatureData</i>	

Functions

<i>vtkLoadStl</i> (fname)	Load the given STL file, and return a vtkPolyData object for it.
<i>vtkPolydataToActor</i> (polydata)	Wrap the provided vtkPolyData object in a mapper and an actor, returning the actor.
<i>vtkPolydataToConnectedEdges</i> (polydata[, angle])	
<i>_vtkPolydataToConnectedEdges</i> (polydata[, angle])	Feature extract from polydata given an angular cut
<i>vtkPolydataEdgeInformation</i> (polydata[, bPlot])	Calculate path information for 2D vtkPolydata lines
<i>_vtkPolydataEdgeInformation</i> (polydata[, bPlot])	Calculate path information for 2D vtkPolydata lines
<i>pyg4PlaneToVtkPlane</i> (pyg4Plane)	Convert Plane to vtkPlane
<i>pyg4ArrayToVtkPolydataLine</i> (pyg4Array)	Convert data array to vtkPolydata
<i>vtkCutterPlane</i> (plane, polydata)	Calculate path information for 2D vtkPolydata lines
<i>_vtkCutterPlane</i> (plane, polydata)	Cutter plane on polydata
<i>extract</i> (inputFileName[, angle, planeQuality, ...])	
<i>test</i> (fileName[, featureIndexList, planeQuality, ...])	

```
class pyg4ometry.features.algos.Line(point, dir)
```

Line class taking a point and dir(ection)

Parameters

dir (*list* or *array* - 3 values) – direction

eval (*par*)

intersectPlane (*plane*)

Intersection of line (self) with plane (unimplemented)

Parameters

plane (*Plane*) – plane object

Returns

Point of intersection between plane and line

Return type

`list` or `array`

`__repr__()`

Return `repr(self)`.

`__iter__()`

`__next__()`

`flatten()`

`class pyg4ometry.features.algos.Plane(point, normal, e2=None, e3=None)`

Plane class taking a point on plane and normal

Parameters

- **point** (`list`, `array`) – point on plane
- **normal** (`list`, `array`) – vector of normal

`intersect(object)`

Intersection between Line or Plane

Parameters

object (`Line` or `Plane`) – Object to intersect with Plane (either Line or Plane)

`intersectLine(line)`

Intersection between plane (self) and line

`intersectPlane(plane)`

Compute intersection between two planes self and plane

`angleBetween(plane)`

Compute angle between two planes (self and plane)

Parameters

plane (`Plane`) – Plane to intersect with self

Returns

angle between two planes

Return type

`float`

`__repr__()`

Return `repr(self)`.

`__iter__()`

`__next__()`

`flatten()`

`class pyg4ometry.features.algos.CoordinateSystem(origin=[0, 0, 0], e1=[1, 0, 0], e2=[0, 1, 0])`

`_commonInit()`

```

makeFromPlanes(p1, p2, p3)

coordinateSystem(rotX=0.0, rotY=0.0, rotZ=0.0)

plane(rotX=0.0, rotY=0.0, rotZ=0.0)

transform(points)

__repr__()
    Return repr(self).

class pyg4ometry.features.algos.vtkViewer
    Simple visualiser for feature extraction only

    _polyDataToActor(polydata, colour=[0, 0, 0, 1], lineWidth=5)
        Wrap the provided vtkPolyData object in a mapper and an actor, returning the actor.

    addPolydata(key, polydata, colour=[0, 0, 0, 1], lineWidth=5, label=None, labelPos=[0, 0, 0])

    addAxis(origin, length=[1, 1, 1], rotation=[[1, 0, 0], [0, 1, 0], [0, 0, 1]], label=False, disableCone=False)

    addPlane(point, p1, p2, length=1)

    addLine(line)

    view(interactive=True)

pyg4ometry.features.algos.vtkLoadStl(fname)
    Load the given STL file, and return a vtkPolyData object for it.

pyg4ometry.features.algos.vtkPolydataToActor(polydata)
    Wrap the provided vtkPolyData object in a mapper and an actor, returning the actor.

pyg4ometry.features.algos.vtkPolydataToConnectedEdges(polydata, angle=89)

pyg4ometry.features.algos._vtkPolydataToConnectedEdges(polydata, angle=89)
    Feature extract from polydata given an angular cut

    Parameters
        angle (float) – Angle between plane > angle is returned

    Returns
        Connected boundaries with > angle

    Return type
        list of vtkPolydata

pyg4ometry.features.algos.vtkPolydataEdgeInformation(polydata, bPlot=False)
    Calculate path information for 2D vtkPolydata lines

    Parameters
        polydata (vtkPolydata or list of vtkPolydata) – Geometry to extract information

    Returns
        Information on each boundary

    Return type
        list of information dict

pyg4ometry.features.algos._vtkPolydataEdgeInformation(polydata, bPlot=False)
    Calculate path information for 2D vtkPolydata lines

```

`pyg4ometry.features.algos.pyg4PlaneToVtkPlane(pyg4Plane)`

Convert Plane to vtkPlane

`pyg4ometry.features.algos.pyg4ArrayToVtkPolydataLine(pyg4Array)`

Convert data array to vtkPolydata

`pyg4ometry.features.algos.vtkCutterPlane(plane, polydata)`

Calculate path information for 2D vtkPolydata lines

Parameters

polydata (*vtkPolydata* or *list* of *vtkPolydata*) – Geometry to extract information

Returns

Information on each boundary

Return type

list of information dict

`pyg4ometry.features.algos._vtkCutterPlane(plane, polydata)`

Cutter plane on polydata

`pyg4ometry.features.algos.extract(inputFileName, angle=89, planeQuality=0.1, circumference=300,
outputFileName=None, planes=[], bViewer=False,
bViewerInteractive=False)`

class `pyg4ometry.features.algos.FeatureData`

readFile(*fileName*)

plotFeature(*iFeature*)

plotCutter(*iCutter*)

`pyg4ometry.features.algos.test(fileName, featureIndexList=[], planeQuality=0.1, circumference=300,
bPlotRadii=False)`

Package Contents

Classes

<i>CoordinateSystem</i>	
<i>Plane</i>	Plane class taking a point on plane and normal
<i>FeatureData</i>	

Functions

```
extract(inputFileName[, angle, planeQuality, ...])
```

```
class pyg4ometry.features.CoordinateSystem(origin=[0, 0, 0], e1=[1, 0, 0], e2=[0, 1, 0])
```

```
    _commonInit()
```

```
    makeFromPlanes(p1, p2, p3)
```

```
    coordinateSystem(rotX=0.0, rotY=0.0, rotZ=0.0)
```

```
    plane(rotX=0.0, rotY=0.0, rotZ=0.0)
```

```
    transform(points)
```

```
    __repr__()
```

```
        Return repr(self).
```

```
class pyg4ometry.features.Plane(point, normal, e2=None, e3=None)
```

```
    Plane class taking a point on plane and normal
```

Parameters

- **point** (*list*, *array*) – point on plane
- **normal** (*list*, *array*) – vector of normal

```
    intersect(object)
```

```
        Intersection between Line or Plane
```

Parameters

object (*Line* or *Plane*) – Object to intersect with Plane (either Line or Plane)

```
    intersectLine(line)
```

```
        Intersection between plane (self) and line
```

```
    intersectPlane(plane)
```

```
        Compute intersection between two planes self and plane
```

```
    angleBetween(plane)
```

```
        Compute angle between two planes (self and plane)
```

Parameters

plane (*Plane*) – Plane to intersect with self

Returns

angle between two planes

Return type

float

```
    __repr__()
```

```
        Return repr(self).
```

```
    __iter__()
```

`__next__()`

`flatten()`

`pyg4ometry.features.extract(inputFileName, angle=89, planeQuality=0.1, circumference=300, outputFileName=None, planes=[], bViewer=False, bViewerInteractive=False)`

`class pyg4ometry.features.FeatureData`

`readFile(fileName)`

`plotFeature(iFeature)`

`plotCutter(iCutter)`

`pyg4ometry.fluka`

Subpackages

`pyg4ometry.fluka.RegionExpression`

Submodules

`pyg4ometry.fluka.RegionExpression.RegionLexer`

Module Contents

Classes

RegionLexer

Functions

serializedATN()

`pyg4ometry.fluka.RegionExpression.RegionLexer.serializedATN()`

`class pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer(input=None, output=sys.stdout)`

Bases: `Lexer`

Parameters

`output` (*TextIO*) –

`atn`

`decisionsToDFA`


```

Whitespace = 1
InLineComment = 2
LineComment = 3
Newline = 4
Integer = 5
RegionName = 6
BodyName = 7
Plus = 8
Minus = 9
Bar = 10
LParen = 11
RParen = 12

channelNames = ['DEFAULT_TOKEN_CHANNEL', 'HIDDEN']

modeNames = ['DEFAULT_MODE']

literalNames = ['<INVALID>', "'+'", "'-'", "'|'", "'('", "')'"]

symbolicNames = ['<INVALID>', 'Whitespace', 'InLineComment', 'LineComment',
'Newline', 'Integer', 'RegionName',...]

ruleNames = ['Whitespace', 'InLineComment', 'LineComment', 'Newline', 'Integer',
'Digit', 'RegionName',...]

grammarFileName = 'RegionLexer.g4'

sempred(localctx, ruleIndex, predIndex)

```

Parameters

- **localctx** (*RuleContext*) –
- **ruleIndex** (*int*) –
- **predIndex** (*int*) –

LineComment_sempred(*localctx*, *predIndex*)

Parameters

- **localctx** (*RuleContext*) –
- **predIndex** (*int*) –

RegionName_sempred(*localctx*, *predIndex*)

Parameters

- **localctx** (*RuleContext*) –
- **predIndex** (*int*) –

pyg4ometry.fluka.RegionExpression.RegionParser

Module Contents

Classes

RegionParser

Functions

serializedATN()

pyg4ometry.fluka.RegionExpression.RegionParser.**serializedATN()**

class pyg4ometry.fluka.RegionExpression.RegionParser.**RegionParser**(*input, output=sys.stdout*)

Bases: Parser

Parameters

- **input** (*TokenStream*) –
- **output** (*TextIO*) –

class **RegionsContext**(*parser, parent=None, invokingState=-1*)

Bases: ParserRuleContext

Parameters

- **parent** (*ParserRuleContext*) –
- **invokingState** (*int*) –

__slots__ = 'parser'

region(*i=None*)

Parameters

- **i** (*Optional[int]*) –

getRuleIndex()

accept(*visitor*)

Parameters

- **visitor** (*ParseTreeVisitor*) –

class **RegionContext**(*parser, parent=None, invokingState=-1*)

Bases: ParserRuleContext

Parameters

- **parent** (*ParserRuleContext*) –
- **invokingState** (*int*) –

```

__slots__ = 'parser'

getRuleIndex()

copyFrom(ctx)
    Parameters
        ctx (ParserRuleContext) –

class ComplexRegionContext(parser, ctx)
    Bases: RegionContext
        Parameters
            ctx (ParserRuleContext) –

    RegionName()

    Integer()

    zoneUnion()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class SimpleRegionContext(parser, ctx)
    Bases: RegionContext
        Parameters
            ctx (ParserRuleContext) –

    RegionName()

    Integer()

    zone()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class ZoneUnionContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –

    __slots__ = 'parser'

    getRuleIndex()

    copyFrom(ctx)
        Parameters
            ctx (ParserRuleContext) –

```

```
class MultipleUnion2Context(parser, ctx)
    Bases: ZoneUnionContext

    Parameters
        ctx (ParserRuleContext) –

    zone(i=None)

    Parameters
        i (Optional[int]) –

    Bar(i=None)

    Parameters
        i (Optional[int]) –

    accept(visitor)

    Parameters
        visitor (ParseTreeVisitor) –

class MultipleUnionContext(parser, ctx)
    Bases: ZoneUnionContext

    Parameters
        ctx (ParserRuleContext) –

    Bar(i=None)

    Parameters
        i (Optional[int]) –

    zone(i=None)

    Parameters
        i (Optional[int]) –

    accept(visitor)

    Parameters
        visitor (ParseTreeVisitor) –

class SingleUnionContext(parser, ctx)
    Bases: ZoneUnionContext

    Parameters
        ctx (ParserRuleContext) –

    Bar()

    zone()

    accept(visitor)

    Parameters
        visitor (ParseTreeVisitor) –

class ZoneContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext

    Parameters
        • parent (ParserRuleContext) –
        • invokingState (int) –
```

```

__slots__ = 'parser'

getRuleIndex()

copyFrom(ctx)
    Parameters
        ctx (ParserRuleContext) –

class ZoneExprContext(parser, ctx)
    Bases: ZoneContext
        Parameters
            ctx (ParserRuleContext) –

        expr()

        BodyName()

        accept(visitor)
            Parameters
                visitor (ParseTreeVisitor) –

class ZoneBodyContext(parser, ctx)
    Bases: ZoneContext
        Parameters
            ctx (ParserRuleContext) –

        BodyName()

        accept(visitor)
            Parameters
                visitor (ParseTreeVisitor) –

class ZoneSubZoneContext(parser, ctx)
    Bases: ZoneContext
        Parameters
            ctx (ParserRuleContext) –

        subZone()

        BodyName()

        accept(visitor)
            Parameters
                visitor (ParseTreeVisitor) –

class ExprContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –

        __slots__ = 'parser'

        getRuleIndex()

```

```
    copyFrom(ctx)
        Parameters
            ctx (ParserRuleContext) –

class UnaryAndBooleanContext(parser, ctx)
    Bases: ExprContext
        Parameters
            ctx (ParserRuleContext) –
    unaryExpression()
    expr()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class OneSubZoneContext(parser, ctx)
    Bases: ExprContext
        Parameters
            ctx (ParserRuleContext) –
    subZone()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class UnaryAndSubZoneContext(parser, ctx)
    Bases: ExprContext
        Parameters
            ctx (ParserRuleContext) –
    subZone()
    expr()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class SingleUnaryContext(parser, ctx)
    Bases: ExprContext
        Parameters
            ctx (ParserRuleContext) –
    unaryExpression()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class SubZoneContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
```

```

        • parent (ParserRuleContext) –
        • invokingState (int) –
__slots__ = 'parser'

LParen()

expr()

RParen()

Minus()

Plus()

BodyName()

getRuleIndex()

accept(visitor)

    Parameters
        visitor (ParseTreeVisitor) –

class UnaryExpressionContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext

        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –
__slots__ = 'parser'

BodyName()

Minus()

Plus()

getRuleIndex()

accept(visitor)

    Parameters
        visitor (ParseTreeVisitor) –

grammarFileName = 'RegionParser.g4'

atn

decisionsToDFA

sharedContextCache

literalNames = ['<INVALID>', '<INVALID>', '<INVALID>', '<INVALID>', '<INVALID>',
'<INVALID>', '<INVALID>',...

symbolicNames = ['<INVALID>', 'Whitespace', 'InLineComment', 'LineComment',
'Newline', 'Integer', 'RegionName',...

```

```
RULE_regions = 0
RULE_region = 1
RULE_zoneUnion = 2
RULE_zone = 3
RULE_expr = 4
RULE_subZone = 5
RULE_unaryExpression = 6

ruleNames = ['regions', 'region', 'zoneUnion', 'zone', 'expr', 'subZone',
'UnaryExpression']

EOF

Whitespace = 1
InLineComment = 2
LineComment = 3
Newline = 4
Integer = 5
RegionName = 6
BodyName = 7
Plus = 8
Minus = 9
Bar = 10
LParen = 11
RParen = 12

regions()
region()
zoneUnion()
zone()
expr()
subZone()
unaryExpression()
```


pyg4ometry.fluka.RegionExpression.RegionParserVisitor

Module Contents

Classes

RegionParserVisitor

class pyg4ometry.fluka.RegionExpression.RegionParserVisitor.**RegionParserVisitor**

Bases: ParseTreeVisitor

visitRegions(*ctx*)

Parameters

ctx (RegionParser.RegionsContext) –

visitSimpleRegion(*ctx*)

Parameters

ctx (RegionParser.SimpleRegionContext) –

visitComplexRegion(*ctx*)

Parameters

ctx (RegionParser.ComplexRegionContext) –

visitMultipleUnion(*ctx*)

Parameters

ctx (RegionParser.MultipleUnionContext) –

visitSingleUnion(*ctx*)

Parameters

ctx (RegionParser.SingleUnionContext) –

visitMultipleUnion2(*ctx*)

Parameters

ctx (RegionParser.MultipleUnion2Context) –

visitZoneExpr(*ctx*)

Parameters

ctx (RegionParser.ZoneExprContext) –

visitZoneSubZone(*ctx*)

Parameters

ctx (RegionParser.ZoneSubZoneContext) –

visitZoneBody(*ctx*)

Parameters

ctx (RegionParser.ZoneBodyContext) –

visitSingleUnary(*ctx*)

Parameters

ctx (`RegionParser.SingleUnaryContext`) –

visitUnaryAndBoolean(*ctx*)

Parameters

ctx (`RegionParser.UnaryAndBooleanContext`) –

visitUnaryAndSubZone(*ctx*)

Parameters

ctx (`RegionParser.UnaryAndSubZoneContext`) –

visitOneSubZone(*ctx*)

Parameters

ctx (`RegionParser.OneSubZoneContext`) –

visitSubZone(*ctx*)

Parameters

ctx (`RegionParser.SubZoneContext`) –

visitUnaryExpression(*ctx*)

Parameters

ctx (`RegionParser.UnaryExpressionContext`) –

Package Contents

Classes

RegionParserVisitor

RegionParser

RegionLexer

class `pyg4ometry.fluka.RegionExpression.RegionParserVisitor`

Bases: `ParseTreeVisitor`

visitRegions(*ctx*)

Parameters

ctx (`RegionParser.RegionsContext`) –

visitSimpleRegion(*ctx*)

Parameters

ctx (`RegionParser.SimpleRegionContext`) –

visitComplexRegion(*ctx*)

Parameters

ctx (`RegionParser.ComplexRegionContext`) –

visitMultipleUnion(*ctx*)

Parameters

ctx (`RegionParser.MultipleUnionContext`) –

visitSingleUnion(*ctx*)

Parameters

ctx (`RegionParser.SingleUnionContext`) –

visitMultipleUnion2(*ctx*)

Parameters

ctx (`RegionParser.MultipleUnion2Context`) –

visitZoneExpr(*ctx*)

Parameters

ctx (`RegionParser.ZoneExprContext`) –

visitZoneSubZone(*ctx*)

Parameters

ctx (`RegionParser.ZoneSubZoneContext`) –

visitZoneBody(*ctx*)

Parameters

ctx (`RegionParser.ZoneBodyContext`) –

visitSingleUnary(*ctx*)

Parameters

ctx (`RegionParser.SingleUnaryContext`) –

visitUnaryAndBoolean(*ctx*)

Parameters

ctx (`RegionParser.UnaryAndBooleanContext`) –

visitUnaryAndSubZone(*ctx*)

Parameters

ctx (`RegionParser.UnaryAndSubZoneContext`) –

visitOneSubZone(*ctx*)

Parameters

ctx (`RegionParser.OneSubZoneContext`) –

visitSubZone(*ctx*)

Parameters

ctx (`RegionParser.SubZoneContext`) –

visitUnaryExpression(*ctx*)

Parameters

ctx ([RegionParser.UnaryExpressionContext](#)) –

class `pyg4ometry.fluka.RegionExpression.RegionParser`(*input*, *output=sys.stdout*)

Bases: `Parser`

Parameters

- **input** (`TokenStream`) –

- **output** (`TextIO`) –

class `RegionsContext`(*parser*, *parent=None*, *invokingState=-1*)

Bases: `ParserRuleContext`

Parameters

- **parent** (`ParserRuleContext`) –

- **invokingState** (`int`) –

__slots__ = 'parser'

region(*i=None*)

Parameters

i (`Optional[int]`) –

getRuleIndex()

accept(*visitor*)

Parameters

visitor (`ParseTreeVisitor`) –

class `RegionContext`(*parser*, *parent=None*, *invokingState=-1*)

Bases: `ParserRuleContext`

Parameters

- **parent** (`ParserRuleContext`) –

- **invokingState** (`int`) –

__slots__ = 'parser'

getRuleIndex()

copyFrom(*ctx*)

Parameters

ctx (`ParserRuleContext`) –

class `ComplexRegionContext`(*parser*, *ctx*)

Bases: `RegionContext`

Parameters

ctx (`ParserRuleContext`) –

RegionName()

Integer()

```

zoneUnion()

accept(visitor)
    Parameters
        visitor (ParseTreeVisitor) –

class SimpleRegionContext(parser, ctx)
    Bases: RegionContext
        Parameters
            ctx (ParserRuleContext) –

    RegionName()

    Integer()

    zone()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class ZoneUnionContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –

    __slots__ = 'parser'

    getRuleIndex()

    copyFrom(ctx)
        Parameters
            ctx (ParserRuleContext) –

class MultipleUnion2Context(parser, ctx)
    Bases: ZoneUnionContext
        Parameters
            ctx (ParserRuleContext) –

    zone(i=None)
        Parameters
            i (Optional[int]) –

    Bar(i=None)
        Parameters
            i (Optional[int]) –

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

```

```
class MultipleUnionContext(parser, ctx)
    Bases: ZoneUnionContext
        Parameters
            ctx (ParserRuleContext) –
        Bar(i=None)
            Parameters
                i (Optional[int]) –
        zone(i=None)
            Parameters
                i (Optional[int]) –
        accept(visitor)
            Parameters
                visitor (ParseTreeVisitor) –
class SingleUnionContext(parser, ctx)
    Bases: ZoneUnionContext
        Parameters
            ctx (ParserRuleContext) –
        Bar()
        zone()
        accept(visitor)
            Parameters
                visitor (ParseTreeVisitor) –
class ZoneContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –
        __slots__ = 'parser'
        getRuleIndex()
        copyFrom(ctx)
            Parameters
                ctx (ParserRuleContext) –
class ZoneExprContext(parser, ctx)
    Bases: ZoneContext
        Parameters
            ctx (ParserRuleContext) –
        expr()
        BodyName()
```

```

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class ZoneBodyContext(parser, ctx)
    Bases: ZoneContext

        Parameters
            ctx (ParserRuleContext) –

    BodyName()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class ZoneSubZoneContext(parser, ctx)
    Bases: ZoneContext

        Parameters
            ctx (ParserRuleContext) –

    subZone()

    BodyName()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class ExprContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext

        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –

    __slots__ = 'parser'

    getRuleIndex()

    copyFrom(ctx)
        Parameters
            ctx (ParserRuleContext) –

class UnaryAndBooleanContext(parser, ctx)
    Bases: ExprContext

        Parameters
            ctx (ParserRuleContext) –

    unaryExpression()

    expr()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

```

```
class OneSubZoneContext(parser, ctx)
    Bases: ExprContext
        Parameters
            ctx (ParserRuleContext) –
        subZone()
        accept(visitor)
            Parameters
                visitor (ParseTreeVisitor) –
class UnaryAndSubZoneContext(parser, ctx)
    Bases: ExprContext
        Parameters
            ctx (ParserRuleContext) –
        subZone()
        expr()
        accept(visitor)
            Parameters
                visitor (ParseTreeVisitor) –
class SingleUnaryContext(parser, ctx)
    Bases: ExprContext
        Parameters
            ctx (ParserRuleContext) –
        unaryExpression()
        accept(visitor)
            Parameters
                visitor (ParseTreeVisitor) –
class SubZoneContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –
    __slots__ = 'parser'
    LParen()
    expr()
    RParen()
    Minus()
    Plus()
    BodyName()
```



```

    getRuleIndex()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class UnaryExpressionContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext

    Parameters
        • parent (ParserRuleContext) –
        • invokingState (int) –

    __slots__ = 'parser'

    BodyName()

    Minus()

    Plus()

    getRuleIndex()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

grammarFileName = 'RegionParser.g4'

atn

decisionsToDFA

sharedContextCache

literalNames = ['<INVALID>', '<INVALID>', '<INVALID>', '<INVALID>', '<INVALID>',
'<INVALID>', '<INVALID>',...]

symbolicNames = ['<INVALID>', 'Whitespace', 'InLineComment', 'LineComment',
'Newline', 'Integer', 'RegionName',...]

RULE_regions = 0

RULE_region = 1

RULE_zoneUnion = 2

RULE_zone = 3

RULE_expr = 4

RULE_subZone = 5

RULE_unaryExpression = 6

ruleNames = ['regions', 'region', 'zoneUnion', 'zone', 'expr', 'subZone',
'unaryExpression']

```

```
EOF
Whitespace = 1
InLineComment = 2
LineComment = 3
Newline = 4
Integer = 5
RegionName = 6
BodyName = 7
Plus = 8
Minus = 9
Bar = 10
LParen = 11
RParen = 12
regions()
region()
zoneUnion()
zone()
expr()
subZone()
unaryExpression()
```

```
class pyg4ometry.fluka.RegionExpression.RegionLexer(input=None, output=sys.stdout)
```

```
    Bases: Lexer
```

```
        Parameters
```

```
            output (TextIO) –
```

```
        atn
```

```
        decisionsToDFA
```

```
        Whitespace = 1
```

```
        InLineComment = 2
```

```
        LineComment = 3
```

```
        Newline = 4
```

```
        Integer = 5
```

```
        RegionName = 6
```

```

BodyName = 7

Plus = 8

Minus = 9

Bar = 10

LParen = 11

RParen = 12

channelNames = ['DEFAULT_TOKEN_CHANNEL', 'HIDDEN']

modeNames = ['DEFAULT_MODE']

literalNames = ['<INVALID>', "'+'", "'-'", "'|'", "'('", "')'"]

symbolicNames = ['<INVALID>', 'Whitespace', 'InLineComment', 'LineComment',
'Newline', 'Integer', 'RegionName',...]

ruleNames = ['Whitespace', 'InLineComment', 'LineComment', 'Newline', 'Integer',
'Digit', 'RegionName',...]

grammarFileName = 'RegionLexer.g4'

sempred(localctx, ruleIndex, predIndex)

```

Parameters

- `localctx` (*RuleContext*) –
- `ruleIndex` (*int*) –
- `predIndex` (*int*) –

`LineComment_sempred(localctx, predIndex)`

Parameters

- `localctx` (*RuleContext*) –
- `predIndex` (*int*) –

`RegionName_sempred(localctx, predIndex)`

Parameters

- `localctx` (*RuleContext*) –
- `predIndex` (*int*) –

Submodules

`pyg4ometry.fluka.Writer`

```

GLOBAL      20000.      -1.      1.      0.      0.      0.
DEFAULTS      0.0      0.0      0.0      0.0      0.0      0.0 EM-CASCA
BEAM      17.5      0.0      10000.0      0.0      0.0      1.0 ELECTRON
GEOBEGIN      3.
              0      0
              0      0

USRBIN      , 10., ELECTRON, -21., 250., 0.1, 2000., ElecTop
USRBIN      , -250., -0.1, -2000., 400., 1., 400., &
USRBIN      , 10., POSITRON, -22., 250., 0.1, 2000., PosTop
USRBIN      , -250., -0.1, -2000., 400., 1., 400., &
USRBIN      , 10., PHOTON, -23., 250., 0.1, 2000., PhotTop
USRBIN      , -250., -0.1, -2000., 400., 1., 400., &
RANDOMIZ      , 1.
START      , 1000.
STOP
END

```

Module Contents

Classes

Writer

Class to write FLUKA input files from a fluka registry object.

class pyg4ometry.fluka.Writer.**Writer**

Class to write FLUKA input files from a fluka registry object.

```

>>> f = Writer()
>>> f.addDetector(flukaRegObject)
>>> f.write("model.inp")

```

```

_flukaFFString =
'*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...'

```

addDetector(*flukaRegistry*)

Set the fluka registry and therefore the model for this writer instance.

write(*fileName*)

Write the output to a given filename. e.g. “model.inp”.

pyg4ometry.fluka.body

Set of classes for FLUKA bodies.

Module Contents

Classes

<i>RPP</i>	Rectangular Parallelepiped
<i>BOX</i>	General Rectangular Parallelepiped
<i>SPH</i>	Sphere
<i>RCC</i>	Right Circular Cylinder
<i>REC</i>	Right Elliptical Cylinder
<i>TRC</i>	Truncated Right-angled Cone
<i>ELL</i>	Ellipsoid of Revolution
<i>WED</i>	Right Angle Wedge
<i>RAW</i>	Base class representing a body as defined in FLUKA
<i>ARB</i>	Arbitrary Convex Polyhedron
<i>XYP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the z-axis)
<i>XZP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the y-axis)
<i>YZP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the x-axis)
<i>PLA</i>	Infinite half-space delimited by the x-y plane (perpendicular to the z-axis) Generic infinite half-space.
<i>XCC</i>	Infinite Circular Cylinder parallel to the x-axis
<i>YCC</i>	Infinite Circular Cylinder parallel to the y-axis
<i>ZCC</i>	Infinite Circular Cylinder parallel to the z-axis
<i>XEC</i>	Infinite Elliptical Cylinder parallel to the x-axis
<i>YEC</i>	Infinite Elliptical Cylinder parallel to the y-axis
<i>ZEC</i>	Infinite Elliptical Cylinder parallel to the z-axis
<i>QUA</i>	Generic quadric

```
class pyg4ometry.fluka.body.RPP(name, xmin, xmax, ymin, ymax, zmin, zmax, transform=None,
                                flukaregistry=None, addRegistry=True, comment="")
```

Bases: BodyMixin

Rectangular Parallelepiped

Parameters

- **name** (*str*) – of body
- **xmin** (*float*) – lower x coordinate of RPP
- **xmax** (*float*) – upper x coordinate of RPP
- **ymin** (*float*) – lower y coordinate of RPP
- **ymax** (*float*) – upper y coordinate of RPP
- **zmin** (*float*) – lower z coordinate of RPP
- **zmax** (*float*) – upper z coordinate of RPP

centre(aabb=None)

rotation()

`lengths()`

`geant4Solid(reg, aabb=None)`

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`class pyg4ometry.fluka.body.BOX(name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None, comment="")`

Bases: `BodyMixin`

General Rectangular Parallelepiped

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the corners.
- **edge1** (*list*) – vector [x, y, z] denoting the first side of the box.
- **edge2** (*list*) – vector [x, y, z] denoting the second side of the box.
- **edge3** (*list*) – vector [x, y, z] denoting the second side of the box.

`centre(aabb=None)`

`rotation()`

`lengths()`

`geant4Solid(greg, aabb=None)`

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`class pyg4ometry.fluka.body.SPH(name, point, radius, transform=None, flukaregistry=None, comment="")`

Bases: `BodyMixin`

Sphere

Parameters

- **name** (*str*) – of body
- **point** (*list*) – position [x, y, z] of the centre of the sphere.
- **radius** (*float*) – radius of the sphere.

`centre(aabb=None)`

`rotation()`

`geant4Solid(reg, aabb=None)`

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

```
class pyg4ometry.fluka.body.RCC(name, face, direction, radius, transform=None, flukaregistry=None,
                                comment="")
```

Bases: BodyMixin

Right Circular Cylinder

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the faces of the cylinder.
- **edge1** (*list*) – vector [x, y, z] denoting the direction along the length of the cylinder.
- **edge2** (*float*) – radius of the cylinder face.

`centre(aabb=None)`

`rotation()`

`geant4Solid(reg, aabb=None)`

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

```
class pyg4ometry.fluka.body.REC(name, face, direction, semiminor, semimajor, transform=None,
                                flukaregistry=None, comment="")
```

Bases: BodyMixin

Right Elliptical Cylinder

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the faces of the cylinder.
- **semiminor** (*list*) – vector [x, y, z] denoting the direction along the semiminor axis of the ellipse.
- **semimajor** (*list*) – vector [x, y, z] denoting the direction along the semimajor axis of the ellipse.

`centre(aabb=None)`

`rotation()`

`geant4Solid(reg, aabb=None)`

`__repr__()`

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.fluka.body.**TRC**(*name, major_centre, direction, major_radius, minor_radius, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin

Truncated Right-angled Cone

Parameters

- **name** (*str*) – of body
- **major_centre** (*list*) – vector [x, y, z] position of the centre of the larger face.
- **direction** (*list*) – vector [x, y, z] pointing from the larger face to the smaller face.
- **major_radius** (*float*) – radius of the larger face.
- **minor_radius** (*float*) – radius of the smaller face.

centre(*aabb=None*)

rotation()

geant4Solid(*registry, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.fluka.body.**ELL**(*name, focus1, focus2, length, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin

Ellipsoid of Revolution

Parameters

- **name** (*str*) – of body
- **focus1** (*list*) – position [x, y, z] denoting of one of the foci.
- **focus2** (*list*) – position [x, y, z] denoting the other focus.
- **length** (*float*) – length of the ellipse axis which the foci lie on.

centre(*aabb=None*)

rotation()

_linearEccentricity()

_semiminor()

geant4Solid(*greg, aabb=None*)

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

class `pyg4ometry.fluka.body.WED(name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None, comment="")`

Bases: `_WED_RAW`

Right Angle Wedge

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the the rectangular corners.
- **edge1** (*list*) – vector [x, y, z] denoting height of the wedge.
- **edge2** (*list*) – vector [x, y, z] denoting width of the wedge.
- **edge3** (*list*) – vector [x, y, z] denoting length of the wedge.

class `pyg4ometry.fluka.body.RAW(name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None, comment="")`

Bases: `_WED_RAW`

Base class representing a body as defined in FLUKA

`__doc__`

class `pyg4ometry.fluka.body.ARB(name, vertices, facenumbers, transform=None, flukaregistry=None, comment="")`

Bases: `BodyMixin`

Arbitrary Convex Polyhedron

Parameters

- **name** (*str*) – of body
- **vertices** (*list*) – Eight vertices which make up the polyhedron as [[x1, y1, z1], [x2, y2, z2], ...]. There must be eight even if only six or seven vertices are needed to make up the polyhedron.
- **facenumbers** (*float*) – The faces of the polyhedron expressed as floats where each digit of the float refers to one of the vertices which makes up that face. Six must always be provided as [1234,8765, ...], even if only four or five faces are needed. Any unneeded faces must be set to 0 (no less than 4 sides). Note that the references to the vertices are not zero-counting. The order of the vertices denoted in the facenumbers must be either all clockwise or all anticlockwise, which if not obeyed will result in erroneous output without warning.

`centre(aabb=None)`

`rotation()`

`_faceNumbersToZeroCountingIndices()`

`_extent()`

```
geant4Solid(greg, aabb=None)
_toTessellatedSolid(verticesAndPolygons, greg, addRegistry)
_getVerticesAndPolygons()
_toVerticesAndPolygons(reverse)
__repr__()
_withLengthSafety(safety, reg)
flukaFreeString()
hash()
```

```
class pyg4ometry.fluka.body.XYP(name, z, transform=None, flukaregistry=None, comment="")
```

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the z-axis)

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the x-y plane on the z-axis. All points less than z are considered to be part of this body.

```
__repr__()
_withLengthSafety(safety, reg)
flukaFreeString()
hash()
toPlane()
```

```
class pyg4ometry.fluka.body.XZP(name, y, transform=None, flukaregistry=None, comment="")
```

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the y-axis)

Parameters

- **name** (*str*) – of body
- **y** (*float*) – position of the x-y plane on the y-axis. All points less than y are considered to be part of this body.

```
__repr__()
_withLengthSafety(safety, reg)
flukaFreeString()
hash()
toPlane()
```

```
class pyg4ometry.fluka.body.YZP(name, x, transform=None, flukaregistry=None, comment="")
```

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the x-axis)

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the x-y plane on the x-axis. All points less than x are considered to be part of this body.

```
__repr__()
```

```
_withLengthSafety(safety, reg)
```

```
flukaFreeString()
```

```
hash()
```

```
toPlane()
```

```
class pyg4ometry.fluka.body.PLA(name, normal, point, transform=None, flukaregistry=None, comment="")
```

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the z-axis) Generic infinite half-space.

Parameters

- **name** (*str*) – of body
- **normal** (*list*) – position of a point on the plane
- **normal** – vector perpendicular to the face of the plane, pointing away from the contents of the half space.

```
rotation()
```

```
__repr__()
```

```
_withLengthSafety(safety, reg=None)
```

```
flukaFreeString()
```

```
hash()
```

```
toPlane()
```

```
class pyg4ometry.fluka.body.XCC(name, y, z, radius, transform=None, flukaregistry=None, comment="")
```

Bases: `_InfiniteCylinderMixin`, `_ShiftableCylinderMixin`

Infinite Circular Cylinder parallel to the x-axis

Parameters

- **name** (*str*) – of body
- **y** (*float*) – position of the centre on the
- **z** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

```
centre(aabb=None)
```

rotation()

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

point()

direction()

class pyg4ometry.fluka.body.**YCC**(*name, z, x, radius, transform=None, flukaregistry=None, comment=""*)

Bases: **_InfiniteCylinderMixin**, **_ShiftableCylinderMixin**

Infinite Circular Cylinder parallel to the y-axis

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the centre on the
- **x** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

point()

direction()

class pyg4ometry.fluka.body.**ZCC**(*name, x, y, radius, transform=None, flukaregistry=None, comment=""*)

Bases: **_InfiniteCylinderMixin**, **_ShiftableCylinderMixin**

Infinite Circular Cylinder parallel to the z-axis

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the centre on the
- **y** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

__repr__()

Return repr(self).

_withLengthSafety(*safety*, *reg=None*)

flukaFreeString()

hash()

point()

direction()

class pyg4ometry.fluka.body.XEC(*name*, *y*, *z*, *ysemi*, *zsemi*, *transform=None*, *flukaregistry=None*, *comment=""*)

Bases: BodyMixin, _ShiftableCylinderMixin

Infinite Elliptical Cylinder parallel to the x-axis

Parameters

- **name** (*str*) – of body
- **y** (*float*) – position of the centre on the
- **z** (*float*) – position of the centre on the
- **ysemi** (*float*) – position of the centre on the
- **zsemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg*, *aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety*, *reg=None*)

flukaFreeString()

hash()

class pyg4ometry.fluka.body.YEC(*name*, *z*, *x*, *zsemi*, *xsemi*, *transform=None*, *flukaregistry=None*, *comment=""*)

Bases: BodyMixin, _ShiftableCylinderMixin

Infinite Elliptical Cylinder parallel to the y-axis

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the centre on the
- **x** (*float*) – position of the centre on the
- **zsemi** (*float*) – position of the centre on the
- **xsemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

class pyg4ometry.fluka.body.**ZEC**(*name, x, y, xsemi, ysemi, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin, _ShiftableCylinderMixin

Infinite Elliptical Cylinder parallel to the z-axis

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the centre on the
- **y** (*float*) – position of the centre on the
- **xsemi** (*float*) – position of the centre on the
- **ysemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

class pyg4ometry.fluka.body.**QUA**(*name, cxx, cyy, czz, cxy, cxz, cyz, cx, cy, cz, c, transform=None, flukaregistry=None, comment="", **kwargs*)

Bases: BodyMixin

Generic quadric

Parameters

- **name** (*str*) – of body
- **cxx** (*float*) – x^2 coefficient
- **cyy** (*float*) – y^2 coefficient
- **czz** (*float*) – z^2 coefficient
- **cxy** (*float*) – xy coefficient

- **cxz** (*float*) – xz coefficient
- **cyz** (*float*) – yz coefficient
- **cx** (*float*) – x coefficient
- **cy** (*float*) – y coefficient
- **cz** (*float*) – z coefficient
- **c** (*constant*) – constant

centre(*aabb=None*)

rotation()

coefficientsMatrix()

static **_quadricMatrixToCoefficients**(*matrix*)

mesh(*lower, upper, capping=True*)

geant4Solid(*reg, aabb=None*)

_withLengthSafety(*safety, reg=None*)

__repr__()

flukaFreeString()

hash()

`pyg4ometry.fluka.boolean_algebra`

Module Contents

Functions

<code><i>expressionToZone</i>(zone, zoneExpr)</code>
<code><i>zoneToDNFZones</i>(zone)</code>
<code><i>regionToAlgebraicExpression</i>(region)</code>
<code><i>zoneToAlgebraicExpression</i>(zone)</code>
<code><i>_getMeshAndAABB</i>(body[, aabb])</code>
<code><i>pruneRegion</i>(reg[, aabb])</code>
<code><i>isZoneContradiction</i>(zone)</code>
<code><i>pruneZone</i>(zone[, aabb0, aabb])</code>
<code><i>squashDegenerateBodies</i>(zone[, bodystore])</code>
<code><i>simplifyRegion</i>(region)</code>
<code><i>_filterRedunantZonesSymbollicaly</i>(zoneData)</code>
<code><i>_filterRedundantZonesMetricCheck</i>(zoneData)</code>
<code><i>nTermsDNF</i>(expr)</code>
<code><i>_nTermsDNFZone</i>(zone)</code>

Attributes

<code><i>_MeshedZoneInfo</i></code>

```
pyg4ometry.fluka.boolean_algebra.expressionToZone(zone, zoneExpr)
pyg4ometry.fluka.boolean_algebra.zoneToDNFZones(zone)
pyg4ometry.fluka.boolean_algebra.regionToAlgebraicExpression(region)
pyg4ometry.fluka.boolean_algebra.zoneToAlgebraicExpression(zone)
pyg4ometry.fluka.boolean_algebra._getMeshAndAABB(body, aabb=None)
pyg4ometry.fluka.boolean_algebra.pruneRegion(reg, aabb=None)
pyg4ometry.fluka.boolean_algebra.isZoneContradiction(zone)
pyg4ometry.fluka.boolean_algebra.pruneZone(zone, aabb0=None, aabb=None)
```



```

pyg4ometry.fluka.boolean_algebra.squashDegenerateBodies(zone, bodystore=None)
pyg4ometry.fluka.boolean_algebra._MeshedZoneInfo
pyg4ometry.fluka.boolean_algebra.simplifyRegion(region)
pyg4ometry.fluka.boolean_algebra._filterRedunantZonesSymbollicaly(zoneData)
pyg4ometry.fluka.boolean_algebra._filterRedundantZonesMetricCheck(zoneData)
pyg4ometry.fluka.boolean_algebra.nTermsDNF(expr)
pyg4ometry.fluka.boolean_algebra._nTermsDNFZone(zone)

```

pyg4ometry.fluka.card

Module Contents

Classes

<i>Card</i>	Card class for representing a FLUKA input card. To construct
-------------	--

Functions

<i>_attempt_float_coercion</i> (string)	
<i>freeFormatStringSplit</i> (string)	Method to split a string in FLUKA FREE format into its components.
<i>main</i> (filein)	

```

class pyg4ometry.fluka.card.Card(keyword, what1=None, what2=None, what3=None, what4=None,
                                   what5=None, what6=None, sdum=None)

```

Card class for representing a FLUKA input card. To construct instances from a of FLUKA input, use the fromFree or fromFixed class method for FREE and FIXED format, respectively.

```
__repr__()
```

Return repr(self).

```
toList()
```

```
toFreeString(delim=', ')
```

```
toFixedString()
```

```
nonesToZero()
```

Return a class instance with same contents as this instance, but with all entries of None set to 0.0 instead.

```
classmethod fromFree(line)
```

```
classmethod fromFixed(line)
```

```
pyg4ometry.fluka.card._attempt_float_coercion(string)
```

```
pyg4ometry.fluka.card.freeFormatStringSplit(string)
```

Method to split a string in FLUKA FREE format into its components.

```
pyg4ometry.fluka.card.main(filein)
```

```
pyg4ometry.fluka.directive
```

Module Contents

Classes

<i>MatrixConvertibleMixin</i>	
<i>Transform</i>	expansion, translation, rotoTranslation can be either a single
<i>RotoTranslation</i>	translation in mm, angles in degrees
<i>RecursiveRotoTranslation</i>	container for dealing with a recursively defined

Functions

<i>rotoTranslationFromTBxyz</i> (name, tbxyz[, flukaregistry])	tbxyz = trait bryan angles in radians
<i>rotoTranslationFromTra2</i> (name, tra2[, flukaregistry])	
<i>_translationTo4DMatrix</i> (translation)	
<i>_expansionFactorTo4DMatrix</i> (factor)	
<i>_rightMultiplyMatrices</i> (matrices)	

```
class pyg4ometry.fluka.directive.MatrixConvertibleMixin
```

```
    toScaleMatrix()
```

```
    toTranslationMatrix()
```

```
    toRotationMatrix()
```

```
    netTranslation()
```

```
    netExpansion()
```

```
    leftMultiplyVector(vector)
```

```
    leftMultiplyRotation(matrix)
```

hash()

```
class pyg4ometry.fluka.directive.Transform(*, expansion=None, translation=None,
                                           rotoTranslation=None, invertRotoTranslation=None)
```

Bases: [MatrixConvertibleMixin](#)

expansion, translation, rotoTranslation can be either a single instance of RotoTranslation or a multiple instances of RotoTranslation and RecursiveRotoTranslation

_expansionsTo4DMatrices()

_translationsTo4DMatrices()

_rotoTranslationsTo4DMatrices()

to4DMatrix()

```
class pyg4ometry.fluka.directive.RotoTranslation(name, axis=None, polar=0.0, azimuth=0.0,
                                                  translation=None, transformationIndex=None,
                                                  flukaregistry=None)
```

Bases: [MatrixConvertibleMixin](#)

translation in mm, angles in degrees

__repr__()

Return repr(self).

to4DMatrix()

toCard()

flukaFreeString()

classmethod fromCard(card)

hasTranslation()

hasRotation()

isPureTranslation()

```
class pyg4ometry.fluka.directive.RecursiveRotoTranslation(name, rotoTranslations)
```

Bases: [collections.abc.MutableSequence](#), [MatrixConvertibleMixin](#)

container for dealing with a recursively defined rototranslation. they must also refer to the same rototrans, i.e., have the same name. for a list of rototranslations supplied:

[a, b, c], the order of evaluation acting on a vector v is $c*b*a*v$. so teh first rototrans is applied first.. and so on.

property transformationIndex

__repr__()

Return repr(self).

__getitem__(i)

_raiseIfDifferentName(name)

__setitem__(i, obj)

`__delitem__(i)``__len__()``insert(i, obj)`

S.insert(index, value) – insert value before index

`to4DMatrix()``flukaFreeString()``_transformationIndices()``areAllTheSameTransformationIndices()``transformationIndex()``pyg4ometry.fluka.directive.rotoTranslationFromTBxyz(name, tbxyz, flukaregistry=None)`

tbxyz = trait bryan angles in radians

`pyg4ometry.fluka.directive.rotoTranslationFromTra2(name, tra2, flukaregistry=None)``pyg4ometry.fluka.directive._translationTo4DMatrix(translation)``pyg4ometry.fluka.directive._expansionFactorTo4DMatrix(factor)``pyg4ometry.fluka.directive._rightMultiplyMatrices(matrices)``pyg4ometry.fluka.elcfield`

Module Contents

Classes

`elcfield`

```
class pyg4ometry.fluka.elcfield.elcfield(maxStepAngle=57, minBoundaryAccuracy=0.05, minStep=0.1,
                                          ex=0, ey=9, ez=0, dpdxFactor=None)
```

`pyg4ometry.fluka.extruder`

Module Contents

Classes

`Extruder`

Base class for all solids

```
class pyg4ometry.fluka.extruder.Extruder(name="", length=1000, regions={}, registry=None)
    Bases: pyg4ometry.geant4.solid.SolidBase
    Base class for all solids
    addRegion(name)
    addPointToRegion(name, pntIndx)
    setRegionToOuterBoundary(name)
    buildCgalPolygons()
    buildGeant4Extrusions()
    plot(decompositions=False)
    mesh()
```

```
pyg4ometry.fluka.flair
```

Module Contents

Classes

Flair

Functions

_getFLUKATemplateFileName()

Attributes

_flukaTemplateFileName

```
pyg4ometry.fluka.flair._flukaTemplateFileName
```

```
pyg4ometry.fluka.flair._getFLUKATemplateFileName()
```

```
class pyg4ometry.fluka.flair.Flair(flukaInputFileName='fluka.inp', extent=None)
```

```
    write(fileName)
```

```
    addMaterialColour(materialName, color=(0, 0, 0, 0))
```

pyg4ometry.fluka.fluka_registry

Module Contents

Classes

<i>FlukaRegistry</i>	Object to store geometry for FLUKA input and output. All of the FLUKA classes can be used without storing them in the Registry. The registry is used to write the FLUKA output file.
<i>RotoTranslationStore</i>	only get by names.
<i>FlukaBodyStore</i>	A MutableMapping is a generic container for associating
<i>BaseCacher</i>	
<i>Cacheable</i>	
<i>HalfSpaceCacher</i>	
<i>InfiniteCylinderCacher</i>	
<i>FlukaBodyStoreExact</i>	

Attributes

<i>logger</i>

pyg4ometry.fluka.fluka_registry.**logger**

class pyg4ometry.fluka.fluka_registry.**FlukaRegistry**

Object to store geometry for FLUKA input and output. All of the FLUKA classes can be used without storing them in the Registry. The registry is used to write the FLUKA output file.

addBody(*body*)

makeBody(*clas*, **args*, ***kwargs*)

getDegenerateBody(*body*)

addRotoTranslation(*rototrans*)

addRegion(*region*, *addBodies=False*)

makeRegionsDNF()

addLattice(*lattice*)

getBody(*name*)

getBodyToRegionsMap()

```

printDefinitions()

regionAABBs(write=None)

latticeAABBs()

addMaterial(material, recursive=False)

getMaterial(name)

addMaterialAssignments(mat, *regions, elc=None, mgn=None)

assignma(material, *regions)

addCard(card)

addTitle(title='FLUKA simulation')

addDefaults(default='EM-CASCA')

addGlobal()

addBeam(energy, particle='ELECTRON')

addBeamPos(xpos=0, ypos=0, zpos=0, xdc=0, ydc=0)

addLowMat(flukaMat, lowENeutron1, lowENeutron2, lowENeutron3)

addLowMatAllMaterials()
    Add LOWMAT card to all materials

addLowPwxs(what1=None, lowerMaterial=None, upperMaterial=None)

addMgnCreat()

addMgnField()

addElcField()

addUsrBin(mesh=0, particle='ENERGY', lunOutput=-21, e1max=1, e2max=1, e3max=1, name='name',
            e1min=-1, e2min=-1, e3min=-1, e1nbin=10, e2nbin=10, e3nbin=10)

addRotprBin(precision=0, rotDefi=0, printEventBin=0, lowerBin=None, upperBin=None)

addUsrBdx(binning, scoringDir, scoringType, type, reg1, reg2, name, area=1.0, lunOutput=-22,
            maxKE=None, minKE=None, nKEbin=None, maxSA=None, minSA=None, nSAbin=None)

addUsrCall()

addUsrOcall()

addUsrDump(mgdraw=100, lun=70, mgdrawOpt=-1, what4=0, sdum=None)

addRandomiz(seedLun=1, seed=54217137)

addStart(maxPrimHistories=1, timeTermSec=None, coreDump=None, eachHistoryOutput=None)

addPhotonuc(what1, mat_low, mat_high, mat_step, sdum='')

addMuphoton(what1, mat_low, mat_high, mat_step)

```

addPairbrem(*what1, what2, what3, mat_low, mat_high, mat_step*)

addDeltaRay(*what1, what2, what3, mat_low, mat_high, mat_step, dsum='NOPRINT'*)

addIonFluct(*what1, what2, what3, mat_low, mat_high, mat_step, dsum='PRIM-ION'*)

printDumps(*detail=1*)

findLastBodyIndex()

Find last body index (if the numbering was performed by a geant4 -> fluka conversion)

findLastRegionIndex()

Find last region index (if the numbering was performed by a geant4 -> fluka conversion)

findLastMaterialIndex()

Find last material index (if the numbering was performed by a geant4 -> fluka conversion)

findLastTransformationIndex()

checkBodyName(*bodyName*)

checkRegionName(*regionName*)

checkMaterialName(*materialName*)

addRegistry(*flukaRegistry, outerRegion=None, rotation=[0, 0, 0], translation=[0, 0, 0],
removeRegions=[], removeRegionDependents=False*)

class pyg4ometry.fluka.fluka_registry.**RotoTranslationStore**

Bases: `collections.abc.MutableMapping`

only get by names.

__getitem__(*name*)

__setitem__(*name, rtrans*)

addRotoTranslation(*rtrans*)

allTransformationIndices()

__delitem__(*key*)

__iter__()

__len__()

flukaFreeString()

class pyg4ometry.fluka.fluka_registry.**FlukaBodyStore**

Bases: `collections.abc.MutableMapping`

A MutableMapping is a generic container for associating key/value pairs.

This class provides concrete generic implementations of all methods except for `__getitem__`, `__setitem__`, `__delitem__`, `__iter__`, and `__len__`.

_bodyNames()

_bodies()


```

    _getCacherFromBody(body)
    make(clas, *args, **kwargs)
    getDegenerateBody(body)
    addBody(body)
    __setitem__(key, value)
    __getitem__(key)
    __delitem__(key)
    __len__()
    __contains__(key)
    __iter__()
    __repr__()
        Return repr(self).
class pyg4ometry.fluka.fluka_registry.BaseCacher(df)
    COLUMNS = ['name', 'body']
    appendData(variables)
    append(body)
    setBody(body)
    addBody(body)
    remove(key)
    make(clas, *args, **kwargs)
    getDegenerateBody(body)
    __repr__()
        Return repr(self).
class pyg4ometry.fluka.fluka_registry.Cacheable(df)
    Bases: BaseCacher
    getDegenerateBody(body)
    getMask(columns, values, predicates)
class pyg4ometry.fluka.fluka_registry.HalfSpaceCacher(df)
    Bases: Cacheable
    COLUMNS = ['name', 'body', 'planeNormal', 'pointOnPlane']
    append(body)
    mask(body)

```

```
class pyg4ometry.fluka.fluka_registry.InfiniteCylinderCacher(df)
    Bases: Cacheable
    COLUMNS = ['name', 'body', 'direction', 'pointOnLine', 'radius']
    append(body)
    mask(body)
    static _cylinderPoint(body)
class pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact
    _bodyNames()
    _bodies()
    make(cls, *args, **kwargs)
    getDegenerateBody(body)
    addBody(body)
    keys()
    values()
    __setitem__(key, value)
    __getitem__(key)
    __delitem__(key)
    __len__()
    __contains__(key)
    __iter__()
    __repr__()
        Return repr(self).
```

pyg4ometry.fluka.lattice

The Lattice class is defined in this module for handling LATTICES in FLUKA geometries.

Module Contents

Classes

Lattice

```
class pyg4ometry.fluka.lattice.Lattice(cellRegion, rotoTranslation, invertRotoTranslation=False,
                                       flukaregistry=None)
```

```
    flukaFreeString(delim=', ')
```

```
    getTransform()
```

```
pyg4ometry.fluka.material
```

Module Contents

Classes

<i>BuiltIn</i>	
<i>multiGroupNeutronCrossSections</i>	
<i>_MatProp</i>	
<i>Material</i>	A FLUKA material consisting of a single element. This corresponds
<i>Compound</i>	A FLUKA compound material. This corresponds to the case in

Functions

<i>predefinedMaterialNames()</i>	
<i>defineBuiltInFlukaMaterials([flukaregistry])</i>	
<i>_appendFractionPairs(card, fractions, fractionTypes)</i>	
<i>_parseFraction(what1, what2)</i>	Returns the (name, fraction, fractionType). This is for handling
<i>_formatFlukaMaterialPair(pair, namePrefix, fractionPrefix)</i>	Names and fractions maybe stored as "negative" numbers in the
<i>_grouper(n, iterable[, fillvalue])</i>	<code>grouper(3, 'ABCDEFGF', 'x') --> ABC DEF Gxx</code>

Attributes

<i>_PREDEFINED_ELEMENTS</i>
<i>_PREDEFINED_COMPOUNDS</i>

```
pyg4ometry.fluka.material._PREDEFINED_ELEMENTS = [('BLCKHOLE', 0, 0, 0), ('VACUUM', 0, 0, 0), ('HYDROGEN', 1.00794, 1.0, 8.37e-05), ('HELIUM', ...
```

```
pyg4ometry.fluka.material._PREDEFINED_COMPOUNDS = [('WATER', 1.0), ('POLYSTYR', 1.06), ('PLASCINT', 1.032), ('PMMA', 1.19), ('BONECOMP', 1.85), ...
```

```
pyg4ometry.fluka.material.predefinedMaterialNames()
```

```
class pyg4ometry.fluka.material.BuiltIn(name, *, atomicNumber=None, atomicMass=None, density=None, flukaregistry=None)
```

```
    __repr__()
```

```
        Return repr(self).
```

```
    flukaFreeString(delim="")
```

```
pyg4ometry.fluka.material.defineBuiltInFlukaMaterials(flukaregistry=None)
```

```
class pyg4ometry.fluka.material.multiGroupNeutronCrossSections(fileName=None)
```

```
    _readFile()
```

```
    findMaterial(Z, A, T, selfShield=False)
```

```
class pyg4ometry.fluka.material._MatProp
```

```
    isGas()
```

```
    makeMatPropCard()
```

```
class pyg4ometry.fluka.material.Material(name, atomicNumber, density, massNumber=None, atomicMass=None, pressure=None, flukaregistry=None, comment="")
```

Bases: [`_MatProp`](#)

A FLUKA material consisting of a single element. This corresponds to the case in FLUKA of a single MATERIAL card with no associated COMPOUND cards, as well as a possible MAT-PROP card (only if a pressure is provided, other options of MAT-PROP are unsupported).

Parameters

- **name** (*str*) – The name of the material
- **atomicNumber** (*int*) – the atomic number, Z, of the element.
- **density** (*float*) – the density in g/cm3 of the material.
- **massNumber** (*int*, *None*) – Optional mass number, will be inferred in FLUKA based on atomicNumber. Allows one to specify a specific isotope.
- **atomicMass** (*float*) – The mass of the atom in g/mole. Will be inferred in FLUKA based on atomicNumber.
- **pressure** (*float*) – Optional pressure if the material is a gas.
- **flukaregistry** ([`FlukaRegistry`](#)) – Optional FlukaRegistry instance the material is to be added to.

```
toCards()
```

```
flukaFreeString(delim=',')
```

__repr__()

Return repr(self).

classmethod fromCard(card, flukaregistry)

rename(newName, recursive=False, iIndex=0)

class pyg4ometry.fluka.material.**Compound**(name, density, fractions, fractionType, pressure=None, flukaregistry=None, comment="")

Bases: [_MatProp](#)

A FLUKA compound material. This corresponds to the case in FLUKA of a single MATERIAL card with one or more associated COMPOUND cards.

Parameters

- **name** (*str*) – The name of the compound.
- **density** (*float*) – The density of the compound in g/cm3
- **fractions** (*list*) – List of (Element, fraction) and (Compound, fraction) tuples corresponding to the fractional proportion of that material.
- **fractionType** (*str*) – The type of the fractions listed in the fractions parameter, either atomic, mass, or volume.
- **flukaregistry** ([FlukaRegistry](#)) – Optional FlukaRegistry instance the Compound is to be added to.

toCards()

flukaFreeString(delim=', ')

classmethod fromCards(cards, flukareg)

__repr__()

Return repr(self).

totalWeighting(densityWeighted=False)

rename(newName, recursive=True, iIndex=0)

pyg4ometry.fluka.material.**_appendFractionPairs**(card, fractions, fractionTypes)

pyg4ometry.fluka.material.**_parseFraction**(what1, what2)

Returns the (name, fraction, fractionType). This is for handling the different permutations found in the FLUKA manual COMPOUND entry.

pyg4ometry.fluka.material.**_formatFlukaMaterialPair**(pair, namePrefix, fractionPrefix)

Names and fractions maybe stored as “negative” numbers in the FLUKA input, permutations between which mean different types of fractions.

pyg4ometry.fluka.material.**_grouper**(n, iterable, fillvalue=None)

`grouper(3, 'ABCDEFG', 'x') -> ABC DEF Gxx` <https://docs.python.org/3/library/itertools.html#recipes>

pyg4ometry.fluka.mgnfield

Module Contents

Classes

<i>mgnfield</i>
<i>mgnfield_mgncreat</i>
<i>mgncreat</i>
<i>mgndata_array</i>

```
class pyg4ometry.fluka.mgnfield.mgnfield(maxStepAngle=57, minBoundaryAccuracy=0.05, minStep=0.1,
                                           bx=0, by=9, bz=0)
```

```
class pyg4ometry.fluka.mgnfield.mgnfield_mgncreat(fieldStrength=1, rotDefi=None, regionLattice=0,
                                                    lowerRegion=None, upperRegion=None,
                                                    stepRegion=1, fieldName='field')
```

```
class pyg4ometry.fluka.mgnfield.mgncreat(analyticalField=0, interpolatedField=0,
                                           interpolationRadius=0, xoffset=0, yoffset=0, xsym=0, ysym=0,
                                           zsym=0, fieldName=None)
```

```
class pyg4ometry.fluka.mgnfield.mgndata_array(mgndata, fieldName)
```

pyg4ometry.fluka.preprocessor

Module Contents

Classes

<i>_IfInfo</i>	Tells us about current conditional and its state at the current line.
<i>_Calc</i>	A node visitor base class that walks the abstract syntax tree and calls a

Functions

```
_degree_input(f)
```

```
_degree_output(f)
```

```
preprocess(filein)
```

```
_parse_preprocessor_define(directive, split_line,
defines)
```

```
_parse_preprocessor_conditional(directive,
split_line, ...)
```

```
_parse_preprocessor_include(directory, directive,
...)
```

Attributes

```
_FLUKA_PREDEFINED_IDS
```

```
_FLUKA_PREDEFINED_FUNCTIONS
```

```
pyg4ometry.fluka.preprocessor._degree_input(f)
```

```
pyg4ometry.fluka.preprocessor._degree_output(f)
```

```
pyg4ometry.fluka.preprocessor._FLUKA_PREDEFINED_IDS
```

```
pyg4ometry.fluka.preprocessor._FLUKA_PREDEFINED_FUNCTIONS
```

```
pyg4ometry.fluka.preprocessor.preprocess(filein)
```

```
class pyg4ometry.fluka.preprocessor._IfInfo(any_branch_satisfied, read_until_next)
```

Tells us about current conditional and its state at the current line.

- If the conditional at this level has been satisfied (e.g. if the IF has been satisfied then we don't read any subsequent ELIF or ELSE clauses, we skip until the next ENDIF).
- If we should read until the next conditional directive (e.g. if the IF or ELIF has been satisfied then we should read all lines following this directive until we reach the next conditional directive).

```
__repr__()
```

Return repr(self).

```
pyg4ometry.fluka.preprocessor._parse_preprocessor_define(directive, split_line, defines)
```

```
pyg4ometry.fluka.preprocessor._parse_preprocessor_conditional(directive, split_line, defines,
if_stack)
```

```
pyg4ometry.fluka.preprocessor._parse_preprocessor_include(directory, directive, split_line,
line_stack)
```

class pyg4ometry.fluka.preprocessor._Calc(*definitions*)

Bases: `ast.NodeVisitor`

A node visitor base class that walks the abstract syntax tree and calls a visitor function for every node found. This function may return a value which is forwarded by the *visit* method.

This class is meant to be subclassed, with the subclass adding visitor methods.

Per default the visitor functions for the nodes are 'visit_' + class name of the node. So a *TryFinally* node visit function would be *visit_TryFinally*. This behavior can be changed by overriding the *visit* method. If no visitor function exists for a node (return value *None*) the *generic_visit* visitor is used instead.

Don't use the *NodeVisitor* if you want to apply changes to nodes during traversing. For this a special visitor exists (*NodeTransformer*) that allows modifications.

op_map

evaluate(*expression*)

visit_BinOp(*node*)

visit_UnaryOp(*node*)

visit_Constant(*node*)

visit_Name(*node*)

visit_Call(*node*)

visit_Expr(*node*)

pyg4ometry.fluka.reader

Module Contents

Classes

<i>Reader</i>	Class to read a FLUKA file.
<i>RegionVisitor</i>	A visitor class for accumulating the region definitions. The body
<i>SensitiveErrorListener</i>	ANTLR4 by default is very passive regarding parsing errors, it will

Functions

<i>_getConstituentMaterialNamesFromCompound</i> (con
<i>_make_body</i> (body_parts, expansion_stack, ...)
<i>main</i> (filein)

Attributes

`_BODY_NAMES`

`pyg4ometry.fluka.reader._BODY_NAMES`

class `pyg4ometry.fluka.reader.Reader(filename)`

Class to read a FLUKA file.

getRegistry()

Get the fluka registry

_load()

Load the FLUKA input file

_findLines()

_parseBodies()

_parseRegions()

_parseCards()

_parseRotDefinis()

_parseGeometryDirective(*line_parts, expansion_stack, translation_stack, transform_stack*)

_parseMaterialAssignments()

_parseLattice()

_parseMaterials()

_splitMaterialCards()

Return a tuple as (Elements, Compounds, Matprop (not yet implemented)). Each is keyed with the name of the material, and the value corresponds to the card (Element) or cards (Compounds) that define the material.

`pyg4ometry.fluka.reader._getConstituentMaterialNamesFromCompound(compoundCards)`

`pyg4ometry.fluka.reader._make_body(body_parts, expansion_stack, translation_stack, transform_stack, flukareg)`

class `pyg4ometry.fluka.reader.RegionVisitor(flukaregistry)`

Bases: `pyg4ometry.fluka.RegionExpression.RegionParserVisitor`

A visitor class for accumulating the region definitions. The body instances are provided at instantiation, and then these are used when traversing the tree to build up a dictionary of region name and `pyfluka.geometry.Region` instances.

visitSimpleRegion(*ctx*)

visitComplexRegion(*ctx*)

visitUnaryAndBoolean(*ctx*)

visitUnaryExpression(*ctx*)

visitUnaryAndSubZone(*ctx*)

visitSingleUnion(*ctx*)

visitMultipleUnion(*ctx*)

visitMultipleUnion2(*ctx*)

visitSubZone(*ctx*)

visitZoneExpr(*ctx*)

visitZoneSubZone(*ctx*)

visitZoneBody(*ctx*)

class pyg4ometry.fluka.reader.**SensitiveErrorListener**

Bases: `antlr4.error.ErrorListener.ErrorListener`

ANTLR4 by default is very passive regarding parsing errors, it will just carry on parsing and potentially build a nonsense-tree. This is not ideal as pyfluka has a very convoluted syntax; we want to be very strict about what our parser can and can't do. For that reason this is a very sensitive error listener, throwing exceptions readily.

syntaxError(*recognizer, offendingSymbol, line, column, msg, e*)

pyg4ometry.fluka.reader.**main**(*filein*)

pyg4ometry.fluka.region

Module Contents

Classes

<i>_Boolean</i>	
<i>Subtraction</i>	
<i>Intersection</i>	
<i>Union</i>	
<i>Zone</i>	Represents a zone which consists of one or more body intersections
<i>Region</i>	Represents a region which consists of a region name, one or more

Functions

<code>bracket_depth(zone)</code>	
<code>bracket_number(zone)</code>	
<code>zone_to_sympy(zone)</code>	
<code>region_to_sympy(region)</code>	
<code>sympy_to_zone(sympy_expr, freg)</code>	
<code>sympy_to_region(sympy_expr, freg[, regionName])</code>	Convert sympy boolean to fluka region
<code>simplify_region(region)</code>	
<code>_generate_name(typename, index, name, isZone, rootname)</code>	Try to generate a sensible name for intermediate
<code>_get_relative_rot_matrix(first, second)</code>	
<code>_get_relative_translation(first, second, aabb)</code>	
<code>_get_relative_rotation(first, second)</code>	
<code>_getRelativeTransform(first, second, aabb)</code>	
<code>_randomName()</code>	Returns a random name that is syntactically correct for use in GDML.
<code>_makeWorldLogicalVolume(reg)</code>	
<code>_getAxisAlignedBoundingBox(aabb, boolean)</code>	aabb should really be a dictionary of

Attributes

<code>logger</code>

`pyg4ometry.fluka.region.logger`

`pyg4ometry.fluka.region.bracket_depth(zone)`

`pyg4ometry.fluka.region.bracket_number(zone)`

`pyg4ometry.fluka.region.zone_to_sympy(zone)`

`pyg4ometry.fluka.region.region_to_sympy(region)`

`pyg4ometry.fluka.region.sympy_to_zone(sympy_expr, freg)`

`pyg4ometry.fluka.region.sympy_to_region(sympy_expr, freg, regionName='name')`

Convert sympy boolean to fluka region Must be the OR of multiple convex zones

`pyg4ometry.fluka.region.simplify_region(region)`

`pyg4ometry.fluka.region._generate_name(typename, index, name, isZone, rootname)`

Try to generate a sensible name for intermediate Geant4 booleans which have no FLUKA analogue.

class `pyg4ometry.fluka.region._Boolean`

`generate_name(index, rootname=None)`

class `pyg4ometry.fluka.region.Subtraction(body)`

Bases: `_Boolean`

class `pyg4ometry.fluka.region.Intersection(body)`

Bases: `_Boolean`

class `pyg4ometry.fluka.region.Union(body)`

Bases: `_Boolean`

class `pyg4ometry.fluka.region.Zone(name=None)`

Bases: `pyg4ometry.fluka.vis.ViewableMixin`

Represents a zone which consists of one or more body intersections and zero or more body subtractions. May also be used to represent subzones, which are zones nested within zones, for example the form +A -B -(+C -D). Once instantiated, intersections and subtractions can be added to this instance with the `addIntersection` and `addSubtraction` method.

Parameters

name (*string*) – Optional name for the zone.

addSubtraction(*body*)

Add a body or subzone as a subtraction to this Zone instance.

Parameters

body (*Body* or *Zone*) – Body or Zone instance.

addIntersection(*body*)

Add a body or subzone as an intersection to this Zone instance.

Parameters

body – Body or Zone instance.

Type

body: *Body* or *Zone*

convertToDNF(*fluka_registry*)

centre(*aabb=None*)

rotation()

tbxyz()

static **_getSolidFromBoolean**(*boolean, g4reg, aabb*)

mesh(*aabb=None*)

geant4Solid(*reg, aabb=None*)

Translate this zone to a geant4solid, adding the constituent primitive solids and any Booleans to the Geant4 registry. Returns the geant4 solid equivalent in geometry to this Zone.

Parameters

- **reg** (*AABB* or *dict*) – The Registry (geant4) instance to which the resulting Geant4 definitions should be added.
- **aabb** – Optional reference AABB or dictionary of body name to AABB instances with which the geant4 solid should be evaluated with respect to. This is the entry point to which solid minimisation can be performed.

_combine_booleans(*body0, start, collection, operation, reg, aabb*)

_geant4MultiUnionSubtraction(*body0, start, reg, aabb*)

dumps()

Returns a string of this Zone instance in the equivalent FLUKA syntax.

dumpsDebug()

Returns a string of this Zone instance in the equivalent FLUKA syntax with extra debug information

withLengthSafety(*bigger_flukareg, smaller_flukareg, shrink_intersections*)

allBodiesToRegistry(*flukaregistry*)

Add all the bodies that constitute this Zone to the provided FlukaRegistry instance.

Parameters

flukaregistry (*FlukaRegistry*) – FlukaRegistry instance to which constituent bodies will be added.

bodies()

Return the set of unique bodies that constitute this Zone.

removeBody(*name*)

Remove a body from this zone by name.

Parameters

name (*string*) – The name of the body to be removed.

makeUnique(*nameSuffix, flukaregistry*)

Get this zone with every constituent body recreated with a unique name by appending nameSuffix.

Parameters

- **nameSuffix** – The string to append to the names of the bodies.
- **flukaregistry** – the FlukaRegistry instance to add the uniquely defined bodies to.

isNull(*aabb=None*)

toDNF(*name*)

isDNF()

leafCount()

class `pyg4ometry.fluka.region.Region`(*name, comment=""*)

Bases: `pyg4ometry.fluka.vis.ViewableMixin`

Represents a region which consists of a region name, one or more zones, and a single material. Metadata may be provided with the comment kwarg, which is used when writing to FLUKA to provide contextual information to the physicist.

Parameters

- **name** (*str*) – The name of this region.

- **material** (*str*) – The name of a material.
- **comment** – Optional descriptive comment.

addZone(*zone*)

Add a Zone instance to this region.

Parameters

zone (*Zone*) – The Zone instance to be added.

addIntersection(*zone*)

addSubtraction(*zone*)

extend(*region*)

removeNullZones()

convertToDNF(*fluka_registry*)

centre(*aabb=None*)

tbxyz()

rotation()

bodies()

Return the set of unique bodies that constitute this Zone.

mesh(*aabb=None*)

geant4Solid(*reg, aabb=None*)

Get the geant4Solid instance corresponding to this Region.

dumps()

flukaFreeString()

withLengthSafety(*bigger_flukareg, smaller_flukareg*)

allBodiesToRegistry(*registry*)

Add all the bodies that constitute this Region to the provided FlukaRegistry instance.

Parameters

flukaregistry (*FlukaRegistry*) – FlukaRegistry instance to which constituent bodies will be added.

zoneGraph(*zoneAABBs=None, aabb=None*)

_zoneGraphPycgal()

connectedZones(*zoneAABBs=None, aabb=None*)

zoneAABBs(*aabb=None*)

aabb(*aabb=None*)

removeBody(*name*)

Remove a body from this region by name.

Parameters

name (*string*) – The name of the body to be removed.

makeUnique(*nameSuffix*, *flukaRegistry*)

Get this Region instance with every constituent body with a unique name by appending *nameSuffix* to each Body instance.

Parameters

- **nameSuffix** – string to append to each Body instance.
- **flukaRegistry** – FlukaRegistry instance to add each newly-defined body to.

isNull(*aabb=None*)

toDNF(*name*)

isDNF()

filterNullZones(*aabb=None*)

leafCount()

simplify()

__repr__()

removeZones(*indices*)

Remove zones by index

`pyg4ometry.fluka.region._get_relative_rot_matrix(first, second)`

`pyg4ometry.fluka.region._get_relative_translation(first, second, aabb)`

`pyg4ometry.fluka.region._get_relative_rotation(first, second)`

`pyg4ometry.fluka.region._getRelativeTransform(first, second, aabb)`

`pyg4ometry.fluka.region._randomName()`

Returns a random name that is syntactically correct for use in GDML.

`pyg4ometry.fluka.region._makeWorldLogicalVolume(reg)`

`pyg4ometry.fluka.region._getAxisAlignedBoundingBox(aabb, boolean)`

aabb should really be a dictionary of {bodyName: extentInstance}.

`pyg4ometry.fluka.vector`

Module Contents

Classes

Three

AABB

Functions

<code>areAABBsOverlapping</code> (first, second)	Check if two AABB instances are overlapping.
<code>pointOnLineClosestToPoint</code> (point, point_on_line, direction)	Line is defined in terms of two vectors: a point on the line and
<code>pointOnPlaneClosestToPoint</code> (planeNormal, plane-Point, point)	Get point on plane which is closest to point not on the plane.
<code>areParallelOrAntiParallel</code> (v1, v2)	

```
class pyg4ometry.fluka.vector.Three(shape, dtype=float, buffer=None, offset=0, strides=None,
                                     order=None)
```

Bases: `numpy.ndarray`

property `x`

property `y`

property `z`

parallel_to(other, tolerance=1e-10)

Check if instance is parallel to some other vector, v

unit()

Get this as a unit vector.

length()

vector length (l2 norm)

dot(other)

cross(other)

__eq__(other)

Return self==value.

__ne__(other)

Return self!=value.

__add__(other)

__radd__(other)

__sub__(other)

__rsub__(other)

__iadd__(other)

__isub__(other)

```
class pyg4ometry.fluka.vector.AABB(lower, upper)
```

__repr__()

Return repr(self).

`__eq__(other)`

Return self==value.

`cornerDistance()`

`classmethod fromMesh(csgmesh)`

`intersects(other)`

`coplanarIntersects(other)`

`envelops(other)`

`intersect(other)`

`union(other)`

`isNull()`

`pyg4ometry.fluka.vector.areAABBsOverlapping(first, second)`

Check if two AABB instances are overlapping.

`pyg4ometry.fluka.vector.pointOnLineClosestToPoint(point, point_on_line, direction)`

Line is defined in terms of two vectors: a point on the line and the direction of the line.

point is the point with which the distance to the line is to be minimised.

`pyg4ometry.fluka.vector.pointOnPlaneClosestToPoint(planeNormal, planePoint, point)`

Get point on plane which is closest to point not on the plane.

`pyg4ometry.fluka.vector.areParallelOrAntiParallel(v1, v2)`

`pyg4ometry.fluka.vis`

Module Contents

Classes

ViewableMixin

`class pyg4ometry.fluka.vis.ViewableMixin`

`view(aabb=None)`

Package Contents

Classes

<i>RPP</i>	Rectangular Parallelepiped
<i>BOX</i>	General Rectangular Parallelepiped
<i>SPH</i>	Sphere
<i>RCC</i>	Right Circular Cylinder
<i>REC</i>	Right Elliptical Cylinder
<i>TRC</i>	Truncated Right-angled Cone
<i>ELL</i>	Ellipsoid of Revolution
<i>WED</i>	Right Angle Wedge
<i>RAW</i>	Base class representing a body as defined in FLUKA
<i>ARB</i>	Arbitrary Convex Polyhedron
<i>XYP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the z-axis)
<i>XZP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the y-axis)
<i>YZP</i>	Infinite half-space delimited by the x-y plane (perpendicular to the x-axis)
<i>PLA</i>	Infinite half-space delimited by the x-y plane (perpendicular to the z-axis) Generic infinite half-space.
<i>XCC</i>	Infinite Circular Cylinder parallel to the x-axis
<i>YCC</i>	Infinite Circular Cylinder parallel to the y-axis
<i>ZCC</i>	Infinite Circular Cylinder parallel to the z-axis
<i>XEC</i>	Infinite Elliptical Cylinder parallel to the x-axis
<i>YEC</i>	Infinite Elliptical Cylinder parallel to the y-axis
<i>ZEC</i>	Infinite Elliptical Cylinder parallel to the z-axis
<i>QUA</i>	Generic quadric
<i>Reader</i>	Class to read a FLUKA file.
<i>Writer</i>	
<i>FlukaRegistry</i>	Object to store geometry for FLUKA input and output. All of the FLUKA classes can be used without storing them in the Registry. The registry is used to write the FLUKA output file.
<i>FlukaBodyStoreExact</i>	
<i>Three</i>	
<i>AABB</i>	
<i>Zone</i>	Represents a zone which consists of one or more body intersections
<i>Region</i>	Represents a region which consists of a region name, one or more
<i>Transform</i>	expansion, translation, rotoTranslation can be either a single
<i>RotoTranslation</i>	translation in mm, angles in degrees
<i>RecursiveRotoTranslation</i>	container for dealing with a recursively defined

continues on next page

Table 2 – continued from previous page

<i>Lattice</i>	
<i>Flair</i>	
<i>BuiltIn</i>	
<i>Material</i>	A FLUKA material consisting of a single element. This corresponds
<i>Compound</i>	A FLUKA compound material. This corresponds to the case in
<i>Card</i>	Card class for representing a FLUKA input card. To construct
<i>Extruder</i>	Base class for all solids

Functions

<i>infinity</i> (inf)	Use this to temporarily modify INFINITY, with it resetting back
<i>bracket_depth</i> (zone)	
<i>bracket_number</i> (zone)	
<i>zone_to_sympy</i> (zone)	
<i>region_to_sympy</i> (region)	
<i>sympy_to_region</i> (sympy_expr, freg[, regionName])	Convert sympy boolean to fluka region

```
class pyg4ometry.fluka.RPP(name, xmin, xmax, ymin, ymax, zmin, zmax, transform=None,
                           flukaregistry=None, addRegistry=True, comment="")
```

Bases: BodyMixin

Rectangular Parallelepiped

Parameters

- **name** (*str*) – of body
- **xmin** (*float*) – lower x coordinate of RPP
- **xmax** (*float*) – upper x coordinate of RPP
- **ymin** (*float*) – lower y coordinate of RPP
- **ymax** (*float*) – upper y coordinate of RPP
- **zmin** (*float*) – lower z coordinate of RPP
- **zmax** (*float*) – upper z coordinate of RPP

centre(*aabb=None*)

rotation()

lengths()

geant4Solid(*reg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.fluka.**BOX**(*name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin

General Rectangular Parallelepiped

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the corners.
- **edge1** (*list*) – vector [x, y, z] denoting the first side of the box.
- **edge2** (*list*) – vector [x, y, z] denoting the second side of the box.
- **edge3** (*list*) – vector [x, y, z] denoting the second side of the box.

centre(*aabb=None*)

rotation()

lengths()

geant4Solid(*greg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.fluka.**SPH**(*name, point, radius, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin

Sphere

Parameters

- **name** (*str*) – of body
- **point** (*list*) – position [x, y, z] of the centre of the sphere.
- **radius** (*float*) – radius of the sphere.

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`class pyg4ometry.fluka.RCC(name, face, direction, radius, transform=None, flukaregistry=None, comment="")`

Bases: BodyMixin

Right Circular Cylinder

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the faces of the cylinder.
- **edge1** (*list*) – vector [x, y, z] denoting the direction along the length of the cylinder.
- **edge2** (*float*) – radius of the cylinder face.

`centre(aabb=None)`

`rotation()`

`geant4Solid(reg, aabb=None)`

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`class pyg4ometry.fluka.REC(name, face, direction, semiminor, semimajor, transform=None, flukaregistry=None, comment="")`

Bases: BodyMixin

Right Elliptical Cylinder

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the faces of the cylinder.
- **semiminor** (*list*) – vector [x, y, z] denoting the direction along the semiminor axis of the ellipse.
- **semimajor** (*list*) – vector [x, y, z] denoting the direction along the semimajor axis of the ellipse.

`centre(aabb=None)`

`rotation()`

`geant4Solid(reg, aabb=None)`

`__repr__()`

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.fluka.**TRC**(*name, major_centre, direction, major_radius, minor_radius, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin

Truncated Right-angled Cone

Parameters

- **name** (*str*) – of body
- **major_centre** (*list*) – vector [x, y, z] position of the centre of the larger face.
- **direction** (*list*) – vector [x, y, z] pointing from the larger face to the smaller face.
- **major_radius** (*float*) – radius of the larger face.
- **minor_radius** (*float*) – radius of the smaller face.

centre(*aabb=None*)

rotation()

geant4Solid(*registry, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.fluka.**ELL**(*name, focus1, focus2, length, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin

Ellipsoid of Revolution

Parameters

- **name** (*str*) – of body
- **focus1** (*list*) – position [x, y, z] denoting of one of the foci.
- **focus2** (*list*) – position [x, y, z] denoting the other focus.
- **length** (*float*) – length of the ellipse axis which the foci lie on.

centre(*aabb=None*)

rotation()

_linearEccentricity()

_semiminor()

geant4Solid(*greg, aabb=None*)

__repr__()

_withLengthSafety(*safety, reg*)

flukaFreeString()

hash()

class pyg4ometry.fluka.**WED**(*name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None, comment=""*)

Bases: **_WED_RAW**

Right Angle Wedge

Parameters

- **name** (*str*) – of body
- **vertex** (*list*) – position [x, y, z] of one of the the rectangular corners.
- **edge1** (*list*) – vector [x, y, z] denoting height of the wedge.
- **edge2** (*list*) – vector [x, y, z] denoting width of the wedge.
- **edge3** (*list*) – vector [x, y, z] denoting length of the wedge.

class pyg4ometry.fluka.**RAW**(*name, vertex, edge1, edge2, edge3, transform=None, flukaregistry=None, comment=""*)

Bases: **_WED_RAW**

Base class representing a body as defined in FLUKA

__doc__

class pyg4ometry.fluka.**ARB**(*name, vertices, facenumbers, transform=None, flukaregistry=None, comment=""*)

Bases: **BodyMixin**

Arbitrary Convex Polyhedron

Parameters

- **name** (*str*) – of body
- **vertices** (*list*) – Eight vertices which make up the polyhedron as [[x1, y1, z1], [x2, y2, z2], ...]. There must be eight even if only six or seven vertices are needed to make up the polyhedron.
- **facenumbers** (*float*) – The faces of the polyhedron expressed as floats where each digit of the float refers to one of the vertices which makes up that face. Six must always be provided as [1234,8765, ...], even if only four or five faces are needed. Any unneeded faces must be set to 0 (no less than 4 sides). Note that the references to the vertices are not zero-counting. The order of the vertices denoted in the facenumbers must be either all clockwise or all anticlockwise, which if not obeyed will result in erroneous output without warning.

centre(*aabb=None*)

rotation()

_faceNumbersToZeroCountingIndices()

_extent()

geant4Solid(*greg, aabb=None*)

`_toTesselatedSolid(verticesAndPolygons, greg, addRegistry)`

`_getVerticesAndPolygons()`

`_toVerticesAndPolygons(reverse)`

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`class pyg4ometry.fluka.XYP(name, z, transform=None, flukaregistry=None, comment="")`

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the z-axis)

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the x-y plane on the z-axis. All points less than z are considered to be part of this body.

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`toPlane()`

`class pyg4ometry.fluka.XZP(name, y, transform=None, flukaregistry=None, comment="")`

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the y-axis)

Parameters

- **name** (*str*) – of body
- **y** (*float*) – position of the x-y plane on the y-axis. All points less than y are considered to be part of this body.

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`toPlane()`

`class pyg4ometry.fluka.YZP(name, x, transform=None, flukaregistry=None, comment="")`

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the x-axis)

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the x-y plane on the x-axis. All points less than x are considered to be part of this body.

`__repr__()`

`_withLengthSafety(safety, reg)`

`flukaFreeString()`

`hash()`

`toPlane()`

class `pyg4ometry.fluka.PLA(name, normal, point, transform=None, flukaregistry=None, comment="")`

Bases: `_HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (perpendicular to the z-axis) Generic infinite half-space.

Parameters

- **name** (*str*) – of body
- **normal** (*list*) – position of a point on the plane
- **normal** – vector perpendicular to the face of the plane, pointing away from the contents of the half space.

`rotation()`

`__repr__()`

`_withLengthSafety(safety, reg=None)`

`flukaFreeString()`

`hash()`

`toPlane()`

class `pyg4ometry.fluka.XCC(name, y, z, radius, transform=None, flukaregistry=None, comment="")`

Bases: `_InfiniteCylinderMixin`, `_ShiftableCylinderMixin`

Infinite Circular Cylinder parallel to the x-axis

Parameters

- **name** (*str*) – of body
- **y** (*float*) – position of the centre on the
- **z** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

`centre(aabb=None)`

`rotation()`

`__repr__()`

Return repr(self).

`_withLengthSafety(safety, reg=None)`

flukaFreeString()

hash()

point()

direction()

class pyg4ometry.fluka.YCC(*name, z, x, radius, transform=None, flukaregistry=None, comment=""*)

Bases: _InfiniteCylinderMixin, _ShiftableCylinderMixin

Infinite Circular Cylinder parallel to the y-axis

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the centre on the
- **x** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

point()

direction()

class pyg4ometry.fluka.ZCC(*name, x, y, radius, transform=None, flukaregistry=None, comment=""*)

Bases: _InfiniteCylinderMixin, _ShiftableCylinderMixin

Infinite Circular Cylinder parallel to the z-axis

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the centre on the
- **y** (*float*) – position of the centre on the
- **radius** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

point()

direction()

class pyg4ometry.fluka.XEC(*name*, *y*, *z*, *ysemi*, *zsemi*, *transform=None*, *flukaregistry=None*, *comment=""*)

Bases: BodyMixin, _ShiftableCylinderMixin

Infinite Elliptical Cylinder parallel to the x-axis

Parameters

- **name** (*str*) – of body
- **y** (*float*) – position of the centre on the
- **z** (*float*) – position of the centre on the
- **ysemi** (*float*) – position of the centre on the
- **zsemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg*, *aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety*, *reg=None*)

flukaFreeString()

hash()

class pyg4ometry.fluka.YEC(*name*, *z*, *x*, *zsemi*, *xsemi*, *transform=None*, *flukaregistry=None*, *comment=""*)

Bases: BodyMixin, _ShiftableCylinderMixin

Infinite Elliptical Cylinder parallel to the y-axis

Parameters

- **name** (*str*) – of body
- **z** (*float*) – position of the centre on the
- **x** (*float*) – position of the centre on the
- **zsemi** (*float*) – position of the centre on the
- **xsemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg*, *aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

class pyg4ometry.fluka.ZEC(*name, x, y, xsemi, ysemi, transform=None, flukaregistry=None, comment=""*)

Bases: BodyMixin, _ShiftableCylinderMixin

Infinite Elliptical Cylinder parallel to the z-axis

Parameters

- **name** (*str*) – of body
- **x** (*float*) – position of the centre on the
- **y** (*float*) – position of the centre on the
- **xsemi** (*float*) – position of the centre on the
- **ysemi** (*float*) – position of the centre on the

centre(*aabb=None*)

rotation()

geant4Solid(*reg, aabb=None*)

__repr__()

Return repr(self).

_withLengthSafety(*safety, reg=None*)

flukaFreeString()

hash()

class pyg4ometry.fluka.QUA(*name, cxx, cyy, czz, cxy, cxz, cyz, cx, cy, cz, c, transform=None, flukaregistry=None, comment="", **kwargs*)

Bases: BodyMixin

Generic quadric

Parameters

- **name** (*str*) – of body
- **cxx** (*float*) – x^2 coefficient
- **cyy** (*float*) – y^2 coefficient
- **czz** (*float*) – z^2 coefficient
- **cxy** (*float*) – xy coefficient
- **cxz** (*float*) – xz coefficient
- **cyz** (*float*) – yz coefficient
- **cx** (*float*) – x coefficient
- **cy** (*float*) – y coefficient

- **cz** (*float*) – z coefficient
- **c** (*constant*) – constant

centre(*aabb=None*)

rotation()

coefficientsMatrix()

static **_quadricMatrixToCoefficients**(*matrix*)

mesh(*lower, upper, capping=True*)

geant4Solid(*reg, aabb=None*)

_withLengthSafety(*safety, reg=None*)

__repr__()

flukaFreeString()

hash()

pyg4ometry.fluka.infinity(*inf*)

Use this to temporarily modify INFINITY, with it resetting back to the default once the block has exited. INFINITY is used throughout the bodies to approximate the infinite size of infinity (elliptical) cylinders, half spaces, and quadrics.

Parameters

inf – the value to temporarily set INFINITY to.

class **pyg4ometry.fluka.Reader**(*filename*)

Class to read a FLUKA file.

getRegistry()

Get the fluka registry

_load()

Load the FLUKA input file

_findLines()

_parseBodies()

_parseRegions()

_parseCards()

_parseRotDefinis()

_parseGeometryDirective(*line_parts, expansion_stack, translation_stack, transform_stack*)

_parseMaterialAssignments()

_parseLattice()

_parseMaterials()

_splitMaterialCards()

Return a tuple as (Elements, Compounds, Matprop (not yet implemented)). Each is keyed with the name of the material, and the value corresponds to the card (Element) or cards (Compounds) that define the material.

class pyg4ometry.fluka.Writer

Class to write FLUKA input files from a fluka registry object.

```
>>> f = Writer()
>>> f.addDetector(flukaRegObject)
>>> f.write("model.inp")
```

_flukaFFString =

'*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...'

addDetector(flukaRegistry)

Set the fluka registry and therefore the model for this writer instance.

write(fileName)

Write the output to a given filename. e.g. "model.inp".

class pyg4ometry.fluka.FlukaRegistry

Object to store geometry for FLUKA input and output. All of the FLUKA classes can be used without storing them in the Registry. The registry is used to write the FLUKA output file.

addBody(body)**makeBody(clas, *args, **kwargs)****getDegenerateBody(body)****addRotoTranslation(rototrans)****addRegion(region, addBodies=False)****makeRegionsDNF()****addLattice(lattice)****getBody(name)****getBodyToRegionsMap()****printDefinitions()****regionAABBs(write=None)****latticeAABBs()****addMaterial(material, recursive=False)****getMaterial(name)****addMaterialAssignments(mat, *regions, elc=None, mgn=None)****assignma(material, *regions)****addCard(card)****addTitle(title='FLUKA simulation')**

```

addDefaults(default='EM-CASCA')

addGlobal()

addBeam(energy, particle='ELECTRON')

addBeamPos(xpos=0, ypos=0, zpos=0, xdc=0, ydc=0)

addLowMat(flukaMat, lowENeutron1, lowENeutron2, lowENeutron3)

addLowMatAllMaterials()
    Add LOWMAT card to all materials

addLowPwxs(what1=None, lowerMaterial=None, upperMaterial=None)

addMgnCreat()

addMgnField()

addElcField()

addUsrBin(mesh=0, particle='ENERGY', lunOutput=-21, e1max=1, e2max=1, e3max=1, name='name',
            e1min=-1, e2min=-1, e3min=-1, e1nbin=10, e2nbin=10, e3nbin=10)

addRotprBin(precision=0, rotDefi=0, printEventBin=0, lowerBin=None, upperBin=None)

addUsrBdx(binning, scoringDir, scoringType, type, reg1, reg2, name, area=1.0, lunOutput=-22,
            maxKE=None, minKE=None, nKEbin=None, maxSA=None, minSA=None, nSAbin=None)

addUsrIcall()

addUsrOcall()

addUsrDump(mgdraw=100, lun=70, mgdrawOpt=-1, what4=0, sdum=None)

addRandomiz(seedLun=1, seed=54217137)

addStart(maxPrimHistories=1, timeTermSec=None, coreDump=None, eachHistoryOutput=None)

addPhotonuc(what1, mat_low, mat_high, mat_step, sdum="")

addMuphoton(what1, mat_low, mat_high, mat_step)

addPairbrem(what1, what2, what3, mat_low, mat_high, mat_step)

addDeltaRay(what1, what2, what3, mat_low, mat_high, mat_step, dsum='NOPRINT')

addIonFluct(what1, what2, what3, mat_low, mat_high, mat_step, dsum='PRIM-ION')

printDumps(detail=1)

findLastBodyIndex()
    Find last body index (if the numbering was performed by a geant4 -> fluka conversion)

findLastRegionIndex()
    Find last region index (if the numbering was performed by a geant4 -> fluka conversion)

findLastMaterialIndex()
    Find last material index (if the numbering was performed by a geant4 -> fluka conversion)

```

```
findLastTransformationIndex()

checkBodyName(bodyName)

checkRegionName(regionName)

checkMaterialName(materialName)

addRegistry(flukaRegistry, outerRegion=None, rotation=[0, 0, 0], translation=[0, 0, 0],
            removeRegions=[], removeRegionDependents=False)
```

```
class pyg4ometry.fluka.FlukaBodyStoreExact
```

```
    _bodyNames()

    _bodies()

    make(cls, *args, **kwargs)

    getDegenerateBody(body)

    addBody(body)

    keys()

    values()

    __setitem__(key, value)

    __getitem__(key)

    __delitem__(key)

    __len__()

    __contains__(key)

    __iter__()

    __repr__()
        Return repr(self).
```

```
class pyg4ometry.fluka.Three(shape, dtype=float, buffer=None, offset=0, strides=None, order=None)
```

```
    Bases: numpy.ndarray
```

```
    property x
```

```
    property y
```

```
    property z
```

```
    parallel_to(other, tolerance=1e-10)
```

```
        Check if instance is parallel to some other vector, v
```

```
    unit()
```

```
        Get this as a unit vector.
```

```
    length()
```

```
        vector length (l2 norm)
```



```

    dot(other)

    cross(other)

    __eq__(other)
        Return self==value.

    __ne__(other)
        Return self!=value.

    __add__(other)

    __radd__(other)

    __sub__(other)

    __rsub__(other)

    __iadd__(other)

    __isub__(other)

class pyg4ometry.fluka.AABB(lower, upper)

    __repr__()
        Return repr(self).

    __eq__(other)
        Return self==value.

    cornerDistance()

    classmethod fromMesh(csgmesh)

    intersects(other)

    coplanarIntersects(other)

    envelops(other)

    intersect(other)

    union(other)

    isNull()

```

```
class pyg4ometry.fluka.Zone(name=None)
```

Bases: [pyg4ometry.fluka.vis.ViewableMixin](#)

Represents a zone which consists of one or more body intersections and zero or more body subtractions. May also be used to represent subzones, which are zones nested within zones, for example the form +A -B -(+C - D). Once instantiated, intersections and subtractions can be added to this instance with the addIntersection and addSubtraction method.

Parameters

name (*string*) – Optional name for the zone.

addSubtraction(*body*)

Add a body or subzone as a subtraction to this Zone instance.

Parameters

body (*Body* or *Zone*) – Body or Zone instance.

addIntersection(*body*)

Add a body or subzone as an intersection to this Zone instance.

Parameters

body – Body or Zone instance.

Type

body: *Body* or *Zone*

convertToDNF(*fluka_registry*)**centre**(*aabb=None*)**rotation**()**tbxyz**()**static** **_getSolidFromBoolean**(*boolean, g4reg, aabb*)**mesh**(*aabb=None*)**geant4Solid**(*reg, aabb=None*)

Translate this zone to a geant4solid, adding the constituent primitive solids and any Booleans to the Geant4 registry. Returns the geant4 solid equivalent in geometry to this Zone.

Parameters

- **reg** (*AABB* or *dict*) – The Registry (geant4) instance to which the resulting Geant4 definitions should be added.
- **aabb** – Optional reference AABB or dictionary of body name to AABB instances with which the geant4 solid should be evaluated with respect to. This is the entry point to which solid minimisation can be performed.

_combine_booleans(*body0, start, collection, operation, reg, aabb*)**_geant4MultiUnionSubtraction**(*body0, start, reg, aabb*)**dumps**()

Returns a string of this Zone instance in the equivalent FLUKA syntax.

dumpsDebug()

Returns a string of this Zone instance in the equivalent FLUKA syntax with extra debug information

withLengthSafety(*bigger_flukareg, smaller_flukareg, shrink_intersections*)**allBodiesToRegistry**(*flukaregistry*)

Add all the bodies that constitute this Zone to the provided FlukaRegistry instance.

Parameters

flukaregistry (*FlukaRegistry*) – FlukaRegistry instance to which constituent bodies will be added.

bodies()

Return the set of unique bodies that constitute this Zone.

removeBody(*name*)

Remove a body from this zone by name.

Parameters

name (*string*) – The name of the body to be removed.

makeUnique(*nameSuffix*, *flukaregistry*)

Get this zone with every constituent body recreated with a unique name by appending *nameSuffix*.

Parameters

- **nameSuffix** – The string to append to the names of the bodies.
- **flukaregistry** – the FlukaRegistry instance to add the uniquely defined bodies to.

isNull(*aabb=None*)

toDNF(*name*)

isDNF()

leafCount()

class `pyg4ometry.fluka.Region`(*name*, *comment=""*)

Bases: `pyg4ometry.fluka.vis.ViewableMixin`

Represents a region which consists of a region name, one or more zones, and a single material. Metadata may be provided with the *comment* kwarg, which is used when writing to FLUKA to provide contextual information to the physicist.

Parameters

- **name** (*str*) – The name of this region.
- **material** (*str*) – The name of a material.
- **comment** – Optional descriptive comment.

addZone(*zone*)

Add a Zone instance to this region.

Parameters

zone (*Zone*) – The Zone instance to be added.

addIntersection(*zone*)

addSubtraction(*zone*)

extend(*region*)

removeNullZones()

convertToDNF(*fluka_registry*)

centre(*aabb=None*)

tbxyz()

rotation()

bodies()

Return the set of unique bodies that constitute this Zone.

mesh(*aabb=None*)

geant4Solid(*reg, aabb=None*)

Get the geant4Solid instance corresponding to this Region.

dumps()

flukaFreeString()

withLengthSafety(*bigger_flukareg, smaller_flukareg*)

allBodiesToRegistry(*registry*)

Add all the bodies that constitute this Region to the provided FlukaRegistry instance.

Parameters

flukaregistry ([FlukaRegistry](#)) – FlukaRegistry instance to which constituent bodies will be added.

zoneGraph(*zoneAABBs=None, aabb=None*)

_zoneGraphPycgal()

connectedZones(*zoneAABBs=None, aabb=None*)

zoneAABBs(*aabb=None*)

aabb(*aabb=None*)

removeBody(*name*)

Remove a body from this region by name.

Parameters

name (*string*) – The name of the body to be removed.

makeUnique(*nameSuffix, flukaregistry*)

Get this Region instance with every constituent body with a unique name by appending nameSuffix to each Body instance.

Parameters

- **nameSuffix** – string to append to each Body instance.
- **flukaregistry** – FlukaRegistry instance to add each newly-defined body to.

isNull(*aabb=None*)

toDNF(*name*)

isDNF()

filterNullZones(*aabb=None*)

leafCount()

simplify()

__repr__()

removeZones(*indices*)

Remove zones by index

```

pyg4ometry.fluka.bracket_depth(zone)
pyg4ometry.fluka.bracket_number(zone)
pyg4ometry.fluka.zone_to_sympy(zone)
pyg4ometry.fluka.region_to_sympy(region)
pyg4ometry.fluka.sympy_to_region(sympy_expr, freq, regionName='name')
    Convert sympy boolean to fluka region Must be the OR of multiple convex zones
class pyg4ometry.fluka.Transform(*, expansion=None, translation=None, rotoTranslation=None,
                                invertRotoTranslation=None)

    Bases: MatrixConvertibleMixin

    expansion, translation, rotoTranslation can be either a single instance of RotoTranslation or a multiple instances
    of RotoTranslation and RecursiveRotoTranslation

    _expansionsTo4DMatrices()

    _translationsTo4DMatrices()

    _rotoTranslationsTo4DMatrices()

    to4DMatrix()

class pyg4ometry.fluka.RotoTranslation(name, axis=None, polar=0.0, azimuth=0.0, translation=None,
                                       transformationIndex=None, flukaregistry=None)

    Bases: MatrixConvertibleMixin

    translation in mm, angles in degrees

    __repr__()
        Return repr(self).

    to4DMatrix()

    toCard()

    flukaFreeString()

    classmethod fromCard(card)

    hasTranslation()

    hasRotation()

    isPureTranslation()

class pyg4ometry.fluka.RecursiveRotoTranslation(name, rotoTranslations)

    Bases: collections.abc.MutableSequence, MatrixConvertibleMixin

    container for dealing with a recursively defined rototranslation. they must also refer to the same rototrans, i.e.,
    have the same name. for a list of rototranslations supplied:

    [a, b, c], the order of evaluation acting on a vector v is  $c*b*a*v$ . so teh first rototrans is applied first.. and so on.

    property transformationIndex

    __repr__()
        Return repr(self).

```

```
__getitem__(i)
_raiseIfDifferentName(name)
__setitem__(i, obj)
__delitem__(i)
__len__()
insert(i, obj)
    S.insert(index, value) – insert value before index
to4DMatrix()
flukaFreeString()
_transformationIndices()
areAllTheSameTransformationIndices()
transformationIndex()
class pyg4ometry.fluka.Lattice(cellRegion, rotoTranslation, invertRotoTranslation=False,
                               flukaregistry=None)
    flukaFreeString(delim=', ')
    getTransform()
class pyg4ometry.fluka.Flair(flukaInputFileName='fluka.inp', extent=None)
    write(fileName)
    addMaterialColour(materialName, color=(0, 0, 0, 0))
class pyg4ometry.fluka.BuiltIn(name, *, atomicNumber=None, atomicMass=None, density=None,
                               flukaregistry=None)
    __repr__()
    Return repr(self).
    flukaFreeString(delim="")
class pyg4ometry.fluka.Material(name, atomicNumber, density, massNumber=None, atomicMass=None,
                                pressure=None, flukaregistry=None, comment="")
```

Bases: `_MatProp`

A FLUKA material consisting of a single element. This corresponds to the case in FLUKA of a single MATERIAL card with no associated COMPOUND cards, as well as a possible MAT-PROP card (only if a pressure is provided, other options of MAT-PROP are unsupported).

Parameters

- **name** (*str*) – The name of the material
- **atomicNumber** (*int*) – the atomic number, Z, of the element.
- **density** (*float*) – the density in g/cm³ of the material.
- **massNumber** (*int*, *None*) – Optional mass number, will be inferred in FLUKA based on atomicNumber. Allows one to specify a specific isotope.

- **atomicMass** (*float*) – The mass of the atom in g/mole. Will be inferred in FLUKA based on atomicNumber.
- **pressure** (*float*) – Optional pressure if the material is a gas.
- **flukaregistry** (*FlukaRegistry*) – Optional FlukaRegistry instance the material is to be added to.

toCards()

flukaFreeString(*delim*=' ')

__repr__()

Return repr(self).

classmethod fromCard(*card*, *flukaregistry*)

rename(*newName*, *recursive*=False, *iIndex*=0)

class pyg4ometry.fluka.**Compound**(*name*, *density*, *fractions*, *fractionType*, *pressure*=None, *flukaregistry*=None, *comment*="")

Bases: *MatProp*

A FLUKA compound material. This corresponds to the case in FLUKA of a single MATERIAL card with one or more associated COMPOUND cards.

Parameters

- **name** (*str*) – The name of the compound.
- **density** (*float*) – The density of the compound in g/cm3
- **fractions** (*list*) – List of (Element, fraction) and (Compound, fraction) tuples corresponding to the fractional proportion of that material.
- **fractionType** (*str*) – The type of the fractions listed in the fractions parameter, either atomic, mass, or volume.
- **flukaregistry** (*FlukaRegistry*) – Optional FlukaRegistry instance the Compound is to be added to.

toCards()

flukaFreeString(*delim*=' ')

classmethod fromCards(*cards*, *flukareg*)

__repr__()

Return repr(self).

totalWeighting(*densityWeighted*=False)

rename(*newName*, *recursive*=True, *iIndex*=0)

class pyg4ometry.fluka.**Card**(*keyword*, *what1*=None, *what2*=None, *what3*=None, *what4*=None, *what5*=None, *what6*=None, *sdum*=None)

Card class for representing a FLUKA input card. To construct instances from a of FLUKA input, use the fromFree or fromFixed class method for FREE and FIXED format, respectively.

__repr__()

Return repr(self).

toList()

toFreeString(*delim*=' ')

toFixedString()

nonesToZero()

Return a class instance with same contents as this instance, but with all entries of None set to 0.0 instead.

classmethod fromFree(*line*)

classmethod fromFixed(*line*)

class pyg4ometry.fluka.**Extruder**(*name*="", *length*=1000, *regions*={}, *registry*=None)

Bases: [pyg4ometry.geant4.solid.SolidBase](#)

Base class for all solids

addRegion(*name*)

addPointToRegion(*name*, *pntIndx*)

setRegionToOuterBoundary(*name*)

buildCgalPolygons()

buildGeant4Extrusions()

plot(*decompositions*=False)

mesh()

[pyg4ometry.freecad](#)

Submodules

[pyg4ometry.freecad.Reader](#)

Module Contents

Classes

[Reader](#)

Functions

MeshToFacetList(mesh)

WriteSMeshFile(mesh, filename)

FacetListAxisAlignedExtent(facetList)

PartFeatureGlobalPlacement(obj, placement)

MeshAnalysis(m)

MeshCleaning(m)

```
class pyg4ometry.freecad.Reader.Reader(fileName, registryOn=True, fileNameAux=None)

    load(fileName)

    simplifyModel(volumeCut=500000.0)

    relabelModel()

    loadAuxiliaryData(fileName, colorByMaterial=True)

    convertStructure()
        Convert file with structure

    setLogicalVolumeMaterial(logicalVolumeName, material='G4_Galactic')

    convertFlat(meshDeviation=0.05, centreName='', globalOffset=_fc.Vector(),
                globalRotation=_fc.Rotation(), extentScale=1.0, daughterMaterial='G4_Galactic',
                storePartCentrePos=False, meshShrinkFactor=1e-06)
        Convert file without structure

    getRegistry()

    printPartFeatures(fileName=None, randomColors=False)
        Print to screen or write to file Part::Features with color and material

    printStructure()

    recursePrintObjectTree(obj)

    recurseObjectTree(obj)

pyg4ometry.freecad.Reader.MeshToFacetList(mesh)

pyg4ometry.freecad.Reader.WriteSMeshFile(mesh, filename)

pyg4ometry.freecad.Reader.FacetListAxisAlignedExtent(facetList)

pyg4ometry.freecad.Reader.PartFeatureGlobalPlacement(obj, placement)

pyg4ometry.freecad.Reader.MeshAnalysis(m)

pyg4ometry.freecad.Reader.MeshCleaning(m)
```

Package Contents

`pyg4ometry.freecad.useFreeCAD = True`

`pyg4ometry.freecad.useFreeCAD = False`

`pyg4ometry.gdml`

Subpackages

`pyg4ometry.gdml.GdmlExpression`

Submodules

`pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval`

Module Contents

Classes

<i>GdmlExpressionEvalVisitor</i>

<i>ExpressionParser</i>

```
class pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor
    Bases: pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor
    visitVariable(ctx)
    visitPrintExpr(ctx)
    visitScientific(ctx)
    visitMultiplyingExpression(ctx)
    visitExpression(ctx)
    visitPowExpression(ctx)
    visitMinExpression(ctx)
    visitMaxExpression(ctx)
    visitMatrixElement(ctx)
    visitParens(ctx)
    visitSignedAtom(ctx)
    visitAtom(ctx)
```

```
visitFunc(ctx)
```

```
visitFuncname(ctx)
```

```
visitConstant(ctx)
```

```
class pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.ExpressionParser
```

```
parse(expression)
```

```
evaluate(parse_tree, define_dict={})
```

```
get_variables(parse_tree)
```

```
pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer
```

Module Contents

Classes

```
GdmlExpressionLexer
```

Functions

```
serializedATN()
```

```
pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.serializedATN()
```

```
class pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer(input=None,
                                                                              out-
                                                                              put=sys.stdout)
```

Bases: Lexer

Parameters

output (TextIO) –

atn

decisionsToDFA

COS = 1

SIN = 2

TAN = 3

ACOS = 4

ASIN = 5

```
ATAN = 6
LN = 7
LOG = 8
SQRT = 9
EXP = 10
POWER = 11
ABS = 12
LPAREN = 13
RPAREN = 14
LBRACKET = 15
RBRACKET = 16
PLUS = 17
MINUS = 18
TIMES = 19
DIV = 20
GT = 21
LT = 22
EQ = 23
COMMA = 24
POINT = 25
POW = 26
PI = 27
EULER = 28
I = 29
MIN = 30
MAX = 31
VARIABLE = 32
SCIENTIFIC_NUMBER = 33
WS = 34
channelNames = ['DEFAULT_TOKEN_CHANNEL', 'HIDDEN']
modeNames = ['DEFAULT_MODE']
```

```

literalNames = ['<INVALID>', "'cos'", "'sin'", "'tan'", "'acos'", "'asin'",
"'atan'", "'log'", "'log10'",...]

symbolicNames = ['<INVALID>', 'COS', 'SIN', 'TAN', 'ACOS', 'ASIN', 'ATAN', 'LN',
'LOG', 'SQRT', 'EXP', 'POWER',...]

ruleNames = ['COS', 'SIN', 'TAN', 'ACOS', 'ASIN', 'ATAN', 'LN', 'LOG', 'SQRT',
'EXP', 'POWER', 'ABS',...]

grammarFileName = 'GdmlExpression.g4'

```

`pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser`

Module Contents

Classes

GdmlExpressionParser

Functions

serializedATN()

`pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.serializedATN()`

`class pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser(input, output=sys.stdout)`

Bases: Parser

Parameters

- **input** (*TokenStream*) –
- **output** (*TextIO*) –

`class EquationContext(parser, parent=None, invokingState=-1)`

Bases: ParserRuleContext

Parameters

- **parent** (*ParserRuleContext*) –
- **invokingState** (*int*) –

`__slots__ = 'parser'`

`expression(i=None)`

Parameters

- **i** (*Optional[int]*) –

```
    relop()

    getRuleIndex()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class ExpressionContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext

    Parameters

        • parent (ParserRuleContext) –
        • invokingState (int) –

    __slots__ = 'parser'

    multiplyingExpression(i=None)
        Parameters
            i (Optional[int]) –

    operatorAddSub(i=None)
        Parameters
            i (Optional[int]) –

    getRuleIndex()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class MultiplyingExpressionContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext

    Parameters

        • parent (ParserRuleContext) –
        • invokingState (int) –

    __slots__ = 'parser'

    powExpression(i=None)
        Parameters
            i (Optional[int]) –

    operatorMulDiv(i=None)
        Parameters
            i (Optional[int]) –

    getRuleIndex()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –
```

```
class OperatorAddSubContext(parser, parent=None, invokingState=-1)
```

```
    Bases: ParserRuleContext
```

```
        Parameters
```

- **parent** (*ParserRuleContext*) –
- **invokingState** (*int*) –

```
    __slots__ = 'parser'
```

```
    PLUS()
```

```
    MINUS()
```

```
    getRuleIndex()
```

```
    accept(visitor)
```

```
        Parameters
```

- visitor** (*ParseTreeVisitor*) –

```
class OperatorMulDivContext(parser, parent=None, invokingState=-1)
```

```
    Bases: ParserRuleContext
```

```
        Parameters
```

- **parent** (*ParserRuleContext*) –
- **invokingState** (*int*) –

```
    __slots__ = 'parser'
```

```
    TIMES()
```

```
    DIV()
```

```
    getRuleIndex()
```

```
    accept(visitor)
```

```
        Parameters
```

- visitor** (*ParseTreeVisitor*) –

```
class PowExpressionContext(parser, parent=None, invokingState=-1)
```

```
    Bases: ParserRuleContext
```

```
        Parameters
```

- **parent** (*ParserRuleContext*) –
- **invokingState** (*int*) –

```
    __slots__ = 'parser'
```

```
    signedAtom(i=None)
```

```
        Parameters
```

- i** (*Optional[int]*) –

```
    POW(i=None)
```

```
        Parameters
```

- i** (*Optional[int]*) –

```
    getRuleIndex()
```

```
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class SignedAtomContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext

        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –

    __slots__ = 'parser'

    PLUS()

    signedAtom()

    MINUS()

    func()

    atom()

    getRuleIndex()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class AtomContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext

        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –

    __slots__ = 'parser'

    scientific()

    matrixElement()

    variable()

    constant()

    LPAREN()

    expression()

    RPAREN()

    getRuleIndex()

    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –
```



```

class ScientificContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –
    __slots__ = 'parser'
    SCIENTIFIC_NUMBER()
    getRuleIndex()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –
class MatrixElementContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –
    __slots__ = 'parser'
    variable()
    LBRACKET()
    expression(i=None)
        Parameters
            i (Optional[int]) –
    RBRACKET()
    COMMA(i=None)
        Parameters
            i (Optional[int]) –
    getRuleIndex()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –
class ConstantContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –
    __slots__ = 'parser'
    PI()

```

```
EULER()

I()

getRuleIndex()

accept(visitor)
    Parameters
        visitor (ParseTreeVisitor) –

class VariableContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –
    __slots__ = 'parser'
    VARIABLE()
    getRuleIndex()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –

class FuncContext(parser, parent=None, invokingState=-1)
    Bases: ParserRuleContext
        Parameters
            • parent (ParserRuleContext) –
            • invokingState (int) –
    __slots__ = 'parser'
    funcname()
    LPAREN()
    expression(i=None)
        Parameters
            i (Optional[int]) –
    RPAREN()
    COMMA(i=None)
        Parameters
            i (Optional[int]) –
    getRuleIndex()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –
```

```
class FuncnameContext(parser, parent=None, invokingState=-1)
```

```
    Bases: ParserRuleContext
```

Parameters

- **parent** (*ParserRuleContext*) –
- **invokingState** (*int*) –

```
    __slots__ = 'parser'
```

```
    COS()
```

```
    TAN()
```

```
    SIN()
```

```
    ACOS()
```

```
    ATAN()
```

```
    ASIN()
```

```
    LOG()
```

```
    LN()
```

```
    EXP()
```

```
    SQRT()
```

```
    POWER()
```

```
    ABS()
```

```
    MIN()
```

```
    MAX()
```

```
    getRuleIndex()
```

```
    accept(visitor)
```

Parameters

- **visitor** (*ParseTreeVisitor*) –

```
class RelopContext(parser, parent=None, invokingState=-1)
```

```
    Bases: ParserRuleContext
```

Parameters

- **parent** (*ParserRuleContext*) –
- **invokingState** (*int*) –

```
    __slots__ = 'parser'
```

```
    EQ()
```

```
    GT()
```

```
    LT()
```

```
    getRuleIndex()
    accept(visitor)
        Parameters
            visitor (ParseTreeVisitor) –
grammarFileName = 'GdmlExpression.g4'
atn
decisionsToDFA
sharedContextCache
literalNames = ['<INVALID>', "'cos'", "'sin'", "'tan'", "'acos'", "'asin'",
"'atan'", "'log'", "'log10'",...]
symbolicNames = ['<INVALID>', 'COS', 'SIN', 'TAN', 'ACOS', 'ASIN', 'ATAN', 'LN',
'LOG', 'SQRT', 'EXP', 'POWER',...]
RULE_equation = 0
RULE_expression = 1
RULE_multiplyingExpression = 2
RULE_operatorAddSub = 3
RULE_operatorMulDiv = 4
RULE_powExpression = 5
RULE_signedAtom = 6
RULE_atom = 7
RULE_scientific = 8
RULE_matrixElement = 9
RULE_constant = 10
RULE_variable = 11
RULE_func = 12
RULE_funcname = 13
RULE_relop = 14
ruleNames = ['equation', 'expression', 'multiplyingExpression', 'operatorAddSub',
'operatorMulDiv',...]
EOF
COS = 1
SIN = 2
TAN = 3
```

ACOS = 4
ASIN = 5
ATAN = 6
LN = 7
LOG = 8
SQRT = 9
EXP = 10
POWER = 11
ABS = 12
LPAREN = 13
RPAREN = 14
LBRACKET = 15
RBRACKET = 16
PLUS = 17
MINUS = 18
TIMES = 19
DIV = 20
GT = 21
LT = 22
EQ = 23
COMMA = 24
POINT = 25
POW = 26
PI = 27
EULER = 28
I = 29
MIN = 30
MAX = 31
VARIABLE = 32
SCIENTIFIC_NUMBER = 33
WS = 34

```
equation()  
expression()  
multiplyingExpression()  
operatorAddSub()  
operatorMulDiv()  
powExpression()  
signedAtom()  
atom()  
scientific()  
matrixElement()  
constant()  
variable()  
func()  
funcname()  
relop()
```

`pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor`

Module Contents

Classes

GdmlExpressionVisitor

class `pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor`

Bases: `ParseTreeVisitor`

visitEquation(*ctx*)

Parameters

ctx (`GdmlExpressionParser.EquationContext`) –

visitExpression(*ctx*)

Parameters

ctx (`GdmlExpressionParser.ExpressionContext`) –

visitMultiplyingExpression(*ctx*)

Parameters

ctx (`GdmlExpressionParser.MultiplyingExpressionContext`) –

visitOperatorAddSub(*ctx*)

Parameters

ctx (`GdmlExpressionParser.OperatorAddSubContext`) –

visitOperatorMulDiv(*ctx*)

Parameters

ctx (`GdmlExpressionParser.OperatorMulDivContext`) –

visitPowExpression(*ctx*)

Parameters

ctx (`GdmlExpressionParser.PowExpressionContext`) –

visitSignedAtom(*ctx*)

Parameters

ctx (`GdmlExpressionParser.SignedAtomContext`) –

visitAtom(*ctx*)

Parameters

ctx (`GdmlExpressionParser.AtomContext`) –

visitScientific(*ctx*)

Parameters

ctx (`GdmlExpressionParser.ScientificContext`) –

visitMatrixElement(*ctx*)

Parameters

ctx (`GdmlExpressionParser.MatrixElementContext`) –

visitConstant(*ctx*)

Parameters

ctx (`GdmlExpressionParser.ConstantContext`) –

visitVariable(*ctx*)

Parameters

ctx (`GdmlExpressionParser.VariableContext`) –

visitFunc(*ctx*)

Parameters

ctx (`GdmlExpressionParser.FuncContext`) –

visitFuncname(*ctx*)

Parameters

ctx (`GdmlExpressionParser.FuncnameContext`) –

visitRelop(*ctx*)

Parameters

ctx (`GdmlExpressionParser.RelopContext`) –

Package Contents

Classes

ExpressionParser

```
class pyg4ometry.gdml.GdmlExpression.ExpressionParser
```

```
    parse(expression)
```

```
    evaluate(parse_tree, define_dict={})
```

```
    get_variables(parse_tree)
```

Submodules

```
pyg4ometry.gdml.Defines
```

Module Contents

Classes

<i>BasicExpression</i>	Holds an expression as a string and can use the expression parser
<i>DefineBase</i>	Common bits for a define. Must have a name and a registry. Adding
<i>ScalarBase</i>	Base class for all scalars (Constants, Quantity, Variable and 'Expression')
<i>Constant</i>	GDML constant define wrapper object
<i>Quantity</i>	GDML quantity define wrapper object
<i>Variable</i>	GDML variable define wrapper object
<i>Expression</i>	General expression, does not have an analogue in GDML
<i>VectorBase</i>	
<i>Position</i>	GDML position define wrapper object
<i>Rotation</i>	GDML rotation define wrapper object
<i>Scale</i>	GDML scale define wrapper object
<i>Matrix</i>	GDML matrix define wrapper object
<i>Auxiliary</i>	Auxiliary information container object

Functions

<code>upgradeToStringExpression</code> (reg, obj)	Take a float, str, ScalarBase and return string expression.
<code>evaluateToFloat</code> (reg, obj)	
<code>upgradeToExpression</code> (reg, obj)	Helper functions that takes a string and returns an expression object or a string
<code>upgradeToVector</code> (var, reg[, type, unit, addRegistry])	Take a list [x,y,z] and create a vector
<code>upgradeToTransformation</code> (var, reg[, addRegistry])	Take a list of lists [[rx,ry,rz],[x,y,z]] and create a transformation [Rotation,Position]
<code>operationReturnType</code> (name, strExpr, v1, v2, type1, ...)	
<code>sin</code> (arg)	Sin of a ScalarBase object, returns a Constant
<code>cos</code> (arg)	Cosine of a ScalarBase object, returns a Constant
<code>tan</code> (arg)	Tangent of a ScalarBase object, returns a Constant
<code>asin</code> (arg)	ArcSin of a ScalarBase object, returns a Constant
<code>acos</code> (arg)	ArcCos of a ScalarBase object, returns a Constant
<code>atan</code> (arg)	ArcTan of a ScalarBase object, returns a Constant
<code>exp</code> (arg)	Exponential of a ScalarBase object, returns a Constant
<code>log</code> (arg)	Natural logarithm of a ScalarBase object, returns a Constant
<code>log10</code> (arg)	Base 10 logarithm of a ScalarBase object, returns a Constant
<code>sqrt</code> (arg)	Square root of a ScalarBase object, returns a Constant
<code>pow</code> (arg, power)	arg raised to power
<code>abs</code> (arg)	absolute value of arg
<code>min</code> (arg1, arg2)	absolute value of arg
<code>max</code> (arg1, arg2)	absolute value of arg
<code>MatrixFromVectors</code> (e, v, name, registry[, eunit, vunit])	Creates a GDML Matrix from an energy and a value vector

class `pyg4ometry.gdml.Defines.BasicExpression`(name, expressionString, registry)

Holds an expression as a string and can use the expression parser in the supplied registry to evaluate it. A registry is required.

Parameters

- **name** (*str*) – Name of the expression object
- **expressionString** (*str*) – Expression itself as a string e.g. “12.0” or “a + 3.0”
- **registry** (`pyg4ometry.geant4.Registry.Registry`) – The registry object to give context for any variables used.

```
>>> r = pyg4ometry.geant4.Registry()
>>> a = BasicExpression("a", "3.0", r)
>>> float(a)
>>> str(a)
```

`eval()`

`variables(allDependents=False)`

`simp()`

__repr__()

Return repr(self).

__float__()

str()

`pyg4ometry.gdml.Defines.upgradeToStringExpression(reg, obj)`

Take a float, str, ScalarBase and return string expression.

Parameters

- **reg** ([Registry](#)) – Registry for lookup in define dictionary
- **obj** ([str](#), [float](#), [ScalarBase](#)) – Object to upgrade

Returns

String expression

Return type

[str](#)

`pyg4ometry.gdml.Defines.evaluateToFloat(reg, obj)`

`pyg4ometry.gdml.Defines.upgradeToExpression(reg, obj)`

Helper functions that takes a string and returns an expression object or a string

`pyg4ometry.gdml.Defines.upgradeToVector(var, reg, type='position', unit='', addRegistry=False)`

Take a list [x,y,z] and create a vector

Parameters

- **var** (*list of* [str](#), [float](#), [Constant](#), [Quantity](#), [Variable](#)) – input list to create a position, rotation or scale
- **reg** ([Registry](#)) – registry
- **type** ([str](#)) – class type of vector (position, rotation, scale)
- **addRegistry** ([bool](#)) – flag to add to registry

`pyg4ometry.gdml.Defines.upgradeToTransformation(var, reg, addRegistry=False)`

Take a list of lists [[rx,ry,rz],[x,y,z]] and create a transformation [Rotation,Position]

Parameters

- **var** (*list of* [str](#), [float](#), [Constant](#), [Quantity](#), [Variable](#)) – input list to create a transformation
- **reg** ([Registry](#)) – registry
- **addRegistry** ([bool](#)) – flag to add to registry

`pyg4ometry.gdml.Defines.operationReturnType(name, strExpr, v1, v2, type1, type2, reg)`

class `pyg4ometry.gdml.Defines.DefineBase(name='', registry=None)`

Common bits for a define. Must have a name and a registry. Adding to the registry can't be done here as it must be done by the derived type.

setName(*name*)

Set name of the object.

Parameters

name ([str](#)) – name of object

setRegistry(*registry*)

class pyg4ometry.gdml.Defines.**ScalarBase**(*typeName*, *name*="", *registry*=None)

Bases: *DefineBase*

Base class for all scalars (Constants, Quantity, Variable and 'Expression')

__radd__

__rmul__

setName(*name*)

Set name of scalar

Parameters

name (*str*) – name of object

setExpression(*expressionString*)

Take a string and make it into the BasicExpression type for this object.

Parameters

expressionString (*str*) – Expression to store.

setRegistry(*registry*)

eval()

Evaluate the expression

Returns

numerical evaluation of Constant

Return type

float

__repr__()

Return repr(self).

__float__()

__add__(*other*)

__sub__(*other*)

__rsub__(*other*)

__mul__(*other*)

__truediv__(*other*)

__rtruediv__(*other*)

__neg__()

__abs__()

__pow__(*power*)

pyg4ometry.gdml.Defines.**sin**(*arg*)

Sin of a ScalarBase object, returns a Constant

Parameters

arg (*Constant*, *Quantity*, *Variable* or *Expression*) – Argument of sin

`pyg4ometry.gdml.Defines.cos(arg)`

Cosine of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of cos

`pyg4ometry.gdml.Defines.tan(arg)`

Tangent of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of tan

`pyg4ometry.gdml.Defines.asin(arg)`

ArcSin of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of asin

`pyg4ometry.gdml.Defines.acos(arg)`

ArcCos of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of acos

`pyg4ometry.gdml.Defines.atan(arg)`

ArcTan of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of tan

`pyg4ometry.gdml.Defines.exp(arg)`

Exponential of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of exp

`pyg4ometry.gdml.Defines.log(arg)`

Natural logarithm of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of log

`pyg4ometry.gdml.Defines.log10(arg)`

Base 10 logarithm of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of log10

`pyg4ometry.gdml.Defines.sqrt(arg)`

Square root of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of sin

`pyg4ometry.gdml.Defines.pow(arg, power)`

arg raised to power

Parameters

- **arg** (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of $x^{**}y$
- **power** (`float`) – y

`pyg4ometry.gdml.Defines.abs(arg)`

absolute value of arg

Parameters

arg (*Constant, Quantity, Variable or Expression*) – Argument of abs(arg)

`pyg4ometry.gdml.Defines.min(arg1, arg2)`

absolute value of arg

Parameters

arg (*Constant, Quantity, Variable or Expression*) – Argument of abs(arg)

`pyg4ometry.gdml.Defines.max(arg1, arg2)`

absolute value of arg

Parameters

arg (*Constant, Quantity, Variable or Expression*) – Argument of abs(arg)

class `pyg4ometry.gdml.Defines.Constant(name, value, registry, addRegistry=True)`

Bases: *ScalarBase*

GDML constant define wrapper object

Parameters

- **name** (*str*) – of constant for registry
- **value** (*float, str, Constant, Quantity, Variable*) – expression for constant
- **registry** (*Registry*) – for storing define
- **addRegistry** (*bool*) – add constant to registry

`__eq__(other)`

Return self==value.

`__ne__(other)`

Return self!=value.

`__lt__(other)`

Return self<value.

`__gt__(other)`

Return self>value.

`__le__(other)`

Return self<=value.

`__ge__(other)`

Return self>=value.

class `pyg4ometry.gdml.Defines.Quantity(name, value, unit, type, registry, addRegistry=True)`

Bases: *ScalarBase*

GDML quantity define wrapper object

Parameters

- **name** (*str*) – of constant for registry
- **value** (*float, str, Constant, Quantity, Variable*) – expression for constant
- **unit** (*str*) – unit of the quantity

- **type** (*not sure*) – type of quantity
- **registry** ([Registry](#)) – for storing define
- **addRegistry** (*bool*) – add constant to registry

__repr__()

Return repr(self).

eval()

Evaluate the expression

Returns

numerical evaluation of Constant

Return type

[float](#)

class pyg4ometry.gdml.Defines.**Variable**(*name, value, registry, addRegistry=True*)

Bases: [ScalarBase](#)

GDML variable define wrapper object

Parameters

- **name** ([str](#)) – of constant for registry
- **value** ([float](#), [str](#), [Constant](#), [Quantity](#), [Variable](#)) – expression for constant
- **registry** ([Registry](#)) – for storing define

class pyg4ometry.gdml.Defines.**Expression**(*name, value, registry, addRegistry=False*)

Bases: [ScalarBase](#)

General expression, does not have an analogue in GDML

Parameters

- **name** ([str](#)) – of constant for registry
- **value** ([float](#), [str](#), [Constant](#), [Quantity](#), [Variable](#)) – expression for constant
- **registry** ([Registry](#)) – for storing define
- **addRegistry** (*bool*) – add constant to registry

__int__()

class pyg4ometry.gdml.Defines.**VectorBase**(*typeName, name, registry*)

__rmul__

__repr__()

Return repr(self).

__add__(*other*)

__sub__(*other*)

__mul__(*other*)

__truediv__(*other*)

setName(*name*)

Set name of vector

Parameters

name (*str*) – name of object

eval()

Evaluate vector

Returns

numerical evaluation of vector

Return type

list of floats

nonzero()

Evaluate vector

Returns

Check if the vector is trivial (all elements zero)

Return type

bool

__getitem__(*key*)

setRegistry(*registry*)

class pyg4ometry.gdml.Defines.**Position**(*name, x, y, z, unit='mm', registry=None, addRegistry=True*)

Bases: [VectorBase](#)

GDML position define wrapper object

Parameters

- **x** (*float, Constant, Quantity, Variable*) – x component of position
- **y** (*float, Constant, Quantity, Variable*) – y component of position
- **z** (*float, Constant, Quantity, Variable*) – z component of position

class pyg4ometry.gdml.Defines.**Rotation**(*name, rx, ry, rz, unit='rad', registry=None, addRegistry=True*)

Bases: [VectorBase](#)

GDML rotation define wrapper object

Parameters

- **rx** (*float, Constant, Quantity, Variable*) – rotation around x axis
- **ry** (*float, Constant, Quantity, Variable*) – rotation around y axis
- **rz** (*float, Constant, Quantity, Variable*) – rotation around z axis

class pyg4ometry.gdml.Defines.**Scale**(*name, sx, sy, sz, unit=None, registry=None, addRegistry=True*)

Bases: [VectorBase](#)

GDML scale define wrapper object

Parameters

- **sx** (*float, Constant, Quantity, Variable*) – x component of scale
- **sy** (*float, Constant, Quantity, Variable*) – y component of scale
- **sz** (*float, Constant, Quantity, Variable*) – z component of scale

class pyg4ometry.gdml.Defines.**Matrix**(*name, coldim, values, registry=None, addRegistry=True*)

GDML matrix define wrapper object

Parameters

- **name** (*str*) – of matrix for registry
- **coldim** – is number of columns
- **coldim** – int
- **values** (*list of float, str, Constant, Quantity, Variable*) – list of values for matrix
- **registry** (*Registry*) – for storing define
- **addRegistry** (*bool*) – add matrix to registry

eval()

Evaluate matrix

Returns

numerical evaluation of matrix

Return type

numpy.array

__repr__()

Return repr(self).

__getitem__(*key*)

pyg4ometry.gdml.Defines.**MatrixFromVectors**(*e, v, name, registry, eunit='eV', vunit=''*)

Creates a GDML Matrix from an energy and a value vector

Parameters

- **name** (*str*) – of matrix of registry
- **e** (*list or numpy.array - shape (1,)*) – energy list/vector in units of eunit
- **v** (*list or numpy.array - shape (1,)*) – value list/vector in units of vunit
- **registry** (*Registry*) – for storing define
- **eunit** (*str*) – unit for the energy vector (default: eV)
- **vunit** (*str*) – unit for the value vector (default: unitless)

class pyg4ometry.gdml.Defines.**Auxiliary**(*auxtype, auxvalue, registry=None, unit="", addRegistry=True*)

Auxiliary information container object

addSubAuxiliary(*aux*)

Add a sub-auxiliary inside the scope of the current auxiliary

Parameters

aux (*object, gdml.Defines.Auxiliary*) – auxiliary definition

pyg4ometry.gdml.Reader

Module Contents

Classes

<i>Reader</i>	Read a GDML file.
---------------	-------------------

Functions

<i>isComment</i> (node)
<i>_StripPointer</i> (name)

pyg4ometry.gdml.Reader.**isComment**(node)

class pyg4ometry.gdml.Reader.**Reader**(*fileName*, *registryOn=True*, *skipMaterials=False*,
reduceNISTMaterialsToPredefined=False)

Read a GDML file.

Parameters

- **fileName** (*str*) – path to gdml file to load
- **registryOn** (*bool*) – whether to build a registry
- **reduceNISTMaterialsToPredefined** (*bool*) – change NIST-named materials to predefined ones

When loading a GDML file that was exported by Geant4, the NIST materials may be fully expanded to include their full element / isotope composition. With the `reduceNISTMaterialsToPredefined` flag set to `True`, these will be ignored and the materials that have a name that matches a NIST one will be ‘reduced’ back to a predefined material by name only.

load()

getRegistry()

parseDefines(*xmldoc*)

parseVector(*node*, *type='position'*, *addRegistry=True*)

parseMaterials(*xmldoc*)

_makeMaterials(*materials*, *elements*, *isotopes*)

Construct the isotopes, elements and materials in that order. These aren’t returned, but simply constructed in and therefore exist in the registry.

parseUserInfo(*xmldoc*)

_parseAuxiliary(*xmlnode*, *register=True*)

`parseSolids(xmldoc)`
`parseBox(node)`
`parseTube(node)`
`parseCutTube(node)`
`parseCone(node)`
`parsePara(node)`
`parseTrd(node)`
`parseTrap(node)`
`parseSphere(node)`
`parseOrb(node)`
`parseTorus(node)`
`parsePolycone(node)`
`parseGenericPolycone(node)`
`parsePolyhedra(node)`
`parseGenericPolyhedra(node)`
`parseEllipticalTube(node)`
`parseEllipsoid(node)`
`parseEllipticalCone(node)`
`parseParaboloid(node)`
`parseHype(node)`
`parseTet(node)`
`parseExtrudedSolid(node)`
`parseTwistedBox(node)`
`parseTwistedTrap(node)`
`parseTwistedTrd(node)`
`parseTwistedTubs(node)`
`parseGenericTrap(node)`
`parseTessellatedSolid(node)`
`parseUnion(node)`
`parseSubtraction(node)`
`parseIntersection(node)`

```

parseMultiUnion(node)
parseOpticalSurface(node)
parseScaledSolid(node)
parseSolidLoop(node)
parseStructure(xmldoc, materialSubstitutionNames=None)
extractStructureNodeData(node, materialSubstitutionNames=None)
parsePhysicalVolumeChildren(node, vol)

```

```
pyg4ometry.gdml.Reader._StripPointer(name)
```

```
pyg4ometry.gdml.Units
```

Module Contents

Functions

```
unit(unitString)
```

Attributes

```
units
```

```
pyg4ometry.gdml.Units.units
```

```
pyg4ometry.gdml.Units.unit(unitString)
```

```
pyg4ometry.gdml.Writer
```

Module Contents

Classes

```
Writer
```

```
class pyg4ometry.gdml.Writer.Writer(prepend="")
```

```
    addDetector(registry)
```

```
write(filename)

writeGMADTesterNoBeamline(gmad, gdml)

writeGmadTester(filenameGmad, filenameGDML, writeDefaultLattice=False, preprocessGDML=True,
                 energy=250)

writeDefaultLattice(filename='lattice.gmad')

checkDefineName(defineName)

checkMaterialName(materialName)

checkSolidName(solidName)

checkLogicalVolumeName(logicalVolumeName)

checkPhysicalVolumeName(physicalVolumeName)

writeVectorVariable(node, vector_var, allow_ref=True, suppress_trivial=True)
    Writes an XML child node for a vector variable - position, rotation, scale. If allow_ref is enabled, it will
    write a ref to a registry define where possible. If suppress_trivial is enabled it won't write vectors with all
    elements zero.

writeDefine(define)

writeMaterialProps(material, oe)

writeMaterial(material)

writeLogicalVolume(lv)

writeAuxiliary(aux, parent=None)

writeAssemblyVolume(lv)

writePhysicalVolume(pv)

writeReplicaVolume(instance)

writeDivisionVolume(instance)

writeParametrisedVolume(instance)

writeSkinSurface(instance)

writeBorderSurface(instance)

writeSolid(solid)
    Dispatch to correct member function based on type string in SolidBase.

getValueOrExpr(var)

getValueOrExprFromInstance(instance, variable, index=None)

writeBox(instance)

writeCons(instance)

writeCutTubs(instance)
```

writeEllipsoid(*instance*)

writeEllipticalCone(*instance*)

writeEllipticalTube(*instance*)

createTwoDimVertex(*x*, *y*)

createSection(*zOrder*, *zPosition*, *xOffset*, *yOffset*, *scalingFactor*)

writeExtrudedSolid(*instance*)

createrzPoint(*r*, *z*)

writeGenericPolycone(*instance*)

writeGenericPolyhedra(*instance*)

createTriangularFacet(*vertex1*, *vertex2*, *vertex3*)

createQuadrangularFacet(*vertex1*, *vertex2*, *vertex3*, *vertex4*)

writeTessellatedSolid(*instance*)

writeHype(*instance*)

writeIntersection(*instance*)

writeOpticalSurface(*instance*)

writeOrb(*instance*)

writePara(*instance*)

writeParaboloid(*instance*)

createzPlane(*rInner*, *rOuter*, *zplane*)

writePolycone(*instance*)

writePolyhedra(*instance*)

writeSphere(*instance*)

writeGenericTrap(*instance*)

createPosition(*name*, *x*, *y*, *z*)

writeTet(*instance*)

writeTorus(*instance*)

writeTrap(*instance*)

writeTrd(*instance*)

writeTubs(*instance*)

writeTwistedBox(*instance*)

writeTwistedTrd(*instance*)

```
writeTwistedTrap(instance)
writeTwistedTubs(instance)
writeUnion(instance)
writeSubtraction(instance)
writeIntersection(instance)
writeMultiUnion(instance)
writeScaled(instance)
```

Package Contents

Classes

<i>Reader</i>	
<i>_Material</i>	This class provides an interface to GDML material definitions.
<i>_Element</i>	This class provides an interface to GDML material definitions. Because of the different options
<i>_Isotope</i>	This class that handles isotopes as components of composite materials. An element can be
<i>Writer</i>	
<i>BasicExpression</i>	Holds an expression as a string and can use the expression parser
<i>DefineBase</i>	Common bits for a define. Must have a name and a registry. Adding
<i>ScalarBase</i>	Base class for all scalars (Constants, Quantity, Variable and 'Expression')
<i>Constant</i>	GDML constant define wrapper object
<i>Quantity</i>	GDML quantity define wrapper object
<i>Variable</i>	GDML variable define wrapper object
<i>Expression</i>	General expression, does not have an analogue in GDML
<i>VectorBase</i>	
<i>Position</i>	GDML position define wrapper object
<i>Rotation</i>	GDML rotation define wrapper object
<i>Scale</i>	GDML scale define wrapper object
<i>Matrix</i>	GDML matrix define wrapper object
<i>Auxiliary</i>	Auxiliary information container object

Functions

<code>isComment(node)</code>	
<code>_StripPointer(name)</code>	
<code>upgradeToStringExpression(reg, obj)</code> <code>evaluateToFloat(reg, obj)</code>	Take a float, str, ScalarBase and return string expression.
<code>upgradeToExpression(reg, obj)</code>	Helper functions that takes a string and returns an expression object or a string
<code>upgradeToVector(var, reg[, type, unit, addRegistry])</code>	Take a list [x,y,z] and create a vector
<code>upgradeToTransformation(var, reg[, addRegistry])</code>	Take a list of lists [[rx,ry,rz],[x,y,z]] and create a transformation [Rotation,Position]
<code>operationReturnType(name, strExpr, v1, v2, type1, ...)</code>	
<code>sin(arg)</code>	Sin of a ScalarBase object, returns a Constant
<code>cos(arg)</code>	Cosine of a ScalarBase object, returns a Constant
<code>tan(arg)</code>	Tangent of a ScalarBase object, returns a Constant
<code>asin(arg)</code>	ArcSin of a ScalarBase object, returns a Constant
<code>acos(arg)</code>	ArcCos of a ScalarBase object, returns a Constant
<code>atan(arg)</code>	ArcTan of a ScalarBase object, returns a Constant
<code>exp(arg)</code>	Exponential of a ScalarBase object, returns a Constant
<code>log(arg)</code>	Natural logarithm of a ScalarBase object, returns a Constant
<code>log10(arg)</code>	Base 10 logarithm of a ScalarBase object, returns a Constant
<code>sqrt(arg)</code>	Square root of a ScalarBase object, returns a Constant
<code>pow(arg, power)</code>	arg raised to power
<code>abs(arg)</code>	absolute value of arg
<code>min(arg1, arg2)</code>	absolute value of arg
<code>max(arg1, arg2)</code>	absolute value of arg
<code>MatrixFromVectors(e, v, name, registry[, eunit, vunit])</code>	Creates a GDML Matrix from an energy and a value vector

`pyg4ometry.gdml.isComment(node)`

class `pyg4ometry.gdml.Reader`(*fileName*, *registryOn=True*, *skipMaterials=False*,
reduceNISTMaterialsToPredefined=False)

Read a GDML file.

Parameters

- **fileName** (*str*) – path to gdml file to load
- **registryOn** (*bool*) – whether to build a registry
- **reduceNISTMaterialsToPredefined** (*bool*) – change NIST-named materials to predefined ones

When loading a GDML file that was exported by Geant4, the NIST materials may be fully expanded to include their full element / isotope composition. With the `reduceNISTMaterialsToPredefined` flag set to `True`, these will be ignored and the materials that have a name that matches a NIST one will be ‘reduced’ back to a predefined material by name only.

load()

getRegistry()

parseDefines(*xmldoc*)

parseVector(*node*, *type*='position', *addRegistry*=True)

parseMaterials(*xmldoc*)

_makeMaterials(*materials*, *elements*, *isotopes*)

Construct the isotopes, elements and materials in that order. These aren't returned, but simply constructed in and therefore exist in the registry.

parseUserInfo(*xmldoc*)

_parseAuxiliary(*xmlnode*, *register*=True)

parseSolids(*xmldoc*)

parseBox(*node*)

parseTube(*node*)

parseCutTube(*node*)

parseCone(*node*)

parsePara(*node*)

parseTrd(*node*)

parseTrap(*node*)

parseSphere(*node*)

parseOrb(*node*)

parseTorus(*node*)

parsePolycone(*node*)

parseGenericPolycone(*node*)

parsePolyhedra(*node*)

parseGenericPolyhedra(*node*)

parseEllipticalTube(*node*)

parseEllipsoid(*node*)

parseEllipticalCone(*node*)

parseParaboloid(*node*)

parseHype(*node*)

parseTet(*node*)

parseExtrudedSolid(*node*)


```

parseTwistedBox(node)
parseTwistedTrap(node)
parseTwistedTrd(node)
parseTwistedTubs(node)
parseGenericTrap(node)
parseTessellatedSolid(node)
parseUnion(node)
parseSubtraction(node)
parseIntersection(node)
parseMultiUnion(node)
parseOpticalSurface(node)
parseScaledSolid(node)
parseSolidLoop(node)
parseStructure(xmldoc, materialSubstitutionNames=None)
extractStructureNodeData(node, materialSubstitutionNames=None)
parsePhysicalVolumeChildren(node, vol)

```

```
pyg4ometry.gdml._StripPointer(name)
```

```
class pyg4ometry.gdml._Material(**kwargs)
```

Bases: MaterialBase

This class provides an interface to GDML material definitions.

Because of the different options for constructing a material instance the constructor is kwarg only. Proxy methods are provided to instantiate particular types of material. Those proxy methods are:

MaterialSingleElement MaterialCompound MaterialPredefined

It is possible to instantiate a material directly through kwargs. The possible kwargs are (but note some are mutually exclusive): name - string density - float atomic_number - int atomic_weight - float number_of_components - int state - string pressure - float pressure_unit - string temperature - float temperature_unit - string

property state_variables

```
add_element_massfraction(element, massfraction)
```

Add an element as a component to a material as a fraction of the material mass. Can only add elements to materials defined as composite.

Inputs:

element - pyg4ometry.geant4.Material.Element instance massfraction - float, 0.0 < massfraction <= 1.0

add_element_natoms(*element*, *natoms*)

Add an element as a component to a material as a number of atoms in the material molecule. Can only add elements to materials defined as composite.

Inputs:

element - pyg4ometry.geant4.Material.Element instance *natoms* - int, number of atoms in the compound molecule

add_material(*material*, *fractionmass*)

Add a material as a component to another material (mixture) as a fraction of the mixture mass. Can only add new materials to materials defined as composite.

Inputs:

material - pyg4ometry.geant4.Material.Material instance *massfraction* - float, 0.0 < massfraction <= 1.0

set_pressure(*value*, *unit*='pascal')

set_temperature(*value*, *unit*='K')

__str__()

Return str(self).

addProperty(*name*, *matrix*)

Add a material property from a matrix.

Parameters

- **name** (*str*) – key of the material property
- **matrix** (*Matrix*) – matrix defining the value(s) of the property

addVecProperty(*name*, *e*, *v*, *eunit*='eV', *vunit*='')

Add a property from an energy and a value vector to this object.

Parameters

- **name** (*str*) – key of property
- **e** (*list* or *numpy.array* - *shape* (1,)) – energy list/vector in units of *eunit*
- **v** (*list* or *numpy.array* - *shape* (1,)) – value list/vector in units of *vunit*
- **eunit** (*str*) – unit for the energy vector (default: eV)
- **vunit** (*str*) – unit for the value vector (default: unitless)

addConstProperty(*name*, *value*, *vunit*='')

Add a constant scalar property to this object.

Parameters

- **name** (*str*) – key of property
- **value** (*str*, *float*, *int*) – constant value for this property
- **vunit** (*str*) – unit for the value vector (default: unitless)

class pyg4ometry.gdml._Element(***kwargs*)

Bases: MaterialBase

This class provides an interface to GDML material definitions. Because of the different options for constructing a material instance the constructor is kwarg only. Proxy methods are provided to instantiate particular types of material. Those proxy methods are:

ElementSimple ElementIsotopeMixture

It is possible to instantiate a material directly through kwargs. The possible kwargs are (but note some are mutually exclusive): name - string symbol - string Z - int A - int n_comp - int

add_isotope(*isotope, abundance*)

Add an isotope as a component to an element as an abundance fraction in the element.

Inputs:

element - pyg4ometry.geant4.Material.Isotope instance abundance - float, $0.0 < \text{abundance} \leq 1.0$

class pyg4ometry.gdml._Isotope(*name, Z, N, a, registry=None*)

Bases: MaterialBase

This class that handles isotopes as components of composite materials. An element can be defined as a mixture of isotopes.

Inputs:

name - string Z - int, atomic number N - int, mass number a - float, molar weight in g/mole

class pyg4ometry.gdml.Writer(*prepend=""*)

addDetector(*registry*)

write(*filename*)

writeGMADTesterNoBeamline(*gmad, gdml*)

writeGmadTester(*filenameGmad, filenameGDML, writeDefaultLattice=False, preprocessGDML=True, energy=250*)

writeDefaultLattice(*filename='lattice.gmad'*)

checkDefineName(*defineName*)

checkMaterialName(*materialName*)

checkSolidName(*solidName*)

checkLogicalVolumeName(*logicalVolumeName*)

checkPhysicalVolumeName(*physicalVolumeName*)

writeVectorVariable(*node, vector_var, allow_ref=True, suppress_trivial=True*)

Writes an XML child node for a vector variable - position, rotation, scale. If allow_ref is enabled, it will write a ref to a registry define where possible. If suppress_trivial is enabled it won't write vectors with all elements zero.

writeDefine(*define*)

writeMaterialProps(*material, oe*)

writeMaterial(*material*)

writeLogicalVolume(*lv*)

writeAuxiliary(*aux, parent=None*)

writeAssemblyVolume(*lv*)

writePhysicalVolume(*pv*)

writeReplicaVolume(*instance*)

writeDivisionVolume(*instance*)

writeParametrisedVolume(*instance*)

writeSkinSurface(*instance*)

writeBorderSurface(*instance*)

writeSolid(*solid*)

Dispatch to correct member function based on type string in SolidBase.

getValueOrExpr(*var*)

getValueOrExprFromInstance(*instance*, *variable*, *index=None*)

writeBox(*instance*)

writeCons(*instance*)

writeCutTubs(*instance*)

writeEllipsoid(*instance*)

writeEllipticalCone(*instance*)

writeEllipticalTube(*instance*)

createTwoDimVertex(*x*, *y*)

createSection(*zOrder*, *zPosition*, *xOffset*, *yOffset*, *scalingFactor*)

writeExtrudedSolid(*instance*)

create rzPoint(*r*, *z*)

writeGenericPolycone(*instance*)

writeGenericPolyhedra(*instance*)

createTriangularFacet(*vertex1*, *vertex2*, *vertex3*)

createQuadrangularFacet(*vertex1*, *vertex2*, *vertex3*, *vertex4*)

writeTessellatedSolid(*instance*)

writeHype(*instance*)

writeIntersection(*instance*)

writeOpticalSurface(*instance*)

writeOrb(*instance*)

writePara(*instance*)

writeParaboloid(*instance*)

create rzPlane(*rInner*, *rOuter*, *zplane*)

```

writePolycone(instance)
writePolyhedra(instance)
writeSphere(instance)
writeGenericTrap(instance)
createPosition(name, x, y, z)
writeTet(instance)
writeTorus(instance)
writeTrap(instance)
writeTrd(instance)
writeTubs(instance)
writeTwistedBox(instance)
writeTwistedTrd(instance)
writeTwistedTrap(instance)
writeTwistedTubs(instance)
writeUnion(instance)
writeSubtraction(instance)
writeIntersection(instance)
writeMultiUnion(instance)
writeScaled(instance)

```

```
class pyg4ometry.gdml.BasicExpression(name, expressionString, registry)
```

Holds an expression as a string and can use the expression parser in the supplied registry to evaluate it. A registry is required.

Parameters

- **name** (*str*) – Name of the expression object
- **expressionString** (*str*) – Expression itself as a string e.g. “12.0” or “a + 3.0”
- **registry** (*pyg4ometry.geant4.Registry.Registry*) – The registry object to give context for any variables used.

```

>>> r = pyg4ometry.geant4.Registry()
>>> a = BasicExpression("a", "3.0", r)
>>> float(a)
>>> str(a)

```

```
eval()
```

```
variables(allDependents=False)
```

```
simp()
```

__repr__()

Return repr(self).

__float__()

str()

`pyg4ometry.gdml.upgradeToStringExpression(reg, obj)`

Take a float, str, ScalarBase and return string expression.

Parameters

- **reg** ([Registry](#)) – Registry for lookup in define dictionary
- **obj** ([str](#), [float](#), [ScalarBase](#)) – Object to upgrade

Returns

String expression

Return type

[str](#)

`pyg4ometry.gdml.evaluateToFloat(reg, obj)`

`pyg4ometry.gdml.upgradeToExpression(reg, obj)`

Helper functions that takes a string and returns an expression object or a string

`pyg4ometry.gdml.upgradeToVector(var, reg, type='position', unit='', addRegistry=False)`

Take a list [x,y,z] and create a vector

Parameters

- **var** (*list of* [str](#), [float](#), [Constant](#), [Quantity](#), [Variable](#)) – input list to create a position, rotation or scale
- **reg** ([Registry](#)) – registry
- **type** ([str](#)) – class type of vector (position, rotation, scale)
- **addRegistry** ([bool](#)) – flag to add to registry

`pyg4ometry.gdml.upgradeToTransformation(var, reg, addRegistry=False)`

Take a list of lists [[rx,ry,rz],[x,y,z]] and create a transformation [Rotation,Position]

Parameters

- **var** (*list of* [str](#), [float](#), [Constant](#), [Quantity](#), [Variable](#)) – input list to create a transformation
- **reg** ([Registry](#)) – registry
- **addRegistry** ([bool](#)) – flag to add to registry

`pyg4ometry.gdml.operationReturnType(name, strExpr, v1, v2, type1, type2, reg)`

class `pyg4ometry.gdml.DefineBase(name="", registry=None)`

Common bits for a define. Must have a name and a registry. Adding to the registry can't be done here as it must be done by the derived type.

setName(*name*)

Set name of the object.

Parameters

name ([str](#)) – name of object

setRegistry(*registry*)

class pyg4ometry.gdml.**ScalarBase**(*typeName*, *name*="", *registry*=None)

Bases: *DefineBase*

Base class for all scalars (Constants, Quantity, Variable and 'Expression')

__radd__

__rmul__

setName(*name*)

Set name of scalar

Parameters

name (*str*) – name of object

setExpression(*expressionString*)

Take a string and make it into the BasicExpression type for this object.

Parameters

expressionString (*str*) – Expression to store.

setRegistry(*registry*)

eval()

Evaluate the expression

Returns

numerical evaluation of Constant

Return type

float

__repr__()

Return repr(self).

__float__()

__add__(*other*)

__sub__(*other*)

__rsub__(*other*)

__mul__(*other*)

__truediv__(*other*)

__rtruediv__(*other*)

__neg__()

__abs__()

__pow__(*power*)

pyg4ometry.gdml.**sin**(*arg*)

Sin of a ScalarBase object, returns a Constant

Parameters

arg (*Constant*, *Quantity*, *Variable* or *Expression*) – Argument of sin

`pyg4ometry.gdml.cos(arg)`

Cosine of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of cos

`pyg4ometry.gdml.tan(arg)`

Tangent of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of tan

`pyg4ometry.gdml.asin(arg)`

ArcSin of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of asin

`pyg4ometry.gdml.acos(arg)`

ArcCos of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of acos

`pyg4ometry.gdml.atan(arg)`

ArcTan of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of tan

`pyg4ometry.gdml.exp(arg)`

Exponential of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of exp

`pyg4ometry.gdml.log(arg)`

Natural logarithm of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of log

`pyg4ometry.gdml.log10(arg)`

Base 10 logarithm of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of log10

`pyg4ometry.gdml.sqrt(arg)`

Square root of a `ScalarBase` object, returns a `Constant`

Parameters

arg (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of sin

`pyg4ometry.gdml.pow(arg, power)`

arg raised to power

Parameters

- **arg** (`Constant`, `Quantity`, `Variable` or `Expression`) – Argument of $x^{**}y$
- **power** (`float`) – y

`pyg4ometry.gdml.abs(arg)`

absolute value of arg

Parameters

arg (*Constant, Quantity, Variable or Expression*) – Argument of abs(arg)

`pyg4ometry.gdml.min(arg1, arg2)`

absolute value of arg

Parameters

arg (*Constant, Quantity, Variable or Expression*) – Argument of abs(arg)

`pyg4ometry.gdml.max(arg1, arg2)`

absolute value of arg

Parameters

arg (*Constant, Quantity, Variable or Expression*) – Argument of abs(arg)

class `pyg4ometry.gdml.Constant(name, value, registry, addRegistry=True)`

Bases: *ScalarBase*

GDML constant define wrapper object

Parameters

- **name** (*str*) – of constant for registry
- **value** (*float, str, Constant, Quantity, Variable*) – expression for constant
- **registry** (*Registry*) – for storing define
- **addRegistry** (*bool*) – add constant to registry

`__eq__(other)`

Return self==value.

`__ne__(other)`

Return self!=value.

`__lt__(other)`

Return self<value.

`__gt__(other)`

Return self>value.

`__le__(other)`

Return self<=value.

`__ge__(other)`

Return self>=value.

class `pyg4ometry.gdml.Quantity(name, value, unit, type, registry, addRegistry=True)`

Bases: *ScalarBase*

GDML quantity define wrapper object

Parameters

- **name** (*str*) – of constant for registry
- **value** (*float, str, Constant, Quantity, Variable*) – expression for constant
- **unit** (*str*) – unit of the quantity

- **type** (*not sure*) – type of quantity
- **registry** ([Registry](#)) – for storing define
- **addRegistry** (*bool*) – add constant to registry

__repr__()

Return repr(self).

eval()

Evaluate the expression

Returns

numerical evaluation of Constant

Return type

[float](#)

class pyg4ometry.gdml.**Variable**(*name, value, registry, addRegistry=True*)

Bases: [ScalarBase](#)

GDML variable define wrapper object

Parameters

- **name** (*str*) – of constant for registry
- **value** (*float, str, Constant, Quantity, Variable*) – expression for constant
- **registry** ([Registry](#)) – for storing define

class pyg4ometry.gdml.**Expression**(*name, value, registry, addRegistry=False*)

Bases: [ScalarBase](#)

General expression, does not have an analogue in GDML

Parameters

- **name** (*str*) – of constant for registry
- **value** (*float, str, Constant, Quantity, Variable*) – expression for constant
- **registry** ([Registry](#)) – for storing define
- **addRegistry** (*bool*) – add constant to registry

__int__()

class pyg4ometry.gdml.**VectorBase**(*typeName, name, registry*)

__rmul__

__repr__()

Return repr(self).

__add__(*other*)

__sub__(*other*)

__mul__(*other*)

__truediv__(*other*)

setName(*name*)

Set name of vector

Parameters

name (*str*) – name of object

eval()

Evaluate vector

Returns

numerical evaluation of vector

Return type

list of floats

nonzero()

Evaluate vector

Returns

Check if the vector is trivial (all elements zero)

Return type

bool

__getitem__(*key*)

setRegistry(*registry*)

class pyg4ometry.gdml.**Position**(*name, x, y, z, unit='mm', registry=None, addRegistry=True*)

Bases: [VectorBase](#)

GDML position define wrapper object

Parameters

- **x** (*float, Constant, Quantity, Variable*) – x component of position
- **y** (*float, Constant, Quantity, Variable*) – y component of position
- **z** (*float, Constant, Quantity, Variable*) – z component of position

class pyg4ometry.gdml.**Rotation**(*name, rx, ry, rz, unit='rad', registry=None, addRegistry=True*)

Bases: [VectorBase](#)

GDML rotation define wrapper object

Parameters

- **rx** (*float, Constant, Quantity, Variable*) – rotation around x axis
- **ry** (*float, Constant, Quantity, Variable*) – rotation around y axis
- **rz** (*float, Constant, Quantity, Variable*) – rotation around z axis

class pyg4ometry.gdml.**Scale**(*name, sx, sy, sz, unit=None, registry=None, addRegistry=True*)

Bases: [VectorBase](#)

GDML scale define wrapper object

Parameters

- **sx** (*float, Constant, Quantity, Variable*) – x component of scale
- **sy** (*float, Constant, Quantity, Variable*) – y component of scale
- **sz** (*float, Constant, Quantity, Variable*) – z component of scale

class pyg4ometry.gdml.**Matrix**(*name, coldim, values, registry=None, addRegistry=True*)

GDML matrix define wrapper object

Parameters

- **name** (*str*) – of matrix for registry
- **coldim** – is number of columns
- **coldim** – int
- **values** (*list of float, str, Constant, Quantity, Variable*) – list of values for matrix
- **registry** (*Registry*) – for storing define
- **addRegistry** (*bool*) – add matrix to registry

eval()

Evaluate matrix

Returns

numerical evaluation of matrix

Return type

numpy.array

__repr__()

Return repr(self).

__getitem__(*key*)

pyg4ometry.gdml.MatrixFromVectors(*e, v, name, registry, eunit='eV', vunit=''*)

Creates a GDML Matrix from an energy and a value vector

Parameters

- **name** (*str*) – of matrix of registry
- **e** (*list or numpy.array - shape (1,)*) – energy list/vector in units of eunit
- **v** (*list or numpy.array - shape (1,)*) – value list/vector in units of vunit
- **registry** (*Registry*) – for storing define
- **eunit** (*str*) – unit for the energy vector (default: eV)
- **vunit** (*str*) – unit for the value vector (default: unitless)

class pyg4ometry.gdml.**Auxiliary**(*auxtype, auxvalue, registry=None, unit="", addRegistry=True*)

Auxiliary information container object

addSubAuxiliary(*aux*)

Add a sub-auxiliary inside the scope of the current auxiliary

Parameters

aux (*object, gdml.Defines.Auxiliary*) – auxiliary definition

pyg4ometry.geant4

Geant4 classes. The classes mainly match those of Geant4

Subpackages

pyg4ometry.geant4.solid

Submodules

pyg4ometry.geant4.solid.Box

Module Contents

Classes

<i>Box</i>	Constructs a box. Note the lengths are full lengths and not half lengths as in Geant4
------------	---

Functions

<i>cubeNet</i> (vecList)

pyg4ometry.geant4.solid.Box.**cubeNet**(*vecList*)

class pyg4ometry.geant4.solid.Box.**Box**(*name*, *pX*, *pY*, *pZ*, *registry*, *lunit*='mm', *addRegistry*=True)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a box. Note the lengths are full lengths and not half lengths as in Geant4

Parameters

- **name** (*str*) – of solid for registry
- **pX** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along x
- **pY** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along y
- **pZ** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along z
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

pyg4ometry.geant4.solid.Cons

Module Contents

Classes

<i>Cons</i>	Constructs a conical section.
-------------	-------------------------------

```
class pyg4ometry.geant4.solid.Cons.Cons(name, pRmin1, pRmax1, pRmin2, pRmax2, pDz, pSPhi, pDPhi,
                                         registry, lunit='mm', aunit='rad', nslice=None,
                                         addRegistry=True)
```

Bases: [*pyg4ometry.geant4.solid.SolidBase.SolidBase*](#)

Constructs a conical section.

Parameters

- **name** (*str*) – of the solid
- **pRMin1** (*float*, *Constant*, *Quantity*, *Variable*) – inner radius at -pDz/2
- **pRMax1** (*float*, *Constant*, *Quantity*, *Variable*) – outer radius at -pDz/2
- **pRMin2** (*float*, *Constant*, *Quantity*, *Variable*) – inner radius at +pDz/2
- **pRMax2** (*float*, *Constant*, *Quantity*, *Variable*) – outer radius at +pDz/2
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*) – length along z
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*) – starting phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*) – angle of segment in radians
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing

checkParameters()

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

pyg4ometry.geant4.solid.CutTubs

Module Contents

Classes

CutTubs

Constructs a cylindrical section with end face cuts. Note pLowNorm and pHighNorm can be

```
class pyg4ometry.geant4.solid.CutTubs.CutTubs(name, pRMin, pRMax, pDz, pSPhi, pDPhi, pLowNorm,
                                             pHighNorm, registry, lunit='mm', aunit='rad',
                                             nslice=None, addRegistry=True)
```

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a cylindrical section with end face cuts. Note pLowNorm and pHighNorm can be lists of floats, Constants, Quantities or Variables.

Parameters

- **name** (*str*) – of solid for registry
- **pRMin** (*float*, *Constant*, *Quantity*, *Variable*) – Inner radius
- **pRMax** (*float*, *Constant*, *Quantity*, *Variable*) – Outer radius
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*) – length along z
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*) – starting phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*) – angle of segment
- **pLowNorm** (*list*) – normal vector of the cut plane at -pDz/2
- **pHighNorm** (*list*) – normal vector of the cut plane at +pDz/2

`__repr__()`

`__str__()`

`mesh()`

pyg4ometry.geant4.solid.Ellipsoid

Module Contents

Classes

Ellipsoid

Constructs an ellipsoid optionally cut by planes perpendicular to the z-axis.

```
class pyg4ometry.geant4.solid.Ellipsoid.Ellipsoid(name, pxSemiAxis, pySemiAxis, pzSemiAxis,
                                                  pzBottomCut, pzTopCut, registry, lunit='mm',
                                                  nslice=None, nstack=None, addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs an ellipsoid optionally cut by planes perpendicular to the z-axis.

Parameters

- **name** (*str*) – of the solid
- **pxSemiAxis** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length of x semi axis
- **pySemiAxis** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length of y semi axis
- **pzSemiAxis** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length of z semi axis
- **pzBottomCut** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – z-position of bottom cut plane
- **pzTopCut** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – z-position of top cut plane
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

`__repr__()`

Return repr(self).

`__str__()`

Return str(self).

`mesh()`

`pyg4ometry.geant4.solid.EllipticalCone`

Module Contents

Classes

<i><code>EllipticalCone</code></i>	Constructs a cone with elliptical cross-section and an
------------------------------------	--

```
class pyg4ometry.geant4.solid.EllipticalCone.EllipticalCone(name, pxSemiAxis, pySemiAxis, zMax,  
                                                         pzTopCut, registry, lunit='mm',  
                                                         nslice=None, nstack=None,  
                                                         addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a cone with elliptical cross-section and an optional cut. Both zMax and pzTopCut are half lengths extending from the centre of the cone, at z=0.

Parameters

- **name** (*str*) – name of the volume
- **pxSemiAxis** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – semiaxis in x at z=0 as a fraction of zMax.
- **pySemiAxis** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – semiaxis in y at z=0 as a fraction of zMax
- **zMax** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length of the cone.
- **pzTopCut** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length of the cut.

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

pyg4ometry.geant4.solid.EllipticalTube

Module Contents

Classes

EllipticalTube

Constructs a tube of elliptical cross-section.

```
class pyg4ometry.geant4.solid.EllipticalTube.EllipticalTube(name, pDx, pDy, pDz, registry,
                                                         lunit='mm', nstack=None,
                                                         nslice=None, addRegistry=True)
```

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a tube of elliptical cross-section.

Parameters

- **name** (*str*) – name of the solid
- **pDx** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in x
- **pDy** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in y
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in z
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

__str__()

mesh()

new meshing based of Tubs meshing

pyg4ometry.geant4.solid.ExtrudedSolid**Module Contents****Classes***ExtrudedSolid*

Construct an extruded solid

```
class pyg4ometry.geant4.solid.ExtrudedSolid.ExtrudedSolid(name, pPolygon, pZslices, registry,  
                                                         lunit='mm', addRegistry=True)
```

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Construct an extruded solid

Parameters

- **name** (*str*) – of solid
- **pPolygon** (*list of lists*) – x-y coordinates of vertices for the polygon.
- **pZslices** (*list of lists*) – z-coordinates of a slice, slice offsets in x-y and scaling
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid

Example: Triangular prism with 2 slices pPoligon = [[x1,y1],[x2,y2],[x3,y3]] - vertices of polygon in clockwise order
zSlices = [[z1,[offsx1, offsy1],scale1],[z2,[offsx2, offsy2],scale2]]

__repr__()

Return repr(self).

__str__()

Return str(self).

polygon_area(vertices)**evaluateParameterWithUnits(varName)****mesh()****pyg4ometry.geant4.solid.GenericPolycone****Module Contents****Classes***GenericPolycone*

Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.

```
class pyg4ometry.geant4.solid.GenericPolycone.GenericPolycone(name, pSPhi, pDPhi, pR, pZ,
                                                                registry, lunit='mm', aunit='rad',
                                                                nslice=None, addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.

Parameters

- **name** (*str*) – of solid
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle phi at start of rotation
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle Phi at end of rotation
- **pR** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – r coordinate
- **pZ** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – z coordinate
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

mesh()

`pyg4ometry.geant4.solid.GenericPolyhedra`

Module Contents

Classes

<code>GenericPolyhedra</code>	Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.
-------------------------------	--

```
class pyg4ometry.geant4.solid.GenericPolyhedra.GenericPolyhedra(name, pSPhi, pDPhi, numSide,
                                                                pR, pZ, registry, lunit='mm',
                                                                aunit='rad', addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.

Parameters

- **name** (*str*) – name
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle Phi at start of rotation
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle Phi at end of rotation
- **numSide** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – number of polygon sides
- **pR** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – r coordinate list
- **pZ** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – z coordinate list

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

mesh()

`pyg4ometry.geant4.solid.GenericTrap`

Module Contents

Classes

GenericTrap

Constructs an arbitrary trapezoid using two quadrilaterals sitting

```
class pyg4ometry.geant4.solid.GenericTrap.GenericTrap(name, v1x, v1y, v2x, v2y, v3x, v3y, v4x, v4y,
                                                    v5x, v5y, v6x, v6y, v7x, v7y, v8x, v8y, dz,
                                                    registry, nstack=20, lunit='mm',
                                                    addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs an arbitrary trapezoid using two quadrilaterals sitting on two parallel planes. Vertices 1-4 define the quadrilateral at -dz and vertices 5-8 define the quadrilateral at +dz. This solid is called Arb8 in GDML notation.

Parameters

- **name** – string, name of the volume
- **v1x** – vertex 1 x position
- **v1y** – vertex 1 y position
- **v2x** – vertex 2 x position
- **v2y** – vertex 2 y position
- **v3x** – vertex 3 x position

- **v3y** – vertex 3 y position
- **v4x** – vertex 4 x position
- **v4y** – vertex 4 y position
- **v5x** – vertex 5 x position
- **v5y** – vertex 5 y position
- **v6x** – vertex 6 x position
- **v6y** – vertex 6 y position
- **v7x** – vertex 7 x position
- **v7y** – vertex 7 y position
- **v8x** – vertex 8 x position
- **v8y** – vertex 8 y position
- **dz** – half length along z
- **registry** ([Registry](#)) – for storing solid

`__repr__()`

`__str__()`

`polygon_area(vertices)`

`get_vertex(index)`

`makeLayers(verts_bot, verts_top)`

`mesh()`

`pyg4ometry.geant4.solid.Hype`

Module Contents

Classes

Hype

Constructs a tube with hyperbolic profile.

```
class pyg4ometry.geant4.solid.Hype.Hype(name, innerRadius, outerRadius, innerStereo, outerStereo, lenZ,
                                         registry, lunit='mm', aunit='rad', nslice=None, nstack=None,
                                         addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a tube with hyperbolic profile.

Parameters

- **name** (*str*) – of solid
- **innerRadius** (*float*, [Constant](#), [Quantity](#), [Variable](#), [Expression](#)) – inner radius

- **outerRadius** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – outer radius
- **innerStereo** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – inner stereo angle
- **outerStereo** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – outer stereo angle
- **lenZ** – length along z
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **ainit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

```
__repr__()  
    Return repr(self).  
  
__str__()  
    Return str(self).  
  
checkParameters()  
  
mesh()
```

`pyg4ometry.geant4.solid.Intersection`

Module Contents

Classes

<i>Intersection</i>	Intersection between two solids
---------------------	---------------------------------

```
class pyg4ometry.geant4.solid.Intersection.Intersection(name, obj1, obj2, tra2, registry,  
                                                    addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Intersection between two solids

Parameters

- **name** (*str*) – of solid
- **obj1** (`pyg4ometry.geant4.solid`) – unrotated, untranslated solid
- **obj2** (`pyg4ometry.geant4.solid`) – solid rotated and translated according to tra
- **tra2** (*list*) – [rot,tra] = [[a,b,g],[dx,dy,dz]]
- **registry** (*Registry*) – for storing solid

```
__repr__()  
    Return repr(self).
```

```

__str__()
    Return str(self).

mesh()

translation()

rotation()

object1()

object2()

```

`pyg4ometry.geant4.solid.Layer`

Module Contents

Classes

Layer

```

class pyg4ometry.geant4.solid.Layer.Layer(p1, p2, p3, p4, z)

    __getitem__(index)

    Rotated(angle)

    __repr__()
        Return repr(self).

```

`pyg4ometry.geant4.solid.MultiUnion`

Module Contents

Classes

MultiUnion

Union between two or more solids.

```

class pyg4ometry.geant4.solid.MultiUnion.MultiUnion(name, objects, transformations, registry,
                                                    addRegistry=True)

```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Union between two or more solids.

Parameters

- **name** (*str*) – of solid
- **objects** – unrotated, untranslated solid objects to form union

- **transformations** – [[rot1,tra1],[rot2,tra2], [rot3,tra3] ..] or [[[a,b,g],[dx,dy,dz]], [[a,b,g],[dx,dy,dz]], [[a,b,g],[dx,dy,dz]], ...]
- **registry** ([Registry](#)) – for storing solid
- **addRegistry** – Add solid to registry

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

`pyg4ometry.geant4.solid.OpticalSurface`

Module Contents

Classes

OpticalSurface

class `pyg4ometry.geant4.solid.OpticalSurface.OpticalSurface`(*name, finish, model, surf_type, value, registry, addRegistry=True*)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

allowed_models = ['glisur', 'unified', 'LUT', 'DAVIS', 'dichroic']

allowed_types = ['dielectric_metal', 'dielectric_dielectric', 'dielectric_LUT', 'dielectric_LUTDAVIS', ...]

allowed_finishes = ['polished', 'polishedfrontpainted', 'polishedbackpainted', 'ground', 'groundfrontpainted', ...]

__repr__()

__str__()

addProperty(*name, matrix*)

Add a property to this surface from a matrix.

Parameters

- **name** (*str*) – key of the surface property
- **matrix** (*Matrix*) – matrix defining the value(s) of the property

addVecProperty(*name, e, v, eunit='eV', vunit=''*)

Add a property from an energy and a value vector to this object.

Parameters

- **name** (*str*) – key of property
- **e** (*list* or *numpy.array* - *shape* (1,)) – energy list/vector in units of eunit

- **v** (*list* or *numpy.array* - *shape* (1,)) – value list/vector in units of *vunit*
- **eunit** (*str*) – unit for the energy vector (default: eV)
- **vunit** (*str*) – unit for the value vector (default: unitless)

addConstProperty(*name*, *value*, *vunit*="")

Add a constant scalar property to this object.

Parameters

- **name** (*str*) – key of property
- **value** (*str*, *float*, *int*) – constant value for this property
- **vunit** (*str*) – unit for the value vector (default: unitless)

`pyg4ometry.geant4.solid.Orb`

Module Contents

Classes

<i>Orb</i>	Constructs a solid sphere.
------------	----------------------------

class `pyg4ometry.geant4.solid.Orb.Orb`(*name*, *pRMax*, *registry*, *lunit*='mm', *nslice*=None, *nstack*=None, *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a solid sphere.

Parameters

- **name** (*str*) – of the sold
- **pRMax** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – outer radius
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

pyg4ometry.geant4.solid.Para

Module Contents

Classes

<i>Para</i>	Constructs a parallelepiped.
-------------	------------------------------

class pyg4ometry.geant4.solid.Para.**Para**(*name*, *pDx*, *pDy*, *pDz*, *pAlpha*, *pTheta*, *pPhi*, *registry*, *lunit*='mm', *aunit*='rad', *addRegistry*=True)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a parallelepiped.

Parameters

- **name** (*str*) – of the volume
- **pX** (*float*, *Constant*, *Quantity*, *Variable*) – length along x
- **pY** (*float*, *Constant*, *Quantity*, *Variable*) – length along y
- **pZ** (*float*, *Constant*, *Quantity*, *Variable*) – length along z
- **pAlpha** (*float*, *Constant*, *Quantity*, *Variable*) – angle formed by the y axis and the plane joining the centres of the faces parallel to the z-x plane at -dy/2 and +dy/2
- **pTheta** (*float*, *Constant*, *Quantity*, *Variable*) – polar angle of the line joining the centres of the faces at -dz/2 and +dz/2 in z
- **pPhi** (*float*, *Constant*, *Quantity*, *Variable*) – azimuthal angle of the line joining the centres of the faces at -dx/2 and +dx/2 in x
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

pyg4ometry.geant4.solid.Paraboloid

Module Contents

Classes

<i>Paraboloid</i>	Constructs a paraboloid with possible cuts along the z axis.
-------------------	--

class pyg4ometry.geant4.solid.Paraboloid.**Paraboloid**(*name, pDz, pR1, pR2, registry, lunit='mm', nslice=16, nstack=8, addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a paraboloid with possible cuts along the z axis.

Parameters

- **name** (*str*) – of solid
- **pDz** (*float, Constant, Quantity, Variable, Expression*) – length along z
- **pR1** (*float, Constant, Quantity, Variable, Expression*) – radius at -Dz/2
- **pR2** (*float, Constant, Quantity, Variable, Expression*) – radius at +Dz/2 (pR2 > pR1)
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

[pyg4ometry.geant4.solid.Plane](#)

Module Contents

Classes

Plane

Constructs a *infinite* plane. Should not be used to construct geant4 geometry.

class pyg4ometry.geant4.solid.Plane.**Plane**(*name, normal, dist, zlength=10000*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a *infinite* plane. Should not be used to construct geant4 geometry.

Parameters

- **name** (*str*) – of object in registry
- **normal** (*tuple*) – normal [x,y,z]
- **dist** (*float*) – distance from origin to plane
- **zlength** (*float*) – large transverse box size to emulate infinite plane

__repr__()

pycsgmesh()

pyg4ometry.geant4.solid.Polycone

Module Contents

Classes

<i>Polycone</i>	Constructs a solid of rotation using an arbitrary 2D surface.
-----------------	---

```
class pyg4ometry.geant4.solid.Polycone.Polycone(name, pSPhi, pDPhi, pZpl, pRMin, pRMax, registry,
                                                lunit='mm', aunit='rad', nslice=None,
                                                addRegistry=True)
```

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a solid of rotation using an arbitrary 2D surface.

Parameters

- **name** (*str*) – of the solid
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – starting rotation angle in radians
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – total rotation angle in radius
- **pZPlns** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – z-positions of planes used
- **pRInr** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – inner radii of surface at each z-plane
- **pROut** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – outer radii of surface at each z-plane
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing

Optional registration as this solid is used as a temporary solid in Polyhedra and needn't be always registered.

`__repr__()`

`__str__()`

`mesh()`

pyg4ometry.geant4.solid.Polyhedra

Module Contents

Classes

<i>Polyhedra</i>	Constructs a polyhedra.
------------------	-------------------------

class pyg4ometry.geant4.solid.Polyhedra.**Polyhedra**(*name*, *pSPhi*, *pDPhi*, *numSide*, *numZPlanes*, *zPlane*, *rInner*, *rOuter*, *registry*, *lunit*='mm', *auunit*='rad', *addRegistry*=True)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a polyhedra.

Parameters

- **name** (*str*) – of solid
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – start phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – delta phi angle
- **numSide** (*int*) – number of sides
- **numZPlanes** (*int*) – number of planes along z
- **zPlane** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – position of z planes
- **rInner** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – tangent distance to inner surface per z plane
- **rOuter** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – tangent distance to outer surface per z plane
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **auunit** (*str*) – angle unit (rad,deg) for solid

`__repr__()`

`__str__()`

`mesh()`

pyg4ometry.geant4.solid.Scaled

Module Contents

Classes

<i>Scaled</i>	Constructs a scaled sold.
---------------	---------------------------

class pyg4ometry.geant4.solid.Scaled.**Scaled**(*name, solid, pX, pY, pZ, registry, addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a scaled sold.

Parameters

- **name** (*str*) – of solid for registry
- **pX** (*float, Constant, Quantity, Variable, Expression*) – scale in x
- **pY** (*float, Constant, Quantity, Variable, Expression*) – scale in y
- **pZ** (*float, Constant, Quantity, Variable, Expression*) – scale in z
- **registry** (*Registry*) – for storing solid

Poram solid

reference for scaling

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

[pyg4ometry.geant4.solid.SolidBase](#)

Module Contents

Classes

SolidBase

Base class for all solids

class pyg4ometry.geant4.solid.SolidBase.**SolidBase**(*name, type, registry=None*)

Base class for all solids

property **name**

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

pyg4ometry.geant4.solid.Sphere

Module Contents

Classes

Sphere

Constructs a section of a spherical shell.

```
class pyg4ometry.geant4.solid.Sphere.Sphere(name, pRmin, pRmax, pSPhi, pDPhi, pSTheta, pDTheta,
                                             registry, lunit='mm', aunit='rad', nslice=None,
                                             nstack=None, addRegistry=True)
```

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a section of a spherical shell.

Parameters

- **name** (*str*) – of object in registry
- **pRmin** (*float*, *Constant*, *Quantity*, *Variable*) – inner radius of the shell
- **pRmax** (*float*, *Constant*, *Quantity*, *Variable*) – outer radius of the shell
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*) – starting phi angle in radians
- **pSTheta** (*float*, *Constant*, *Quantity*, *Variable*) – starting theta angle in radians
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*) – delta phi angle in radians
- **pDTheta** (*float*, *Constant*, *Quantity*, *Variable*) – delta theta angle in radians
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

checkParameters()

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

working off $0 < \phi < 2\pi$ $0 < \theta < \pi$

pyg4ometry.geant4.solid.Subtraction

Module Contents

Classes

<i>Subtraction</i>	Subtraction between two solids
--------------------	--------------------------------

```
class pyg4ometry.geant4.solid.Subtraction.Subtraction(name, obj1, obj2, tra2, registry,
                                                    addRegistry=True)
```

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Subtraction between two solids

Parameters

- **name** (*str*) – of solid
- **obj1** (*pyg4ometry.geant4.solid*) – unrotated, untranslated solid
- **obj2** (*pyg4ometry.geant4.solid*) – solid rotated and translated according to tra2
- **tra2** (*list*) – [rot,tra] = [[a,b,g],[dx,dy,dz]]
- **registry** (*Registry*) – for storing solid

`__repr__()`

`__str__()`

`mesh()`

`translation()`

`rotation()`

`object1()`

`object2()`

pyg4ometry.geant4.solid.TessellatedSolid

Module Contents

Classes

<i>TessellatedSolid</i>	Constructs a tessellated solid
-------------------------	--------------------------------

Functions

<code>createTessellatedSolid(name, polygons, reg)</code>	Args:
--	-------

```
class pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid(name, meshTess, registry,
                                                                meshtype=MeshType.Freecad,
                                                                addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a tessellated solid

Parameters

- **name** (*str*) – of solid
- **mesh** (*Freecad, Gdml or Stl*) – mesh
- **registry** (*Registry*) – for storing solid
- **meshtype** (*MeshType.Freecad*) – type of mesh

```
class MeshType
```

```
    Freecad = 1
```

```
    Gdml = 2
```

```
    Stl = 3
```

```
    __repr__()
```

```
        Return repr(self).
```

```
    __str__()
```

```
        Return str(self).
```

```
    addVertex(vertex)
```

```
    addTriangle(triangle)
```

```
    removeDuplicateVertices()
```

```
    mesh()
```

```
pyg4ometry.geant4.solid.TessellatedSolid.createTessellatedSolid(name, polygons, reg)
```

Args:

name: Name of the tessallated solid polygons: list of polygons (list of points given in clockwise order). All polygons should have the same number of points. reg: registry

Returns: TessellatedSolid

pyg4ometry.geant4.solid.Tet

Module Contents

Classes

<i>Tet</i>	Constructs a tetrahedra.
------------	--------------------------

```
class pyg4ometry.geant4.solid.Tet.Tet(name, anchor, p2, p3, p4, registry, lunit='mm',
                                       degeneracyFlag=False, addRegistry=True)
```

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a tetrahedra.

Parameters

- **name** – of the solid
- **anchor** (*list*) – point 1 (anchor point)
- **p2** (*list*) – point 2
- **p3** (*list*) – point 3
- **p4** (*list*) – point 4
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **degeneracyFlag** – bool, indicates degeneracy of points

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

pyg4ometry.geant4.solid.Torus

Module Contents

Classes

<i>Torus</i>	Constructs a torus.
--------------	---------------------

```
class pyg4ometry.geant4.solid.Torus.Torus(name, pRmin, pRmax, pRtor, pSPhi, pDPhi, registry,
                                           lunit='mm', aunit='rad', nslice=None, nstack=None,
                                           addRegistry=True)
```

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a torus.

Parameters

- **name** (*str*) – string, name of the volume
- **pRmin** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – inner radius
- **pRmax** – outer radius
- **pRtor** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – swept radius of torus
- **pSphi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – start phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – delta phi angle
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

`__repr__()``__str__()``mesh()``pyg4ometry.geant4.solid.Trap`**Module Contents****Classes**

<i>Trap</i>	Constructs a general trapezoid.
-------------	---------------------------------

class `pyg4ometry.geant4.solid.Trap.Trap`(*name*, *pDz*, *pTheta*, *pDPhi*, *pDy1*, *pDx1*, *pDx2*, *pAlp1*, *pDy2*, *pDx3*, *pDx4*, *pAlp2*, *registry*, *lunit*='mm', *aunit*='rad', *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a general trapezoid.

Parameters

- **name** (*str*) – name of the volume
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length along z
- **pTheta** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – polar angle of the line joining the centres of the faces at +/-pDz
- **pPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – azimuthal angle of the line joining the centres of the faces at +/-pDz
- **pDy1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half-length at -pDz

- **pDx1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length along x of the side at $y=-pDy1$
- **pDx2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length along x of the side at $y=+pDy1$
- **pAlp1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle wrt the y axis from the centre of the side (lower endcap)
- **pDy2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half-length at $+pDz$
- **pDx3** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half-length of the side at $y=-pDy2$ of the face at $+pDz$
- **pDx4** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half-length of the side at $y=+pDy2$ of the face at $+pDz$
- **pAlp2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle wrt the y axis from the centre of the side (upper endcap)

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

pyg4ometry.geant4.solid.Trd

Module Contents

Classes

<i>Trd</i>	Constructs a trapezoid.
------------	-------------------------

class pyg4ometry.geant4.solid.Trd.**Trd**(*name*, *pDx1*, *pDx2*, *pDy1*, *pDy2*, *pDz*, *registry*, *lunit='mm'*, *addRegistry=True*)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a trapezoid.

Parameters

- **name** (*str*) – of the solid
- **pDx1** (*float*, *Constant*, *Quantity*, *Variable*) – length along x at the surface positioned at $-dz/2$
- **pDx2** (*float*, *Constant*, *Quantity*, *Variable*) – length along x at the surface positioned at $+dz/2$
- **pDy1** (*float*, *Constant*, *Quantity*, *Variable*) – length along y at the surface positioned at $-dz/2$
- **pDy2** (*float*, *Constant*, *Quantity*, *Variable*) – length along y at the surface positioned at $+dz/2$

- **dz** (*float*, *Constant*, *Quantity*, *Variable*) – length along the z axis
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid

mesh()

`pyg4ometry.geant4.solid.Tubs`

Module Contents

Classes

<i>Tubs</i>	Constructs a cylindrical section.
-------------	-----------------------------------

class `pyg4ometry.geant4.solid.Tubs.Tubs`(*name*, *pRMin*, *pRMax*, *pDz*, *pSPhi*, *pDPhi*, *registry*, *lunit*='mm', *auunit*='rad', *nslice*=None, *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a cylindrical section.

Parameters

- **name** (*str*) – of solid for registry
- **pRMin** (*float*, *Constant*, *Quantity*, *Variable*) – inner radius
- **pRMax** (*float*, *Constant*, *Quantity*, *Variable*) – outer radius
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*) – full length along z
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*) – starting phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*) – angle of segment in phi
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **auunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

pyg4ometry.geant4.solid.TwistedBox

Module Contents

Classes

<i>TwistedBox</i>	Constructs a box that is twisted though angle twisted angle
-------------------	---

```
class pyg4ometry.geant4.solid.TwistedBox.TwistedBox(name, twistedangle, pDx, pDy, pDz, registry,
                                                    lunit='mm', aunit='rad', nstack=None, refine=0,
                                                    addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`, `pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid`

Constructs a box that is twisted though angle twisted angle

Parameters

- **name** (*str*) – of the solid
- **twistedangle** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – twist angle, must be less than $\pi/2$
- **pDx** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in x
- **pDy** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in y
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in z
- **refine** (*int*) – number of steps to iteratively smoothen the mesh by doubling the number of vertices at every step
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nstack** (*int*) – Not written

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

makeLayers(*p1*, *p2*, *p3*, *p4*, *pDz*, *theta*, *nstack*)

mesh_old()

makeLayerEdge(*pDx*, *pDy*, *pTwistedAngle*=0, *nx*=20, *ny*=20)

mesh()

pyg4ometry.geant4.solid.TwistedSolid

Module Contents

Classes

TwistedSolid

class pyg4ometry.geant4.solid.TwistedSolid.**TwistedSolid**

makeFaceFromLayer(*layer*, *reverse=False*)

makeSide(*pal*, *pbl*, *pau*, *pbu*, *zl*, *zu*, *nsl*)

p = point a = first b = second u = upper l = lower

meshFromLayers(*layers*, *nsl*)

pyg4ometry.geant4.solid.TwistedTrap

Module Contents

Classes

TwistedTrap

Constructs a general trapezoid with a twist around one axis.

class pyg4ometry.geant4.solid.TwistedTrap.**TwistedTrap**(*name*, *twistedAngle*, *pDz*, *pTheta*, *pDPhi*,
pDy1, *pDx1*, *pDx2*, *pDy2*, *pDx3*, *pDx4*, *pAlp*,
registry, *lunit='mm'*, *aunit='rad'*,
nstack=None, *addRegistry=True*)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*, *pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid*

Constructs a general trapezoid with a twist around one axis.

Parameters

- **name** (*str*) – of the solid
- **twistedAngle** – angle of twist (<90 deg)
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along z
- **pDx1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along x of the side at y=-pDy1/2
- **pDx2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along x of the side at y=+pDy1/2
- **pTheta** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – polar angle of the line joining the centres of the faces at +/-pDz/2

- **pPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – azimuthal angle of the line joining the centres of the faces at $-/+pDz/2$
- **pDy1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length at $-pDz/2$
- **pDy2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length at $+pDz/2$
- **pDx3** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length of the side at $y=-pDy2$ of the face at $+pDz/2$
- **pDx4** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length of the side at $y=+pDy2$ of the face at $+pDz/2$
- **pAlp** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle wrt the y axi from the centre of the side
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid

checkParameters()

__repr__()

Return repr(self).

__str__()

Return str(self).

makeLayers(*pl1*, *pl2*, *pl3*, *pl4*, *pu1*, *pu2*, *pu3*, *pu4*, *pDz*, *twist*, *theta*, *nsl*)

mesh()

`pyg4ometry.geant4.solid.TwistedTrd`

Module Contents

Classes

TwistedTrd

Constructs a twisted general trapezoid.

class `pyg4ometry.geant4.solid.TwistedTrd.TwistedTrd`(*name*, *twistedangle*, *pDx1*, *pDx2*, *pDy1*, *pDy2*, *pDz*, *registry*, *lunit*='mm', *aunit*='rad', *nstack*=None, *refine*=0, *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`, `pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid`

Constructs a twisted general trapezoid.

Parameters

- **name** (*str*) – of solid
- **twistedangle** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – twist angle, must be less than 0.5π

- **pDx1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in x at surface positioned at -pDz/2
- **pDx2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in x at surface positioned at +pDz/2
- **pDy1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in y at surface positioned at -pDz/2
- **pDy2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in y at surface positioned at +pDz/2
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in z
- **refine** (*int*) – number of steps to iteratively smoothen the mesh by doubling the number of vertices at every step
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

makeLayers(*pl1*, *pl2*, *pl3*, *pl4*, *pu1*, *pu2*, *pu3*, *pu4*, *pDz*, *theta*, *ns1*)

mesh()

pyg4ometry.geant4.solid.TwistedTubs

Module Contents

Classes

TwistedTubs

Creates a twisted tube segment. Note that only 1 constructor is supported.

```
class pyg4ometry.geant4.solid.TwistedTubs.TwistedTubs(name, endinnerrad, endouterrad, zlen, phi,
twistedangle, registry, lunit='mm',
aunit='rad', nslice=None, nstack=None,
addRegistry=True)
```

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Creates a twisted tube segment. Note that only 1 constructor is supported.

Parameters

- **name** (*str*) – of solid

- **endinnerrad** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – inner radius at the end of the segment
- **endinnerrad** – outer radius at the end of the segment
- **zlen** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length of the tube segment
- **twistedangle** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – twist angle
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **ainit** (*str*) – angle unit (rad,deg) for solid
- **nslice** – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

makeLayers(*verts_bot*, *verts_top*)

mesh()

pyg4ometry.geant4.solid.TwoVector

Module Contents

Classes

TwoVector

class pyg4ometry.geant4.solid.TwoVector.**TwoVector**(*xIn*, *yIn*)

Rotated(*angle*)

__repr__()

Return repr(self).

__getitem__(*index*)

__add__(*other*)

__sub__(*other*)

__mul__(*other*)

__rmul__(*other*)

```

__truediv__(other)

__abs__()

dot(other)

cross(other)

```

`pyg4ometry.geant4.solid.Union`

Module Contents

Classes

Union

Union between two solids

class `pyg4ometry.geant4.solid.Union.Union`(*name, obj1, obj2, tra2, registry, addRegistry=True*)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Union between two solids

Parameters

- **name** (*str*) – of solid
- **obj1** (`pyg4ometry.geant4.solid`) – unrotated, untranslated solid
- **obj2** (`pyg4ometry.geant4.solid`) – solid rotated and translated according to *tra2*
- **tra2** (*list*) – [rot,tra] = [[a,b,g],[dx,dy,dz]]
- **registry** (*Registry*) – for storing solid

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

translation()

rotation()

object1()

object2()

pyg4ometry.geant4.solid.Wedge

Module Contents

Classes

<i>Wedge</i>	Constructs a <i>infinite</i> wedge. Should not be used to construct geant4 geometry.
--------------	--

class pyg4ometry.geant4.solid.Wedge.**Wedge**(*name*, *pRMax*=1000, *pSPhi*=0, *pDPhi*=1.5, *halfzlength*=10000, *nslice*=None)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a *infinite* wedge. Should not be used to construct geant4 geometry.

Parameters

- **name** (*str*) – of object in registry
- **normal** (*tuple*) – normal [x,y,z]
- **dist** (*float*) – distance from origin to plane
- **zlength** (*float*) – large transverse box size to emulate infinite plane

`__repr__()`

`pycsgmesh()`

Package Contents

Classes

<i>TwoVector</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Plane</i>	Constructs a <i>infinite</i> plane. Should not be used to construct geant4 geometry.
<i>_SolidBase</i>	Base class for all solids
<i>Wedge</i>	Constructs a <i>infinite</i> wedge. Should not be used to construct geant4 geometry.
<i>_SolidBase</i>	Base class for all solids
<i>Box</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Tubs</i>	
<i>_SolidBase</i>	Base class for all solids
<i>CutTubs</i>	Constructs a cylindrical section with end face cuts. Note <i>pLowNorm</i> and <i>pHighNorm</i> can be
<i>_SolidBase</i>	Base class for all solids

continues on next page

Table 3 – continued from previous page

<i>Sphere</i>	Constructs a section of a spherical shell.
<i>_SolidBase</i>	Base class for all solids
<i>Cons</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Ellipsoid</i>	Constructs an ellipsoid optionally cut by planes perpendicular to the z-axis.
<i>_SolidBase</i>	Base class for all solids
<i>Trd</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Torus</i>	Constructs a torus.
<i>_SolidBase</i>	Base class for all solids
<i>_GenericPolyhedra</i>	Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.
<i>Polycone</i>	Constructs a solid of rotation using an arbitrary 2D surface.
<i>_GenericPolyhedra</i>	Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.
<i>_SolidBase</i>	Base class for all solids
<i>Polyhedra</i>	Constructs a polyhedra.
<i>_SolidBase</i>	Base class for all solids
<i>ExtrudedSolid</i>	Construct an extruded solid
<i>_SolidBase</i>	Base class for all solids
<i>Union</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Intersection</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Subtraction</i>	Subtraction between two solids
<i>_SolidBase</i>	Base class for all solids
<i>OpticalSurface</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Para</i>	Constructs a parallelepiped.
<i>_SolidBase</i>	Base class for all solids
<i>Trap</i>	Constructs a general trapezoid.
<i>_SolidBase</i>	Base class for all solids
<i>Orb</i>	
<i>_SolidBase</i>	Base class for all solids
<i>EllipticalTube</i>	Constructs a tube of elliptical cross-section.
<i>_SolidBase</i>	Base class for all solids
<i>EllipticalCone</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Paraboloid</i>	Constructs a paraboloid with possible cuts along the z axis.
<i>_SolidBase</i>	Base class for all solids
<i>Hype</i>	Constructs a tube with hyperbolic profile.
<i>_SolidBase</i>	Base class for all solids

continues on next page

Table 3 – continued from previous page

<i>Tet</i>	
<i>_SolidBase</i>	Base class for all solids
<i>_TwistedSolid</i>	
<i>_TwoVector</i>	
<i>_Layer</i>	
<i>TwistedBox</i>	
<i>_SolidBase</i>	Base class for all solids
<i>_TwistedSolid</i>	
<i>_TwoVector</i>	
<i>_Layer</i>	
<i>TwistedTrap</i>	
<i>_SolidBase</i>	Base class for all solids
<i>_TwoVector</i>	
<i>_Layer</i>	
<i>_TwistedSolid</i>	
<i>TwistedTrd</i>	
<i>_SolidBase</i>	Base class for all solids
<i>TwistedTubs</i>	
<i>_SolidBase</i>	Base class for all solids
<i>_GenericPolyhedra</i>	Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.
<i>GenericPolycone</i>	
<i>_SolidBase</i>	Base class for all solids
<i>GenericPolyhedra</i>	
<i>_SolidBase</i>	Base class for all solids
<i>GenericTrap</i>	Constructs an arbitrary trapezoid using two quadrilaterals sitting
<i>_SolidBase</i>	Base class for all solids
<i>TessellatedSolid</i>	
<i>_SolidBase</i>	Base class for all solids
<i>MultiUnion</i>	
<i>_SolidBase</i>	Base class for all solids
<i>Scaled</i>	

continues on next page

Table 3 – continued from previous page

<i>SolidBase</i>	Base class for all solids
Functions	
<i>cubeNet</i> (vecList)	
<i>rad2deg</i> (rad)	Convert rad in radians into degrees
<i>deg2rad</i> (deg)	Convert deg in degrees into radians
<i>grad2rad</i> (gradians)	Convert rad in gradians into radians
<i>tbxyz2axisangle</i> (rv)	Tait-Bryan x-y-z rotation to axis-angle representation
<i>matrix2axisangle</i> (matrix)	Convert 3x3 transformation matrix to axis angle representation
<i>axisangle2matrix</i> (axis, angle)	Convert axis angle to transformation matrix
<i>matrix2tbxyz</i> (matrix)	Convert rotation matrix to Tait-Bryan angles.
<i>axisangle2tbxyz</i> (axis, angle)	Convert axis and angle to tait bryan angles
<i>tbxyz2matrix</i> (angles)	Convert tait bryan angles to a single passive rotation matrix.
<i>tbzyx2matrix</i> (angles)	Convert Tait-Bryan angles to a single passive rotation matrix.
<i>matrix_from</i> (v_from, v_to)	Returns the rotation matrix that rotates v_from to parallel to
<i>_rodrigues_anti_parallel</i> (v_from, v_to)	v_from = vector FROM
<i>are_parallel</i> (vector_1, vector_2[, tolerance])	Check if vector vector_1 is parallel to vector vector_2 down to
<i>are_anti_parallel</i> (vector_1, vector_2[, tolerance])	Check if vector vector_1 is parallel to vector vector_2 down to
<i>reverse</i> (angles)	Invert the rotation represented by these angles.
<i>two_fold_orientation</i> (v1, v2, e1, e2)	matrix_from will align one vector with another, but there are
<i>rad2deg</i> (rad)	Convert rad in radians into degrees
<i>deg2rad</i> (deg)	Convert deg in degrees into radians
<i>grad2rad</i> (gradians)	Convert rad in gradians into radians
<i>tbxyz2axisangle</i> (rv)	Tait-Bryan x-y-z rotation to axis-angle representation
<i>matrix2axisangle</i> (matrix)	Convert 3x3 transformation matrix to axis angle representation
<i>axisangle2matrix</i> (axis, angle)	Convert axis angle to transformation matrix
<i>matrix2tbxyz</i> (matrix)	Convert rotation matrix to Tait-Bryan angles.
<i>axisangle2tbxyz</i> (axis, angle)	Convert axis and angle to tait bryan angles
<i>tbxyz2matrix</i> (angles)	Convert tait bryan angles to a single passive rotation matrix.
<i>tbzyx2matrix</i> (angles)	Convert Tait-Bryan angles to a single passive rotation matrix.
<i>matrix_from</i> (v_from, v_to)	Returns the rotation matrix that rotates v_from to parallel to
<i>_rodrigues_anti_parallel</i> (v_from, v_to)	v_from = vector FROM
<i>are_parallel</i> (vector_1, vector_2[, tolerance])	Check if vector vector_1 is parallel to vector vector_2 down to
<i>are_anti_parallel</i> (vector_1, vector_2[, tolerance])	Check if vector vector_1 is parallel to vector vector_2 down to

continues on next page

Table 4 – continued from previous page

<i>reverse</i> (angles)	Invert the rotation represented by these angles.
<i>two_fold_orientation</i> (v1, v2, e1, e2)	matrix_from will align one vector with another, but there are
<i>rad2deg</i> (rad)	Convert rad in radians into degrees
<i>deg2rad</i> (deg)	Convert deg in degrees into radians
<i>grad2rad</i> (gradians)	Convert rad in gradians into radians
<i>tbxyz2axisangle</i> (rv)	Tait-Bryan x-y-z rotation to axis-angle representation
<i>matrix2axisangle</i> (matrix)	Convert 3x3 transformation matrix to axis angle representation
<i>axisangle2matrix</i> (axis, angle)	Convert axis angle to transformation matrix
<i>matrix2tbxyz</i> (matrix)	Convert rotation matrix to Tait-Bryan angles.
<i>axisangle2tbxyz</i> (axis, angle)	Convert axis and angle to tait bryan angles
<i>tbxyz2matrix</i> (angles)	Convert tait bryan angles to a single passive rotation matrix.
<i>tbzyx2matrix</i> (angles)	Convert Tait-Bryan angles to a single passive rotation matrix.
<i>matrix_from</i> (v_from, v_to)	Returns the rotation matrix that rotates v_from to parallel to
<i>_rodrigues_anti_parallel</i> (v_from, v_to)	v_from = vector FROM
<i>are_parallel</i> (vector_1, vector_2[, tolerance])	Check if vector vector_1 is parallel to vector vector_2 down to
<i>are_anti_parallel</i> (vector_1, vector_2[, tolerance])	Check if vector vector_1 is parallel to vector vector_2 down to
<i>reverse</i> (angles)	Invert the rotation represented by these angles.
<i>two_fold_orientation</i> (v1, v2, e1, e2)	matrix_from will align one vector with another, but there are
<i>_cubeNet</i> (vecList)	
<i>_cubeNet</i> (vecList)	
<i>createTessellatedSolid</i> (name, polygons, reg)	Args:
<i>rad2deg</i> (rad)	Convert rad in radians into degrees
<i>deg2rad</i> (deg)	Convert deg in degrees into radians
<i>grad2rad</i> (gradians)	Convert rad in gradians into radians
<i>tbxyz2axisangle</i> (rv)	Tait-Bryan x-y-z rotation to axis-angle representation
<i>matrix2axisangle</i> (matrix)	Convert 3x3 transformation matrix to axis angle representation
<i>axisangle2matrix</i> (axis, angle)	Convert axis angle to transformation matrix
<i>matrix2tbxyz</i> (matrix)	Convert rotation matrix to Tait-Bryan angles.
<i>axisangle2tbxyz</i> (axis, angle)	Convert axis and angle to tait bryan angles
<i>tbxyz2matrix</i> (angles)	Convert tait bryan angles to a single passive rotation matrix.
<i>tbzyx2matrix</i> (angles)	Convert Tait-Bryan angles to a single passive rotation matrix.
<i>matrix_from</i> (v_from, v_to)	Returns the rotation matrix that rotates v_from to parallel to
<i>_rodrigues_anti_parallel</i> (v_from, v_to)	v_from = vector FROM
<i>are_parallel</i> (vector_1, vector_2[, tolerance])	Check if vector vector_1 is parallel to vector vector_2 down to
<i>are_anti_parallel</i> (vector_1, vector_2[, tolerance])	Check if vector vector_1 is parallel to vector vector_2 down to
<i>reverse</i> (angles)	Invert the rotation represented by these angles.

continues on next page

Table 4 – continued from previous page

<code>two_fold_orientation(v1, v2, e1, e2)</code>	matrix_from will align one vector with another, but there are
---	---

class pyg4ometry.geant4.solid.**TwoVector**(*xIn, yIn*)

Rotated(*angle*)

__repr__()

Return repr(self).

__getitem__(*index*)

__add__(*other*)

__sub__(*other*)

__mul__(*other*)

__rmul__(*other*)

__truediv__(*other*)

__abs__()

dot(*other*)

cross(*other*)

class pyg4ometry.geant4.solid.**_SolidBase**(*name, type, registry=None*)

Base class for all solids

property *name*

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.**Plane**(*name, normal, dist, zlength=10000*)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a *infinite* plane. Should not be used to construct geant4 geometry.

Parameters

- **name** (*str*) – of object in registry
- **normal** (*tuple*) – normal [x,y,z]

- **dist** (*float*) – distance from origin to plane
- **zlength** (*float*) – large transverse box size to emulate infinite plane

`__repr__()`

`pycsgmesh()`

class `pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)`

Base class for all solids

property `name`

`evaluateParameter(obj)`

`evaluateParameterWithUnits(varName)`

`_addProperty(attribute)`

`_setProperty(attribute, value)`

`_getProperty(attribute)`

`_twoPiValueCheck(attribute, aunit='rad')`

Raises a `ValueError` if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** $2 \times \pi$.

`conver2Tessellated()`

return a `TessellatedSolid` instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.Wedge(name, pRMax=1000, pSPhi=0, pDPhi=1.5, halfzlength=10000, nslice=None)`

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a *infinite* wedge. Should not be used to construct geant4 geometry.

Parameters

- **name** (*str*) – of object in registry
- **normal** (*tuple*) – normal [x,y,z]
- **dist** (*float*) – distance from origin to plane
- **zlength** (*float*) – large transverse box size to emulate infinite plane

`__repr__()`

`pycsgmesh()`

class `pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)`

Base class for all solids

property `name`

`evaluateParameter(obj)`

`evaluateParameterWithUnits(varName)`

`_addProperty(attribute)`

`_setProperty(attribute, value)`

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit*='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

`pyg4ometry.geant4.solid.cubeNet`(*vecList*)

class `pyg4ometry.geant4.solid.Box`(*name*, *pX*, *pY*, *pZ*, *registry*, *lunit*='mm', *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a box. Note the lengths are full lengths and not half lengths as in Geant4

Parameters

- **name** (*str*) – of solid for registry
- **pX** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along x
- **pY** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along y
- **pZ** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along z
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

class `pyg4ometry.geant4.solid._SolidBase`(*name*, *type*, *registry*=None)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit*='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.Tubs`(*name*, *pRMin*, *pRMax*, *pDz*, *pSPhi*, *pDPhi*, *registry*, *lunit*='mm', *aunit*='rad', *nslice*=None, *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a cylindrical section.

Parameters

- **name** (*str*) – of solid for registry
- **pRMin** (*float*, *Constant*, *Quantity*, *Variable*) – inner radius
- **pRMax** (*float*, *Constant*, *Quantity*, *Variable*) – outer radius
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*) – full length along z
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*) – starting phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*) – angle of segment in phi
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()**class** pyg4ometry.geant4.solid._SolidBase(*name, type, registry=None*)

Base class for all solids

property name**evaluateParameter**(*obj*)**evaluateParameterWithUnits**(*varName*)**_addProperty**(*attribute*)**_setProperty**(*attribute, value*)**_getProperty**(*attribute*)**_twoPiValueCheck**(*attribute, aunit='rad'*)Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.**conver2Tessellated()**

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.CutTubs(*name, pRMin, pRMax, pDz, pSPhi, pDPhi, pLowNorm, pHighNorm, registry, lunit='mm', aunit='rad', nslice=None, addRegistry=True*)Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a cylindrical section with end face cuts. Note pLowNorm and pHighNorm can be lists of floats, Constants, Quantities or Variables.

Parameters

- **name** (*str*) – of solid for registry
- **pRMin** (*float*, *Constant*, *Quantity*, *Variable*) – Inner radius

- **pRMax** (*float*, *Constant*, *Quantity*, *Variable*) – Outer radius
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*) – length along z
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*) – starting phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*) – angle of segment
- **pLowNorm** (*list*) – normal vector of the cut plane at -pDz/2
- **pHighNorm** (*list*) – normal vector of the cut plane at +pDz/2

__repr__()

__str__()

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name*, *type*, *registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.**Sphere**(*name*, *pRmin*, *pRmax*, *pSPhi*, *pDPhi*, *pSTheta*, *pDTheta*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *nstack=None*, *addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a section of a spherical shell.

Parameters

- **name** (*str*) – of object in registry
- **pRmin** (*float*, *Constant*, *Quantity*, *Variable*) – inner radius of the shell
- **pRmax** (*float*, *Constant*, *Quantity*, *Variable*) – outer radius of the shell
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*) – starting phi angle in radians
- **pSTheta** (*float*, *Constant*, *Quantity*, *Variable*) – starting theta angle in radians
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*) – delta phi angle in radians
- **pDTheta** (*float*, *Constant*, *Quantity*, *Variable*) – delta theta angle in radians
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid

- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

checkParameters()

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

working off $0 < \phi < 2\pi$ $0 < \theta < \pi$

class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)

Base class for all solids

property name

evaluateParameter(obj)

evaluateParameterWithUnits(varName)

_addProperty(attribute)

_setProperty(attribute, value)

_getProperty(attribute)

_twoPiValueCheck(attribute, aunit='rad')

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** $2 \times \pi$.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.Cons(name, pRmin1, pRmax1, pRmin2, pRmax2, pDz, pSPhi, pDPhi, registry, lunit='mm', aunit='rad', nslice=None, addRegistry=True)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a conical section.

Parameters

- **name** (*str*) – of the solid
- **pRmin1** (*float*, *Constant*, *Quantity*, *Variable*) – inner radius at $-pDz/2$
- **pRmax1** (*float*, *Constant*, *Quantity*, *Variable*) – outer radius at $-pDz/2$
- **pRmin2** (*float*, *Constant*, *Quantity*, *Variable*) – inner radius at $+pDz/2$
- **pRmax2** (*float*, *Constant*, *Quantity*, *Variable*) – outer radius at $+pDz/2$
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*) – length along z
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*) – starting phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*) – angle of segment in radians
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid

- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing

checkParameters()

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name, type, registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.Ellipsoid(*name, pxSemiAxis, pySemiAxis, pzSemiAxis, pzBottomCut, pzTopCut, registry, lunit='mm', nslice=None, nstack=None, addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs an ellipsoid optionally cut by planes perpendicular to the z-axis.

Parameters

- **name** (*str*) – of the solid
- **pxSemiAxis** (*float, Constant, Quantity, Variable, Expression*) – length of x semi axis
- **pySemiAxis** (*float, Constant, Quantity, Variable, Expression*) – length of y semi axis
- **pzSemiAxis** (*float, Constant, Quantity, Variable, Expression*) – length of z semi axis
- **pzBottomCut** (*float, Constant, Quantity, Variable, Expression*) – z-position of bottom cut plane
- **pzTopCut** (*float, Constant, Quantity, Variable, Expression*) – z-position of top cut plane
- **registry** (*Registry*) – for storing solid

- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name, type, registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.Trd(*name, pDx1, pDx2, pDy1, pDy2, pDz, registry, lunit='mm', addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a trapezoid.

Parameters

- **name** (*str*) – of the solid
- **pDx1** (*float, Constant, Quantity, Variable*) – length along x at the surface positioned at -dz/2
- **pDx2** (*float, Constant, Quantity, Variable*) – length along x at the surface positioned at +dz/2
- **pDy1** (*float, Constant, Quantity, Variable*) – length along y at the surface positioned at -dz/2
- **pDy2** (*float, Constant, Quantity, Variable*) – length along y at the surface positioned at +dz/2
- **dz** (*float, Constant, Quantity, Variable*) – length along the z axis
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid

mesh()

class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)

Base class for all solids

property name

evaluateParameter(obj)

evaluateParameterWithUnits(varName)

_addProperty(attribute)

_setProperty(attribute, value)

_getProperty(attribute)

_twoPiValueCheck(attribute, aunit='rad')

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.Torus(name, pRmin, pRmax, pRtor, pSPhi, pDPhi, registry, lunit='mm',
aunit='rad', nslice=None, nstack=None, addRegistry=True)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a torus.

Parameters

- **name** (*str*) – string, name of the volume
- **pRmin** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – inner radius
- **pRmax** – outer radius
- **pRtor** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – swept radius of torus
- **pSphi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – start phi angle
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – delta phi angle
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

__str__()

mesh()

class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid._GenericPolyhedra`(*name, pSPhi, pDPhi, numSide, pR, pZ, registry, lunit='mm', aunit='rad', addRegistry=True*)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.

Parameters

- **name** (*str*) – name
- **pSPhi** (*float, Constant, Quantity, Variable, Expression*) – angle Phi at start of rotation
- **pDPhi** (*float, Constant, Quantity, Variable, Expression*) – angle Phi at end of rotation
- **numSide** (*float, Constant, Quantity, Variable, Expression*) – number of polygon sides
- **pR** (*list of float, Constant, Quantity, Variable, Expression*) – r coordinate list
- **pZ** (*list of float, Constant, Quantity, Variable, Expression*) – z coordinate list

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

mesh()

class `pyg4ometry.geant4.solid.Polycone`(*name, pSPhi, pDPhi, pZpl, pRMin, pRMax, registry, lunit='mm', aunit='rad', nslice=None, addRegistry=True*)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a solid of rotation using an arbitrary 2D surface.

Parameters

- **name** (*str*) – of the solid

- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – starting rotation angle in radians
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – total rotation angle in radius
- **pZPlns** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – z-positions of planes used
- **pRInr** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – inner radii of surface at each z-plane
- **pROut** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – outer radii of surface at each z-plane
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing

Optional registration as this solid is used as a temporary solid in Polyhedra and needn't be always registered.

__repr__()

__str__()

mesh()

class pyg4ometry.geant4.solid._GenericPolyhedra(*name*, *pSPhi*, *pDPhi*, *numSide*, *pR*, *pZ*, *registry*, *lunit*='mm', *aunit*='rad', *addRegistry*=True)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.

Parameters

- **name** (*str*) – name
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle Phi at start of rotation
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle Phi at end of rotation
- **numSide** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – number of polygon sides
- **pR** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – r coordinate list
- **pZ** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – z coordinate list

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name, type, registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.Polyhedra(*name, pSPhi, pDPhi, numSide, numZPlanes, zPlane, rInner, rOuter, registry, lunit='mm', aunit='rad', addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a polyhedra.

Parameters

- **name** (*str*) – of solid
- **pSPhi** (*float, Constant, Quantity, Variable, Expression*) – start phi angle
- **pDPhi** (*float, Constant, Quantity, Variable, Expression*) – delta phi angle
- **numSide** (*int*) – number of sides
- **numZPlanes** (*int*) – number of planes along z
- **zPlane** (*list of float, Constant, Quantity, Variable, Expression*) – position of z planes
- **rInner** (*list of float, Constant, Quantity, Variable, Expression*) – tangent distance to inner surface per z plane
- **rOuter** (*list of float, Constant, Quantity, Variable, Expression*) – tangent distance to outer surface per z plane
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid

__repr__()

__str__()

mesh()

```
class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)
```

Base class for all solids

property name

evaluateParameter(obj)

evaluateParameterWithUnits(varName)

_addProperty(attribute)

_setProperty(attribute, value)

_getProperty(attribute)

_twoPiValueCheck(attribute, aunit='rad')

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

```
class pyg4ometry.geant4.solid.ExtrudedSolid(name, pPolygon, pZslices, registry, lunit='mm',
                                             addRegistry=True)
```

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Construct an extruded solid

Parameters

- **name** (*str*) – of solid
- **pPolygon** (*list of lists*) – x-y coordinates of vertices for the polygon.
- **pZslices** (*list of lists*) – z-coordinates of a slice, slice offsets in x-y and scaling
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid

Example: Triangular prism with 2 slices pPoligon = [[x1,y1],[x2,y2],[x3,y3]] - vertices of polygon in clockwise order zSlices = [[z1,[offsx1, offsy1],scale1],[z2,[offsx2, offsy2],scale2]]

__repr__()

Return repr(self).

__str__()

Return str(self).

polygon_area(vertices)

evaluateParameterWithUnits(varName)

mesh()

```
class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)
```

Base class for all solids

property name

evaluateParameter(obj)

evaluateParameterWithUnits(varName)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit*='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.Union`(*name*, *obj1*, *obj2*, *tra2*, *registry*, *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Union between two solids

Parameters

- **name** (*str*) – of solid
- **obj1** (`pyg4ometry.geant4.solid`) – unrotated, untranslated solid
- **obj2** (`pyg4ometry.geant4.solid`) – solid rotated and translated according to *tra2*
- **tra2** (*list*) – [rot,tra] = [[a,b,g],[dx,dy,dz]]
- **registry** (`Registry`) – for storing solid

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

translation()

rotation()

object1()

object2()

`pyg4ometry.geant4.solid.rad2deg`(*rad*)

Convert rad in radians into degrees

Parameters

rad (*float*) – Input in radians

Returns

rad in degrees

Return type

float

`pyg4ometry.geant4.solid.deg2rad`(*deg*)

Convert deg in degrees into radians

Parameters

deg (*float*) – Input in degrees

Returns

deg in radians

Return type

float

pyg4ometry.geant4.solid.**grad2rad**(*gradians*)

Convert rad in gradients into radians

Parameters**gradians** (*float*) – Input in gradients**Returns**

gradians in radians

Return type

float

pyg4ometry.geant4.solid.**tbxyz2axisangle**(*rv*)Tait-Bryan x-y-z rotation to axis-angle representation Algorithm from <http://www.sedris.org/wg8home/Documents/WG80485.pdf>

For converting rotation angles to an active axis/angle pair for use in pycsg. Order of rotation: x->y->z.

Parameters**rv** (*float*(3)) – rotation angles**Returns**

[axis,angle]

Return type

list(list(3),float)

pyg4ometry.geant4.solid.**matrix2axisangle**(*matrix*)

Convert 3x3 transformation matrix to axis angle representation

Parameters**matrix** (*array*(3,3)) – 3x3 rotation matrix array**Returns**

[axis,angle]

Return type

list(list(3),float)

pyg4ometry.geant4.solid.**axisangle2matrix**(*axis*, *angle*)

Convert axis angle to transformation matrix

Parameters

- **axis** (*list/array*(3)) – axis for rotation
- **angle** (*float*) – rotation angle

Returns

transformation matrix

Return type

array(3,3)

pyg4ometry.geant4.solid.**matrix2tbxyz**(*matrix*)

Convert rotation matrix to Tait-Bryan angles. Order of rotation is x -> y -> z.

Parameters

matrix – active (positive angle = anti-clockwise rotation about that axis when looking at the axis) matrix.

Returns

[x, y, z] Tait-Bryan angles in a list.

Return type

list(3)

pyg4ometry.geant4.solid.**axisangle2tbxyz**(axis, angle)

Convert axis and angle to tait bryan angles

Parameters

- **axis** (list/array(3)) – axis for rotation
- **angle** (float) – rotation angle

Returns

tait bryan angles (x-y-z)

Return type

array(3)

pyg4ometry.geant4.solid.**tbxyz2matrix**(angles)

Convert tait bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters

angles – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

pyg4ometry.geant4.solid.**tbzyx2matrix**(angles)

Convert Tait-Bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters

angles – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

pyg4ometry.geant4.solid.**matrix_from**(v_from, v_to)

Returns the rotation matrix that rotates v_from to parallel to v_to.

Useful for ensuring a given face points in a certain direction.

v_from and v_to should be array-like three-vectors.

pyg4ometry.geant4.solid.**_rodrigues_anti_parallel**(v_from, v_to)

v_from = vector FROM v_to = vector TO

source: http://en.citizendium.org/wiki/Rotation_matrix#Case_that_.22from.22_and_.22to.22_vectors_are_anti-parallel

`pyg4ometry.geant4.solid.are_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector `vector_1` is parallel to vector `vector_2` down to some tolerance.

Parameters

- **vector_1** (*array*) – First input vector
- **vector_2** (*array*) – Second input vector
- **tolerance** (*float*) – Tolerance for calculation

Returns

if vectors are parallel

Return type

bool

`pyg4ometry.geant4.solid.are_anti_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector `vector_1` is parallel to vector `vector_2` down to some tolerance.

Parameters

- **vector_1** (*array*) – First input vector
- **vector_2** (*array*) – Second input vector
- **tolerance** (*float*) – Tolerance for calculation

Returns

if vectors are antiparallel

Return type

bool

`pyg4ometry.geant4.solid.reverse(angles)`

Invert the rotation represented by these angles.

`pyg4ometry.geant4.solid.two_fold_orientation(v1, v2, e1, e2)`

`matrix_from` will align one vector with another, but there are an infinite number of such matrices that align two vectors. This further constrains the rotation by introducing a second pair of vectors.

`v1` start `v` `v2` end `v` `e1` start `e` `e2` end `v`

class `pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)`

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a `ValueError` if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` over $2 \times \pi$.

conver2Tessellated()

return a `TessellatedSolid` instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.**Intersection**(*name, obj1, obj2, tra2, registry, addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Intersection between two solids

Parameters

- **name** (*str*) – of solid
- **obj1** (*pyg4ometry.geant4.solid*) – unrotated, untranslated solid
- **obj2** (*pyg4ometry.geant4.solid*) – solid rotated and translated according to tra
- **tra2** (*list*) – [rot,tra] = [[a,b,g],[dx,dy,dz]]
- **registry** (*Registry*) – for storing solid

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

translation()

rotation()

object1()

object2()

pyg4ometry.geant4.solid.**rad2deg**(*rad*)

Convert rad in radians into degrees

Parameters

rad (*float*) – Input in radians

Returns

rad in degrees

Return type

float

pyg4ometry.geant4.solid.**deg2rad**(*deg*)

Convert deg in degrees into radians

Parameters

deg (*float*) – Input in degrees

Returns

deg in radians

Return type

float

pyg4ometry.geant4.solid.**grad2rad**(*gradians*)

Convert rad in radians into radians

Parameters

gradians (*float*) – Input in radians

Returns

gradians in radians

Return type

float

`pyg4ometry.geant4.solid.tbxyz2axisangle(rv)`

Tait-Bryan x-y-z rotation to axis-angle representation Algorithm from <http://www.sedris.org/wg8home/Documents/WG80485.pdf>

For converting rotation angles to an active axis/angle pair for use in pycsg. Order of rotation: x->y->z.

Parameters

rv (*float*(3)) – rotation angles

Returns

[axis,angle]

Return type

list(list(3),float)

`pyg4ometry.geant4.solid.matrix2axisangle(matrix)`

Convert 3x3 transformation matrix to axis angle representation

Parameters

matrix (*array*(3,3)) – 3x3 rotation matrix array

Returns

[axis,angle]

Return type

list(list(3),float)

`pyg4ometry.geant4.solid.axisangle2matrix(axis, angle)`

Convert axis angle to transformation matrix

Parameters

- **axis** (*list/array*(3)) – axis for rotation
- **angle** (*float*) – rotation angle

Returns

transformation matrix

Return type

array(3,3)

`pyg4ometry.geant4.solid.matrix2tbxyz(matrix)`

Convert rotation matrix to Tait-Bryan angles. Order of rotation is x -> y -> z.

Parameters

matrix – active (positive angle = anti-clockwise rotation about that axis when looking at the axis) matrix.

Returns

[x, y, z] Tait-Bryan angles in a list.

Return type

list(3)

`pyg4ometry.geant4.solid.axisangle2tbxyz(axis, angle)`

Convert axis and angle to tait bryan angles

Parameters

- **axis** (*list/array(3)*) – axis for rotation
- **angle** (*float*) – rotation angle

Returns

tait bryan angles (x-y-z)

Return type

array(3)

`pyg4ometry.geant4.solid.tbxyz2matrix(angles)`

Convert tait bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters

angles – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

`pyg4ometry.geant4.solid.tbzyx2matrix(angles)`

Convert Tait-Bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters

angles – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

`pyg4ometry.geant4.solid.matrix_from(v_from, v_to)`

Returns the rotation matrix that rotates v_from to parallel to v_to.

Useful for ensuring a given face points in a certain direction.

v_from and v_to should be array-like three-vectors.

`pyg4ometry.geant4.solid._rodrigues_anti_parallel(v_from, v_to)`

v_from = vector FROM v_to = vector TO

source: http://en.citizendium.org/wiki/Rotation_matrix#Case_that_.22from.22_and_.22to.22_vectors_are_anti-parallel

`pyg4ometry.geant4.solid.are_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector vector_1 is parallel to vector vector_2 down to some tolerance.

Parameters

- **vector_1** (*array*) – First input vector
- **vector_2** (*array*) – Second input vector
- **tolerance** (*float*) – Tolerance for calculation

Returns

if vectors are parallel

Return type

bool

pyg4ometry.geant4.solid.**are_anti_parallel**(vector_1, vector_2, tolerance=1e-10)

Check if vector vector_1 is parallel to vector vector_2 down to some tolerance.

Parameters

- **vector_1** (array) – First input vector
- **vector_2** (array) – Second input vector
- **tolerance** (float) – Tolerance for calculation

Returns

if vectors are antiparallel

Return type

bool

pyg4ometry.geant4.solid.**reverse**(angles)

Invert the rotation represented by these angles.

pyg4ometry.geant4.solid.**two_fold_orientation**(v1, v2, e1, e2)

matrix_from will align one vector with another, but there are an infinite number of such matrices that align two vectors. This further constrains the rotation by introducing a second pair of vectors.

v1 start v v2 end v e1 start e e2 end v

class pyg4ometry.geant4.solid.**_SolidBase**(name, type, registry=None)

Base class for all solids

property name

evaluateParameter(obj)

evaluateParameterWithUnits(varName)

_addProperty(attribute)

_setProperty(attribute, value)

_getProperty(attribute)

_twoPiValueCheck(attribute, aunit='rad')

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.**Subtraction**(name, obj1, obj2, tra2, registry, addRegistry=True)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Subtraction between two solids

Parameters

- **name** (str) – of solid
- **obj1** (pyg4ometry.geant4.solid) – unrotated, untranslated solid
- **obj2** (pyg4ometry.geant4.solid) – solid rotated and translated according to tra2
- **tra2** (list) – [rot,tra] = [[a,b,g],[dx,dy,dz]]

- **registry** ([Registry](#)) – for storing solid

__repr__()

__str__()

mesh()

translation()

rotation()

object1()

object2()

`pyg4ometry.geant4.solid.rad2deg(rad)`

Convert rad in radians into degrees

Parameters

rad (*float*) – Input in radians

Returns

rad in degrees

Return type

float

`pyg4ometry.geant4.solid.deg2rad(deg)`

Convert deg in degrees into radians

Parameters

deg (*float*) – Input in degrees

Returns

deg in radians

Return type

float

`pyg4ometry.geant4.solid.grad2rad(gradians)`

Convert rad in gradians into radians

Parameters

gradians (*float*) – Input in gradians

Returns

gradians in radians

Return type

float

`pyg4ometry.geant4.solid.tbxyz2axisangle(rv)`

Tait-Bryan x-y-z rotation to axis-angle representation Algorithm from <http://www.sedris.org/wg8home/Documents/WG80485.pdf>

For converting rotation angles to an active axis/angle pair for use in pycsg. Order of rotation: x->y->z.

Parameters

rv (*float*(3)) – rotation angles

Returns

[axis,angle]

Return type`list(list(3),float)``pyg4ometry.geant4.solid.matrix2axisangle(matrix)`

Convert 3x3 transformation matrix to axis angle representation

Parameters**matrix** (`array(3,3)`) – 3x3 rotation matrix array**Returns**`[axis,angle]`**Return type**`list(list(3),float)``pyg4ometry.geant4.solid.axisangle2matrix(axis, angle)`

Convert axis angle to transformation matrix

Parameters

- **axis** (`list/array(3)`) – axis for rotation
- **angle** (`float`) – rotation angle

Returns

transformation matrix

Return type`array(3,3)``pyg4ometry.geant4.solid.matrix2tbxyz(matrix)`

Convert rotation matrix to Tait-Bryan angles. Order of rotation is x -> y -> z.

Parameters**matrix** – active (positive angle = anti-clockwise rotation about that axis when looking at the axis) matrix.**Returns**`[x, y, z]` Tait-Bryan angles in a list.**Return type**`list(3)``pyg4ometry.geant4.solid.axisangle2tbxyz(axis, angle)`

Convert axis and angle to tait bryan angles

Parameters

- **axis** (`list/array(3)`) – axis for rotation
- **angle** (`float`) – rotation angle

Returns

tait bryan angles (x-y-z)

Return type`array(3)``pyg4ometry.geant4.solid.tbxyz2matrix(angles)`

Convert tait bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters**angles** – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

`pyg4ometry.geant4.solid.tbzyx2matrix(angles)`

Convert Tait-Bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters**angles** – list of angles: x, y, z**Returns**

rotation matrix

Return type

array(3,3)

`pyg4ometry.geant4.solid.matrix_from(v_from, v_to)`

Returns the rotation matrix that rotates v_from to parallel to v_to.

Useful for ensuring a given face points in a certain direction.

v_from and v_to should be array-like three-vectors.

`pyg4ometry.geant4.solid._rodrigues_anti_parallel(v_from, v_to)`

v_from = vector FROM v_to = vector TO

source: http://en.citizendium.org/wiki/Rotation_matrix#Case_that_.22from.22_and_.22to.22_vectors_are_anti-parallel`pyg4ometry.geant4.solid.are_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector vector_1 is parallel to vector vector_2 down to some tolerance.

Parameters

- **vector_1** (array) – First input vector
- **vector_2** (array) – Second input vector
- **tolerance** (float) – Tolerance for calculation

Returns

if vectors are parallel

Return type

bool

`pyg4ometry.geant4.solid.are_anti_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector vector_1 is parallel to vector vector_2 down to some tolerance.

Parameters

- **vector_1** (array) – First input vector
- **vector_2** (array) – Second input vector
- **tolerance** (float) – Tolerance for calculation

Returns

if vectors are antiparallel

Return type

bool

`pyg4ometry.geant4.solid.reverse(angles)`

Invert the rotation represented by these angles.

`pyg4ometry.geant4.solid.two_fold_orientation(v1, v2, e1, e2)`

`matrix_from` will align one vector with another, but there are an infinite number of such matrices that align two vectors. This further constrains the rotation by introducing a second pair of vectors.

v1 start *v* *v2* end *v* *e1* start *e* *e2* end *v*

class `pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)`

Base class for all solids

property `name`

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *unit*='rad')

Raises a `ValueError` if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a `TessellatedSolid` instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.OpticalSurface(name, finish, model, surf_type, value, registry, addRegistry=True)`

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

`allowed_models` = ['glisur', 'unified', 'LUT', 'DAVIS', 'dichroic']

`allowed_types` = ['dielectric_metal', 'dielectric_dielectric', 'dielectric_LUT', 'dielectric_LUTDAVIS', ...]

`allowed_finishes` = ['polished', 'polishedfrontpainted', 'polishedbackpainted', 'ground', 'groundfrontpainted', ...]

__repr__()

__str__()

addProperty(*name*, *matrix*)

Add a property to this surface from a matrix.

Parameters

- **name** (*str*) – key of the surface property
- **matrix** (*Matrix*) – matrix defining the value(s) of the property

addVecProperty(*name*, *e*, *v*, *eunit*='eV', *vunit*='')

Add a property from an energy and a value vector to this object.

Parameters

- **name** (*str*) – key of property

- **e** (*list* or *numpy.array* - *shape* (1,)) – energy list/vector in units of eunit
- **v** (*list* or *numpy.array* - *shape* (1,)) – value list/vector in units of vunit
- **eunit** (*str*) – unit for the energy vector (default: eV)
- **vunit** (*str*) – unit for the value vector (default: unitless)

addConstProperty(*name*, *value*, *vunit=""*)

Add a constant scalar property to this object.

Parameters

- **name** (*str*) – key of property
- **value** (*str*, *float*, *int*) – constant value for this property
- **vunit** (*str*) – unit for the value vector (default: unitless)

class pyg4ometry.geant4.solid._SolidBase(*name*, *type*, *registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

pyg4ometry.geant4.solid._cubeNet(*vecList*)

class pyg4ometry.geant4.solid.Para(*name*, *pDx*, *pDy*, *pDz*, *pAlpha*, *pTheta*, *pPhi*, *registry*, *lunit='mm'*,
aunit='rad', *addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a parallelepiped.

Parameters

- **name** (*str*) – of the volume
- **pX** (*float*, *Constant*, *Quantity*, *Variable*) – length along x
- **pY** (*float*, *Constant*, *Quantity*, *Variable*) – length along y
- **pZ** (*float*, *Constant*, *Quantity*, *Variable*) – length along z
- **pAlpha** (*float*, *Constant*, *Quantity*, *Variable*) – angle formed by the y axis and the plane joining the centres of the faces parallel to the z-x plane at -dy/2 and +dy/2
- **pTheta** (*float*, *Constant*, *Quantity*, *Variable*) – polar angle of the line joining the centres of the faces at -dz/2 and +dz/2 in z

- **pPhi** (*float*, *Constant*, *Quantity*, *Variable*) – azimuthal angle of the line joining the centres of the faces at $-dx/2$ and $+dz/2$ in z
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name*, *type*, *registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** $2 \times \pi$.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

pyg4ometry.geant4.solid._cubeNet(*vecList*)

class pyg4ometry.geant4.solid.Trap(*name*, *pDz*, *pTheta*, *pDPhi*, *pDy1*, *pDx1*, *pDx2*, *pAlp1*, *pDy2*, *pDx3*, *pDx4*, *pAlp2*, *registry*, *lunit='mm'*, *aunit='rad'*, *addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a general trapezoid.

Parameters

- **name** (*str*) – name of the volume
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length along z
- **pTheta** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – polar angle of the line joining the centres of the faces at $-/+pDz$
- **pPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – azimuthal angle of the line joining the centres of the faces at $-/+pDz$
- **pDy1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half-length at $-pDz$
- **pDx1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length along x of the side at $y=-pDy1$

- **pDx2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length along x of the side at y=+pDy1
- **pAlp1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle wrt the y axis from the centre of the side (lower endcap)
- **pDy2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half-length at +pDz
- **pDx3** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half-length of the side at y=-pDy2 of the face at +pDz
- **pDx4** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half-length of the side at y=+pDy2 of the face at +pDz
- **pAlp2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle wrt the y axis from the centre of the side (upper endcap)

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name*, *type*, *registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.Orb(*name*, *pRMax*, *registry*, *lunit='mm'*, *nslice=None*, *nstack=None*, *addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Constructs a solid sphere.

Parameters

- **name** (*str*) – of the sold
- **pRMax** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – outer radius
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **nslice** (*int*) – number of phi elements for meshing

- **nstack** (*int*) – number of theta elements for meshing

```
__repr__()
```

Return repr(self).

```
__str__()
```

Return str(self).

```
mesh()
```

```
class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)
```

Base class for all solids

property name

```
evaluateParameter(obj)
```

```
evaluateParameterWithUnits(varName)
```

```
_addProperty(attribute)
```

```
_setProperty(attribute, value)
```

```
_getProperty(attribute)
```

```
_twoPiValueCheck(attribute, aunit='rad')
```

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` over 2 x pi.

```
conver2Tessellated()
```

return a TessellatedSolid instance based on the mesh of this solid.

```
class pyg4ometry.geant4.solid.EllipticalTube(name, pDx, pDy, pDz, registry, lunit='mm', nstack=None,
                                             nslice=None, addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a tube of elliptical cross-section.

Parameters

- **name** (*str*) – name of the solid
- **pDx** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in x
- **pDy** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in y
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length in z
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

```
__repr__()
```

```
__str__()
```

```
mesh()
```

new meshing based of Tubs meshing

```
class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)
```

Base class for all solids

property name

evaluateParameter(obj)

evaluateParameterWithUnits(varName)

_addProperty(attribute)

_setProperty(attribute, value)

_getProperty(attribute)

_twoPiValueCheck(attribute, aunit='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

```
class pyg4ometry.geant4.solid.EllipticalCone(name, pxSemiAxis, pySemiAxis, zMax, pzTopCut, registry,
                                             lunit='mm', nslice=None, nstack=None,
                                             addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a cone with elliptical cross-section and an optional cut. Both `zMax` and `pzTopCut` are half lengths extending from the centre of the cone, at `z=0`.

Parameters

- **name** (*str*) – name of the volume
- **pxSemiAxis** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – semiaxis in x at `z=0` as a fraction of `zMax`.
- **pySemiAxis** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – semiaxis in y at `z=0` as a fraction of `zMax`
- **zMax** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length of the cone.
- **pzTopCut** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – half length of the cut.

__repr__()

Return `repr(self)`.

__str__()

Return `str(self)`.

mesh()

```
class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)
```

Base class for all solids

property name

evaluateParameter(obj)

evaluateParameterWithUnits(varName)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit*='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.Paraboloid`(*name*, *pDz*, *pR1*, *pR2*, *registry*, *lunit*='mm', *nslice*=16, *nstack*=8, *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a paraboloid with possible cuts along the z axis.

Parameters

- **name** (*str*) – of solid
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along z
- **pR1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – radius at -Dz/2
- **pR2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – radius at +Dz/2 (pR2 > pR1)
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

class `pyg4ometry.geant4.solid._SolidBase`(*name*, *type*, *registry*=None)

Base class for all solids

property *name*

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit*='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

```
class pyg4ometry.geant4.solid.Hype(name, innerRadius, outerRadius, innerStereo, outerStereo, lenZ,  
                                registry, lunit='mm', aunit='rad', nslice=None, nstack=None,  
                                addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a tube with hyperbolic profile.

Parameters

- **name** (*str*) – of solid
- **innerRadius** (*float, Constant, Quantity, Variable, Expression*) – inner radius
- **outerRadius** (*float, Constant, Quantity, Variable, Expression*) – outer radius
- **innerStereo** (*float, Constant, Quantity, Variable, Expression*) – inner stereo angle
- **outerStereo** (*float, Constant, Quantity, Variable, Expression*) – outer stereo angle
- **lenZ** – length along z
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()**mesh()**

```
class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)
```

Base class for all solids

property name**evaluateParameter(obj)****evaluateParameterWithUnits(varName)****_addProperty(attribute)****_setProperty(attribute, value)****_getProperty(attribute)**

__twoPiValueCheck(*attribute*, *aunit*='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** $2 \times \pi$.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.Tet`(*name*, *anchor*, *p2*, *p3*, *p4*, *registry*, *lunit*='mm', *degeneracyFlag*=False, *addRegistry*=True)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a tetrahedra.

Parameters

- **name** – of the solid
- **anchor** (*list*) – point 1 (anchor point)
- **p2** (*list*) – point 2
- **p3** (*list*) – point 3
- **p4** (*list*) – point 4
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **degeneracyFlag** – bool, indicates degeneracy of points

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

class `pyg4ometry.geant4.solid._SolidBase`(*name*, *type*, *registry*=None)

Base class for all solids

property **name**

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

__twoPiValueCheck(*attribute*, *aunit*='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** $2 \times \pi$.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid._TwistedSolid`

makeFaceFromLayer(*layer*, *reverse*=False)

makeSide(*pal, pbl, pau, pbu, zl, zu, nsl*)

p = point a = first b = second u = upper l = lower

meshFromLayers(*layers, nsl*)

class pyg4ometry.geant4.solid._TwoVector(*xIn, yIn*)

Rotated(*angle*)

__repr__()

Return repr(self).

__getitem__(*index*)

__add__(*other*)

__sub__(*other*)

__mul__(*other*)

__rmul__(*other*)

__truediv__(*other*)

__abs__()

dot(*other*)

cross(*other*)

class pyg4ometry.geant4.solid._Layer(*p1, p2, p3, p4, z*)

__getitem__(*index*)

Rotated(*angle*)

__repr__()

Return repr(self).

class pyg4ometry.geant4.solid.TwistedBox(*name, twistedangle, pDx, pDy, pDz, registry, lunit='mm',
aunit='rad', nstack=None, refine=0, addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#), [pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid](#)

Constructs a box that is twisted though angle twisted angle

Parameters

- **name** (*str*) – of the solid
- **twistedangle** (*float, Constant, Quantity, Variable, Expression*) – twist angle, must be less than $\pi/2$
- **pDx** (*float, Constant, Quantity, Variable, Expression*) – length in x
- **pDy** (*float, Constant, Quantity, Variable, Expression*) – length in y
- **pDz** (*float, Constant, Quantity, Variable, Expression*) – length in z
- **refine** (*int*) – number of steps to iteratively smoothen the mesh by doubling the number of vertices at every step
- **registry** (*Registry*) – for storing solid

- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nstack** (*int*) – Not written

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

makeLayers(*p1, p2, p3, p4, pDz, theta, nstack*)

mesh_old()

makeLayerEdge(*pDx, pDy, pTwistedAngle=0, nx=20, ny=20*)

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name, type, registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid._TwistedSolid

makeFaceFromLayer(*layer, reverse=False*)

makeSide(*pal, pbl, pau, pbu, zl, zu, nsl*)

p = point a = first b = second u = upper l = lower

meshFromLayers(*layers, nsl*)

class pyg4ometry.geant4.solid._TwoVector(*xln, yln*)

Rotated(*angle*)

__repr__()

Return repr(self).

__getitem__(*index*)

```
__add__(other)
__sub__(other)
__mul__(other)
__rmul__(other)
__truediv__(other)
__abs__()
dot(other)
cross(other)
```

```
class pyg4ometry.geant4.solid._Layer(p1, p2, p3, p4, z)
```

```
__getitem__(index)
Rotated(angle)
__repr__()
    Return repr(self).
```

```
class pyg4ometry.geant4.solid.TwistedTrap(name, twistedAngle, pDz, pTheta, pDPhi, pDy1, pDx1, pDx2,
                                          pDy2, pDx3, pDx4, pAlp, registry, lunit='mm', aunit='rad',
                                          nstack=None, addRegistry=True)
```

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#), [pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid](#)

Constructs a general trapezoid with a twist around one axis.

Parameters

- **name** (*str*) – of the solid
- **twistedAngle** – angle of twist (<90 deg)
- **pDz** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along z
- **pDx1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along x of the side at $y=-pDy1/2$
- **pDx2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length along x of the side at $y=+pDy1/2$
- **pTheta** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – polar angle of the line joining the centres of the faces at $-/+pDz/2$
- **pPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – azimuthal angle of the line joining the centres of the faces at $-/+pDz/2$
- **pDy1** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length at $-pDz/2$
- **pDy2** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length at $+pDz/2$
- **pDx3** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length of the side at $y=-pDy2/2$ of the face at $+pDz/2$
- **pDx4** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – length of the side at $y=+pDy2/2$ of the face at $+pDz/2$

- **pAlp** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle wrt the y axi from the centre of the side
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid

checkParameters()

__repr__()

Return repr(self).

__str__()

Return str(self).

makeLayers(*pl1*, *pl2*, *pl3*, *pl4*, *pu1*, *pu2*, *pu3*, *pu4*, *pDz*, *twist*, *theta*, *nsl*)

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name*, *type*, *registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid._TwoVector(*xIn*, *yIn*)

Rotated(*angle*)

__repr__()

Return repr(self).

__getitem__(*index*)

__add__(*other*)

__sub__(*other*)

__mul__(*other*)

__rmul__(*other*)

__truediv__(*other*)

```
__abs__()  
  
dot(other)  
  
cross(other)  
  
class pyg4ometry.geant4.solid._Layer(p1, p2, p3, p4, z)  
  
    __getitem__(index)  
  
    Rotated(angle)  
  
    __repr__()  
        Return repr(self).  
  
class pyg4ometry.geant4.solid._TwistedSolid  
  
    makeFaceFromLayer(layer, reverse=False)  
  
    makeSide(pal, pbl, pau, pbu, zl, zu, nsl)  
        p = point a = first b = second u = upper l = lower  
  
    meshFromLayers(layers, nsl)  
  
class pyg4ometry.geant4.solid.TwistedTrd(name, twistedangle, pDx1, pDx2, pDy1, pDy2, pDz, registry,  
                                         lunit='mm', aunit='rad', nstack=None, refine=0,  
                                         addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`, `pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid`

Constructs a twisted general trapezoid.

Parameters

- **name** (*str*) – of solid
- **twistedangle** (*float, Constant, Quantity, Variable, Expression*) – twist angle, must be less than 0.5*pi
- **pDx1** (*float, Constant, Quantity, Variable, Expression*) – length in x at surface positioned at -pDz/2
- **pDx2** (*float, Constant, Quantity, Variable, Expression*) – length in x at surface positioned at +pDz/2
- **pDy1** (*float, Constant, Quantity, Variable, Expression*) – length in y at surface positioned at -pDz/2
- **pDy2** (*float, Constant, Quantity, Variable, Expression*) – length in y at surface positioned at +pDz/2
- **pDz** (*float, Constant, Quantity, Variable, Expression*) – length in z
- **refine** (*int*) – number of steps to iteratively smoothen the mesh by doubling the number of vertices at every step
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nstack** (*int*) – number of theta elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

makeLayers(*pl1, pl2, pl3, pl4, pu1, pu2, pu3, pu4, pDz, theta, nsl*)

mesh()

class pyg4ometry.geant4.solid._SolidBase(*name, type, registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.TwistedTubs(*name, endinnerrad, endouterrad, zlen, phi, twistedangle, registry, lunit='mm', aunit='rad', nslice=None, nstack=None, addRegistry=True*)

Bases: [pyg4ometry.geant4.solid.SolidBase.SolidBase](#)

Creates a twisted tube segment. Note that only 1 constructor is supported.

Parameters

- **name** (*str*) – of solid
- **endinnerrad** (*float, Constant, Quantity, Variable, Expression*) – inner radius at the end of the segment
- **endinnerrad** – outer radius at the end of the segment
- **zlen** (*float, Constant, Quantity, Variable, Expression*) – length of the tube segment
- **twistedangle** (*float, Constant, Quantity, Variable, Expression*) – twist angle
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** – number of phi elements for meshing

- **nstack** (*int*) – number of theta elements for meshing

```
__repr__()  
    Return repr(self).  
  
__str__()  
    Return str(self).  
  
makeLayers(verts_bot, verts_top)  
  
mesh()
```

```
class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)  
    Base class for all solids  
  
    property name  
  
    evaluateParameter(obj)  
  
    evaluateParameterWithUnits(varName)  
  
    _addProperty(attribute)  
  
    _setProperty(attribute, value)  
  
    _getProperty(attribute)  
  
    _twoPiValueCheck(attribute, aunit='rad')  
        Raises a ValueError if the attribute is over pyg4ometry.config.twoPiComparisonTolerance over 2 x pi.  
  
    conver2Tessellated()  
        return a TessellatedSolid instance based on the mesh of this solid.
```

```
class pyg4ometry.geant4.solid._GenericPolyhedra(name, pSPhi, pDPhi, numSide, pR, pZ, registry,  
                                                lunit='mm', aunit='rad', addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.

Parameters

- **name** (*str*) – name
- **pSPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle Phi at start of rotation
- **pDPhi** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – angle Phi at end of rotation
- **numSide** (*float*, *Constant*, *Quantity*, *Variable*, *Expression*) – number of polygon sides
- **pR** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – r coordinate list
- **pZ** (*list of float*, *Constant*, *Quantity*, *Variable*, *Expression*) – z coordinate list

```
__repr__()  
    Return repr(self).
```


__str__()

Return str(self).

checkParameters()

mesh()

class pyg4ometry.geant4.solid.**GenericPolycone**(*name, pSPhi, pDPhi, pR, pZ, registry, lunit='mm', aunit='rad', nslice=None, addRegistry=True*)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.

Parameters

- **name** (*str*) – of solid
- **pSPhi** (*float, Constant, Quantity, Variable, Expression*) – angle phi at start of rotation
- **pDPhi** (*float, Constant, Quantity, Variable, Expression*) – angle Phi at end of rotation
- **pR** (*list of float, Constant, Quantity, Variable, Expression*) – r coordinate
- **pZ** (*list of float, Constant, Quantity, Variable, Expression*) – z coordinate
- **registry** (*Registry*) – for storing solid
- **lunit** (*str*) – length unit (nm,um,mm,m,km) for solid
- **aunit** (*str*) – angle unit (rad,deg) for solid
- **nslice** (*int*) – number of phi elements for meshing

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

mesh()

class pyg4ometry.geant4.solid.**_SolidBase**(*name, type, registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.GenericPolyhedra`(*name, pSPhi, pDPhi, numSide, pR, pZ, registry, lunit='mm', aunit='rad', addRegistry=True*)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a solid of rotation using an arbitrary 2D surface defined by a series of (r,z) coordinates.

Parameters

- **name** (*str*) – name
- **pSPhi** (*float, Constant, Quantity, Variable, Expression*) – angle Phi at start of rotation
- **pDPhi** (*float, Constant, Quantity, Variable, Expression*) – angle Phi at end of rotation
- **numSide** (*float, Constant, Quantity, Variable, Expression*) – number of polygon sides
- **pR** (*list of float, Constant, Quantity, Variable, Expression*) – r coordinate list
- **pZ** (*list of float, Constant, Quantity, Variable, Expression*) – z coordinate list

__repr__()

Return repr(self).

__str__()

Return str(self).

checkParameters()

mesh()

class `pyg4ometry.geant4.solid._SolidBase`(*name, type, registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

```
class pyg4ometry.geant4.solid.GenericTrap(name, v1x, v1y, v2x, v2y, v3x, v3y, v4x, v4y, v5x, v5y, v6x,
                                         v6y, v7x, v7y, v8x, v8y, dz, registry, nstack=20, lunit='mm',
                                         addRegistry=True)
```

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs an arbitrary trapezoid using two quadrilaterals sitting on two parallel planes. Vertices 1-4 define the quadrilateral at -dz and vertices 5-8 define the quadrilateral at +dz. This solid is called Arb8 in GDML notation.

Parameters

- **name** – string, name of the volume
- **v1x** – vertex 1 x position
- **v1y** – vertex 1 y position
- **v2x** – vertex 2 x position
- **v2y** – vertex 2 y position
- **v3x** – vertex 3 x position
- **v3y** – vertex 3 y position
- **v4x** – vertex 4 x position
- **v4y** – vertex 4 y position
- **v5x** – vertex 5 x position
- **v5y** – vertex 5 y position
- **v6x** – vertex 6 x position
- **v6y** – vertex 6 y position
- **v7x** – vertex 7 x position
- **v7y** – vertex 7 y position
- **v8x** – vertex 8 x position
- **v8y** – vertex 8 y position
- **dz** – half length along z
- **registry** (`Registry`) – for storing solid

```
__repr__()
```

```
__str__()
```

```
polygon_area(vertices)
```

```
get_vertex(index)
```

```
makeLayers(verts_bot, verts_top)
```

```
mesh()
```

```
class pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)
```

Base class for all solids

```
property name
```

```
evaluateParameter(obj)
```

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute*, *value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute*, *aunit*='rad')

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** $2 \times \pi$.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.TessellatedSolid`(*name*, *meshTess*, *registry*,
meshtype=`MeshType.Freecad`, *addRegistry*=`True`)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Constructs a tessellated solid

Parameters

- **name** (*str*) – of solid
- **mesh** (*Freecad*, *Gdml* or *Stl*) – mesh
- **registry** (*Registry*) – for storing solid
- **meshtype** (*MeshType.Freecad*) – type of mesh

class `MeshType`

Freecad = 1

Gdml = 2

Stl = 3

__repr__()

Return repr(self).

__str__()

Return str(self).

addVertex(*vertex*)

addTriangle(*triangle*)

removeDuplicateVertices()

mesh()

`pyg4ometry.geant4.solid.createTessellatedSolid`(*name*, *polygons*, *reg*)

Args:

name: Name of the tessallated solid polygons: list of polygons (list of points given in clockwise order). All polygons should have the same number of points. *reg*: registry

Returns: TessellatedSolid

class `pyg4ometry.geant4.solid._SolidBase`(*name*, *type*, *registry*=`None`)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` **over** 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

class `pyg4ometry.geant4.solid.MultiUnion`(*name, objects, transformations, registry, addRegistry=True*)

Bases: `pyg4ometry.geant4.solid.SolidBase.SolidBase`

Union between two or more solids.

Parameters

- **name** (*str*) – of solid
- **objects** – unrotated, untranslated solid objects to form union
- **transformations** – `[[rot1,tra1],[rot2,tra2], [rot3,tra3] ..]` or `[[[a,b,g],[dx,dy,dz]], [[a,b,g],[dx,dy,dz]], [[a,b,g],[dx,dy,dz]], ...]`
- **registry** (*Registry*) – for storing solid
- **addRegistry** – Add solid to registry

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

`pyg4ometry.geant4.solid.rad2deg`(*rad*)

Convert rad in radians into degrees

Parameters

rad (*float*) – Input in radians

Returns

rad in degrees

Return type

float

`pyg4ometry.geant4.solid.deg2rad`(*deg*)

Convert deg in degrees into radians

Parameters

deg (*float*) – Input in degrees

Returns

deg in radians

Return type

float

`pyg4ometry.geant4.solid.grad2rad(gradians)`

Convert rad in gradians into radians

Parameters

gradians (*float*) – Input in gradians

Returns

gradians in radians

Return type

float

`pyg4ometry.geant4.solid.tbxyz2axisangle(rv)`

Tait-Bryan x-y-z rotation to axis-angle representation Algorithm from <http://www.sedris.org/wg8home/Documents/WG80485.pdf>

For converting rotation angles to an active axis/angle pair for use in pycsg. Order of rotation: x->y->z.

Parameters

rv (*float*(3)) – rotation angles

Returns

[axis,angle]

Return type

list(list(3),float)

`pyg4ometry.geant4.solid.matrix2axisangle(matrix)`

Convert 3x3 transformation matrix to axis angle representation

Parameters

matrix (*array*(3,3)) – 3x3 rotation matrix array

Returns

[axis,angle]

Return type

list(list(3),float)

`pyg4ometry.geant4.solid.axisangle2matrix(axis, angle)`

Convert axis angle to transformation matrix

Parameters

- **axis** (*list/array*(3)) – axis for rotation
- **angle** (*float*) – rotation angle

Returns

transformation matrix

Return type

array(3,3)

`pyg4ometry.geant4.solid.matrix2tbxyz(matrix)`

Convert rotation matrix to Tait-Bryan angles. Order of rotation is x -> y -> z.

Parameters

matrix – active (positive angle = anti-clockwise rotation about that axis when looking at the axis) matrix.

Returns

[x, y, z] Tait-Bryan angles in a list.

Return type

list(3)

pyg4ometry.geant4.solid.**axisangle2tbxyz**(axis, angle)

Convert axis and angle to tait bryan angles

Parameters

- **axis** (list/array(3)) – axis for rotation
- **angle** (float) – rotation angle

Returns

tait bryan angles (x-y-z)

Return type

array(3)

pyg4ometry.geant4.solid.**tbxyz2matrix**(angles)

Convert tait bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters

angles – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

pyg4ometry.geant4.solid.**tbzyx2matrix**(angles)

Convert Tait-Bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters

angles – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

pyg4ometry.geant4.solid.**matrix_from**(v_from, v_to)

Returns the rotation matrix that rotates v_from to parallel to v_to.

Useful for ensuring a given face points in a certain direction.

v_from and v_to should be array-like three-vectors.

pyg4ometry.geant4.solid.**_rodrigues_anti_parallel**(v_from, v_to)

v_from = vector FROM v_to = vector TO

source: http://en.citizendium.org/wiki/Rotation_matrix#Case_that_.22from.22_and_.22to.22_vectors_are_anti-parallel

`pyg4ometry.geant4.solid.are_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector `vector_1` is parallel to vector `vector_2` down to some tolerance.

Parameters

- **vector_1** (*array*) – First input vector
- **vector_2** (*array*) – Second input vector
- **tolerance** (*float*) – Tolerance for calculation

Returns

if vectors are parallel

Return type

bool

`pyg4ometry.geant4.solid.are_anti_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector `vector_1` is parallel to vector `vector_2` down to some tolerance.

Parameters

- **vector_1** (*array*) – First input vector
- **vector_2** (*array*) – Second input vector
- **tolerance** (*float*) – Tolerance for calculation

Returns

if vectors are antiparallel

Return type

bool

`pyg4ometry.geant4.solid.reverse(angles)`

Invert the rotation represented by these angles.

`pyg4ometry.geant4.solid.two_fold_orientation(v1, v2, e1, e2)`

`matrix_from` will align one vector with another, but there are an infinite number of such matrices that align two vectors. This further constrains the rotation by introducing a second pair of vectors.

`v1` start `v` `v2` end `v` `e1` start `e` `e2` end `v`

class `pyg4ometry.geant4.solid._SolidBase(name, type, registry=None)`

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a `ValueError` if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` over $2 \times \pi$.

conver2Tessellated()

return a `TessellatedSolid` instance based on the mesh of this solid.

class pyg4ometry.geant4.solid.Scaled(*name, solid, pX, pY, pZ, registry, addRegistry=True*)

Bases: *pyg4ometry.geant4.solid.SolidBase.SolidBase*

Constructs a scaled sold.

Parameters

- **name** (*str*) – of solid for registry
- **pX** (*float, Constant, Quantity, Variable, Expression*) – scale in x
- **pY** (*float, Constant, Quantity, Variable, Expression*) – scale in y
- **pZ** (*float, Constant, Quantity, Variable, Expression*) – scale in z
- **registry** (*Registry*) – for storing solid

Poram solid

reference for scaling

__repr__()

Return repr(self).

__str__()

Return str(self).

mesh()

class pyg4ometry.geant4.solid.SolidBase(*name, type, registry=None*)

Base class for all solids

property name

evaluateParameter(*obj*)

evaluateParameterWithUnits(*varName*)

_addProperty(*attribute*)

_setProperty(*attribute, value*)

_getProperty(*attribute*)

_twoPiValueCheck(*attribute, aunit='rad'*)

Raises a ValueError if the attribute is over `pyg4ometry.config.twoPiComparisonTolerance` over 2 x pi.

conver2Tessellated()

return a TessellatedSolid instance based on the mesh of this solid.

Submodules

`pyg4ometry.geant4.AssemblyVolume`

Module Contents

Classes

AssemblyVolume

AssemblyVolume : similar to a logical volume but does not have a sense of

class pyg4ometry.geant4.AssemblyVolume.**AssemblyVolume**(name, registry=None, addRegistry=True)

AssemblyVolume : similar to a logical volume but does not have a sense of shape, material or field :param name: of assembly volume :type name: str :param registry: :type registry: :param addRegistry: :type addRegistry: bool

__repr__()

Return repr(self).

add(physicalVolume)

_getDaughterMeshesByName(name)

_getDaughterMeshesByIndex(index)

_getDaughterMeshes()

Get daughter meshes for overlap checking. return [daughterMesh,...],[daughterBoundingMesh,...][daughterName,...]

_getPVMeshes(pv)

Can technically return more than one mesh if the daughter is also an assembly.

_getPhysicalDaughterMesh(pv, warn=True)

Return a (cloned from the lv) mesh of a given pv with rotation,scale, translation evaluated.

clipGeometry(newSolid, rotation=(0, 0, 0), position=(0, 0, 0), runit='rad', punit='mm', replace=False, depth=0, solidUsageCount=_defaultdict(int), lvUsageCount=_defaultdict(int))

Clip the geometry to newSolid, placed with rotation and position.

extent(includeBoundingSolid=True)

depth(depth=0)

Depth for LV-PV tree

getAABBMesh()

return CSG.core (symmetric around the origin) axis aligned bounding box mesh

logicalVolume(material='G4_Galactic', solidName='worldSolid')

Return an logical volume of this this assembly volume, in effect adding a cuboid solid and material of this logical volume, retaining all of the relative daughter placements.

makeWorldVolume(material='G4_Galactic')

dumpStructure(depth=0)

pyg4ometry.geant4.BorderSurface

Module Contents

Classes

BorderSurface

```
class pyg4ometry.geant4.BorderSurface.BorderSurface(name, physref1, physref2, surface_property,
                                                    registry, addRegistry=True)
```

Bases: `pyg4ometry.geant4.SurfaceBase.SurfaceBase`

```
__repr__()
```

```
pyg4ometry.geant4.DivisionVolume
```

Module Contents

Classes

<i>DivisionVolume</i>	DivisionVolume: G4PVDivision
-----------------------	------------------------------

```
class pyg4ometry.geant4.DivisionVolume.DivisionVolume(name, logicalVolume, motherVolume, axis,
                                                    ndivisions=-1, width=-1, offset=0,
                                                    registry=None, addRegistry=True,
                                                    unit='mm')
```

Bases: `pyg4ometry.geant4.PhysicalVolume.PhysicalVolume`

DivisionVolume: G4PVDivision

Parameters

- **name** – of physical volume
- **logical** – volume to be placed
- **mother** – logical volume,
- **axis** – kXAxis, kYAxis, kZAxis, kRho, kPhi
- **ncopies** – number of replicas
- **width** – spacing between replicas along axis
- **offset** – of grid

```
class Axis
```

```
    kXAxis = 1
```

```
    kYAxis = 2
```

```
    kZAxis = 3
```

```
    kRho = 4
```

```
    kPhi = 5
```

```
    getMotherSize()
```

```
    checkAxis(allowed_axes)
```

```
    divideBox(offset, width, ndiv)
```

```
    divideTubs(offset, width, ndiv)
```

```
divideCons(offset, width, ndiv)
dividePara(offset, width, ndiv)
divideTrd(offset, width, ndiv)
dividePolycone(offset, width, ndiv)
dividePolyhedra(offset, width, ndiv)
createDivisionMeshes()
__repr__()
extent(includeBoundingSolid=True)
```

pyg4ometry.geant4.LogicalVolume

Module Contents

Classes

LogicalVolume

LogicalVolume : G4LogicalVolume

Functions

_solid2tessellated(solid)

pyg4ometry.geant4.LogicalVolume._solid2tessellated(solid)

```
class pyg4ometry.geant4.LogicalVolume.LogicalVolume(solid, material, name, registry=None,
                                                    addRegistry=True, **kwargs)
```

LogicalVolume : G4LogicalVolume

Parameters

- **solid** –
- **material** –
- **name** (*str*) –
- **registry** –
- **addRegistry** (*bool*) –

__repr__()

Return repr(self).

reMesh(*recursive=False*)

Regenerate the visualisation for this logical volume. Required if the geometry is modified and overlap checking is subsequently required or revisualisation.

add(*physicalVolume*)

Add physical volume to this logicalVolume

Parameters

physicalVolume (*PhysicalVolume*, *ReplicaVolume*, *ParameterisedVolume*, *DivisionVolume*) – physical volume to add

addBDSIMObject(*bdsimobject*)

_getPhysicalDaughterMesh(*pv*, *warn=True*)

Return a (cloned from the lv) mesh of a given pv with rotation,scale, translation evaluated.

cullDaughtersOutsideSolid(*solid*, *rotation=None*, *position=None*)

Given a solid with a placement rotation and position inside this logical volume, remove (cull) any daughters that would not lie entirely within it. The rotation and position are applied to the solid w.r.t. the frame of this logical volume.

Parameters

- **rotation** (*list(float, float, float)* or *None* - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or *None* - 3 values in mm) – translation of the solid w.r.t. this lv

transformDaughters(*rotation=(0, 0, 0)*, *position=(0, 0, 0)*, *runit='rad'*, *punit='mm'*)

Transform the daughter volumes (without clipping)

Parameters

- **rotation** (*list(float, float, float)* or *None* - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or *None* - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)

replaceSolid(*newSolid*, *rotation=(0, 0, 0)*, *position=(0, 0, 0)*, *runit='rad'*, *punit='mm'*)

Replace the outer solid with optional position and rotation

Parameters

- **newSolid** (*pyg4ometry.geant4.solid*) – object to clip the geometry to
- **rotation** (*list(float, float, float)* or *None* - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or *None* - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)

clipGeometry(*newSolid*, *rotation=(0, 0, 0)*, *position=(0, 0, 0)*, *runit='rad'*, *punit='mm'*, *replace=False*, *depth=0*, *solidUsageCount=_defaultdict(int)*, *lvUsageCount=_defaultdict(int)*)

Clip the geometry to newSolid, placed with rotation and position.

Parameters

- **newSolid** (*pyg4ometry.geant4.solid*) – object to clip the geometry to

- **rotation** (*list(float, float, float)* or *None* - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or *None* - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)
- **replace** (*bool*) – replace the outer solid or not
- **depth** (*int*) – recursion depth (DO NOT USE)
- **solidUsageCount** (*defaultdict*) – solid name dictionary for replacement recursion (DO NOT USE)
- **lvUsageCount** (*defaultdict*) – lv name dictionary for replacement recursion (DO NOT USE)

changeSolidAndTrimGeometry(*newSolid, rotation=(0, 0, 0), position=(0, 0, 0), runit='rad', punit='mm'*)

Change the solid of this logical volume, remove any daughters that will lie outside it, and form new Boolean intersection solids for any daughters that cross the boundary (intersect) it. The rotation and translation are with respect to the original frame and all daughters will now be replaced with respect to the new frame. Therefore, that same rotation and translation should be use to re-place this logical volume if desired. The default is none though, so the frame would nominally remain the same.

Parameters

- **newSolid** (*any of pyg4ometry.geant4.solid*) – new solid to use for this logical volume
- **rotation** (*list(float, float, float)* or *None* - 3 values in radians) – Tait-Bryan rotation for the new solid w.r.t. old frame (i.e. current lv)
- **position** (*list(float, float, float)* or *None* - 3 values in mm) – translation for the new solid w.r.t. old frame (i.e. current lv)

checkOverlaps(*recursive=False, coplanar=False, debugIO=False, printOut=True, nOverlapsDetected=[0]*)

Check based on the meshes in each logical volume if there are any geometrical overlaps. By default, overlaps are checked between daughter volumes and with the mother volume itself (protrusion). Coplanar overlaps may also be checked (default on).

Print out will be given for any overlaps detected and the visualiser will show the colour coded overlaps.

Parameters

- **recursive** – bool - Whether to descend into the daughter volumes and check their contents also.
- **coplanar** – bool - Whether to check for coplanar overlaps
- **debugIO** – bool - Print out for every check made
- **printOut** – bool - (internal) Whether to print out a summary of N overlaps detected
- **nOverlapsDetected** – [int] - (internal) counter for recursion - ignore

setSolid(*solid*)

Set (replace) the outer solid. Does not change the placement of the daughters in the volume. If there is a transformation then use `replaceSolid`

makeSolidTessellated()

Make solid tessellated. Sometimes useful when a boolean cannot be visualised in Geant4

addAuxiliaryInfo(*auxiliary*)

Add auxiliary information to logical volume :param auxiliary: auxiliary information for the logical volume
:type auxiliary: tuple or list

extent(*includeBoundingSolid=False*)

Compute the axis aligned extent of the logical volume.

Parameters

includeBoundingSolid (*bool*) – Include the bounding solid or not

depth(*depth=0*)

Depth for LV-PV tree

clipSolid(*lengthSafety=1e-06*)

Assuming the solid of this LV is a Box, reduce its dimensions and re-placement all daughters to reduce the box to the minimum (axis-aligned) bounding box. This updates the dimensions of the box and the translation of each daughter physical volume.

Parameters

lengthSafety (*float*) – safety length

makeLogicalPhysicalNameSets()

Return a set of logical names and physical names used in this logical volume and any daughters. This is built up recursively by checking all daughters etc.

findLogicalByName(*name*)

Return a list of LogicalVolume instances used inside this logical volume as daughters (at any level inside) with the given name.

Parameters

name (*str*) – lv name

makeMaterialNameSet()

Return a set of material names used in this logical volume and any daughters. This is built up recursively by checking all daughters etc etc.

assemblyVolume(*materialName='G4_AIR0x7f8441173ac0'*)

Return an assembly volume of this logical volume, in effect removing the solid and material of this logical volume, but retaining all of the relative daughter placements.

makeWorldVolume(*worldMaterial='G4_Galactic'*)

This will create a container box according to the extents of this logical volume: an axis-aligned bounding-box. It will be filled with the given material (predefined by name) and assigned as the world volume (outermost) of the registry according to this logical volume.

dumpStructure(*depth=0*)

pyg4ometry.geant4.Loop

Module Contents

Classes

Loop

```
class pyg4ometry.geant4.Loop.Loop(variable, loopFrom, loopTo, loopStep, objectsToLoop)
    expandLoop()
```

pyg4ometry.geant4.ParameterisedVolume

Module Contents

Classes

ParameterisedVolume

ParametrisedVolume

```
class pyg4ometry.geant4.ParameterisedVolume.ParameterisedVolume(name, logicalVolume,
                                                                motherVolume, ncopies,
                                                                paramData, transforms,
                                                                registry=None,
                                                                addRegistry=True)
```

Bases: [pyg4ometry.geant4.ReplicaVolume.ReplicaVolume](#)

ParametrisedVolume :param name: of parametrised volume :type name: str :param logical: volume to be placed :type logical: logicalVolume :param mother: volume logical volume :type mother: logicalVolume :param ncopies: number of parametrised volumes :type ncopies: int

```
class BoxDimensions(pX, pY, pZ, lunit='mm')
```

```
class TubeDimensions(pRMin, pRMax, pDz, pSPhi, pDPhi, lunit='mm', aunit='rad')
```

```
class ConeDimensions(pRMin1, pRMax1, pRMin2, pRMax2, pDz, pSPhi, pDPhi, lunit='mm', aunit='rad')
```

```
class OrbDimensions(pRMax, lunit='mm')
```

```
class SphereDimensions(pRMin, pRMax, pSPhi, pDPhi, pSTheta, pDTheta, lunit='mm', aunit='rad')
```

```
class TorusDimensions(pRMin, pRMax, pRTor, pSPhi, pDPhi, lunit='mm', aunit='rad')
```

```
class HypeDimensions(innerRadius, outerRadius, innerStereo, outerStereo, lenZ, lunit='mm', aunit='rad')
```

```
class ParaDimensions(pX, pY, pZ, pAlpha, pTheta, pPhi, lunit='mm', aunit='rad')
```

```
class TrdDimensions(pX1, pX2, pY1, pY2, pZ, lunit='mm')
```



```

class TrapDimensions(pDz, pTheta, pDPhi, pDy1, pDx1, pDx2, pAlp1, pDy2, pDx3, pDx4, pAlp2,
                    lunit='mm', aunit='rad')

class PolyconeDimensions(pSPhi, pDPhi, pZpl, pRMin, pRMax, lunit='mm', aunit='rad')

class PolyhedraDimensions(pSPhi, pDPhi, numSide, pZpl, pRMin, pRMax, lunit='mm', aunit='rad')

class EllipsoidDimensions(pxSemiAxis, pySemiAxis, pzSemiAxis, pzBottomCut, pzTopCut, lunit='mm')

createParameterisedMeshes()

__repr__()

extent(includeBoundingSolid=True)

```

pyg4ometry.geant4.PhysicalVolume

Module Contents

Classes

<i>PhysicalVolume</i>	PhysicalVolume	:	G4VPhysicalVolume, G4PVPlacement
---------------------------------------	----------------	---	-------------------------------------

```

class pyg4ometry.geant4.PhysicalVolume.PhysicalVolume(rotation, position, logicalVolume, name,
                                                         motherVolume, registry=None,
                                                         copyNumber=0, addRegistry=True,
                                                         scale=None)

```

PhysicalVolume : G4VPhysicalVolume, G4PVPlacement

Parameters

- **rotation** – [float,float,float] - rotations about x,y,z axes of mother volume
- **position** – [float,float,float] - translation with respect to mother volume
- **logicalVolume** – [*pyg4ometry.geant4.LogicalVolume*](#) - instance to place
- **name** – str - name of this placement
- **motherVolume** – [*pyg4ometry.geant4.LogicalVolume*](#) - mother volume to place into
- **registry** – [*pyg4ometry.geant4.Registry*](#) - registry to register to
- **copyNumber** – int - copy number of the placement that can be used for sensitivity
- **addRegistry** – bool - whether to add to the registry or not

```
__repr__()
```

Return repr(self).

```
extent(includeBoundingSolid=True)
```

```
getAABBMesh()
```

return CSG.core (symmetric around the origin) axis aligned bounding box mesh

pyg4ometry.geant4.Registry

Module Contents

Classes

<i>Registry</i>	Object to store geometry for input and output.
<i>GeometryComplexityInformation</i>	

Functions

<i>solidName</i> (var)	
<i>removeprefix</i> (string, prefix, /)	
<i>AnalyseGeometryComplexity</i> (logicalVolume)	Analyse a geometry tree starting from a logical volume.
<i>_UpdateComplexity</i> (lv, info)	
<i>AnalyseGeometryStructure</i> (registry[, lv_name, debug, ...])	Produce a pandas dataframe representing the structure of the geometry.
<i>DumpGeometryStructureTree</i> (lv[, depth])	

pyg4ometry.geant4.Registry.**solidName**(var)

pyg4ometry.geant4.Registry.**removeprefix**(string, prefix, /)

Parameters

- **string** (*str*) –
- **prefix** (*str*) –

Return type

str

class pyg4ometry.geant4.Registry.**Registry**

Object to store geometry for input and output. All of the pyg4ometry classes can be used without storing them in the Registry. The registry is used to write the GDML output file. A registry needs to be used in conjunction with GDML Define objects for evaluation of expressions.

clear()

Empty all internal structures

getExpressionParser()

registerSolidEdit(solid)

addMaterial(material, dontWarnIfAlreadyAdded=False)

Register a material with this registry.

Parameters**material** (**Material**) – Material object for storage**transferMaterial**(*material*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a material to this registry. This can operate on a Material, an Isotope and an Element instance.

addSolid(*solid*)

Register a solid with this registry.

Parameters**solid** (*One of the geant4 solids*) – Solid object for storage**transferSolid**(*solid*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a solid to this registry. Doesn't handle any members' transferal - only the solid itself.

Parameters**solid** (*One of the geant4 solids*) – Solid object for storage**addLogicalVolume**(*volume*)

Register a logical volume with this registry. Also accepts Assembly Volumes.

Parameters**volume** (**LogicalVolume**) – LogicalVolume object for storage**transferLogicalVolume**(*volume*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a logical volume to this registry. Doesn't handle any members' transferal - only the logical volume itself.

addPhysicalVolume(*volume*)

Registry a physical volume with this registry.

Parameters**volume** (**PhysicalVolume**) – PhysicalVolume object for storage**transferPhysicalVolume**(*volume*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a physical volume to this registry. Doesn't handle any members' transferal - only the physical volume itself.

addSurface(*surface*)

Register a surface with this registry.

Parameters**surface** (**pyg4ometry.geant4.BorderSurface** or **pyg4ometry.geant4.SkinSurface**) – Surface**transferSurface**(*surface*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a surface to this registry.

addAuxiliary(*auxiliary*)**addDefine**(*define*)

Register a define with this registry.

Parameters**define** (**Constant**, **Quantity**, **Variable**, **Matrix**) – Definition object for storage**transferDefine**(*define*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a single define from another registry to this one. No checking on previous registry or not.

transferDefines(*var*, *otherRegistry*, *incrementRenameDict*={}, *userRenameDict*=None)

This function tolerates all types of defines including vector ones.

Transfer defines from one registry to another recursively. A define may not be part of the old registry so won't be added to this one. A define may be a vector or composite and its 'bits' may be in the (old) registry so each part should be checked.

In "3x + 2", "x" would be a variable". In "3.5*2" there would be no variables.

setWorld(*worldIn*)

The argument can either be the name of logical volume of the world or the `pyg4ometry.geant4.LogicalVolume` instance of the world volume. The term world is used to refer to the outermost volume of the hierarchy.

setWorldVolume(*worldIn*)

An alias for some of us who can't remember.

_orderMaterialList(*materials*, *materials_ordered*=[])

orderMaterials()

Need to have a ordered list of all material entities for writing to GDML. GDML needs to have the isotopes/elements/materials defined in use order

orderLogicalVolumes(*lvName*, *first*=True)

Need to have an ordered list from most basic (solid) object upto physical/logical volumes for writing to GDML. GDML needs to have the solids/booleans/volumes defined in order

addVolumeRecursive(*volume*, *collapseAssemblies*=False, *incrementRenameDict*=None, *userRenameDict*=None)

Transfer a volume hierarchy to this registry. Any objects that had a registry set to another will be set to this one and will be owned by it effectively. :param volume: PhysicalVolume or LogicalVolume or AssemblyVolume. :type volume: `pyg4ometry.geant4.PhysicalVolume`, `pyg4ometry.geant4.LogicalVolume`, `pyg4ometry.geant4.AssemblyVolume`. :param collapseAssemblies: if True, daughters of AssemblyVolume's will be attached directly to the mother of the assembly and the AssemblyVolume itself will be eliminated from the geometry tree :param incrementRenameDict: ignore - dictionary used internally for potentially incrementing names :param userRenameDict: a dictionary of find/replace regex strings to be used to rename volumes/materials/etc.

In the case where some object or variable has a name (e.g. 'X') that already exists in this registry, it will be incremented to 'X_1'.

addAndCollapseAssemblyVolumeRecursive(*assemblyPV*, *motherVol*, *positions*, *rotations*, *scales*, *names*, *incrementRenameDict*, *userRenameDict*)

Transfer and collapse an AssemblyVolume hierarchy to this registry. Daughter volumes are copied, renamed, and attached to the supplied mother LogicalVolume with the correct position/rotation. Any objects that had a registry set to another will be set to this one and will be owned by it effectively. :param assemblyPV: the PhysicalVolume that places an AssemblyVolume :param motherVol: the LogicalVolume to which all daughters of the AssemblyVolume (including any daughters of nested AssemblyVolume's) should be attached :param positions: a list (initially empty) to hold position objects for each AssemblyVolume in a hierarchy of nested assemblies (used to correctly set the transformation of daughters within the motherVol) :param rotations: a list (initially empty) to hold rotation objects for each AssemblyVolume in a hierarchy of nested assemblies (used to correctly set the transformation of daughters within the motherVol) :param scales: a list (initially empty) to hold scale objects for each AssemblyVolume in a hierarchy of nested assemblies (used to correctly set the transformation of daughters within the motherVol) :param names: a list (initially empty) to hold the names of each AssemblyVolume in a hierarchy of nested assemblies (used to set unique names for the daughters within the motherVol) :param incrementRenameDict: ignore - dictionary

used internally for potentially incrementing names :param userRenameDict: a dictionary of find/replace regex strings to be used to rename volumes/materials/etc.

transferSolidDefines(solid, incrementRenameDict={}, userRenameDict=None)

For each parameter in a given solid (unique to each) check if it's a define and transfer that over.

volumeTree(lvName)

Not sure what this method is used for

solidTree(solidName)

Not sure what this method is used for

getWorldVolume()

printStats()

structureAnalysis(lv_name=None, debug=False, level=0, df=None)

_findDictByName(dic, nameFragment)

Find a object which name matches (or partially matches) nameFragment, returns a list of objects

findSolidByName(nameFragment='box')

Find a solid which name matches (or partially matches) nameFragment, returns a list of solids

findMaterialByName(nameFragment='G4_AIR')

Find a material which name matches (or partially matches) nameFragment, returns a list of materials

findLogicalVolumeByName(nameFragment='World')

Find a logical volume which name matches (or partially matches) nameFragment, returns a list of LogicalVolumes

findPhysicalVolumeByName(nameFragment)

Find a physical volume which name matches (or partially matches) nameFragment, returns a list of LogicalVolumes

class pyg4ometry.geant4.Registry.**GeometryComplexityInformation**

printSummary(boolDepthLimit=3)

pyg4ometry.geant4.Registry.**AnalyseGeometryComplexity**(logicalVolume)

Analyse a geometry tree starting from a logical volume. Produces an instance of *GeometryComplexityInformation* with summary information. Provides:

- count per solid type
- number of daughters per logical volume
- dictionary of N daughters for each logical volume name
- depth count of Boolean solids

ie a Boolean of a Boolean returns 2, a Boolean of two primitives returns 1

- a dictionary of boolean depth for each logical volume name

Example:

```
info = AnalyseGeometryComplexity(lv)
info.printSummary()
```

```
pyg4ometry.geant4.Registry._UpdateComplexity(lv, info)
```

```
pyg4ometry.geant4.Registry.AnalyseGeometryStructure(registry, lv_name=None, debug=False, level=0,
                                                    df=None)
```

Produce a pandas dataframe representing the structure of the geometry.

```
pyg4ometry.geant4.Registry.DumpGeometryStructureTree(lv, depth=0)
```

```
pyg4ometry.geant4.ReplicaVolume
```

Module Contents

Classes

<i>ReplicaVolume</i>	ReplicaVolume: G4PVReplica
----------------------	----------------------------

```
class pyg4ometry.geant4.ReplicaVolume.ReplicaVolume(name, logicalVolume, motherVolume, axis,
                                                    nreplicas, width, offset=0, registry=None,
                                                    addRegistry=True, wunit='mm', ounit='mm')
```

Bases: [pyg4ometry.geant4.PhysicalVolume.PhysicalVolume](#)

ReplicaVolume: G4PVReplica

Parameters

- **name** – of physical volume
- **logical** – volume to be placed
- **mother** – logical volume,
- **axis** – kXAxis, kYAxis, kZAxis, kRho, kPhi
- **ncopies** – number of replicas
- **width** – spacing between replicas along axis
- **offset** – of grid

```
class Axis
```

```
    kXAxis = 1
```

```
    kYAxis = 2
```

```
    kZAxis = 3
```

```
    kRho = 4
```

```
    kPhi = 5
```

```
    GetAxisName()
```

```
    _checkInternalOverlaps(debugIO=False, nOverlapsDetected=[0])
```

Check if there are overlaps with the nominal mother volume. ie it possible to provide an incorrect mother volume / logical volume and parameterisation.

createReplicaMeshes()

getPhysicalVolumes()

return a list of temporary (ie not added to the relevant registry) PhysicalVolume instances with appropriate transforms including any daughter ReplicaVolumes.

The exception is for kRho axis where new unique solids and logical volumes are required. Therefore, these are added to the registry and inadvertently to the mother LV as PVS.

__repr__()

extent(*includeBoundingSolid=True*)

`pyg4ometry.geant4.SkinSurface`

Module Contents

Classes

SkinSurface

```
class pyg4ometry.geant4.SkinSurface.SkinSurface(name, volumeref, surface_property, registry,
                                                addRegistry=True)
```

Bases: `pyg4ometry.geant4.SurfaceBase.SurfaceBase`

__repr__()

`pyg4ometry.geant4.SurfaceBase`

Module Contents

Classes

SurfaceBase

```
class pyg4ometry.geant4.SurfaceBase.SurfaceBase(name, type, surface_property, registry, addRegistry)
```

_chkType(*obj, t, prop*)

pyg4ometry.geant4._Material

Module Contents

Classes

<i>MaterialBase</i>	
<i>Material</i>	This class provides an interface to GDML material definitions.
<i>Element</i>	This class provides an interface to GDML material definitions. Because of the different options
<i>Isotope</i>	This class that handles isotopes as components of composite materials. An element can be

Functions

<code>getNistMaterialDict()</code>	
<code>getNistMaterialList()</code>	
<code>getNistElementZToName()</code>	
<code>_getClassVariables(obj)</code>	
<code>_makeNISTCompoundList()</code>	
<code>_safeName(name)</code>	
<code>loadNISTMaterialDict()</code>	
<code>nist_materials_name_lookup(name)</code>	
<code>nist_materials_z_lookup(z)</code>	
<code>nist_element_2geant4Element(name[, reg])</code>	This returns an instance of either ElementSimple or ElementIsotopeMixture.
<code>nist_material_2geant4Material(name[, reg])</code>	
<code>MaterialPredefined(name[, registry])</code>	Proxy method to construct a NIST compound material - this is just a handle as nothing
<code>MaterialArbitrary(name[, registry])</code>	Just a name of a material. WARNING: It is left to the
<code>MaterialSingleElement(name, atomic_number, ...[, ...])</code>	Proxy method to construct a simple material - full description of the element contained is contained in one definition
<code>MaterialCompound(name, density, number_of_components)</code>	Proxy method to construct a composite material - can be any mixture of Elements and/or Materials
<code>ElementSimple(name, symbol, Z, A[, registry])</code>	Proxy method to construct a simple element - full description of the element contained is contained in one definition
<code>ElementIsotopeMixture(name, symbol, n_comp[, ...])</code>	Proxy method to construct a composite element - a mixture of predefined isotopes

Attributes

<code>_nistMaterialDict</code>
<code>_nistMaterialList</code>
<code>_nistElementZToName</code>

pyg4ometry.geant4._Material._nistMaterialDict

pyg4ometry.geant4._Material._nistMaterialList

pyg4ometry.geant4._Material._nistElementZToName

pyg4ometry.geant4._Material.getNistMaterialDict()

pyg4ometry.geant4._Material.getNistMaterialList()

pyg4ometry.geant4._Material.getNistElementZToName()

pyg4ometry.geant4._Material._getClassVariables(*obj*)

pyg4ometry.geant4._Material._makeNISTCompoundList()

pyg4ometry.geant4._Material._safeName(*name*)

pyg4ometry.geant4._Material.loadNISTMaterialDict()

pyg4ometry.geant4._Material.nist_materials_name_lookup(*name*)

pyg4ometry.geant4._Material.nist_materials_z_lookup(*z*)

pyg4ometry.geant4._Material.nist_element_2geant4Element(*name*, *reg=None*)

This returns an instance of either ElementSimple or ElementIsotopeMixture.

pyg4ometry.geant4._Material.nist_material_2geant4Material(*name*, *reg=None*)

pyg4ometry.geant4._Material.MaterialPredefined(*name*, *registry=None*)

Proxy method to construct a NIST compound material - this is just a handle as nothing needs to be additionally defined for a NIST compound. A check is performed on the name to ensure it is a valid NIST specifier.

Inputs:

name - string

pyg4ometry.geant4._Material.MaterialArbitrary(*name*, *registry=None*)

Just a name of a material. WARNING: It is left to the user to ensure that the name is valid.

Inputs:

name - string

pyg4ometry.geant4._Material.MaterialSingleElement(*name*, *atomic_number*, *atomic_weight*, *density*,
registry=None, *tolerateZeroDensity=False*)

Proxy method to construct a simple material - full description of the element contained is contained in one definition

Inputs:

name - string atomic_number - int, number of protons, commonly known as 'Z' atomic_weight - molar weight in g/mole, commonly known as 'A' density - float, material density in g/cm3

pyg4ometry.geant4._Material.MaterialCompound(*name*, *density*, *number_of_components*, *registry=None*,
tolerateZeroDensity=False, *state=None*)

Proxy method to construct a composite material - can be a mixture of Elements and/or Materials

Inputs:

name - string density - float, material density in g/cm3 number_of_components - int, number of components in the mixture

`pyg4ometry.geant4._Material.ElementSimple(name, symbol, Z, A, registry=None)`

Proxy method to construct a simple element - full description of the element contained is contained in one definition

Inputs:

name - string symbol - string, chemical formula of the compound Z - int, Atomic number A - float, mass number

`pyg4ometry.geant4._Material.ElementIsotopeMixture(name, symbol, n_comp, registry=None, state=None)`

Proxy method to construct a composite element - a mixture of predefined isotopes

Inputs:

name - string symbol - string, chemical formula of the compound n_comp - int, number of isotope components

`class pyg4ometry.geant4._Material.MaterialBase(name, state=None, registry=None)`

`_addToRegistry()`

`get_material_object(material)`

`set_registry(registry, dontWarnIfAlreadyAdded=False)`

`set_state(state)`

`__repr__()`

Return repr(self).

`class pyg4ometry.geant4._Material.Material(**kwargs)`

Bases: [MaterialBase](#)

This class provides an interface to GDML material definitions.

Because of the different options for constructing a material instance the constructor is kwarg only. Proxy methods are provided to instantiate particular types of material. Those proxy methods are:

MaterialSingleElement MaterialCompound MaterialPredefined

It is possible to instantiate a material directly through kwargs. The possible kwargs are (but note some are mutually exclusive): name - string density - float atomic_number - int atomic_weight - float number_of_components - int state - string pressure - float pressure_unit - string temperature - float temperature_unit - string

property state_variables

`add_element_massfraction(element, massfraction)`

Add an element as a component to a material as a fraction of the material mass. Can only add elements to materials defined as composite.

Inputs:

element - `pyg4ometry.geant4.Material.Element` instance massfraction - float, $0.0 < \text{massfraction} \leq 1.0$

`add_element_natoms(element, natoms)`

Add an element as a component to a material as a number of atoms in the material molecule. Can only add elements to materials defined as composite.

Inputs:

element - `pyg4ometry.geant4.Material.Element` instance natoms - int, number of atoms in the compound molecule

add_material(*material*, *fractionmass*)

Add a material as a component to another material (mixture) as a fraction of the mixture mass. Can only add new materials to materials defined as composite.

Inputs:

material - pyg4ometry.geant4.Material.Material instance *massfraction* - float, $0.0 < \text{massfraction} \leq 1.0$

set_pressure(*value*, *unit*='pascal')

set_temperature(*value*, *unit*='K')

__str__()

Return str(self).

addProperty(*name*, *matrix*)

Add a material property from a matrix.

Parameters

- **name** (*str*) – key of the material property
- **matrix** (*Matrix*) – matrix defining the value(s) of the property

addVecProperty(*name*, *e*, *v*, *eunit*='eV', *vunit*='')

Add a property from an energy and a value vector to this object.

Parameters

- **name** (*str*) – key of property
- **e** (*list* or *numpy.array* - *shape* (1,)) – energy list/vector in units of *eunit*
- **v** (*list* or *numpy.array* - *shape* (1,)) – value list/vector in units of *vunit*
- **eunit** (*str*) – unit for the energy vector (default: eV)
- **vunit** (*str*) – unit for the value vector (default: unitless)

addConstProperty(*name*, *value*, *vunit*='')

Add a constant scalar property to this object.

Parameters

- **name** (*str*) – key of property
- **value** (*str*, *float*, *int*) – constant value for this property
- **vunit** (*str*) – unit for the value vector (default: unitless)

class pyg4ometry.geant4._Material.Element(***kwargs*)

Bases: *MaterialBase*

This class provides an interface to GDML material definitions. Because of the different options for constructing a material instance the constructor is kwarg only. Proxy methods are provided to instantiate particular types of material. Those proxy methods are:

ElementSimple ElementIsotopeMixture

It is possible to instantiate a material directly through kwargs. The possible kwargs are (but note some are mutually exclusive): *name* - string *symbol* - string *Z* - int *A* - int *n_comp* - int

add_isotope(*isotope*, *abundance*)

Add an isotope as a component to an element as an abundance fraction in the element.

Inputs:

element - pyg4ometry.geant4.Material.Isotope instance abundance - float, $0.0 < \text{abundance} \leq 1.0$

class pyg4ometry.geant4._Material.Isotope(*name*, *Z*, *N*, *a*, *registry=None*)

Bases: [*MaterialBase*](#)

This class that handles isotopes as components of composite materials. An element can be defined as a mixture of isotopes.

Inputs:

name - string Z - int, atomic number N - int, mass number a - float, molar weight in g/mole

Package Contents

Classes

<i>_OverlapType</i>	
<i>LogicalVolume</i>	LogicalVolume : G4LogicalVolume
<i>PhysicalVolume</i>	
<i>AssemblyVolume</i>	AssemblyVolume : similar to a logical volume but does not have a sense of
<i>_PhysicalVolume</i>	PhysicalVolume : G4VPhysicalVolume, G4PVPlacement
<i>_OverlapType</i>	
<i>ReplicaVolume</i>	ReplicaVolume: G4PVReplica
<i>_ReplicaVolume</i>	ReplicaVolume: G4PVReplica
<i>ParameterisedVolume</i>	
<i>_PhysicalVolume</i>	PhysicalVolume : G4VPhysicalVolume, G4PVPlacement
<i>DivisionVolume</i>	DivisionVolume: G4PVDivision
<i>LogicalVolume</i>	LogicalVolume : G4LogicalVolume
<i>SurfaceBase</i>	
<i>SkinSurface</i>	
<i>PhysicalVolume</i>	
<i>SurfaceBase</i>	
<i>BorderSurface</i>	
<i>Registry</i>	
<i>GeometryComplexityInformation</i>	
<i>MaterialBase</i>	
<i>Material</i>	This class provides an interface to GDML material definitions.
<i>Element</i>	This class provides an interface to GDML material definitions. Because of the different options
<i>Isotope</i>	This class that handles isotopes as components of composite materials. An element can be

Functions

<i>IsAReplica</i> (logicalVolume)	Utility function to test if an LV is really a replica volume. A replica is a special case
<i>_solid2tessellated</i> (solid)	
<i>solidName</i> (var)	
<i>removeprefix</i> (string, prefix, /)	
<i>AnalyseGeometryComplexity</i> (logicalVolume)	Analyse a geometry tree starting from a logical volume.
<i>_UpdateComplexity</i> (lv, info)	
<i>AnalyseGeometryStructure</i> (registry[, lv_name, debug, ...])	Produce a pandas dataframe representing the structure of the geometry.
<i>DumpGeometryStructureTree</i> (lv[, depth])	
<i>getNistMaterialDict</i> ()	
<i>getNistMaterialList</i> ()	
<i>getNistElementZToName</i> ()	
<i>_getClassVariables</i> (obj)	
<i>_makeNISTCompoundList</i> ()	
<i>_safeName</i> (name)	
<i>loadNISTMaterialDict</i> ()	
<i>nist_materials_name_lookup</i> (name)	
<i>nist_materials_z_lookup</i> (z)	
<i>nist_element_2geant4Element</i> (name[, reg])	This returns an instance of either ElementSimple or ElementIsotopeMixture.
<i>nist_material_2geant4Material</i> (name[, reg])	
<i>MaterialPredefined</i> (name[, registry])	Proxy method to construct a NIST compound material - this is just a handle as nothing
<i>MaterialArbitrary</i> (name[, registry])	Just a name of a material. WARNING: It is left to the
<i>MaterialSingleElement</i> (name, atomic_number, ...[, ...])	Proxy method to construct a simple material - full description of the element contained is contained in one definition
<i>MaterialCompound</i> (name, density, number_of_components)	Proxy method to construct a composite material - can be any mixture of Elements and/or Materials
<i>ElementSimple</i> (name, symbol, Z, A[, registry])	Proxy method to construct a simple element - full description of the element contained is contained in one definition
<i>ElementIsotopeMixture</i> (name, symbol, n_comp[, ...])	Proxy method to construct a composite element - a mixture of predefined isotopes

Attributes

`_nistMaterialDict``_nistMaterialList``_nistElementZToName`

`pyg4ometry.geant4.IsAReplica(logicalVolume)`

Utility function to test if an LV is really a replica volume. A replica is a special case where we have an in-effect dummy mother and is detectable by there only being 1 daughter and it being a `ReplicaVolume` instance.

class `pyg4ometry.geant4._OverlapType`

protrusion = 1

overlap = 2

coplanar = 3

`pyg4ometry.geant4._solid2tessellated(solid)`

class `pyg4ometry.geant4.LogicalVolume(solid, material, name, registry=None, addRegistry=True, **kwargs)`

LogicalVolume : G4LogicalVolume

Parameters

- **solid** –
- **material** –
- **name** (*str*) –
- **registry** –
- **addRegistry** (*bool*) –

__repr__()

Return repr(self).

reMesh(recursive=False)

Regenerate the visualisation for this logical volume. Required if the geometry is modified and overlap checking is subsequently required or revisualisation.

add(physicalVolume)

Add physical volume to this logicalVolume

Parameters

physicalVolume (`PhysicalVolume`, `ReplicaVolume`, `ParameterisedVolume`, `DivisionVolume`) – physical volume to add

addBDSIMObject(bdsimobject)

_getPhysicalDaughterMesh(pv, warn=True)

Return a (cloned from the lv) mesh of a given pv with rotation,scale, translation evaluated.

cullDaughtersOutsideSolid(*solid*, *rotation*=None, *position*=None)

Given a solid with a placement rotation and position inside this logical volume, remove (cull) any daughters that would not lie entirely within it. The rotation and position are applied to the solid w.r.t. the frame of this logical volume.

Parameters

- **rotation** (*list(float, float, float)* or None - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or None - 3 values in mm) – translation of the solid w.r.t. this lv

transformDaughters(*rotation*=(0, 0, 0), *position*=(0, 0, 0), *runit*='rad', *punit*='mm')

Transform the daughter volumes (without clipping)

Parameters

- **rotation** (*list(float, float, float)* or None - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or None - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)

replaceSolid(*newSolid*, *rotation*=(0, 0, 0), *position*=(0, 0, 0), *runit*='rad', *punit*='mm')

Replace the outer solid with optional position and rotation

Parameters

- **newSolid** (*pyg4ometry.geant4.solid*) – object to clip the geometry to
- **rotation** (*list(float, float, float)* or None - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or None - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)

clipGeometry(*newSolid*, *rotation*=(0, 0, 0), *position*=(0, 0, 0), *runit*='rad', *punit*='mm', *replace*=False, *depth*=0, *solidUsageCount*=_defaultdict(int), *lvUsageCount*=_defaultdict(int))

Clip the geometry to newSolid, placed with rotation and position.

Parameters

- **newSolid** (*pyg4ometry.geant4.solid*) – object to clip the geometry to
- **rotation** (*list(float, float, float)* or None - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or None - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)
- **replace** (*bool*) – replace the outer solid or not

- **depth** (*int*) – recursion depth (DO NOT USE)
- **solidUsageCount** (*defaultdict*) – solid name dictionary for replacement recursion (DO NOT USE)
- **lvUsageCount** (*defaultdict*) – lv name dictionary for replacement recursion (DO NOT USE)

changeSolidAndTrimGeometry(*newSolid*, *rotation*=(0, 0, 0), *position*=(0, 0, 0), *runit*='rad', *punit*='mm')

Change the solid of this logical volume, remove any daughters that will lie outside it, and form new Boolean intersection solids for any daughters that cross the boundary (intersect) it. The rotation and translation are with respect to the original frame and all daughters will now be replaced with respect to the new frame. Therefore, that same rotation and translation should be use to re-place this logical volume if desired. The default is none though, so the frame would nominally remain the same.

Parameters

- **newSolid** (*any of pyg4ometry.geant4.solid*) – new solid to use for this logical volume
- **rotation** (*list(float, float, float) or None - 3 values in radians*) – Tait-Bryan rotation for the new solid w.r.t. old frame (i.e. current lv)
- **position** (*list(float, float, float) or None - 3 values in mm*) – translation for the new solid w.r.t. old frame (i.e. current lv)

checkOverlaps(*recursive=False*, *coplanar=False*, *debugIO=False*, *printOut=True*, *nOverlapsDetected=[0]*)

Check based on the meshes in each logical volume if there are any geometrical overlaps. By default, overlaps are checked between daughter volumes and with the mother volume itself (protrusion). Coplanar overlaps may also be checked (default on).

Print out will be given for any overlaps detected and the visualiser will show the colour coded overlaps.

Parameters

- **recursive** – bool - Whether to descend into the daughter volumes and check their contents also.
- **coplanar** – bool - Whether to check for coplanar overlaps
- **debugIO** – bool - Print out for every check made
- **printOut** – bool - (internal) Whether to print out a summary of N overlaps detected
- **nOverlapsDetected** – [int] - (internal) counter for recursion - ignore

setSolid(*solid*)

Set (replace) the outer solid. Does not change the placement of the daughters in the volume. If there is a transformation then use `replaceSolid`

makeSolidTessellated()

Make solid tessellated. Sometimes useful when a boolean cannot be visualised in Geant4

addAuxiliaryInfo(*auxiliary*)

Add auxiliary information to logical volume :param auxiliary: auxiliary information for the logical volume
:type auxiliary: tuple or list

extent(*includeBoundingSolid=False*)

Compute the axis aligned extent of the logical volume.

Parameters

- **includeBoundingSolid** (*bool*) – Include the bounding solid or not

depth(*depth=0*)

Depth for LV-PV tree

clipSolid(*lengthSafety=1e-06*)

Assuming the solid of this LV is a Box, reduce its dimensions and re-placement all daughters to reduce the box to the minimum (axis-aligned) bounding box. This updates the dimensions of the box and the translation of each daughter physical volume.

Parameters

lengthSafety (*float*) – safety length

makeLogicalPhysicalNameSets()

Return a set of logical names and physical names used in this logical volume and any daughters. This is built up recursively by checking all daughters etc.

findLogicalByName(*name*)

Return a list of LogicalVolume instances used inside this logical volume as daughters (at any level inside) with the given name.

Parameters

name (*str*) – lv name

makeMaterialNameSet()

Return a set of material names used in this logical volume and any daughters. This is built up recursively by checking all daughters etc etc.

assemblyVolume(*materialName='G4_AIR0x7f8441173ac0'*)

Return an assembly volume of this logical volume, in effect removing the solid and material of this logical volume, but retaining all of the relative daughter placements.

makeWorldVolume(*worldMaterial='G4_Galactic'*)

This will create a container box according to the extents of this logical volume: an axis-aligned bounding-box. It will be filled with the given material (predefined by name) and assigned as the world volume (outermost) of the registry according to this logical volume.

dumpStructure(*depth=0*)

class pyg4ometry.geant4.**PhysicalVolume**(*rotation, position, logicalVolume, name, motherVolume, registry=None, copyNumber=0, addRegistry=True, scale=None*)

PhysicalVolume : G4VPhysicalVolume, G4PVPlacement

Parameters

- **rotation** – [float,float,float] - rotations about x,y,z axes of mother volume
- **position** – [float,float,float] - translation with respect to mother volume
- **logicalVolume** – *pyg4ometry.geant4.LogicalVolume* - instance to place
- **name** – str - name of this placement
- **motherVolume** – *pyg4ometry.geant4.LogicalVolume* - mother volume to place into
- **registry** – *pyg4ometry.geant4.Registry* - registry to register to
- **copyNumber** – int - copy number of the placement that can be used for sensitivity
- **addRegistry** – bool - whether to add to the registry or not

__repr__()

Return repr(self).

extent(*includeBoundingSolid=True*)

getAABBMesh()

return CSG.core (symmetric around the origin) axis aligned bounding box mesh

class pyg4ometry.geant4.**AssemblyVolume**(*name, registry=None, addRegistry=True*)

AssemblyVolume : similar to a logical volume but does not have a sense of shape, material or field :param name: of assembly volume :type name: str :param registry: :type registry: :param addRegistry: :type addRegistry: bool

__repr__()

Return repr(self).

add(*physicalVolume*)

_getDaughterMeshesByName(*name*)

_getDaughterMeshesByIndex(*index*)

_getDaughterMeshes()

Get daughter meshes for overlap checking. return [daughterMesh,...],[daughterBoundingMesh,...][daughterName,...]

_getPVMeshes(*pv*)

Can technically return more than one mesh if the daughter is also an assembly.

_getPhysicalDaughterMesh(*pv, warn=True*)

Return a (cloned from the lv) mesh of a given pv with rotation,scale, translation evaluated.

clipGeometry(*newSolid, rotation=(0, 0, 0), position=(0, 0, 0), runit='rad', punit='mm', replace=False, depth=0, solidUsageCount=_defaultdict(int), lvUsageCount=_defaultdict(int)*)

Clip the geometry to newSolid, placed with rotation and position.

extent(*includeBoundingSolid=True*)

depth(*depth=0*)

Depth for LV-PV tree

getAABBMesh()

return CSG.core (symmetric around the origin) axis aligned bounding box mesh

logicalVolume(*material='G4_Galactic', solidName='worldSolid'*)

Return an logical volume of this this assembly volume, in effect adding a cuboid solid and material of this logical volume, retaining all of the relative daughter placements.

makeWorldVolume(*material='G4_Galactic'*)

dumpStructure(*depth=0*)

class pyg4ometry.geant4.**_PhysicalVolume**(*rotation, position, logicalVolume, name, motherVolume, registry=None, copyNumber=0, addRegistry=True, scale=None*)

PhysicalVolume : G4VPhysicalVolume, G4PVPlacement

Parameters

- **rotation** – [float,float,float] - rotations about x,y,z axes of mother volume
- **position** – [float,float,float] - translation with respect to mother volume
- **logicalVolume** – [pyg4ometry.geant4.LogicalVolume](#) - instance to place

- **name** – str - name of this placement
- **motherVolume** – [pyg4ometry.geant4.LogicalVolume](#) - mother volume to place into
- **registry** – [pyg4ometry.geant4.Registry](#) - registry to register to
- **copyNumber** – int - copy number of the placement that can be used for sensitivity
- **addRegistry** – bool - whether to add to the registry or not

__repr__()

Return repr(self).

extent(*includeBoundingSolid=True*)

getAABBMesh()

return CSG.core (symmetric around the origin) axis aligned bounding box mesh

class [pyg4ometry.geant4._OverlapType](#)

protrusion = 1

overlap = 2

coplanar = 3

class [pyg4ometry.geant4.ReplicaVolume](#)(*name, logicalVolume, motherVolume, axis, nreplicas, width, offset=0, registry=None, addRegistry=True, wunit='mm', ounit='mm'*)

Bases: [pyg4ometry.geant4.PhysicalVolume.PhysicalVolume](#)

ReplicaVolume: G4PVReplica

Parameters

- **name** – of physical volume
- **logical** – volume to be placed
- **mother** – logical volume,
- **axis** – kXAxis,kYAxis,kZAxis,kRho,kPhi
- **ncopies** – number of replicas
- **width** – spacing between replicas along axis
- **offset** – of grid

class [Axis](#)

kXAxis = 1

kYAxis = 2

kZAxis = 3

kRho = 4

kPhi = 5

GetAxisName()

_checkInternalOverlaps(*debugIO=False, nOverlapsDetected=[0]*)

Check if there are overlaps with the nominal mother volume. ie it possible to provide an incorrect mother volume / logical volume and parameterisation.

createReplicaMeshes()

getPhysicalVolumes()

return a list of temporary (ie not added to the relevant registry) PhysicalVolume instances with appropriate transforms including any daughter ReplicaVolumes.

The exception is for kRho axis where new unique solids and logical volumes are required. Therefore, these are added to the registry and inadvertently to the mother LV as PVS.

__repr__()

extent(*includeBoundingSolid=True*)

class pyg4ometry.geant4._ReplicaVolume(*name, logicalVolume, motherVolume, axis, nreplicas, width, offset=0, registry=None, addRegistry=True, wunit='mm', ounit='mm'*)

Bases: [pyg4ometry.geant4.PhysicalVolume.PhysicalVolume](#)

ReplicaVolume: G4PVReplica

Parameters

- **name** – of physical volume
- **logical** – volume to be placed
- **mother** – logical volume,
- **axis** – kXAxis, kYAxis, kZAxis, kRho, kPhi
- **ncopies** – number of replicas
- **width** – spacing between replicas along axis
- **offset** – of grid

class Axis

kXAxis = 1

kYAxis = 2

kZAxis = 3

kRho = 4

kPhi = 5

GetAxisName()

_checkInternalOverlaps(*debugIO=False, nOverlapsDetected=[0]*)

Check if there are overlaps with the nominal mother volume. ie it possible to provide an incorrect mother volume / logical volume and parameterisation.

createReplicaMeshes()

getPhysicalVolumes()

return a list of temporary (ie not added to the relevant registry) PhysicalVolume instances with appropriate transforms including any daughter ReplicaVolumes.

The exception is for kRho axis where new unique solids and logical volumes are required. Therefore, these are added to the registry and inadvertently to the mother LV as PVS.

__repr__()**extent**(includeBoundingSolid=True)

```
class pyg4ometry.geant4.ParameterisedVolume(name, logicalVolume, motherVolume, ncopies, paramData,
                                             transforms, registry=None, addRegistry=True)
```

Bases: `pyg4ometry.geant4.ReplicaVolume.ReplicaVolume`

ParametrisedVolume :param name: of parametrised volume :type name: str :param logical: volume to be placed :type logical: logicalVolume :param mother: volume logical volume :type mother: logicalVolume :param ncopies: number of parametrised volumes :type ncopies: int

```
class BoxDimensions(pX, pY, pZ, lunit='mm')
```

```
class TubeDimensions(pRMin, pRMax, pDz, pSPhi, pDPhi, lunit='mm', aunit='rad')
```

```
class ConeDimensions(pRMin1, pRMax1, pRMin2, pRMax2, pDz, pSPhi, pDPhi, lunit='mm', aunit='rad')
```

```
class OrbDimensions(pRMax, lunit='mm')
```

```
class SphereDimensions(pRMin, pRMax, pSPhi, pDPhi, pSTheta, pDTheta, lunit='mm', aunit='rad')
```

```
class TorusDimensions(pRMin, pRMax, pRTor, pSPhi, pDPhi, lunit='mm', aunit='rad')
```

```
class HypeDimensions(innerRadius, outerRadius, innerStereo, outerStereo, lenZ, lunit='mm', aunit='rad')
```

```
class ParaDimensions(pX, pY, pZ, pAlpha, pTheta, pPhi, lunit='mm', aunit='rad')
```

```
class TrdDimensions(pX1, pX2, pY1, pY2, pZ, lunit='mm')
```

```
class TrapDimensions(pDz, pTheta, pDPhi, pDy1, pDx1, pDx2, pAlp1, pDy2, pDx3, pDx4, pAlp2,
                      lunit='mm', aunit='rad')
```

```
class PolyconeDimensions(pSPhi, pDPhi, pZpl, pRMin, pRMax, lunit='mm', aunit='rad')
```

```
class PolyhedraDimensions(pSPhi, pDPhi, numSide, pZpl, pRMin, pRMax, lunit='mm', aunit='rad')
```

```
class EllipsoidDimensions(pxSemiAxis, pySemiAxis, pzSemiAxis, pzBottomCut, pzTopCut, lunit='mm')
```

createParameterisedMeshes()**__repr__()****extent**(includeBoundingSolid=True)

```
class pyg4ometry.geant4._PhysicalVolume(rotation, position, logicalVolume, name, motherVolume,
                                           registry=None, copyNumber=0, addRegistry=True,
                                           scale=None)
```

PhysicalVolume : G4VPhysicalVolume, G4PVPlacement

Parameters

- **rotation** – [float,float,float] - rotations about x,y,z axes of mother volume

- **position** – [float,float,float] - translation with respect to mother volume
- **logicalVolume** – [pyg4ometry.geant4.LogicalVolume](#) - instance to place
- **name** – str - name of this placement
- **motherVolume** – [pyg4ometry.geant4.LogicalVolume](#) - mother volume to place into
- **registry** – [pyg4ometry.geant4.Registry](#) - registry to register to
- **copyNumber** – int - copy number of the placement that can be used for sensitivity
- **addRegistry** – bool - whether to add to the registry or not

__repr__()

Return repr(self).

extent(*includeBoundingSolid=True*)

getAABBMesh()

return CSG.core (symmetric around the origin) axis aligned bounding box mesh

```
class pyg4ometry.geant4.DivisionVolume(name, logicalVolume, motherVolume, axis, ndivisions=-1,
                                       width=-1, offset=0, registry=None, addRegistry=True,
                                       unit='mm')
```

Bases: [pyg4ometry.geant4.PhysicalVolume.PhysicalVolume](#)

DivisionVolume: G4PVDivision

Parameters

- **name** – of physical volume
- **logical** – volume to be placed
- **mother** – logical volume,
- **axis** – kXAxis,kYAxis,kZAxis,kRho,kPhi
- **ncopies** – number of replicas
- **width** – spacing between replicas along axis
- **offset** – of grid

class Axis

kXAxis = 1

kYAxis = 2

kZAxis = 3

kRho = 4

kPhi = 5

getMotherSize()

checkAxis(*allowed_axes*)

divideBox(*offset, width, ndiv*)

divideTubs(*offset, width, ndiv*)


```

divideCons(offset, width, ndiv)
dividePara(offset, width, ndiv)
divideTrd(offset, width, ndiv)
dividePolycone(offset, width, ndiv)
dividePolyhedra(offset, width, ndiv)
createDivisionMeshes()
__repr__()
extent(includeBoundingSolid=True)

```

class pyg4ometry.geant4.**LogicalVolume**(*solid, material, name, registry=None, addRegistry=True, **kwargs*)

LogicalVolume : G4LogicalVolume

Parameters

- **solid** –
- **material** –
- **name** (*str*) –
- **registry** –
- **addRegistry** (*bool*) –

__repr__()
Return repr(self).

reMesh(*recursive=False*)
Regenerate the visualisation for this logical volume. Required if the geometry is modified and overlap checking is subsequently required or revisualisation.

add(*physicalVolume*)
Add physical volume to this logicalVolume

Parameters

physicalVolume (*PhysicalVolume, ReplicaVolume, ParameterisedVolume, DivisionVolume*) – physical volume to add

addBDSIMObject(*bdsimobject*)

_getPhysicalDaughterMesh(*pv, warn=True*)
Return a (cloned from the lv) mesh of a given pv with rotation, scale, translation evaluated.

cullDaughtersOutsideSolid(*solid, rotation=None, position=None*)
Given a solid with a placement rotation and position inside this logical volume, remove (cull) any daughters that would not lie entirely within it. The rotation and position are applied to the solid w.r.t. the frame of this logical volume.

Parameters

- **rotation** (*list(float, float, float)* or *None* – 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or *None* – 3 values in mm) – translation of the solid w.r.t. this lv

transformDaughters(rotation=(0, 0, 0), position=(0, 0, 0), runit='rad', punit='mm')

Transform the daughter volumes (without clipping)

Parameters

- **rotation** (*list(float, float, float)* or *None* - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or *None* - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)

replaceSolid(newSolid, rotation=(0, 0, 0), position=(0, 0, 0), runit='rad', punit='mm')

Replace the outer solid with optional position and rotation

Parameters

- **newSolid** (*pyg4ometry.geant4.solid*) – object to clip the geometry to
- **rotation** (*list(float, float, float)* or *None* - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or *None* - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)

clipGeometry(newSolid, rotation=(0, 0, 0), position=(0, 0, 0), runit='rad', punit='mm', replace=False, depth=0, solidUsageCount=_defaultdict(int), lvUsageCount=_defaultdict(int))

Clip the geometry to newSolid, placed with rotation and position.

Parameters

- **newSolid** (*pyg4ometry.geant4.solid*) – object to clip the geometry to
- **rotation** (*list(float, float, float)* or *None* - 3 values in radians) – Tait-Bryan angles for rotation of the solid w.r.t. this lv
- **position** (*list(float, float, float)* or *None* - 3 values in mm) – translation of the solid w.r.t. this lv
- **runit** (*str*) – angular unit for rotation (rad,deg)
- **punit** (*str*) – length unit for position (m,mm,km)
- **replace** (*bool*) – replace the outer solid or not
- **depth** (*int*) – recursion depth (DO NOT USE)
- **solidUsageCount** (*defaultdict*) – solid name dictionary for replacement recursion (DO NOT USE)
- **lvUsageCount** (*defaultdict*) – lv name dictionary for replacement recursion (DO NOT USE)

changeSolidAndTrimGeometry(newSolid, rotation=(0, 0, 0), position=(0, 0, 0), runit='rad', punit='mm')

Change the solid of this logical volume, remove any daughters that will lie outside it, and form new Boolean intersection solids for any daughters that cross the boundary (intersect) it. The rotation and translation are with respect to the original frame and all daughters will now be replaced with respect to the new frame.

Therefore, that same rotation and translation should be use to re-place this logical volume if desired. The default is none though, so the frame would nominally remain the same.

Parameters

- **newSolid** (*any of pyg4ometry.geant4.solid*) – new solid to use for this logical volume
- **rotation** (*list(float, float, float) or None - 3 values in radians*) – Tait-Bryan rotation for the new solid w.r.t. old frame (i.e. current lv)
- **position** (*list(float, float, float) or None - 3 values in mm*) – translation for the new solid w.r.t. old frame (i.e. current lv)

checkOverlaps(*recursive=False, coplanar=False, debugIO=False, printOut=True, nOverlapsDetected=0*)

Check based on the meshes in each logical volume if there are any geometrical overlaps. By default, overlaps are checked between daughter volumes and with the mother volume itself (protrusion). Coplanar overlaps may also be checked (default on).

Print out will be given for any overlaps detected and the visualiser will show the colour coded overlaps.

Parameters

- **recursive** – bool - Whether to descend into the daughter volumes and check their contents also.
- **coplanar** – bool - Whether to check for coplanar overlaps
- **debugIO** – bool - Print out for every check made
- **printOut** – bool - (internal) Whether to print out a summary of N overlaps detected
- **nOverlapsDetected** – [int] - (internal) counter for recursion - ignore

setSolid(*solid*)

Set (replace) the outer solid. Does not change the placement of the daughters in the volume. If there is a transformation then use `replaceSolid`

makeSolidTessellated()

Make solid tessellated. Sometimes useful when a boolean cannot be visualised in Geant4

addAuxiliaryInfo(*auxiliary*)

Add auxiliary information to logical volume :param auxiliary: auxiliary information for the logical volume
:type auxiliary: tuple or list

extent(*includeBoundingSolid=False*)

Compute the axis aligned extent of the logical volume.

Parameters

includeBoundingSolid (*bool*) – Include the bounding solid or not

depth(*depth=0*)

Depth for LV-PV tree

clipSolid(*lengthSafety=1e-06*)

Assuming the solid of this LV is a Box, reduce its dimensions and re-placement all daughters to reduce the box to the minimum (axis-aligned) bounding box. This updates the dimensions of the box and the translation of each daughter physical volume.

Parameters

lengthSafety (*float*) – safety length

makeLogicalPhysicalNameSets()

Return a set of logical names and physical names used in this logical volume and any daughters. This is built up recursively by checking all daughters etc.

findLogicalByName(name)

Return a list of LogicalVolume instances used inside this logical volume as daughters (at any level inside) with the given name.

Parameters

name (*str*) – lv name

makeMaterialNameSet()

Return a set of material names used in this logical volume and any daughters. This is built up recursively by checking all daughters etc etc.

assemblyVolume(materialName='G4_AIR0x7f8441173ac0')

Return an assembly volume of this logical volume, in effect removing the solid and material of this logical volume, but retaining all of the relative daughter placements.

makeWorldVolume(worldMaterial='G4_Galactic')

This will create a container box according to the extents of this logical volume: an axis-aligned bounding-box. It will be filled with the given material (predefined by name) and assigned as the world volume (outermost) of the registry according to this logical volume.

dumpStructure(depth=0)

```
class pyg4ometry.geant4.SurfaceBase(name, type, surface_property, registry, addRegistry)
```

```
    _chkType(obj, t, prop)
```

```
class pyg4ometry.geant4.SkinSurface(name, volumeref, surface_property, registry, addRegistry=True)
```

```
    Bases: pyg4ometry.geant4.SurfaceBase.SurfaceBase
```

```
    __repr__()
```

```
class pyg4ometry.geant4.PhysicalVolume(rotation, position, logicalVolume, name, motherVolume,
                                         registry=None, copyNumber=0, addRegistry=True, scale=None)
```

PhysicalVolume : G4VPhysicalVolume, G4PVPlacement

Parameters

- **rotation** – [float,float,float] - rotations about x,y,z axes of mother volume
- **position** – [float,float,float] - translation with respect to mother volume
- **logicalVolume** – [pyg4ometry.geant4.LogicalVolume](#) - instance to place
- **name** – str - name of this placement
- **motherVolume** – [pyg4ometry.geant4.LogicalVolume](#) - mother volume to place into
- **registry** – [pyg4ometry.geant4.Registry](#) - registry to register to
- **copyNumber** – int - copy number of the placement that can be used for sensitivity
- **addRegistry** – bool - whether to add to the registry or not

```
__repr__()
```

Return repr(self).

```
extent(includeBoundingSolid=True)
```

```

getAABBMesh()
    return CSG.core (symmetric around the origin) axis aligned bounding box mesh

class pyg4ometry.geant4.SurfaceBase(name, type, surface_property, registry, addRegistry)

    _chkType(obj, t, prop)

class pyg4ometry.geant4.BorderSurface(name, physref1, physref2, surface_property, registry,
                                       addRegistry=True)

    Bases: pyg4ometry.geant4.SurfaceBase.SurfaceBase

    __repr__()

pyg4ometry.geant4.solidName(var)

pyg4ometry.geant4.removeprefix(string, prefix, /)

```

Parameters

- **string** (*str*) –
- **prefix** (*str*) –

Return type

str

```
class pyg4ometry.geant4.Registry
```

Object to store geometry for input and output. All of the pyg4ometry classes can be used without storing them in the Registry. The registry is used to write the GDML output file. A registry needs to be used in conjunction with GDML Define objects for evaluation of expressions.

```
clear()
```

Empty all internal structures

```
getExpressionParser()
```

```
registerSolidEdit(solid)
```

```
addMaterial(material, dontWarnIfAlreadyAdded=False)
```

Register a material with this registry.

Parameters

material (*Material*) – Material object for storage

```
transferMaterial(material, incrementRenameDict={}, userRenameDict=None)
```

Transfer a material to this registry. This can operate on a Material, an Isotope and an Element instance.

```
addSolid(solid)
```

Register a solid with this registry.

Parameters

solid (*One of the geant4 solids*) – Solid object for storage

```
transferSolid(solid, incrementRenameDict={}, userRenameDict=None)
```

Transfer a solid to this registry. Doesn't handle any members' transferal - only the solid itself.

Parameters

solid (*One of the geant4 solids*) – Solid object for storage

addLogicalVolume(*volume*)

Register a logical volume with this registry. Also accepts Assembly Volumes.

Parameters

volume ([LogicalVolume](#)) – LogicalVolume object for storage

transferLogicalVolume(*volume*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a logical volume to this registry. Doesn't handle any members' transferal - only the logical volume itself.

addPhysicalVolume(*volume*)

Registry a physical volume with this registry.

Parameters

volume ([PhysicalVolume](#)) – PhysicalVolume object for storage

transferPhysicalVolume(*volume*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a physical volume to this registry. Doesn't handle any members' transferal - only the physical volume itself.

addSurface(*surface*)

Register a surface with this registry.

Parameters

surface ([pyg4ometry.geant4.BorderSurface](#) or [pyg4ometry.geant4.SkinSurface](#)) – Surface

transferSurface(*surface*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a surface to this registry.

addAuxiliary(*auxiliary*)**addDefine**(*define*)

Register a define with this registry.

Parameters

define ([Constant](#), [Quantity](#), [Variable](#), [Matrix](#)) – Definition object for storage

transferDefine(*define*, *incrementRenameDict*={}, *userRenameDict*=None)

Transfer a single define from another registry to this one. No checking on previous registry or not.

transferDefines(*var*, *otherRegistry*, *incrementRenameDict*={}, *userRenameDict*=None)

This function tolerates all types of defines including vector ones.

Transfer defines from one registry to another recursively. A define may not be part of the old registry so won't be added to this one. A define may be a vector or composite and its 'bits' may be in the (old) registry so each part should be checked.

In " $3x + 2$ ", "x" would be a variable". In " $3.5*2$ " there would be no variables.

setWorld(*worldIn*)

The argument can either be the name of logical volume of the world or the [pyg4ometry.geant4.LogicalVolume](#) instance of the world volume. The term world is used to refer to the outermost volume of the hierarchy.

setWorldVolume(*worldIn*)

An alias for some of us who can't remember.

_orderMaterialList(*materials*, *materials_ordered*=[])

orderMaterials()

Need to have a ordered list of all material entities for writing to GDML. GDML needs to have the isotopes/elements/materials defined in use order

orderLogicalVolumes(*lvName*, *first=True*)

Need to have an ordered list from most basic (solid) object upto physical/logical volumes for writing to GDML. GDML needs to have the solids/booleans/volumes defined in order

addVolumeRecursive(*volume*, *collapseAssemblies=False*, *incrementRenameDict=None*, *userRenameDict=None*)

Transfer a volume hierarchy to this registry. Any objects that had a registry set to another will be set to this one and will be owned by it effectively. :param volume: PhysicalVolume or LogicalVolume or AssemblyVolume. :type volume: pyg4ometry.geant4.PhysicalVolume, pyg4ometry.geant4.LogicalVolume, pyg4ometry.geant4.AssemblyVolume. :param collapseAssemblies: if True, daughters of AssemblyVolume's will be attached directly to the mother of the assembly and the AssemblyVolume itself will be eliminated from the geometry tree :param incrementRenameDict: ignore - dictionary used internally for potentially incrementing names :param userRenameDict: a dictionary of find/replace regex strings to be used to rename volumes/materials/etc.

In the case where some object or variable has a name (e.g. 'X') that already exists in this registry, it will be incremented to 'X_1'.

addAndCollapseAssemblyVolumeRecursive(*assemblyPV*, *motherVol*, *positions*, *rotations*, *scales*, *names*, *incrementRenameDict*, *userRenameDict*)

Transfer and collapse an AssemblyVolume hierarchy to this registry. Daughter volumes are copied, renamed, and attached to the supplied mother LogicalVolume with the correct position/rotation. Any objects that had a registry set to another will be set to this one and will be owned by it effectively. :param assemblyPV: the PhysicalVolume that places an AssemblyVolume :param motherVol: the LogicalVolume to which all daughters of the AssemblyVolume (including any daughters of nested AssemblyVolume's) should be attached :param positions: a list (initially empty) to hold position objects for each AssemblyVolume in a hierarchy of nested assemblies (used to correctly set the transformation of daughters within the motherVol) :param rotations: a list (initially empty) to hold rotation objects for each AssemblyVolume in a hierarchy of nested assemblies (used to correctly set the transformation of daughters within the motherVol) :param scales: a list (initially empty) to hold scale objects for each AssemblyVolume in a hierarchy of nested assemblies (used to correctly set the transformation of daughters within the motherVol) :param names: a list (initially empty) to hold the names of each AssemblyVolume in a hierarchy of nested assemblies (used to set unique names for the daughters within the motherVol) :param incrementRenameDict: ignore - dictionary used internally for potentially incrementing names :param userRenameDict: a dictionary of find/replace regex strings to be used to rename volumes/materials/etc.

transferSolidDefines(*solid*, *incrementRenameDict={}*, *userRenameDict=None*)

For each parameter in a given solid (unique to each) check if it's a define and transfer that over.

volumeTree(*lvName*)

Not sure what this method is used for

solidTree(*solidName*)

Not sure what this method is used for

getWorldVolume()**printStats()****structureAnalysis(*lv_name=None*, *debug=False*, *level=0*, *df=None*)**

`_findDictByName(dic, nameFragment)`

Find a object which name matches (or partially matches) nameFragment, returns a list of objects

`findSolidByName(nameFragment='box')`

Find a solid which name matches (or partially matches) nameFragment, returns a list of solids

`findMaterialByName(nameFragment='G4_AIR')`

Find a material which name matches (or partially matches) nameFragment, returns a list of materials

`findLogicalVolumeByName(nameFragment='World')`

Find a logical volume which name matches (or partially matches) nameFragment, returns a list of LogicalVolumes

`findPhysicalVolumeByName(nameFragment)`

Find a physical volume which name matches (or partially matches) nameFragment, returns a list of LogicalVolumes

`class pyg4ometry.geant4.GeometryComplexityInformation`

`printSummary(boolDepthLimit=3)`

`pyg4ometry.geant4.AnalyseGeometryComplexity(logicalVolume)`

Analyse a geometry tree starting from a logical volume. Produces an instance of [*GeometryComplexityInformation*](#) with summary information. Provides:

- count per solid type
- number of daughters per logical volume
- dictionary of N daughters for each logical volume name
- depth count of Boolean solids

ie a Boolean of a Boolean returns 2, a Boolean of two primitives returns 1

- a dictionary of boolean depth for each logical volume name

Example:

```
info = AnalyseGeometryComplexity(lv)
info.printSummary()
```

`pyg4ometry.geant4._UpdateComplexity(lv, info)`

`pyg4ometry.geant4.AnalyseGeometryStructure(registry, lv_name=None, debug=False, level=0, df=None)`

Produce a pandas dataframe representing the structure of the geometry.

`pyg4ometry.geant4.DumpGeometryStructureTree(lv, depth=0)`

`pyg4ometry.geant4._nistMaterialDict`

`pyg4ometry.geant4._nistMaterialList`

`pyg4ometry.geant4._nistElementZToName`

`pyg4ometry.geant4.getNistMaterialDict()`

`pyg4ometry.geant4.getNistMaterialList()`

`pyg4ometry.geant4.getNistElementZToName()`

`pyg4ometry.geant4._getClassVariables(obj)`

`pyg4ometry.geant4._makeNISTCompoundList()`

`pyg4ometry.geant4._safeName(name)`

`pyg4ometry.geant4.loadNISTMaterialDict()`

`pyg4ometry.geant4.nist_materials_name_lookup(name)`

`pyg4ometry.geant4.nist_materials_z_lookup(z)`

`pyg4ometry.geant4.nist_element_2geant4Element(name, reg=None)`

This returns an instance of either `ElementSimple` or `ElementIsotopeMixture`.

`pyg4ometry.geant4.nist_material_2geant4Material(name, reg=None)`

`pyg4ometry.geant4.MaterialPredefined(name, registry=None)`

Proxy method to construct a NIST compound material - this is just a handle as nothing needs to be additionally defined for a NIST compound. A check is performed on the name to ensure it is a valid NIST specifier.

Inputs:

name - string

`pyg4ometry.geant4.MaterialArbitrary(name, registry=None)`

Just a name of a material. WARNING: It is left to the user to ensure that the name is valid.

Inputs:

name - string

`pyg4ometry.geant4.MaterialSingleElement(name, atomic_number, atomic_weight, density, registry=None, tolerateZeroDensity=False)`

Proxy method to construct a simple material - full description of the element contained is contained in one definition

Inputs:

name - string atomic_number - int, number of protons, commonly known as 'Z' atomic_weight - molar weight in g/mole, commonly known as 'A' density - float, material density in g/cm3

`pyg4ometry.geant4.MaterialCompound(name, density, number_of_components, registry=None, tolerateZeroDensity=False, state=None)`

Proxy method to construct a composite material - can be any mixture of Elements and/or Materials

Inputs:

name - string density - float, material density in g/cm3 number_of_components - int, number of components in the mixture

`pyg4ometry.geant4.ElementSimple(name, symbol, Z, A, registry=None)`

Proxy method to construct a simple element - full description of the element contained is contained in one definition

Inputs:

name - string symbol - string, chemical formula of the compound Z - int, Atomic number A - float, mass number

`pyg4ometry.geant4.ElementIsotopeMixture(name, symbol, n_comp, registry=None, state=None)`

Proxy method to construct a composite element - a mixture of predefined isotopes

Inputs:

name - string symbol - string, chemical formula of the compound n_comp - int, number of isotope components

```
class pyg4ometry.geant4.MaterialBase(name, state=None, registry=None)
```

```
    _addToRegistry()
```

```
    get_material_object(material)
```

```
    set_registry(registry, dontWarnIfAlreadyAdded=False)
```

```
    set_state(state)
```

```
    __repr__()
```

Return repr(self).

```
class pyg4ometry.geant4.Material(**kwargs)
```

Bases: [MaterialBase](#)

This class provides an interface to GDML material definitions.

Because of the different options for constructing a material instance the constructor is kwarg only. Proxy methods are provided to instantiate particular types of material. Those proxy methods are:

MaterialSingleElement MaterialCompound MaterialPredefined

It is possible to instantiate a material directly through kwargs. The possible kwargs are (but note some are mutually exclusive): name - string density - float atomic_number - int atomic_weight - float number_of_components - int state - string pressure - float pressure_unit - string temperature - float temperature_unit - string

property state_variables

```
add_element_massfraction(element, massfraction)
```

Add an element as a component to a material as a fraction of the material mass. Can only add elements to materials defined as composite.

Inputs:

element - pyg4ometry.geant4.Material.Element instance massfraction - float, 0.0 < massfraction <= 1.0

```
add_element_natoms(element, natoms)
```

Add an element as a component to a material as a number of atoms in the material molecule. Can only add elements to materials defined as composite.

Inputs:

element - pyg4ometry.geant4.Material.Element instance natoms - int, number of atoms in the compound molecule

```
add_material(material, fractionmass)
```

Add a material as a component to another material (mixture) as a fraction of the mixture mass. Can only add new materials to materials defined as composite.

Inputs:

material - pyg4ometry.geant4.Material.Material instance massfraction - float, 0.0 < massfraction <= 1.0

```
set_pressure(value, unit='pascal')
```

```
set_temperature(value, unit='K')
```

`__str__()`

Return str(self).

`addProperty(name, matrix)`

Add a material property from a matrix.

Parameters

- **name** (*str*) – key of the material property
- **matrix** (*Matrix*) – matrix defining the value(s) of the property

`addVecProperty(name, e, v, eunit='eV', vunit='')`

Add a property from an energy and a value vector to this object.

Parameters

- **name** (*str*) – key of property
- **e** (*list or numpy.array - shape (1,)*) – energy list/vector in units of eunit
- **v** (*list or numpy.array - shape (1,)*) – value list/vector in units of vunit
- **eunit** (*str*) – unit for the energy vector (default: eV)
- **vunit** (*str*) – unit for the value vector (default: unitless)

`addConstProperty(name, value, vunit='')`

Add a constant scalar property to this object.

Parameters

- **name** (*str*) – key of property
- **value** (*str, float, int*) – constant value for this property
- **vunit** (*str*) – unit for the value vector (default: unitless)

`class pyg4ometry.geant4.Element(**kwargs)`

Bases: *MaterialBase*

This class provides an interface to GDML material definitions. Because of the different options for constructing a material instance the constructor is kwarg only. Proxy methods are provided to instantiate particular types of material. Those proxy methods are:

ElementSimple ElementIsotopeMixture

It is possible to instantiate a material directly through kwargs. The possible kwargs are (but note some are mutually exclusive): name - string symbol - string Z - int A - int n_comp - int

`add_isotope(isotope, abundance)`

Add an isotope as a component to an element as an abundance fraction in the element.

Inputs:

element - pyg4ometry.geant4.Material.Isotope instance abundance - float, 0.0 < abundance <= 1.0

`class pyg4ometry.geant4.Isotope(name, Z, N, a, registry=None)`

Bases: *MaterialBase*

This class that handles isotopes as components of composite materials. An element can be defined as a mixture of isotopes.

Inputs:

name - string Z - int, atomic number N - int, mass number a - float, molar weight in g/mole

pyg4ometry

`pyg4ometry.gui`

Submodules

`pyg4ometry.gui.GeometryModel`

Module Contents

Classes

GeometryModel

Functions

extensionFromPath(fileName)

nameFromPath(fileName)

`pyg4ometry.gui.GeometryModel.extensionFromPath(fileName)`

`pyg4ometry.gui.GeometryModel.nameFromPath(fileName)`

class `pyg4ometry.gui.GeometryModel.GeometryModel`

createNewRegistry(name, type)

loadNewRegistry(fileName)

`pyg4ometry.gui.MainWindow`

Module Contents

Classes

MainWindow

Functions

main()

class pyg4ometry.gui.MainWindow.**MainWindow**

Bases: PyQt5.QtWidgets.QMainWindow

initModel()

initUI()

slotModelChanged(*signal*)

setDisplayRenderer(*iRenderer*)

openFileNameDialog()

saveFileDialog()

pyg4ometry.gui.MainWindow.**main()**

pyg4ometry.gui.QVTKRenderWindowInteractor

Module Contents

Classes

QVTKRenderWindowInteractor

A QVTKRenderWindowInteractor for Python and Qt.
Uses a

Functions

QVTKRenderWidgetConeExample()

A simple example that uses the QVTKRenderWindow-
Interactor class.

_qt_key_to_key_sym(key)

Convert a Qt key into a vtk keysym.

Attributes

PyQtImpl

QVTKRWIBase

_keysyms

pyg4ometry.gui.QVTKRenderWindowInteractor.PyQtImpl

pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRWIBase

class pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor(*parent=None*,
***kw*)

Bases: PyQt5.QtWidgets.QWidget

A QVTKRenderWindowInteractor for Python and Qt. Uses a vtkGenericRenderWindowInteractor to handle the interactions. Use GetRenderWindow() to get the vtkRenderWindow. Create with the keyword stereo=1 in order to generate a stereo-capable window.

The user interface is summarized in vtkInteractorStyle.h:

- Keypress j / Keypress t: toggle between joystick (position

sensitive) and trackball (motion sensitive) styles. In joystick style, motion occurs continuously as long as a mouse button is pressed. In trackball style, motion occurs when the mouse button is pressed and the mouse pointer moves.

- Keypress c / Keypress o: toggle between camera and object

(actor) modes. In camera mode, mouse events affect the camera position and focal point. In object mode, mouse events affect the actor that is under the mouse pointer.

- Button 1: rotate the camera around its focal point (if camera

mode) or rotate the actor around its origin (if actor mode). The rotation is in the direction defined from the center of the renderer's viewport towards the mouse position. In joystick mode, the magnitude of the rotation is determined by the distance the mouse is from the center of the render window.

- Button 2: pan the camera (if camera mode) or translate the actor

(if object mode). In joystick mode, the direction of pan or translation is from the center of the viewport towards the mouse position. In trackball mode, the direction of motion is the direction the mouse moves. (Note: with 2-button mice, pan is defined as <Shift>-Button 1.)

- Button 3: zoom the camera (if camera mode) or scale the actor

(if object mode). Zoom in/increase scale if the mouse position is in the top half of the viewport; zoom out/decrease scale if the mouse position is in the bottom half. In joystick mode, the amount of zoom is controlled by the distance of the mouse pointer from the horizontal centerline of the window.

- Keypress 3: toggle the render window into and out of stereo

mode. By default, red-blue stereo pairs are created. Some systems support Crystal Eyes LCD stereo glasses; you have to invoke SetStereoTypeToCrystalEyes() on the rendering window. Note: to use stereo you also need to pass a stereo=1 keyword argument to the constructor.

- Keypress e: exit the application.
- Keypress f: fly to the picked point
- Keypress p: perform a pick operation. The render window interactor

has an internal instance of vtkCellPicker that it uses to pick.

- Keypress r: reset the camera view along the current view

direction. Centers the actors and moves the camera so that all actors are visible.

- Keypress s: modify the representation of all actors so that they are surfaces.

- Keypress u: invoke the user-defined function. Typically, this

keypress will bring up an interactor that you can type commands in.

- Keypress w: modify the representation of all actors so that they are wireframe.

_CURSOR_MAP

__getattr__(*attr*)

Makes the object behave like a `vtkGenericRenderWindowInteractor`

Finalize()

Call internal cleanup method on VTK objects

CreateTimer(*obj, evt*)

DestroyTimer(*obj, evt*)

TimerEvent()

CursorChangedEvent(*obj, evt*)

Called when the `CursorChangedEvent` fires on the render window.

HideCursor()

Hides the cursor.

ShowCursor()

Shows the cursor.

closeEvent(*evt*)

sizeHint()

paintEngine()

paintEvent(*ev*)

resizeEvent(*ev*)

_GetCtrlShift(*ev*)

static _getPixelRatio()

_setEventInformation(*x, y, ctrl, shift, key, repeat=0, keysum=None*)

enterEvent(*ev*)

leaveEvent(*ev*)

mousePressEvent(*ev*)

mouseReleaseEvent(*ev*)

mouseMoveEvent(*ev*)

keyPressEvent(*ev*)

keyReleaseEvent(*ev*)

wheelEvent(*ev*)

GetRenderWindow()

Render()

`pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWidgetConeExample()`

A simple example that uses the QVTKRenderWindowInteractor class.

`pyg4ometry.gui.QVTKRenderWindowInteractor._keysyms`

`pyg4ometry.gui.QVTKRenderWindowInteractor._qt_key_to_key_sym(key)`

Convert a Qt key into a vtk keysym.

This is essentially copied from the c++ implementation in GUISupport/Qt/QVTKInteractorAdapter.cxx.

`pyg4ometry.gui.example1`

Module Contents

Classes

<code>QVTKRenderWindowInteractor</code>	A QVTKRenderWindowInteractor for Python and Qt. Uses a
---	---

Functions

<code>QVTKRenderWidgetConeExample()</code>	A simple example that uses the QVTKRenderWindow- Interactor class.
<code>_qt_key_to_key_sym(key)</code>	Convert a Qt key into a vtk keysym.

Attributes

<code>PyQtImpl</code>
<code>QVTKRWIBase</code>
<code>QVTKRWIBase</code>
<code>PyQtImpl</code>
<code>QVTKRWIBaseClass</code>
<code>_keysyms</code>

`pyg4ometry.gui.example1.PyQtImpl`


```

pyg4ometry.gui.example1.QVTKRWIBase = 'QWidget'

pyg4ometry.gui.example1.QVTKRWIBase

pyg4ometry.gui.example1.PyQtImpl = 'PyQt5'

pyg4ometry.gui.example1.QVTKRWIBaseClass

class pyg4ometry.gui.example1.QVTKRenderWindowInteractor(parent=None, **kw)

```

Bases: [QVTKRWIBaseClass](#)

A QVTKRenderWindowInteractor for Python and Qt. Uses a vtkGenericRenderWindowInteractor to handle the interactions. Use GetRenderWindow() to get the vtkRenderWindow. Create with the keyword stereo=1 in order to generate a stereo-capable window.

The user interface is summarized in vtkInteractorStyle.h:

- Keypress j / Keypress t: toggle between joystick (position sensitive) and trackball (motion sensitive) styles. In joystick style, motion occurs continuously as long as a mouse button is pressed. In trackball style, motion occurs when the mouse button is pressed and the mouse pointer moves.
- Keypress c / Keypress o: toggle between camera and object (actor) modes. In camera mode, mouse events affect the camera position and focal point. In object mode, mouse events affect the actor that is under the mouse pointer.
- Button 1: rotate the camera around its focal point (if camera mode) or rotate the actor around its origin (if actor mode). The rotation is in the direction defined from the center of the renderer's viewport towards the mouse position. In joystick mode, the magnitude of the rotation is determined by the distance the mouse is from the center of the render window.
- Button 2: pan the camera (if camera mode) or translate the actor (if object mode). In joystick mode, the direction of pan or translation is from the center of the viewport towards the mouse position. In trackball mode, the direction of motion is the direction the mouse moves. (Note: with 2-button mice, pan is defined as <Shift>-Button 1.)
- Button 3: zoom the camera (if camera mode) or scale the actor (if object mode). Zoom in/increase scale if the mouse position is in the top half of the viewport; zoom out/decrease scale if the mouse position is in the bottom half. In joystick mode, the amount of zoom is controlled by the distance of the mouse pointer from the horizontal centerline of the window.
- Keypress 3: toggle the render window into and out of stereo mode. By default, red-blue stereo pairs are created. Some systems support Crystal Eyes LCD stereo glasses; you have to invoke SetStereoTypeToCrystalEyes() on the rendering window. Note: to use stereo you also need to pass a stereo=1 keyword argument to the constructor.
- Keypress e: exit the application.
- Keypress f: fly to the picked point
- Keypress p: perform a pick operation. The render window interactor has an internal instance of vtkCellPicker that it uses to pick.
- Keypress r: reset the camera view along the current view direction. Centers the actors and moves the camera so that all actors are visible.
- Keypress s: modify the representation of all actors so that they

are surfaces.

- Keypress u: invoke the user-defined function. Typically, this keypress will bring up an interactor that you can type commands in.
- Keypress w: modify the representation of all actors so that they are wireframe.

_CURSOR_MAP

__getattr__(*attr*)

Makes the object behave like a `vtkGenericRenderWindowInteractor`

Finalize()

Call internal cleanup method on VTK objects

CreateTimer(*obj, evt*)

DestroyTimer(*obj, evt*)

TimerEvent()

CursorChangedEvent(*obj, evt*)

Called when the `CursorChangedEvent` fires on the render window.

HideCursor()

Hides the cursor.

ShowCursor()

Shows the cursor.

closeEvent(*evt*)

sizeHint()

paintEngine()

paintEvent(*ev*)

resizeEvent(*ev*)

_GetCtrlShift(*ev*)

static _getPixelRatio()

_setEventInformation(*x, y, ctrl, shift, key, repeat=0, keysum=None*)

enterEvent(*ev*)

leaveEvent(*ev*)

mousePressEvent(*ev*)

mouseReleaseEvent(*ev*)

mouseMoveEvent(*ev*)

keyPressEvent(*ev*)

keyReleaseEvent(*ev*)

wheelEvent(*ev*)

GetRenderWindow()

Render()

pyg4ometry.gui.example1.QVTKRenderWindowConeExample()

A simple example that uses the QVTKRenderWindowInteractor class.

pyg4ometry.gui.example1._keysyms

pyg4ometry.gui.example1._qt_key_to_key_sym(*key*)

Convert a Qt key into a vtk keysym.

This is essentially copied from the c++ implementation in GUISupport/Qt/QVTKInteractorAdapter.cxx.

Package Contents

Classes

QVTKRenderWindowInteractor

GeometryModel

MainWindow

QVTKRenderWindowInteractor

GeometryModel

Functions

main()

QVTKRenderWidgetConeExample()

A simple example that uses the QVTKRenderWindow-Interactor class.

_qt_key_to_key_sym(*key*)

Convert a Qt key into a vtk keysym.

extensionFromPath(*fileName*)

nameFromPath(*fileName*)

Attributes

*PyQtImpl**QVTKRWIBase**_keysyms*

class pyg4ometry.gui.QVTKRenderWindowInteractor(*parent=None, **kw*)

Bases: PyQt5.QtWidgets.QWidget

A QVTKRenderWindowInteractor for Python and Qt. Uses a vtkGenericRenderWindowInteractor to handle the interactions. Use GetRenderWindow() to get the vtkRenderWindow. Create with the keyword stereo=1 in order to generate a stereo-capable window.

The user interface is summarized in vtkInteractorStyle.h:

- Keypress j / Keypress t: toggle between joystick (position

sensitive) and trackball (motion sensitive) styles. In joystick style, motion occurs continuously as long as a mouse button is pressed. In trackball style, motion occurs when the mouse button is pressed and the mouse pointer moves.

- Keypress c / Keypress o: toggle between camera and object

(actor) modes. In camera mode, mouse events affect the camera position and focal point. In object mode, mouse events affect the actor that is under the mouse pointer.

- Button 1: rotate the camera around its focal point (if camera

mode) or rotate the actor around its origin (if actor mode). The rotation is in the direction defined from the center of the renderer's viewport towards the mouse position. In joystick mode, the magnitude of the rotation is determined by the distance the mouse is from the center of the render window.

- Button 2: pan the camera (if camera mode) or translate the actor

(if object mode). In joystick mode, the direction of pan or translation is from the center of the viewport towards the mouse position. In trackball mode, the direction of motion is the direction the mouse moves. (Note: with 2-button mice, pan is defined as <Shift>-Button 1.)

- Button 3: zoom the camera (if camera mode) or scale the actor

(if object mode). Zoom in/increase scale if the mouse position is in the top half of the viewport; zoom out/decrease scale if the mouse position is in the bottom half. In joystick mode, the amount of zoom is controlled by the distance of the mouse pointer from the horizontal centerline of the window.

- Keypress 3: toggle the render window into and out of stereo

mode. By default, red-blue stereo pairs are created. Some systems support Crystal Eyes LCD stereo glasses; you have to invoke SetStereoTypeToCrystalEyes() on the rendering window. Note: to use stereo you also need to pass a stereo=1 keyword argument to the constructor.

- Keypress e: exit the application.
- Keypress f: fly to the picked point
- Keypress p: perform a pick operation. The render window interactor

has an internal instance of vtkCellPicker that it uses to pick.

- Keypress r: reset the camera view along the current view direction. Centers the actors and moves the camera so that all actors are visible.
- Keypress s: modify the representation of all actors so that they are surfaces.
- Keypress u: invoke the user-defined function. Typically, this keypress will bring up an interactor that you can type commands in.
- Keypress w: modify the representation of all actors so that they are wireframe.

_CURSOR_MAP

__getattr__(*attr*)

Makes the object behave like a `vtkGenericRenderWindowInteractor`

Finalize()

Call internal cleanup method on VTK objects

CreateTimer(*obj, evt*)

DestroyTimer(*obj, evt*)

TimerEvent()

CursorChangedEvent(*obj, evt*)

Called when the `CursorChangedEvent` fires on the render window.

HideCursor()

Hides the cursor.

ShowCursor()

Shows the cursor.

closeEvent(*evt*)

sizeHint()

paintEngine()

paintEvent(*ev*)

resizeEvent(*ev*)

_GetCtrlShift(*ev*)

static _getPixelRatio()

_setEventInformation(*x, y, ctrl, shift, key, repeat=0, keysum=None*)

enterEvent(*ev*)

leaveEvent(*ev*)

mousePressEvent(*ev*)

mouseReleaseEvent(*ev*)

```
mouseMoveEvent(ev)

keyPressEvent(ev)

keyReleaseEvent(ev)

wheelEvent(ev)

GetRenderWindow()

Render()

class pyg4ometry.gui.GeometryModel

    createNewRegistry(name, type)

    loadNewRegistry(fileName)

class pyg4ometry.gui.MainWindow
    Bases: PyQt5.QtWidgets.QMainWindow
    initModel()

    initUI()

    slotModelChanged(signal)

    setDisplayRenderer(iRenderer)

    openFileNameDialog()

    saveFileDialog()

pyg4ometry.gui.main()

pyg4ometry.gui.PyQtImpl

pyg4ometry.gui.QVTKRWIBase

class pyg4ometry.gui.QVTKRenderWindowInteractor(parent=None, **kw)
```

```
    Bases: PyQt5.QtWidgets.QWidget
```

A QVTKRenderWindowInteractor for Python and Qt. Uses a vtkGenericRenderWindowInteractor to handle the interactions. Use GetRenderWindow() to get the vtkRenderWindow. Create with the keyword stereo=1 in order to generate a stereo-capable window.

The user interface is summarized in vtkInteractorStyle.h:

- Keypress j / Keypress t: toggle between joystick (position

sensitive) and trackball (motion sensitive) styles. In joystick style, motion occurs continuously as long as a mouse button is pressed. In trackball style, motion occurs when the mouse button is pressed and the mouse pointer moves.

- Keypress c / Keypress o: toggle between camera and object

(actor) modes. In camera mode, mouse events affect the camera position and focal point. In object mode, mouse events affect the actor that is under the mouse pointer.

- Button 1: rotate the camera around its focal point (if camera

mode) or rotate the actor around its origin (if actor mode). The rotation is in the direction defined from the center of the renderer's viewport towards the mouse position. In joystick mode, the magnitude of the rotation is determined by the distance the mouse is from the center of the render window.

- Button 2: pan the camera (if camera mode) or translate the actor

(if object mode). In joystick mode, the direction of pan or translation is from the center of the viewport towards the mouse position. In trackball mode, the direction of motion is the direction the mouse moves. (Note: with 2-button mice, pan is defined as <Shift>-Button 1.)

- Button 3: zoom the camera (if camera mode) or scale the actor

(if object mode). Zoom in/increase scale if the mouse position is in the top half of the viewport; zoom out/decrease scale if the mouse position is in the bottom half. In joystick mode, the amount of zoom is controlled by the distance of the mouse pointer from the horizontal centerline of the window.

- Keypress 3: toggle the render window into and out of stereo

mode. By default, red-blue stereo pairs are created. Some systems support Crystal Eyes LCD stereo glasses; you have to invoke `SetStereoTypeToCrystalEyes()` on the rendering window. Note: to use stereo you also need to pass a `stereo=1` keyword argument to the constructor.

- Keypress e: exit the application.
- Keypress f: fly to the picked point
- Keypress p: perform a pick operation. The render window interactor

has an internal instance of `vtkCellPicker` that it uses to pick.

- Keypress r: reset the camera view along the current view

direction. Centers the actors and moves the camera so that all actors are visible.

- Keypress s: modify the representation of all actors so that they

are surfaces.

- Keypress u: invoke the user-defined function. Typically, this

keypress will bring up an interactor that you can type commands in.

- Keypress w: modify the representation of all actors so that they

are wireframe.

_CURSOR_MAP

__getattr__(*attr*)

Makes the object behave like a `vtkGenericRenderWindowInteractor`

Finalize()

Call internal cleanup method on VTK objects

CreateTimer(*obj*, *evt*)

DestroyTimer(*obj*, *evt*)

TimerEvent()

CursorChangedEvent(*obj*, *evt*)

Called when the `CursorChangedEvent` fires on the render window.

HideCursor()

Hides the cursor.

ShowCursor()

Shows the cursor.

closeEvent(*evt*)

sizeHint()

paintEngine()

paintEvent(*ev*)

resizeEvent(*ev*)

_GetCtrlShift(*ev*)

static _getPixelRatio()

_setEventInformation(*x, y, ctrl, shift, key, repeat=0, keysum=None*)

enterEvent(*ev*)

leaveEvent(*ev*)

mousePressEvent(*ev*)

mouseReleaseEvent(*ev*)

mouseMoveEvent(*ev*)

keyPressEvent(*ev*)

keyReleaseEvent(*ev*)

wheelEvent(*ev*)

GetRenderWindow()

Render()

pyg4ometry.gui.QVTKRenderWindowConeExample()

A simple example that uses the QVTKRenderWindowInteractor class.

pyg4ometry.gui._keysyms

pyg4ometry.gui._qt_key_to_key_sym(*key*)

Convert a Qt key into a vtk keysym.

This is essentially copied from the c++ implementation in GUIsupport/Qt/QVTKInteractorAdapter.cxx.

pyg4ometry.gui.extensionFromPath(*fileName*)

pyg4ometry.gui.nameFromPath(*fileName*)

class pyg4ometry.gui.GeometryModel

createNewRegistry(*name, type*)

loadNewRegistry(*fileName*)

pyg4ometry.io

Submodules

pyg4ometry.io.ROOTTGeo

Module Contents

Classes

Reader

Functions

<code>_isClose(a, b[, relativeTolerance, absoluteTolerance])</code>	return True if values are close to each other. As judged by the
<code>rootMatrix2pyg4ometry(matrix, reader)</code>	
<code>rootShape2pyg4ometry(shape, reader[, warnAboutBadShapes])</code>	

pyg4ometry.io.ROOTTGeo.**_isClose**(*a, b, relativeTolerance=1e-05, absoluteTolerance=0.0*)

return True if values are close to each other. As judged by the absolute difference between a and b is greater than relative_tolerance times that difference. Implementation from PEP 485 (python > 3.5)

pyg4ometry.io.ROOTTGeo.**rootMatrix2pyg4ometry**(*matrix, reader*)

pyg4ometry.io.ROOTTGeo.**rootShape2pyg4ometry**(*shape, reader, warnAboutBadShapes=True*)

```
class pyg4ometry.io.ROOTTGeo.Reader(fileName, upgradeVacuumToG4Galactic=True,
                                     solidsToTessellate=None, suffixSeparator='__',
                                     warnAboutBadShapes=True, maximumDepth=1000000.0,
                                     dontLoadOverlapNodes=True)
```

getRegistry()

load(*upgradeVacuumToG4Galactic=True, warnAboutBadShapes=True, dontLoadOverlapNodes=True*)

_ROOTMatStateToGeant4MatState(*rootMaterialState*)

Based on <https://root.cern.ch/doc/master/classTGeoMaterial.html#a8b69c72f90711a29726087e029e39c61>
enum TGeoMaterial::EGeoMaterialState

loadMaterials(*upgradeVacuumToG4Galactic*)

_makeG4Element(*rootElement*)

_makeG4Isotope(*rootIsotope*)

recurseVolumeTree(*volume, thisDepth, maximumDepth, warnAboutBadShapes=True, dontLoadOverlapNodes=True*)

pyg4ometry.misc

Submodules

pyg4ometry.misc.NestedSolids

Module Contents

Functions

<code>NestedBoxes(nameBase, bx, by, bz, registry[, lunit, ...])</code>	Creates a list of geant4.Box starting the bx, by, bz in size and each element is dx, dy, dz smaller
--	---

pyg4ometry.misc.NestedSolids.**NestedBoxes**(*nameBase, bx, by, bz, registry, lunit='mm', dx=0, dy=0, dz=0, N=0*)

Creates a list of geant4.Box starting the bx, by, bz in size and each element is dx, dy, dz smaller for each iteration

Parameters

- **nameBase** (*str*) – name stub for solid
- **bx** (*float*) – box max x size
- **by** (*float*) – box max y size
- **bz** (*float*) – box max z size
- **registry** (*Registry*) – Registry object
- **lunit** (*str*) – length unit (mm/m etc)
- **dx** (*float*) – decrement in x for each iteration
- **dy** (*float*) – decrement in y for each iteration
- **dz** (*float*) – decrement in z for each iteration
- **N** (*int*) – number of iterations

pyg4ometry.misc.TestUtils

Module Contents

Functions

```
md5_file(fname)
```

```
diffFiles(file1, file2)
```

```
compareFilesWithAssert(file1, file2)
```

```
compareNumericallyWithAssert(file1, file2)
```

```
compareMeshInfo(meshInfo1, meshInfo2)
```

```
compareGdmlNumericallyWithAssert(file1, file2)
```

```
pyg4ometry.misc.TestUtils.md5_file(fname)
```

```
pyg4ometry.misc.TestUtils.diffFiles(file1, file2)
```

```
pyg4ometry.misc.TestUtils.compareFilesWithAssert(file1, file2)
```

```
pyg4ometry.misc.TestUtils.compareNumericallyWithAssert(file1, file2)
```

```
pyg4ometry.misc.TestUtils.compareMeshInfo(meshInfo1, meshInfo2)
```

```
pyg4ometry.misc.TestUtils.compareGdmlNumericallyWithAssert(file1, file2)
```

Package Contents

Functions

<i>NestedBoxes</i> (nameBase, bx, by, bz, registry[, lunit, ...])	Creates a list of geant4.Box starting the bx, by, bz in size and each element is dx, dy, dz smaller
---	---

```
md5_file(fname)
```

```
diffFiles(file1, file2)
```

```
compareFilesWithAssert(file1, file2)
```

```
compareNumericallyWithAssert(file1, file2)
```

```
compareMeshInfo(meshInfo1, meshInfo2)
```

```
compareGdmlNumericallyWithAssert(file1, file2)
```

```
pyg4ometry.misc.NestedBoxes(nameBase, bx, by, bz, registry, lunit='mm', dx=0, dy=0, dz=0, N=0)
```

Creates a list of geant4.Box starting the bx, by, bz in size and each element is dx, dy, dz smaller for each iteration

Parameters

- **nameBase** (*str*) – name stub for solid
- **bx** (*float*) – box max x size
- **by** (*float*) – box max y size
- **bz** (*float*) – box max z size
- **registry** (*Registry*) – Registry object
- **lunit** (*str*) – length unit (mm/m etc)
- **dx** (*float*) – decrement in x for each iteration
- **dy** (*float*) – decrement in y for each iteration
- **dz** (*float*) – decrement in z for each iteration
- **N** (*int*) – number of iterations

```
pyg4ometry.misc.md5_file(fname)
```

```
pyg4ometry.misc.diffFiles(file1, file2)
```

```
pyg4ometry.misc.compareFilesWithAssert(file1, file2)
```

```
pyg4ometry.misc.compareNumericallyWithAssert(file1, file2)
```

```
pyg4ometry.misc.compareMeshInfo(meshInfo1, meshInfo2)
```

```
pyg4ometry.misc.compareGdmlNumericallyWithAssert(file1, file2)
```

```
pyg4ometry.montecarlo
```

Submodules

```
pyg4ometry.montecarlo.beam
```

Module Contents

Classes

Beam

```
class pyg4ometry.montecarlo.beam.Beam(pos=[0, 0, 0], dir=[0, 0, 1], energy=1, particleType='e-')
    flukaString()
    bdsimString()
```

pyg4ometry.montecarlo.boundary

Module Contents

Classes

Boundary

```
class pyg4ometry.montecarlo.boundary.Boundary(name='boundary1', pos=[0, 0, 0], rot=[0, 0, 0],
                                              type='plane', size=1, length=1)
```

```
    flukaString()
```

```
    bdsimString()
```

pyg4ometry.montecarlo.config

Module Contents

Classes

Config

```
class pyg4ometry.montecarlo.config.Config(nHistory=1, seed=1)
```

```
    flukaString()
```

```
    bdsimString()
```

pyg4ometry.montecarlo.scoring

Module Contents

Classes

Scoring

```
class pyg4ometry.montecarlo.scoring.Scoring(n1=10, low1=-10, high1=10, n2=10, low2=-10,
                                              high2=10, n3=10, low3=-10, high3=10,
                                              coordType='cartesian', type='')
```

```
    flukaString()
```

```
    bdsimString()
```

Package Contents

Classes

*Beam**Scoring**Boundary**Config*

```
class pyg4ometry.montecarlo.Beam(pos=[0, 0, 0], dir=[0, 0, 1], energy=1, particleType='e-')
    flukaString()
    bdsimString()

class pyg4ometry.montecarlo.Scoring(n1=10, low1=-10, high1=10, n2=10, low2=-10, high2=10, n3=10,
                                     low3=-10, high3=10, coordType='cartesian', type='')
    flukaString()
    bdsimString()

class pyg4ometry.montecarlo.Boundary(name='boundary1', pos=[0, 0, 0], rot=[0, 0, 0], type='plane',
                                     size=1, length=1)
    flukaString()
    bdsimString()

class pyg4ometry.montecarlo.Config(nHistory=1, seed=1)
    flukaString()
    bdsimString()
```

pyg4ometry.pycgal

Submodules

pyg4ometry.pycgal.HalfPlane

Module Contents

Functions

halfPlaneMesh([planes])

```
pyg4ometry.pycgal.HalfPlane.halfPlaneMesh(planes=None)
```

```
pyg4ometry.pycgal.core
```

Module Contents

Classes

<i>CSG</i>
<i>PolygonProcessing</i>
<i>PolyhedronProcessing</i>

Functions

<i>do_intersect</i> (csg1, csg2)
<i>intersecting_meshes</i> (csgList)

```
class pyg4ometry.pycgal.core.CSG
    classmethod fromPolygons(polygons, **kwargs)
    toVerticesAndPolygons()
    clone()
    rotate(axisIn, angleDeg)
    translate(disp)
    scale(*args)
    getNumberPolys()
    getNumberVertices()
    vertexCount()
    polygonCount()
    intersect(csg2)
    union(csg2)
    subtract(csg2)
    inverse()
```

coplanarIntersection(*csg*)

Compute the coplanar surfaces between self and csg

classmethod cube(*center*=[0, 0, 0], *radius*=[1, 1, 1])

Construct an axis-aligned solid cuboid. Optional parameters are *center* and *radius*, which default to [0, 0, 0] and [1, 1, 1]. The radius can be specified using a single number or a list of three numbers, one for each axis.

Example code:

```
cube = CSG.cube(  
    center=[0, 0, 0],  
    radius=1  
)
```

volume()

area()

isNull()

isClosed()

isTriangleMesh()

isOutwardOriented()

info()

pyg4ometry.pycgal.core.**do_intersect**(*csg1*, *csg2*)

pyg4ometry.pycgal.core.**intersecting_meshes**(*csgList*)

class pyg4ometry.pycgal.core.**PolygonProcessing**

classmethod windingNumber(*pgon*)

return the winding number of pgon :param pgon: list of points [[x1,y1], [x2,y2], ...] :type pgon: List[List[x1,y1], ...] returns: Integer winding number

classmethod reversePolygon(*pgon*)

return reversed polygon :param pgon: list of points [[x1,y1], [x2,y2], ...] :type pgon: List[List[x1,y1], ...] returns: List[List[x1,y1], ...]

classmethod makePolygonFromList(*pgon*, *type*="")

Convert list of points [[x1,y1], [x2,y2], ...] to cgal Polygon_2

Parameters

- **pgon** (*List[List[x,y], ...]*) – list of points [[x1,y1], [x2,y2], ...]
- **type** – Class of polygon (Polygon_2_EPICK, Polygon_2_EPECK, Partition_traits_2_Polygon_2_EPECK)
- **type** – str

returns: Polygon_2

classmethod `makeListFromPolygon(pgon)`

Convert 2D polygon to list of points `[[x1,y1], [x2,y2], ...]`

Parameters

`pgon` (*Polygon_2_EPECK* or *Polygon_2_EPICK*) – cgal *Polygon_2* input

returns: `[[x1,y1], [x2,y2], ...]`

classmethod `decomposePolygon2d(pgon)`

Decompose general 2D polygon (*pgon*) to convex 2D polygons

Parameters

`pgon` (*List(List[2])*) – list of *pgon* points (which are lists) `[[x1,y1], [x2,y2], ...]`

returns: List of polygons [*pgon1*, *pgon2*, ...]

classmethod `decomposePolygon2dWithHoles(pgonOuter, pgonHoles)`

Decompose general 2D polygon with holes (*pgon*) to convex 2D polygons

Parameters

- **`pgonOuter`** – list of *pgon* points (which are lists) `[[x1,y1], [x2,y2], ...]`

- **`pgonHoles`** – List of polygons [*pgon1*, *pgon2*, ...]

returns: List of polygons [*pgon1*, *pgon2*, ...]

classmethod `triangulatePolygon2d(pgon)`

Triangulate general 2D polygon

Parameters

`pgonOuter` – list of *pgon* points (which are lists) `[[x1,y1], [x2,y2], ...]`

returns: List of triangles `[[[x1,y1], [x2,y2], [x3,y3]], [[x1,y1], [x2,y2], [x3,y3]], ...]`

class `pyg4ometry.pycgal.core.PolyhedronProcessing`

classmethod `surfaceMesh_to_Polyhedron(sm)`

classmethod `nefPolyhedron_to_convexPolyhedra(np)`

classmethod `polyhedron_to_numpyArrayPlanes(p)`

`pyg4ometry.pycgal.pythonHelpers`

Module Contents

Functions

`polygon_to_numpy(polygon)`

`_draw_polygon_2(p2)`

`draw_polygon_2_list(pl)`

`draw_polygon_2(p2)`

Draw a CGAL polygon for debugging etc

```
pyg4ometry.pycgal.pythonHelpers.polygon_to_numpy(polygon)
```

```
pyg4ometry.pycgal.pythonHelpers._draw_polygon_2(p2)
```

```
pyg4ometry.pycgal.pythonHelpers.draw_polygon_2_list(pl)
```

```
pyg4ometry.pycgal.pythonHelpers.draw_polygon_2(p2)
```

Draw a CGAL polygon for debugging etc

Parameters

p2 (*Polygon_2*, *Polygon_with_holes_2*, *List_Polygon_with_holes_2*) – *Polygon_2* for plotting

Package Contents

Classes

CSG

PolygonProcessing

PolyhedronProcessing

Functions

do_intersect(*csg1*, *csg2*)

intersecting_meshes(*csgList*)

```
class pyg4ometry.pycgal.CSG
```

```
    classmethod fromPolygons(polygons, **kwargs)
```

```
    toVerticesAndPolygons()
```

```
    clone()
```

```
    rotate(axisIn, angleDeg)
```

```
    translate(disp)
```

```
    scale(*args)
```

```
    getNumberPolys()
```

```
    getNumberVertices()
```

```
    vertexCount()
```

polygonCount()

intersect(*csg2*)

union(*csg2*)

subtract(*csg2*)

inverse()

coplanarIntersection(*csg*)

Compute the coplanar surfaces between self and *csg*

classmethod cube(*center*=[0, 0, 0], *radius*=[1, 1, 1])

Construct an axis-aligned solid cuboid. Optional parameters are *center* and *radius*, which default to [0, 0, 0] and [1, 1, 1]. The radius can be specified using a single number or a list of three numbers, one for each axis.

Example code:

```
cube = CSG.cube(
    center=[0, 0, 0],
    radius=1
)
```

volume()

area()

isNull()

isClosed()

isTriangleMesh()

isOutwardOriented()

info()

`pyg4ometry.pycgal.do_intersect(csg1, csg2)`

`pyg4ometry.pycgal.intersecting_meshes(csgList)`

class `pyg4ometry.pycgal.PolygonProcessing`

classmethod windingNumber(*pgon*)

return the winding number of *pgon* :param *pgon*: list of points [[x1,y1], [x2,y2], ...] :type *pgon*: List[List[x1,y1], ...] returns: Integer winding number

classmethod reversePolygon(*pgon*)

return reversed polygon :param *pgon*: list of points [[x1,y1], [x2,y2], ...] :type *pgon*: List[List[x1,y1], ...] returns: List[List[x1,y1], ...]

classmethod makePolygonFromList(*pgon*, *type*='')

Convert list of points [[x1,y1], [x2,y2], ...] to cgal Polygon_2

Parameters

- **pgon** (*List[List[x,y], ...]*) – list of points [[x1,y1], [x2,y2], ...]

- **type** – Class of polygon (Polygon_2_EPICK, Polygon_2_EPECK, Partition_traits_2_Polygon_2_EPECK)
- **type** – str

returns: Polygon_2

classmethod **makeListFromPolygon**(*pgon*)

Convert 2D polygon to list of points $[[x_1, y_1], [x_2, y_2], \dots]$

Parameters

pgon (*Polygon_2_EPECK* or *Polygon_2_EPICK*) – cgal Polygon_2 input

returns: $[[x_1, y_1], [x_2, y_2], \dots]$

classmethod **decomposePolygon2d**(*pgon*)

Decompose general 2D polygon (pgon) to convex 2D polygons

Parameters

pgon (*List(List[2])*) – list of pgon points (which are lists) $[[x_1, y_1], [x_2, y_2], \dots]$

returns: List of polygons [pgon1, pgon2, ...]

classmethod **decomposePolygon2dWithHoles**(*pgonOuter*, *pgonHoles*)

Decompose general 2D polygon with holes (pgon) to convex 2D polygons

Parameters

- **pgonOuter** – list of pgon points (which are lists) $[[x_1, y_1], [x_2, y_2], \dots]$

- **pgonHoles** – List of polygons [pgon1, pgon2, ...]

returns: List of polygons [pgon1, pgon2, ...]

classmethod **triangulatePolygon2d**(*pgon*)

Triangulate general 2D polygon

Parameters

pgonOuter – list of pgon points (which are lists) $[[x_1, y_1], [x_2, y_2], \dots]$

returns: List of triangles $[[[x_1, y_1], [x_2, y_2], [x_3, y_3]], [[x_1, y_1], [x_2, y_2], [x_3, y_3]], \dots]$

class `pyg4ometry.pycgal.PolyhedronProcessing`

classmethod **surfaceMesh_to_Polyhedron**(*sm*)

classmethod **nefPolyhedron_to_convexPolyhedra**(*np*)

classmethod **polyhedron_to_numpyArrayPlanes**(*p*)

`pyg4ometry.pycgal_old`

Submodules

`pyg4ometry.pycgal_old.cgal`

Module Contents

Functions

pycsgmesh2NefPolyhedron(mesh)

pycsgmeshWritePolygon(mesh[, fileName])

numpyPolygonConvex(polygonnp)

Attributes*g**f**vertexfacet_to_polyhedron**convexpolyhedron_to_planes**surfacemesh_print**polyhedron_write**nefpolyhedron_write**surfacemesh_write**polyhedron_print**nefpolyhedron_print**delete_polyhedron**delete_nefpolyhedron**delete_surfacemesh**polyhedron_to_nefpolyhedron**nefpolyhedron_to_polyhedron**nefpolyhedron_to_surfacemesh**nefpolyhedron_to_convexpolyhedra**vertex_to_polygon**polygon_to_vertex**delete_polygon**polygon_to_convexpolygons**delete_polygonlist*`pyg4ometry.pycgal_old.cgal.g``pyg4ometry.pycgal_old.cgal.f``pyg4ometry.pycgal_old.cgal.vertexfacet_to_polyhedron`

```
pyg4ometry.pycgal_old.cgal.convexpolyhedron_to_planes
pyg4ometry.pycgal_old.cgal.surfacemesh_print
pyg4ometry.pycgal_old.cgal.polyhedron_write
pyg4ometry.pycgal_old.cgal.nefpolyhedron_write
pyg4ometry.pycgal_old.cgal.surfacemesh_write
pyg4ometry.pycgal_old.cgal.polyhedron_print
pyg4ometry.pycgal_old.cgal.nefpolyhedron_print
pyg4ometry.pycgal_old.cgal.delete_polyhedron
pyg4ometry.pycgal_old.cgal.delete_nefpolyhedron
pyg4ometry.pycgal_old.cgal.delete_surfacemesh
pyg4ometry.pycgal_old.cgal.polyhedron_to_nefpolyhedron
pyg4ometry.pycgal_old.cgal.nefpolyhedron_to_polyhedron
pyg4ometry.pycgal_old.cgal.nefpolyhedron_to_surfacemesh
pyg4ometry.pycgal_old.cgal.nefpolyhedron_to_convexpolyhedra
pyg4ometry.pycgal_old.cgal.vertex_to_polygon
pyg4ometry.pycgal_old.cgal.polygon_to_vertex
pyg4ometry.pycgal_old.cgal.delete_polygon
pyg4ometry.pycgal_old.cgal.polygon_to_convexpolygons
pyg4ometry.pycgal_old.cgal.delete_polygonlist
pyg4ometry.pycgal_old.cgal.pycsgmesh2NefPolyhedron(mesh)
pyg4ometry.pycgal_old.cgal.pycsgmeshWritePolygon(mesh, fileName='mesh.pol')
pyg4ometry.pycgal_old.cgal.numpyPolygonConvex(polygonnp)
```

Package Contents

Functions

pycsgmesh2NefPolyhedron(*mesh*)

pycsgmeshWritePolygon(*mesh*[, *fileName*])

numpyPolygonConvex(*polygonnp*)

Attributes*g**f**vertexfacet_to_polyhedron**convexpolyhedron_to_planes**surfacemesh_print**polyhedron_write**nefpolyhedron_write**surfacemesh_write**polyhedron_print**nefpolyhedron_print**delete_polyhedron**delete_nefpolyhedron**delete_surfacemesh**polyhedron_to_nefpolyhedron**nefpolyhedron_to_polyhedron**nefpolyhedron_to_surfacemesh**nefpolyhedron_to_convexpolyhedra**vertex_to_polygon**polygon_to_vertex**delete_polygon**polygon_to_convexpolygons**delete_polygonlist*`pyg4ometry.pycgal_old.g``pyg4ometry.pycgal_old.f``pyg4ometry.pycgal_old.vertexfacet_to_polyhedron`


```
pyg4ometry.pycgal_old.convexpolyhedron_to_planes
pyg4ometry.pycgal_old.surfacemesh_print
pyg4ometry.pycgal_old.polyhedron_write
pyg4ometry.pycgal_old.nefpolyhedron_write
pyg4ometry.pycgal_old.surfacemesh_write
pyg4ometry.pycgal_old.polyhedron_print
pyg4ometry.pycgal_old.nefpolyhedron_print
pyg4ometry.pycgal_old.delete_polyhedron
pyg4ometry.pycgal_old.delete_nefpolyhedron
pyg4ometry.pycgal_old.delete_surfacemesh
pyg4ometry.pycgal_old.polyhedron_to_nefpolyhedron
pyg4ometry.pycgal_old.nefpolyhedron_to_polyhedron
pyg4ometry.pycgal_old.nefpolyhedron_to_surfacemesh
pyg4ometry.pycgal_old.nefpolyhedron_to_convexpolyhedra
pyg4ometry.pycgal_old.vertex_to_polygon
pyg4ometry.pycgal_old.polygon_to_vertex
pyg4ometry.pycgal_old.delete_polygon
pyg4ometry.pycgal_old.polygon_to_convexpolygons
pyg4ometry.pycgal_old.delete_polygonlist
pyg4ometry.pycgal_old.pycsgmesh2NefPolyhedron(mesh)
pyg4ometry.pycgal_old.pycsgmeshWritePolygon(mesh, fileName='mesh.pol')
pyg4ometry.pycgal_old.numpyPolygonConvex(polygonnp)
```

pyg4ometry.pycsg

Package Contents

```
pyg4ometry.pycsg.__version__ = '0.3.3'
```

pyg4ometry.pyoce

Submodules

pyg4ometry.pyoce.Reader

Module Contents

Classes

Reader

class pyg4ometry.pyoce.Reader.**Reader**(*fileName*)

readStepFile(*fileName*)

freeShapes()

traverse(*label=None*)

pyg4ometry.pyoce.pythonHelpers

Module Contents

Functions

shapeTopologyCount(shape[, countType, ignore-
Type])

shapeTopology(shape)

findOCCShapeByName(shapeTool, shapeName) Find a shape by its name

findOCCShapeByTreeNode(label, shapeTreeNode)

get_TDataStd_Name_From_Label(label)

get_TDataStd_TreeNode_From_Label(label)

get_XCAFDoc_Location_From_Label(label)

get_shapeTypeString(st, label)

gp_XYZ_numpy(xyz)

test(fileName)

```
pyg4ometry.pyoce.pythonHelpers.shapeTopologyCount(shape, countType=_TopAbs.TopAbs_FACE,
                                                    ignoreType=_TopAbs.TopAbs_VERTEX)
```

```
pyg4ometry.pyoce.pythonHelpers.shapeTopology(shape)
```

```
pyg4ometry.pyoce.pythonHelpers.findOCCShapeByName(shapeTool, shapeName)
```

Find a shape by its name

Parameters

- **shapeTool** (*ShapeTool*) – OpenCascade ShapeTool
- **shapeName** (*str*) – Name of the shape

```
pyg4ometry.pyoce.pythonHelpers.findOCCShapeByTreeNode(label, shapeTreeNode)
```

```
pyg4ometry.pyoce.pythonHelpers.get_TDataStd_Name_From_Label(label)
```

```
pyg4ometry.pyoce.pythonHelpers.get_TDataStd_TreeNode_From_Label(label)
```

```
pyg4ometry.pyoce.pythonHelpers.get_XCAFDoc_Location_From_Label(label)
```

```
pyg4ometry.pyoce.pythonHelpers.get_shapeTypeString(st, label)
```

```
pyg4ometry.pyoce.pythonHelpers.gp_XYZ_numpy(xyz)
```

```
pyg4ometry.pyoce.pythonHelpers.test(fileName)
```

Package Contents

Classes

Reader

```
class pyg4ometry.pyoce.Reader(fileName)
```

```
    readStepFile(fileName)
```

```
    freeShapes()
```

```
    traverse(label=None)
```

```
pyg4ometry.stl
```

Submodules

```
pyg4ometry.stl.Reader
```

Module Contents

Classes

<code>_Facet</code>	
<code>Reader</code>	STL file reader

class `pyg4ometry.stl.Reader._Facet`(*normal*=(0, 0, 0))

add_vertex(*xyztuple*)

dump()

class `pyg4ometry.stl.Reader.Reader`(*filename*, *solidname*='stl_tessellated', *scale*=1, *centre*=False, *registry*=None, *forcebinary*=False)

STL file reader

Parameters

- **filename** (*str*) – Input STL filename
- **solidname** (*str*) – Name of the solid to be created
- **scale** (*float*) – Scaling of STL (e.g. for units)
- **centre** (*boolean*) – Flag to centre STL solid
- **registry** (*Registry*) – Registry to add solid to
- **forcebinary** (*boolean*) – Forces to load this STL file in binary format, otherwise the file format is determined from whether it starts with the string 'solid'

_load_ascii(*data*)

 Load ASCII STL file from bytes instance

_load_binary(*data*)

 Load binary STL file from bytes instance

extent()

 Compute the axis aligned extent of the STL solid.

Returns

 list of minima and maxima in 3 axes

Return type

 [[xmin,ymin,zmin],[xmax, ymax, zmax]]

extentCentre()

 Translate STL mesh to centre of the extent

translate(*translation*=[0, 0, 0])

 Translate STL mesh by translation

Parameters

translation (*list*(3) or *array*(3)) – Vector to translate mesh

getSolid()

Get geant4.solid

Returns

G4Tessellated for STL

Return type

TessellatedSolid

getRegistry()

Return registry

visualise()

View solid directly by using a dummy world

writeDefaultGDML(filename='default', gmad_tester=False)

Write the tessellated solid loaded from STL to GDML. The placement has no rotation or translation. The world material is G4_Galactic, the solid material is G4_Fe.

Parameters

- **filename** (*str*) – Output file name
- **gmad_tester** (*boolean*) – Flag for writing BDSIM gmad tester

Package Contents

Classes

_Facet

Reader

```
class pyg4ometry.stl._Facet(normal=(0, 0, 0))
```

```
    add_vertex(xyztup)
```

```
    dump()
```

```
class pyg4ometry.stl.Reader(filename, solidname='stl_tessellated', scale=1, centre=False, registry=None,
                             forcebinary=False)
```

STL file reader

Parameters

- **filename** (*str*) – Input STL filename
- **solidname** (*str*) – Name of the solid to be created
- **scale** (*float*) – Scaling of STL (e.g. for units)
- **centre** (*boolean*) – Flag to centre STL solid
- **registry** (*Registry*) – Registry to add solid to
- **forcebinary** (*boolean*) – Forces to load this STL file in binary format, otherwise the file format is determined from whether it starts with the string 'solid'

_load_ascii(*data*)

Load ASCII STL file from bytes instance

_load_binary(*data*)

Load binary STL file from bytes instance

extent()

Compute the axis aligned extent of the STL solid.

Returns

list of minima and maxima in 3 axes

Return type

[[xmin,ymin,zmin],[xmax, ymax, zmax]]

extentCentre()

Translate STL mesh to centre of the extent

translate(*translation*=[0, 0, 0])

Translate STL mesh by translation

Parameters

translation (*list*(3) or *array*(3)) – Vector to translate mesh

getSolid()

Get geant4.solid

Returns

G4Tesselated for STL

Return type

TessellatedSolid

getRegistry()

Return registry

visualise()

View solid directly by using a dummy world

writeDefaultGDML(*filename*='default', *gmad_tester*=False)

Write the tessellated solid loaded from STL to GDML. The placement has no rotation or translation. The world material is G4_Galactic, the solid material is G4_Fe.

Parameters

- **filename** (*str*) – Output file name
- **gmad_tester** (*boolean*) – Flag for writing BDSIM gmad tester

`pyg4ometry.visualisation`

Submodules

`pyg4ometry.visualisation.Convert`

Module Contents

Functions

`mkVtkIdList(it)`

`pycsgMeshToVtkPolyData(mesh)`

`vtkPolyDataToNumpy(data)`

`pycsgMeshToObj(mesh, fileName)`

`pyg42VtkTransformation(mtra, tra)`

`vtkTransformation2PyG4(vt)`

`pycsgMeshToStl(mesh, fileName)`

`pyg4ometry.visualisation.Convert.mkVtkIdList(it)`

`pyg4ometry.visualisation.Convert.pycsgMeshToVtkPolyData(mesh)`

`pyg4ometry.visualisation.Convert.vtkPolyDataToNumpy(data)`

`pyg4ometry.visualisation.Convert.pycsgMeshToObj(mesh, fileName)`

`pyg4ometry.visualisation.Convert.pyg42VtkTransformation(mtra, tra)`

`pyg4ometry.visualisation.Convert.vtkTransformation2PyG4(vt)`

`pyg4ometry.visualisation.Convert.pycsgMeshToStl(mesh, fileName)`

`pyg4ometry.visualisation.Mesh`

Module Contents

Classes

`OverlapType`

`Mesh`

Functions

<code>_getBoundingBox(aMesh[, rotationMatrix, translation, ...])</code>	Axes aligned bounding box. Can also provide a rotation and
<code>_getBoundingBoxMesh(boundingBox)</code>	

class pyg4ometry.visualisation.Mesh.**OverlapType**

protrusion = 1

overlap = 2

coplanar = 3

class pyg4ometry.visualisation.Mesh.**Mesh**(*solid*)

remesh()

addOverlapMesh(*mesh*)

getLocalMesh()

getBoundingBox(*rotationMatrix=None, translation=None*)

Axes aligned bounding box. Can also provide a rotation and a translation (applied in that order) to the vertices.

getBoundingBoxMesh()

pyg4ometry.visualisation.Mesh.**_getBoundingBox**(*aMesh, rotationMatrix=None, translation=None, nameForError=""*)

Axes aligned bounding box. Can also provide a rotation and a translation (applied in that order) to the vertices.

pyg4ometry.visualisation.Mesh.**_getBoundingBoxMesh**(*boundingBox*)

pyg4ometry.visualisation.Plot

Module Contents

Functions

<code>AddCutterDataToPlot(filename[, projection, ax, ...])</code>	Add cutter data to a plot or draw in a new figure if no axis object given.
---	--

pyg4ometry.visualisation.Plot.**AddCutterDataToPlot**(*filename, projection='zx', ax=None, unitsFactor=1.0, colour='k', linewidth=0.5, alpha=1.0*)

Add cutter data to a plot or draw in a new figure if no axis object given.

Parameters

- **filename** (*str*) – file name of exported cutter data from visualiser.

- **projection** (*str*) – which two axes to plot in a 2D plot from the 3D data. e.g. zx, zy.
- **ax** (*matplotlib.axes._axes.Axes*) – matplotlib axes instance.
- **unitsFactor** (*float*) – numerical scaling factor for coordinates (e.g. 0.001 for plotting in meters).
- **colour** (*str*) – matplotlib.pyplot.plot colour argument
- **linewidth** (*float*) – matplotlib.pyplot.plot linewidth argument
- **alpha** (*float*) – matplotlib.pyplot.plot alpha argument for transparency

If used without an axes, a new figure and axes will be created.

The project specifies the order of plotting, so 'zx' would give z in the plot's x axis and 'x' in plot's y axis. Any combination of 'x', 'y' and 'z' are accepted and must be different. e.g. any of 'xy', 'xz', 'yx', 'yz', 'zx', 'zy'.

pyg4ometry.visualisation.RenderWriter

Module Contents

Classes

RenderWriter

class pyg4ometry.visualisation.RenderWriter.**RenderWriter**

```

addLogicalVolumeRecursive(logical, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]))

addMesh(logical)

addInstance(logical, transformation, translation)

write(outputDirectory)

```

pyg4ometry.visualisation.ViewerBase

Module Contents

Classes

ViewerBase

Base class for all viewers and exporters. Handles unique meshes and their instances

Functions

```
_daughterSubtractedMesh(lv)
```

```
pyg4ometry.visualisation.ViewerBase._daughterSubtractedMesh(lv)
```

```
class pyg4ometry.visualisation.ViewerBase.ViewerBase
```

Base class for all viewers and exporters. Handles unique meshes and their instances

```
clear()
```

```
setSubtractDaughters(subtractDaughters=True)
```

```
addLogicalVolume(lv, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]),  
visOptions=_VisOptions(representation='wireframe'), depth=0, name=None)
```

Add a logical volume to viewer (recursively)

Parameters

- **mtra** (*matrix*(3,3)) – Transformation matrix for logical volume
- **tra** (*array*(3)) – Displacement for logical volume
- **visOptions** (*VisualisationOptions*) – VisualisationOptions for the lv mesh

```
addFlukaRegions(fluka_registry, max_region=1000000, debugIO=False)
```

```
addMesh(name, mesh)
```

Add a single mesh

Parameters

- **name** (*str*) – Name of mesh (e.g logical volume name)
- **mesh** (*CSG*) – Mesh to be added

```
addInstance(name, transformation, translation, instanceName="")
```

Add a new instance for mesh with name

Parameters

- **name** (*str*) – name of mesh to add instance
- **transformation** (*array*(3,3)) – Transformation matrix for instance
- **translation** (*array*(3)) – Translation for instance
- **instanceName** (*str*) – Name of the instance e.g PV

```
addVisOptions(name, visOption)
```

Add vis options to mesh with name

Parameters

- **name** (*str*) – name of mesh
- **visOptions** (*VisualisationOptions*) –

addPbrOptions(*name*, *pbrOption*)

Add pbr options to mesh with name

Parameters

- **name** (*str*) – name of mesh
- **pbrOptions** (*PbrOptions*) –

addMaterialVisOption(*materialName*, *visOption*)

Append a visualisation option instance to the dictionary of materials.

Parameters

- **materialName** (*str*) – material name to match
- **visOption** (*VisualisationOptions*) – instance

addMaterialPbrOption(*materialName*, *visOption*)

Append a visualisation option instance to the dictionary of materials.

Parameters

- **materialName** (*str*) – material name to match
- **visOption** (*VisualisationOptions*) – instance

setDefaultVisOptions(*visOption*)

getDefaultVisOptions()

getMaterialVisOptions(*material*)

setOverlapVisOption(*visOption*)

setCoplanarVisOption(*visOption*)

setProtusionVisOption(*visOption*)

getOverlapVisOptions(*overlaptype*)

removeInvisible()

Remove wireframe or transparent instances from self

scaleScene(*scaleFactor*)

exportGLTFScene(*gltfFileName*='test.gltf', *singleInstance*=False)

Export entire scene as gltf file, filename extension dictates binary (glb) or readable json (gltf) *singleInstance* is a Boolean flag to suppress all but one instance

exportGLTFAssets(*gltfFileName*='test.gltf')

Export all the assets (meshes) without all the instances. The position of the asset is the position of the first instance

exportThreeJSScene(*fileNameBase*='test', *lightBoxHDR*='concrete_tunnel_02_4k.hdr')

html based on https://threejs.org/examples/#webgl_loader_gltf HRDI https://polyhaven.com/a/concrete_tunnel_02

npm install node wget https://dl.polyhaven.org/file/ph-assets/HDRIs/exr/4k/concrete_tunnel_02_4k.exr
conver EXR to HDR e.g. <https://convertio.co/exr-hdr/> python -m http.server 8000 open test.html

dumpMeshQuality()

__repr__()

Return repr(self).

pyg4ometry.visualisation.VisualisationOptions**Module Contents****Classes**

<i>VisualisationOptions</i>	Basic holder for visualisation parameters, i.e. colour and opacity.
-----------------------------	---

Functions

<i>randomColour()</i>	Return random RGB tuple
<i>hexRGBToRGBTriplet(value)</i>	
<i>loadPredefined()</i>	Construct a ColorMap initialised with default colours for various materials.
<i>getPredefinedMaterialVisOptions()</i>	
<i>makeVisualisationOptionsDictFromPredefined(C</i>	

Attributes

<i>_predefinedMaterialVisOptions</i>

pyg4ometry.visualisation.VisualisationOptions._predefinedMaterialVisOptions**pyg4ometry.visualisation.VisualisationOptions.randomColour()**

Return random RGB tuple

pyg4ometry.visualisation.VisualisationOptions.hexRGBToRGBTriplet(value)

```
class pyg4ometry.visualisation.VisualisationOptions(representation='surface',  
                                                    colour=[0.5, 0.5,  
                                                    0.5], alpha=0.5,  
                                                    visible=True,  
                                                    lineWidth=1, ran-  
                                                    domColour=False,  
                                                    depth=0)
```

Basic holder for visualisation parameters, i.e. colour and opacity. Construct an instance then modify members.

__repr__()

Return repr(self).

getColour()

Return the colour and generate a random colour if flagged.

_generateRandomColour()

`pyg4ometry.visualisation.VisualisationOptions.loadPredefined()`

Construct a ColorMap initialised with default colours for various materials.

Lods from package resource files.

`pyg4ometry.visualisation.VisualisationOptions.getPredefinedMaterialVisOptions()`

`pyg4ometry.visualisation.VisualisationOptions.makeVisualisationOptionsDictFromPredefined(ColourMap)`

`pyg4ometry.visualisation.VtkExporter`

Module Contents

Classes

VtkExporter

Attributes

_WITH_PARAVIEW

_WITH_PARAVIEW

`pyg4ometry.visualisation.VtkExporter._WITH_PARAVIEW = True`

`pyg4ometry.visualisation.VtkExporter._WITH_PARAVIEW = False`

class `pyg4ometry.visualisation.VtkExporter.VtkExporter(path='.')`

export_to_Paraview(*reg, fileName='Paraview_model.pvsm', model=True, df_model=None, df_color=None*)

Method that exports the visible logical volumes of the registry *reg* into Paraview VTK files (.vtm) and creates a Paraview State file (.pvsm) for the Paraview Model.

df_model and *df_color* are optional: if they are not given, the Paraview model will take the materials colors

Parameters

- **reg** – Registry
- **fileName** – Name of the Paraview State file (.pvsm)
- **model** – Boolean informing whether we work with a “model” GDML or a “simple” GDML

True: it will consider that each daughter volume of the GDML world volume will need a .vtm file False: it will create one .vtm file for the whole GDML :param df_model: (optional) pandas.DataFrame linking the NAME of the element and their TYPE :param df_color: (optional) pandas.DataFrame linking the TYPE with its specific R G B color

export_to_VTK(reg, model=True, df_model=None, df_color=None)

Method that exports the visible logical volumes of the registry reg into Paraview VTK files (.vtm).

Parameters

- **reg** – Registry
- **model** – Boolean informing whether we work with a “model” GDML or a “simple” GDML

True: it will consider that each daughter volume of the GDML world volume will need a .vtm file False: it will create one .vtm file for the whole GDML :param df_model: (optional) pandas.DataFrame linking the NAME of the element and their TYPE :param df_color: (optional) pandas.DataFrame linking the TYPE with its specific R G B color

fill_color_dico(df_gdml, df_model, df_color)

Method that fills a dictionary linking the logical volumes names and their respective color based on the pandas.DataFrame df_color linking the TYPE of the elements and their respective RGB color values.

Parameters

- **df_gdml** – pandas.DataFrame that represents the GDML Structure
- **df_model** – pandas.DataFrame linking the NAME of the element and their TYPE
- **df_color** – pandas.DataFrame linking the TYPE with its specific R G B color

Returns: A dictionary with the keys R, G and B whose item is a dictionary with as keys the logical volumes names and as item the respective RGB value

add_logical_world_volume(lv, model, color_dico={'R': {}, 'G': {}, 'B': {}}, rotation=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), translation=_np.array([0, 0, 0]))

Method that receives the logical world volume and calls the necessary methods to write the .vtm file(s).

Parameters

- **lv** – Logical world volume
- **model** – Boolean informing whether we work with a “model” GDML or a “simple” GDML

True: it will consider that each daughter volume of the GDML world volume will need a .vtm file False: it will create one .vtm file for the whole GDML :param color_dico: A dictionary with the keys R, G and B whose item is a dictionary with as keys the logical volumes names and as item the respective RGB value :param rotation: (optional) numpy.matrix :param translation: (optional) numpy.array

add_logical_volume_recursive(lv, rotation, translation, color_dico, first_level=True)

Method that receives a logical volume and add calls addMesh() on its mesh. The method is recursive and will be called on all the daughter logical volumes of the logical volume.

Parameters

- **lv** – Logical volume
- **rotation** – numpy.matrix
- **translation** – numpy.array
- **color_dico** – A dictionary with the keys R, G and B whose item is a dictionary

with as keys the logical volumes names and as item the respective RGB value :param first_level: Boolean indicating if we are at the first level of recursivity

addMesh(*solid_name, mesh, rotation, translation, visOptions=None*)

Method that converts the mesh into VtkPolyData (.vtp file) and gives it the correct rotation, translation and color.

Parameters

- **solid_name** – Name of the logical volume solid
- **mesh** – Mesh of the logical volume
- **rotation** – numpy.matrix
- **translation** – numpy.array
- **visOptions** – VisualisationOptions instance

getMaterialVisOptions(*name*)

Method that “cleans” the logical volume material string.

Parameters

name (*str*) – raw name of the logical volume material

Returns: clean name of the logical volume material

getElementName(*logicalVolumeName*)

Method that “cleans” the logical volume name string.

Parameters

logicalVolumeName (*str*) – raw name of the logical volume

Returns: clean name of the logical volume

countVisibleDaughters(*lv, element_name, n=0*)

Method that counts the number of “visible” daughter logical volumes of the mother logical volume lv.

Parameters

- **lv** (*pyg4ometry.geant4.LogicalVolume*) – logical volume
- **element_name** (*str*) – name of the element
- **n** (*int*) – number of “visible” daughter volumes

Returns: n

pyg4ometry.visualisation.VtkViewer

Module Contents

Classes

<i>VtkViewer</i>	Visualiser.
<i>VtkViewerColoured</i>	Visualiser that extends VtkViewer. Uses "flat" interpolation and introduces control over colours.
<i>VtkViewerColouredMaterial</i>	Extension of VtkViewerColoured that uses a default material dictionary for
<i>MouseInteractorNamePhysicalVolume</i>	

Functions

<i>axesFromExtents</i> (extent)	
<i>viewLogicalVolumeDifference</i> (referenceLV, otherLV[, ...])	param referenceLV LogicalVolume instance to view viewed in red.

Attributes

<i>PubViewer</i>

class pyg4ometry.visualisation.VtkViewer.VtkViewer(*size*=(1024, 1024), *interpolation*='none', ***kwargs*)

Visualiser.

Parameters

- **size** – (int,int) - (nPixelsHorizontal, nPixelsVeritcal), default (1024,1024)
- **interpolation** – (str) - one of “none”, “flat”, “gouraud”, “phong”

Examples

```
>>> v = VtkViewer()
>>> v.addLogicalVolume(someLV)
>>> v.view()
```

addAxes(*length*=20.0, *origin*=(0, 0, 0))

Add x,y,z axis to the scene.

Parameters

- **length** – float - length of each axis in mm
- **origin** – (float,float,float) - (x,y,z) of origin in mm

addAxesWidget()

setOpacity(*v*, *iActor=-1*)

setWireframe(*iActor=-1*)

setSurface(*iActor=-1*)

setOpacityOverlap(*v*, *iActor=-1*)

setWireframeOverlap(*iActor=-1*)

setSurfaceOverlap(*iActor=-1*)

setRandomColours(*seed=0*)

setCutterOrigin(*dimension*, *origin*)

Parameters

- **dimension** – str - 'x', 'y', or 'z'
- **origin** – list([x,y,z])

setCutterNormal(*dimension*, *normal*)

Parameters

- **dimension** – str - 'x', 'y', or 'z'
- **normal** – list([x,y,z]) - should be unit vector

addMaterialVisOption(*materialName*, *visOptionInstance*)

Append a visualisation option instance to the dictionary of materials.

Parameters

- **materialName** – str - material name to match
- **visOptionInstance** – VisualisationOptions instance

setMaterialVisOptions(*materialDict*)

Replace the (by default None) dictionary for materials to colours :param materialDict: {"materialName": VisualisationOptions}

See also VisualisationOptions.

setCameraFocusPosition(*focalPoint=[0, 0, 0]*, *position=[100, 100, 100]*)

start()

exportOBJScene(*fileName='scene'*)

exportVRMLScene(*fileName='scene'*)

exportGLTFScene(*fileName='scene.gltf'*)

exportScreenShot(*fileName='screenshot.png'*, *rgba=True*)

Write the render window view to an image file.

Image types supported are: BMP, JPEG, PNM, PNG, PostScript, TIFF. The default parameters are used for all writers, change as needed.

Parameters

- **fileName** – The file name, if no extension then PNG is assumed.
- **renWin** – The render window.
- **rgba** – Used to set the buffer type.

Returns

addLogicalVolume(*logical*, *mtra*=*_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])*, *tra*=*_np.array([0, 0, 0])*, *recursive*=*True*, *addWorld*=*True*)

addLogicalVolumeBounding(*logical*)

addSolid(*solid*, *rotation*=[0, 0, 0], *position*=[0, 0, 0], *representation*='surface', *colour*=[0.5, 0.5, 0.5], *opacity*=0.2)

Add a solid to the view with an optional rotation and translation.

Parameters

- **solid** (any *solid* in *pyg4ometry.geant4.solid*) – solid to add to the view
- **rotation** (*list(float, float, float)* - 3 values) – list of TB rotation angles in radians
- **position** (*list(float, float, float)* - 3 values) – translation in global from from centre in mm
- **representation** (*str*) – the way to visualise it, e.g. 'surface' or 'wireframe'
- **colour** (*list(float, float, float)* - 3 values ranging from 0 - 1) – normalised rgb colour to use for the solid mesh
- **opacity** (*float*, from 0 to 1) – the opacity of the solid if surface style

addBooleanSolidRecursive(*solid*, *mtra*=*_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])*, *tra*=*_np.array([0, 0, 0])*, *first*=*True*)

Parameters

solid (*pyg4ometry.geant4.solid.SolidBase*) – *pyg4ometry.geant4.solid* instance.

Other parameters are for internal recursion and don't need to be provided.

Render only a Boolean solid. If one of the constituent solids is also a Boolean, visualise those too. The resultant Boolean is shown in solid form and each constituent in a wireframe. In the case of a null mesh, only the constituents can be shown.

addMeshSimple(*csgMesh*, *visOptions*=*_VisOptions()*, *clip*=*False*, *name*='mesh')

addLogicalVolumeRecursive(*logical*, *mtra*=*_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])*, *tra*=*_np.array([0, 0, 0])*)

addMesh(*pv_name*, *solid_name*, *mesh*, *mtra*, *tra*, *localmeshes*, *filters*, *mappers*, *mapperMap*, *actors*, *actorMap*, *visOptions*=*None*, *overlap*=*False*, *cutters*=*True*, *clippers*=*False*)

view(*interactive*=*True*, *resetCamera*=*True*)

_getCutterData(*axis*='x', *scaling*=1.0)

exportCutterSection(*filename*, *normal*='x', *scaling*=1.0)

Export the section lines in plane perpendicular to normal. Exported as json text.

Parameters

- **filename** – (str) - name of file to export to

- **normal** – (str) - one of “x”, “y” or “z”
- **scaling** – (float) - multiplier for all cutter line coordinates on export

Examples

```
>>> v.exportCutterSection("xz-section.dat", normal="y", scaling=1000)
```

```
viewSection(dir='x')
```

```
addCutterPlane(position, normal, colour=None)
```

Add a cutting plane at position=[x,y,z] with normal [nx,ny,nz].

Parameters

- **position** – [float, float, float] - (x,y,z) position in scene (mm)
- **normal** – [float, float, float] - (nx,ny,z) normal unit vector
- **colour** – None or [float, float, float] - [r,g,b] in range [0:1]

Cutters are stored in self.usercutters.

```
addActor(actor)
```

```
getOverlapVisOptions(overlaptype)
```

```
getMaterialVisOptions(pv)
```

```
_getDefaultVis(pv)
```

```
printViewParameters()
```

```
class pyg4ometry.visualisation.VtkViewer.VtkViewerColoured(*args, defaultColour=None,
                                                            materialVisOptions=None, **kwargs)
```

Bases: [VtkViewer](#)

Visualiser that extends VtkViewer. Uses “flat” interpolation and introduces control over colours.

Keyword Arguments

- **materialVisOptions**: {“materialName”: VisualisationOptions or list or tuple, ... }
- **interpolation** (str): see [VtkViewer](#)
- **defaultColour** (str): “random” or [r,g,b]

Examples

```
>>> vMaterialMap = VtkViewerColoured(materialVisOptions={"G4_WATER": [0,0,1]})
>>> vRandom = VtkViewerColoured(defaultColour="random")
>>> vColoured = VtkViewerColoured(defaultColour=[0.1,0.1,0.1])
>>> vColourAlpha = VtkViewerColoured(defaultColour=[0.1,0.1,0.1,0.5])
```

of use visualisation options instances

```
>>> vo = pyg4ometry.visualisation.VisualisationOptions()
>>> vo.colour = [0.1,1.0,0.5]
>>> vo.alpha = 0.3
>>> options = {'G4_WATER': vo}
>>> vis = VtkViewerColoured(materialVisOptions=options)
```

If the value in the materialVisOptions is a list or a tuple, it will be upgraded to a VisualisationOptions instance.

_getDefaultVis(pv)

pyg4ometry.visualisation.VtkViewer.**PubViewer**

class pyg4ometry.visualisation.VtkViewer.**VtkViewerColouredMaterial**(*args, **kwargs)

Bases: *PubViewer*

Extension of VtkViewerColoured that uses a default material dictionary for several common materials. Material colours are in defined Colour.py for many Geant4, FLUKA and BDSIM materials.

class pyg4ometry.visualisation.VtkViewer.**MouseInteractorNamePhysicalVolume**(renderer,
vtkviewer)

Bases: vtk.vtkInteractorStyleTrackballCamera

rightButtonPressEvent(obj, event)

pyg4ometry.visualisation.VtkViewer.**axesFromExtents**(extent)

pyg4ometry.visualisation.VtkViewer.**viewLogicalVolumeDifference**(referenceLV, otherLV,
otherTranslation=[0, 0, 0],
otherRotation=[0, 0, 0],
viewDifference=True)

Parameters

- **referenceLV** (pyg4ometry.geant4.LogicalVolume.) – LogicalVolume instance to view viewed in red.
- **referenceLV** – LogicalVolume instance to view viewed in blue.
- **otherTranslation** ([float, float, float]) – Translation (in native units, mm) of otherLV w.r.t. referenceLV
- **otherRotation** ([float, float, float]) – Rotation (in native units, rad) of other LV w.r.t. referenceLV.

View the shapes of 2 logical volumes without their contents. The reference one will be red and the ‘other’ one will be blue.

The other one may optionally be translated and rotated (Tait-Bryant x,y,z) relative to the reference.

pyg4ometry.visualisation.VtkViewerNew

Module Contents

Classes

<i>VtkViewerNew</i>	
<i>VtkViewerColouredNew</i>	Visualiser that extends VtkViewer. Uses "flat" interpolation and introduces control over colours.
<i>VtkViewerColouredMaterialNew</i>	Extension of VtkViewerColoured that uses a default material dictionary for
<i>MouseInteractorNamePhysicalVolume</i>	

```

class pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew
    Bases: pyg4ometry.visualisation.ViewerBase
    initVtk()
    clear()
    addAxes(length=20.0, origin=(0, 0, 0))
        Add x,y,z axis to the scene.
        Parameters
            • length – float - length of each axis in mm
            • origin – (float,float,float) - (x,y,z) of origin in mm
    setAxes(iAxes, length, origin)
    addAxesWidget()
    addCutter(name, origin, normal)
    setCutter(name, origin, normal)
    addCutterWidget()
    exportCutter(name, fileName)
    getCutterPolydata(name)
    addClipper(origin, normal, bClipperCutter=False, bClipperCloseCuts=True)
    setClipper(origin, normal)
    addClipperWidget()
    updateClipperPlaneCallback(obj, event)
    _polydata2Actor(polydata)
    buildPipelines()
    buildPipelinesSeparate()
    buildPipelinesAppend()
    buildPipelinesTransformed()

```

render()

view(*interactive=True, resetCamera=False*)

__repr__()

addTracks(*pd*)

addScoringMesh(*gd*)

```
class pyg4ometry.visualisation.VtkViewerNew.VtkViewerColouredNew(*args, defaultColour=None,
                                                                materialVisOptions=None,
                                                                **kwargs)
```

Bases: [VtkViewerNew](#)

Visualiser that extends VtkViewer. Uses “flat” interpolation and introduces control over colours.

Keyword Arguments

- **materialVisOptions**: {“materialName”: VisualisationOptions or list or tuple, ... }
- **interpolation** (str): see VtkViewer
- **defaultColour** (str): “random” or [r,g,b]

Examples

```
>>> vMaterialMap = VtkViewerColoured(materialVisOptions={"G4_WATER": [0,0,1]})
>>> vRandom = VtkViewerColoured(defaultColour="random")
>>> vColoured = VtkViewerColoured(defaultColour=[0.1,0.1,0.1])
>>> vColourAlpha = VtkViewerColoured(defaultColour=[0.1,0.1,0.1,0.5])
```

of use visualisation options instances

```
>>> vo = pyg4ometry.visualisation.VisualisationOptions()
>>> vo.colour = [0.1,1.0,0.5]
>>> vo.alpha = 0.3
>>> options = {'G4_WATER': vo}
>>> vis = VtkViewerColoured(materialVisOptions=options)
```

If the value in the materialVisOptions is a list or a tuple, it will be upgraded to a VisualisationOptions instance.

_getDefaultVis(*pv*)

```
class pyg4ometry.visualisation.VtkViewerNew.VtkViewerColouredMaterialNew(*args, **kwargs)
```

Bases: [VtkViewerColouredNew](#)

Extension of VtkViewerColoured that uses a default material dictionary for several common materials. Material colours are in defined Colour.py for many Geant4, FLUKA and BDSIM materials.

```
class pyg4ometry.visualisation.VtkViewerNew.MouseInteractorNamePhysicalVolume(renderer,
                                                                              vtkviewer)
```

Bases: [vtk.vtkInteractorStyleTrackballCamera](#)

removeHighLight()

removeHighLightText()

rightButtonPressEvent(*obj, event*)

pyg4ometry.visualisation.Writer**Module Contents****Functions**

<code>writeVtkPolyDataAsSTLFile(fileName, meshes)</code>	<code>meshes</code> : list of triFilters
--	--

`pyg4ometry.visualisation.Writer.writeVtkPolyDataAsSTLFile(fileName, meshes)`
`meshes` : list of triFilters

Package Contents**Classes**

<i>Mesh</i>	
<i>OverlapType</i>	
<i>ViewerBase</i>	Base class for all viewers and exporters. Handles unique meshes and their instances
<i>VisualisationOptions</i>	
<i>VtkViewer</i>	
<i>VtkViewerColoured</i>	Visualiser that extends VtkViewer. Uses "flat" interpolation and introduces control over colours.
<i>VtkViewerColouredMaterial</i>	Extension of VtkViewerColoured that uses a default material dictionary for
<i>MouseInteractorNamePhysicalVolume</i>	
<i>_VisOptions</i>	Basic holder for visualisation parameters, i.e. colour and opacity.
<i>VtkViewerNew</i>	
<i>VtkViewerColouredNew</i>	Visualiser that extends VtkViewer. Uses "flat" interpolation and introduces control over colours.
<i>VtkViewerColouredMaterialNew</i>	Extension of VtkViewerColoured that uses a default material dictionary for
<i>MouseInteractorNamePhysicalVolume</i>	
<i>RenderWriter</i>	
<i>VtkExporter</i>	

Functions

<code>_getBoundingBox(aMesh[, rotationMatrix, translation, ...])</code>	Axes aligned bounding box. Can also provide a rotation and
<code>_getBoundingBoxMesh(boundingBox)</code>	
<code>randomColour()</code> <code>hexRGBToRGBTriplet(value)</code>	Return random RGB tuple
<code>loadPredefined()</code> <code>getPredefinedMaterialVisOptions()</code>	Construct a ColorMap initialised with default colours for various materials.
<code>makeVisualisationOptionsDictFromPredefined(C</code> <code>_getPredefinedMaterialVisOptions()</code>	
<code>axesFromExtents(extent)</code>	
<code>viewLogicalVolumeDifference(referenceLV, otherLV[, ...])</code>	param referenceLV LogicalVolume instance to view viewed in red.
<code>_getPredefinedMaterialVisOptions()</code>	
<code>mkVtkIdList(it)</code>	
<code>pycsgMeshToVtkPolyData(mesh)</code>	
<code>vtkPolyDataToNumpy(data)</code>	
<code>pycsgMeshToObj(mesh, fileName)</code>	
<code>pyg42VtkTransformation(mtra, tra)</code>	
<code>vtkTransformation2PyG4(vt)</code>	
<code>pycsgMeshToStl(mesh, fileName)</code>	

Attributes

<code>_predefinedMaterialVisOptions</code>
<code>PubViewer</code>
<code>_WITH_PARAVIEW</code>


```
class pyg4ometry.visualisation.Mesh(solid)
```

```
    remesh()
```

```
    addOverlapMesh(mesh)
```

```
    getLocalMesh()
```

```
    getBoundingBox(rotationMatrix=None, translation=None)
```

Axes aligned bounding box. Can also provide a rotation and a translation (applied in that order) to the vertices.

```
    getBoundingBoxMesh()
```

```
class pyg4ometry.visualisation.OverlapType
```

```
    protrusion = 1
```

```
    overlap = 2
```

```
    coplanar = 3
```

```
pyg4ometry.visualisation._getBoundingBox(aMesh, rotationMatrix=None, translation=None,
                                         nameForError="")
```

Axes aligned bounding box. Can also provide a rotation and a translation (applied in that order) to the vertices.

```
pyg4ometry.visualisation._getBoundingBoxMesh(boundingBox)
```

```
class pyg4ometry.visualisation.ViewerBase
```

Base class for all viewers and exporters. Handles unique meshes and their instances

```
    clear()
```

```
    setSubtractDaughters(subtractDaughters=True)
```

```
    addLogicalVolume(lv, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]),
                    visOptions=_VisOptions(representation='wireframe'), depth=0, name=None)
```

Add a logical volume to viewer (recursively)

Parameters

- **mtra** (*matrix*(3,3)) – Transformation matrix for logical volume
- **tra** (*array*(3)) – Displacement for logical volume
- **visOptions** (*VisualisationOptions*) – VisualisationOptions for the lv mesh

```
    addFlukaRegions(fluka_registry, max_region=1000000, debugIO=False)
```

```
    addMesh(name, mesh)
```

Add a single mesh

Parameters

- **name** (*str*) – Name of mesh (e.g logical volume name)
- **mesh** (*CSG*) – Mesh to be added

addInstance(*name, transformation, translation, instanceName=""*)

Add a new instance for mesh with name

Parameters

- **name** (*str*) – name of mesh to add instance
- **transformation** (*array(3,3)*) – Transformation matrix for instance
- **translation** (*array(3)*) – Translation for instance
- **instanceName** (*str*) – Name of the instance e.g PV

addVisOptions(*name, visOption*)

Add vis options to mesh with name

Parameters

- **name** (*str*) – name of mesh
- **visOptions** (*VisualisationOptions*) –

addPbrOptions(*name, pbrOption*)

Add pbr options to mesh with name

Parameters

- **name** (*str*) – name of mesh
- **pbrOptions** (*PbrOptions*) –

addMaterialVisOption(*materialName, visOption*)

Append a visualisation option instance to the dictionary of materials.

Parameters

- **materialName** (*str*) – material name to match
- **visOption** (*VisualisationOptions*) – instance

addMaterialPbrOption(*materialName, visOption*)

Append a visualisation option instance to the dictionary of materials.

Parameters

- **materialName** (*str*) – material name to match
- **visOption** (*VisualisationOptions*) – instance

setDefaultVisOptions(*visOption*)

getDefaultVisOptions()

getMaterialVisOptions(*material*)

setOverlapVisOption(*visOption*)

setCoplanarVisOption(*visOption*)

setProtusionVisOption(*visOption*)

getOverlapVisOptions(*overlaptype*)

removeInvisible()

Remove wireframe or transparent instances from self

scaleScene(*scaleFactor*)

exportGLTFScene(*gltfFileName='test.gltf', singleInstance=False*)

Export entire scene as gltf file, filename extension dictates binary (glb) or readable json (gltf) *singleInstance* is a Boolean flag to suppress all but one instance

exportGLTFAssets(*gltfFileName='test.gltf'*)

Export all the assets (meshes) without all the instances. The position of the asset is the position of the first instance

exportThreeJSScene(*fileNameBase='test', lightBoxHDR='concrete_tunnel_02_4k.hdr'*)

html based on https://threejs.org/examples/#webgl_loader_gltf HRDI https://polyhaven.com/a/concrete_tunnel_02

npm install node wget https://dl.polyhaven.org/file/ph-assets/HDRIs/exr/4k/concrete_tunnel_02_4k.exr
 conver EXR to HDR e.g. <https://convertio.co/exr-hdr/> python -m http.server 8000 open test.html

dumpMeshQuality()

__repr__()

Return repr(self).

pyg4ometry.visualisation._predefinedMaterialVisOptions

pyg4ometry.visualisation.randomColour()

Return random RGB tuple

pyg4ometry.visualisation.hexRGBToRGBTriplet(*value*)

class pyg4ometry.visualisation.VisualisationOptions(*representation='surface', colour=[0.5, 0.5, 0.5], alpha=0.5, visible=True, lineWidth=1, randomColour=False, depth=0*)

Basic holder for visualisation parameters, i.e. colour and opacity. Construct an instance then modify members.

__repr__()

Return repr(self).

getColour()

Return the colour and generate a random colour if flagged.

_generateRandomColour()

pyg4ometry.visualisation.loadPredefined()

Construct a ColorMap initialised with default colours for various materials.

Lods from package resource files.

pyg4ometry.visualisation.getPredefinedMaterialVisOptions()

pyg4ometry.visualisation.makeVisualisationOptionsDictFromPredefined(*ColourMap*)

pyg4ometry.visualisation._getPredefinedMaterialVisOptions()

class pyg4ometry.visualisation.VtkViewer(*size=(1024, 1024), interpolation='none', **kwargs*)

Visualiser.

Parameters

- **size** – (int,int) - (nPixelsHorizontal, nPixelsVeritcal), default (1024,1024)
- **interpolation** – (str) - one of “none”, “flat”, “gouraud”, “phong”

Examples

```
>>> v = VtkViewer()
>>> v.addLogicalVolume(someLV)
>>> v.view()
```

addAxes(*length=20.0, origin=(0, 0, 0)*)

Add x,y,z axis to the scene.

Parameters

- **length** – float - length of each axis in mm
- **origin** – (float,float,float) - (x,y,z) of origin in mm

addAxesWidget()

setOpacity(*v, iActor=-1*)

setWireframe(*iActor=-1*)

setSurface(*iActor=-1*)

setOpacityOverlap(*v, iActor=-1*)

setWireframeOverlap(*iActor=-1*)

setSurfaceOverlap(*iActor=-1*)

setRandomColours(*seed=0*)

setCutterOrigin(*dimension, origin*)

Parameters

- **dimension** – str - 'x', 'y', or 'z'
- **origin** – list([x,y,z])

setCutterNormal(*dimension, normal*)

Parameters

- **dimension** – str - 'x', 'y', or 'z'
- **normal** – list([x,y,z]) - should be unit vector

addMaterialVisOption(*materialName, visOptionInstance*)

Append a visualisation option instance to the dictionary of materials.

Parameters

- **materialName** – str - material name to match
- **visOptionInstance** – [VisualisationOptions](#) instance

setMaterialVisOptions(*materialDict*)

Replace the (by default None) dictionary for materials to colours :param materialDict: {"materialName": VisualisationOptions}

See also [VisualisationOptions](#).

setCameraFocusPosition(*focalPoint=[0, 0, 0], position=[100, 100, 100]*)

```

start()

exportOBJScene(fileName='scene')

exportVRMLScene(fileName='scene')

exportGLTFScene(fileName='scene.gltf')

exportScreenShot(fileName='screenshot.png', rgba=True)

```

Write the render window view to an image file.

Image types supported are: BMP, JPEG, PNM, PNG, PostScript, TIFF. The default parameters are used for all writers, change as needed.

Parameters

- **fileName** – The file name, if no extension then PNG is assumed.
- **renWin** – The render window.
- **rgba** – Used to set the buffer type.

Returns

```
addLogicalVolume(logical, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]),
                recursive=True, addWorld=True)
```

```
addLogicalVolumeBounding(logical)
```

```
addSolid(solid, rotation=[0, 0, 0], position=[0, 0, 0], representation='surface', colour=[0.5, 0.5, 0.5],
         opacity=0.2)
```

Add a solid to the view with an optional rotation and translation.

Parameters

- **solid** (any *solid* in `pyg4ometry.geant4.solid`) – solid to add to the view
- **rotation** (`list(float, float, float)` - 3 values) – list of TB rotation angles in radians
- **position** (`list(float, float, float)` - 3 values) – translation in global from from centre in mm
- **representation** (`str`) – the way to visualise it, e.g. ‘surface’ or ‘wireframe’
- **colour** (`list(float, float, float)` - 3 values ranging from 0 - 1) – normalised rgb colour to use for the solid mesh
- **opacity** (`float, from 0 to 1`) – the opacity of the solid if surface style

```
addBooleanSolidRecursive(solid, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]),
                        first=True)
```

Parameters

solid (`pyg4ometry.geant4.solid.SolidBase`) – `pyg4ometry.geant4.solid` instance.

Other parameters are for internal recursion and don’t need to be provided.

Render only a Boolean solid. If one of the constituent solids is also a Boolean, visualise those too. The resultant Boolean is shown in solid form and each constituent in a wireframe. In the case of a null mesh, only the constituents can be shown.

```
addMeshSimple(csgMesh, visOptions=_VisOptions(), clip=False, name='mesh')
```

```
addLogicalVolumeRecursive(logical, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]))
```

```
addMesh(pv_name, solid_name, mesh, mtra, tra, localmeshes, filters, mappers, mapperMap, actors, actorMap, visOptions=None, overlap=False, cutters=True, clippers=False)
```

```
view(interactive=True, resetCamera=True)
```

```
_getCutterData(axis='x', scaling=1.0)
```

```
exportCutterSection(filename, normal='x', scaling=1.0)
```

Export the section lines in plane perpendicular to normal. Exported as json text.

Parameters

- **filename** – (str) - name of file to export to
- **normal** – (str) - one of “x”, “y” or “z”
- **scaling** – (float) - multiplier for all cutter line coordinates on export

Examples

```
>>> v.exportCutterSection("xz-section.dat", normal="y", scaling=1000)
```

```
viewSection(dir='x')
```

```
addCutterPlane(position, normal, colour=None)
```

Add a cutting plane at position=[x,y,z] with normal [nx,ny,nz].

Parameters

- **position** – [float, float, float] - (x,y,z) position in scene (mm)
- **normal** – [float, float, float] - (nx,ny,z) normal unit vector
- **colour** – None or [float, float, float] - [r,g,b] in range [0:1]

Cutters are stored in self.usercutters.

```
addActor(actor)
```

```
getOverlapVisOptions(overlaptype)
```

```
getMaterialVisOptions(pv)
```

```
_getDefaultVis(pv)
```

```
printViewParameters()
```

```
class pyg4ometry.visualisation.VtkViewerColoured(*args, defaultColour=None, materialVisOptions=None, **kwargs)
```

Bases: [VtkViewer](#)

Visualiser that extends VtkViewer. Uses “flat” interpolation and introduces control over colours.

Keyword Arguments

- **materialVisOptions**: {“materialName”: [VisualisationOptions](#) or list or tuple, ... }
- **interpolation** (str): see [VtkViewer](#)
- **defaultColour** (str): “random” or [r,g,b]

Examples

```
>>> vMaterialMap = VtkViewerColoured(materialVisOptions={"G4_WATER":[0,0,1]})
>>> vRandom = VtkViewerColoured(defaultColour="random")
>>> vColoured = VtkViewerColoured(defaultColour=[0.1,0.1,0.1])
>>> vColourAlpha = VtkViewerColoured(defaultColour=[0.1,0.1,0.1,0.5])
```

of use visualisation options instances

```
>>> vo = pyg4ometry.visualisation.VisualisationOptions()
>>> vo.colour = [0.1,1.0,0.5]
>>> vo.alpha = 0.3
>>> options = {'G4_WATER':vo}
>>> vis = VtkViewerColoured(materialVisOptions=options)
```

If the value in the materialVisOptions is a list or a tuple, it will be upgraded to a *VisualisationOptions* instance.

_getDefaultVis(pv)

pyg4ometry.visualisation.**PubViewer**

class pyg4ometry.visualisation.**VtkViewerColouredMaterial**(*args, **kwargs)

Bases: *PubViewer*

Extension of VtkViewerColoured that uses a default material dictionary for several common materials. Material colours are in defined Colour.py for many Geant4, FLUKA and BDSIM materials.

class pyg4ometry.visualisation.**MouseInteractorNamePhysicalVolume**(renderer, vtkviewer)

Bases: vtk.vtkInteractorStyleTrackballCamera

rightButtonPressEvent(obj, event)

pyg4ometry.visualisation.**axesFromExtents**(extent)

pyg4ometry.visualisation.**viewLogicalVolumeDifference**(referenceLV, otherLV, otherTranslation=[0, 0, 0], otherRotation=[0, 0, 0], viewDifference=True)

Parameters

- **referenceLV** (pyg4ometry.geant4.LogicalVolume.) – LogicalVolume instance to view viewed in red.
- **referenceLV** – LogicalVolume instance to view viewed in blue.
- **otherTranslation** ([float, float, float]) – Translation (in native units, mm) of otherLV w.r.t. referenceLV
- **otherRotation** ([float, float, float]) – Rotation (in native units, rad) of other LV w.r.t. referenceLV.

View the shapes of 2 logical volumes without their contents. The reference one will be red and the ‘other’ one will be blue.

The other one may optionally be translated and rotated (Tait-Bryant x,y,z) relative to the reference.

class pyg4ometry.visualisation.**_VisOptions**(representation='surface', colour=[0.5, 0.5, 0.5], alpha=0.5, visible=True, lineWidth=1, randomColour=False, depth=0)

Basic holder for visualisation parameters, i.e. colour and opacity. Construct an instance then modify members.

__repr__()

Return repr(self).

getColour()

Return the colour and generate a random colour if flagged.

_generateRandomColour()

pyg4ometry.visualisation._getPredefinedMaterialVisOptions()

class pyg4ometry.visualisation.VtkViewerNew

Bases: *pyg4ometry.visualisation.ViewerBase*

initVtk()

clear()

addAxes(*length=20.0, origin=(0, 0, 0)*)

Add x,y,z axis to the scene.

Parameters

- **length** – float - length of each axis in mm
- **origin** – (float,float,float) - (x,y,z) of origin in mm

setAxes(*iAxes, length, origin*)

addAxesWidget()

addCutter(*name, origin, normal*)

setCutter(*name, origin, normal*)

addCutterWidget()

exportCutter(*name, fileName*)

getCutterPolydata(*name*)

addClipper(*origin, normal, bClipperCutter=False, bClipperCloseCuts=True*)

setClipper(*origin, normal*)

addClipperWidget()

updateClipperPlaneCallback(*obj, event*)

_polydata2Actor(*polydata*)

buildPipelines()

buildPipelinesSeparate()

buildPipelinesAppend()

buildPipelinesTransformed()

render()

view(*interactive=True, resetCamera=False*)

`__repr__()`

`addTracks(pd)`

`addScoringMesh(gd)`

```
class pyg4ometry.visualisation.VtkViewerColouredNew(*args, defaultColour=None,
                                                    materialVisOptions=None, **kwargs)
```

Bases: [VtkViewerNew](#)

Visualiser that extends VtkViewer. Uses “flat” interpolation and introduces control over colours.

Keyword Arguments

- **materialVisOptions**: {“materialName”: [VisualisationOptions](#) or list or tuple, ... }
- **interpolation** (str): see [VtkViewer](#)
- **defaultColour** (str): “random” or [r,g,b]

Examples

```
>>> vMaterialMap = VtkViewerColoured(materialVisOptions={"G4_WATER": [0,0,1]})
>>> vRandom = VtkViewerColoured(defaultColour="random")
>>> vColoured = VtkViewerColoured(defaultColour=[0.1,0.1,0.1])
>>> vColourAlpha = VtkViewerColoured(defaultColour=[0.1,0.1,0.1,0.5])
```

of use visualisation options instances

```
>>> vo = pyg4ometry.visualisation.VisualisationOptions()
>>> vo.colour = [0.1,1.0,0.5]
>>> vo.alpha = 0.3
>>> options = {'G4_WATER': vo}
>>> vis = VtkViewerColoured(materialVisOptions=options)
```

If the value in the materialVisOptions is a list or a tuple, it will be upgraded to a [VisualisationOptions](#) instance.

`_getDefaultVis(pv)`

```
class pyg4ometry.visualisation.VtkViewerColouredMaterialNew(*args, **kwargs)
```

Bases: [VtkViewerColouredNew](#)

Extension of VtkViewerColoured that uses a default material dictionary for several common materials. Material colours are in defined Colour.py for many Geant4, FLUKA and BDSIM materials.

```
class pyg4ometry.visualisation.MouseInteractorNamePhysicalVolume(renderers, vtkviewer)
```

Bases: `vtk.vtkInteractorStyleTrackballCamera`

`removeHighLight()`

`removeHighLightText()`

`rightButtonPressEvent(obj, event)`

```
class pyg4ometry.visualisation.RenderWriter
```

```
addLogicalVolumeRecursive(logical, mtra=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), tra=_np.array([0, 0, 0]))
```

```
addMesh(logical)

addInstance(logical, transformation, translation)

write(outputDirectory)

pyg4ometry.visualisation.mkVtkIdList(it)

pyg4ometry.visualisation.pycsgMeshToVtkPolyData(mesh)

pyg4ometry.visualisation.vtkPolyDataToNumpy(data)

pyg4ometry.visualisation.pycsgMeshToObj(mesh, fileName)

pyg4ometry.visualisation.pyg42VtkTransformation(mtra, tra)

pyg4ometry.visualisation.vtkTransformation2PyG4(vt)

pyg4ometry.visualisation.pycsgMeshToStl(mesh, fileName)

pyg4ometry.visualisation._WITH_PARAVIEW = False

class pyg4ometry.visualisation.VtkExporter(path='.')
```

```
export_to_Paraview(reg, fileName='Paraview_model.pvsm', model=True, df_model=None,
                  df_color=None)
```

Method that exports the visible logical volumes of the registry reg into Paraview VTK files (.vtm) and creates a Paraview State file (.pvsm) for the Paraview Model.

df_model and df_color are optional: if they are not given, the Paraview model will take the materials colors

Parameters

- **reg** – Registry
- **fileName** – Name of the Paraview State file (.pvsm)
- **model** – Boolean informing whether we work with a “model” GDML or a “simple” GDML

True: it will consider that each daughter volume of the GDML world volume will need a .vtm file False: it will create one .vtm file for the whole GDML :param df_model: (optional) pandas.DataFrame linking the NAME of the element and their TYPE :param df_color: (optional) pandas.DataFrame linking the TYPE with its specific R G B color

```
export_to_VTK(reg, model=True, df_model=None, df_color=None)
```

Method that exports the visible logical volumes of the registry reg into Paraview VTK files (.vtm).

Parameters

- **reg** – Registry
- **model** – Boolean informing whether we work with a “model” GDML or a “simple” GDML

True: it will consider that each daughter volume of the GDML world volume will need a .vtm file False: it will create one .vtm file for the whole GDML :param df_model: (optional) pandas.DataFrame linking the NAME of the element and their TYPE :param df_color: (optional) pandas.DataFrame linking the TYPE with its specific R G B color

```
fill_color_dico(df_gdml, df_model, df_color)
```

Method that fills a dictionary linking the logical volumes names and their respective color based on the pandas.DataFrame df_color linking the TYPE of the elements and their respective RGB color values.

Parameters

- **df_gdml** – pandas.DataFrame that represents the GDML Structure
- **df_model** – pandas.DataFrame linking the NAME of the element and their TYPE
- **df_color** – pandas.DataFrame linking the TYPE with its specific R G B color

Returns: A dictionary with the keys R, G and B whose item is a dictionary with as keys the logical volumes names and as item the respective RGB value

add_logical_world_volume(*lv, model, color_dico*={'R': {}, 'G': {}, 'B': {}}, *rotation*=_np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]), *translation*=_np.array([0, 0, 0]))

Method that receives the logical world volume and calls the necessary methods to write the .vtm file(s).

Parameters

- **lv** – Logical world volume
- **model** – Boolean informing whether we work with a “model” GDML or a “simple” GDML

True: it will consider that each daughter volume of the GDML world volume will need a .vtm file False: it will create one .vtm file for the whole GDML :param color_dico: A dictionary with the keys R, G and B whose item is a dictionary with as keys the logical volumes names and as item the respective RGB value :param rotation: (optional) numpy.matrix :param translation: (optional) numpy.array

add_logical_volume_recursive(*lv, rotation, translation, color_dico, first_level=True*)

Method that receives a logical volume and add calls addMesh() on its mesh. The method is recursive and will be called on all the daughter logical volumes of the logical volume.

Parameters

- **lv** – Logical volume
- **rotation** – numpy.matrix
- **translation** – numpy.array
- **color_dico** – A dictionary with the keys R, G and B whose item is a dictionary

with as keys the logical volumes names and as item the respective RGB value :param first_level: Boolean indicating if we are at the first level of recursivity

addMesh(*solid_name, mesh, rotation, translation, visOptions=None*)

Method that converts the mesh into VtkPolyData (.vtp file) and gives it the correct rotation, translation and color.

Parameters

- **solid_name** – Name of the logical volume solid
- **mesh** – Mesh of the logical volume
- **rotation** – numpy.matrix
- **translation** – numpy.array
- **visOptions** – VisualisationOptions instance

getMaterialVisOptions(*name*)

Method that “cleans” the logical volume material string.

Parameters

- **name** (*str*) – raw name of the logical volume material

Returns: clean name of the logical volume material

getElementName(*logicalVolumeName*)

Method that “cleans” the logical volume name string.

Parameters

logicalVolumeName (*str*) – raw name of the logical volume

Returns: clean name of the logical volume

countVisibleDaughters(*lv, element_name, n=0*)

Method that counts the number of “visible” daughter logical volumes of the mother logical volume lv.

Parameters

- **lv** (`pyg4ometry.geant4.LogicalVolume`) – logical volume
- **element_name** (*str*) – name of the element
- **n** (*int*) – number of “visible” daughter volumes

Returns: n

5.1.2 Submodules

`pyg4ometry.cli`

Module Contents

Functions

<code>_loadFile(fileName)</code>
<code>_writeFile(fileName, reg)</code>
<code>_parseStrMultipletAsFloat(strTriplet)</code>
<code>_parseStrPythonAsSolid(reg, strPython)</code>
<code>_parseStrPythonAsDict(strPython)</code>
<code>_printCitation()</code>
<code>cli([inputFileName, view, bounding, checkOverlaps, ...])</code>
<code>main()</code>

`pyg4ometry.cli._loadFile(fileName)`

`pyg4ometry.cli._writeFile(fileName, reg)`

`pyg4ometry.cli._parseStrMultipletAsFloat(strTriplet)`

`pyg4ometry.cli._parseStrPythonAsSolid(reg, strPython)`

```
pyg4ometry.cli._parseStrPythonAsDict(strPython)
```

```
pyg4ometry.cli._printCitation()
```

```
pyg4ometry.cli.cli(inputFileName=None, view=False, bounding=False, checkOverlaps=False, analysis=False,
                  nullMeshException=False, compareFileName=None, appendFileName=None,
                  lvName=None, info=None, exchangeLvName=None, clip=None, solid=None,
                  translation=None, rotation=None, materials=None, outputFileName=None,
                  planeCutterData=None, planeCutterOutputFileName=None, featureData=None,
                  featureDataOutputFileName=None, gltfScale=None, verbose=None)
```

```
pyg4ometry.cli.main()
```

pyg4ometry.config

We check that sweep angles aren't greater than 2 pi. This is the tolerance for rounding errors. The default is around float precision.

Module Contents

Classes

```
meshingType
```

```
SolidDefaults
```

Functions

```
backendName()
```

```
setGlobalMeshSliceAndStack(value)
```

Attributes

```
twoPiComparisonTolerance
```

```
meshing
```

```
meshingNullException
```

```
doMeshing
```

```
pyg4ometry.config.twoPiComparisonTolerance = 1e-07

class pyg4ometry.config.meshingType
    pycsg = 1
    cgal_sm = 2
    cgal_np = 3

pyg4ometry.config.meshing
pyg4ometry.config.meshingNullException = True
pyg4ometry.config.backendName()
pyg4ometry.config.doMeshing = True
pyg4ometry.config.setGlobalMeshSliceAndStack(value)

class pyg4ometry.config.SolidDefaults
    class Tubs
        nslice = 16
    class Cons
        nslice = 16
    class CutTubs
        nslice = 16
    class Ellipsoid
        nslice = 8
        nstack = 8
    class EllipticalCone
        nslice = 16
        nstack = 16
    class EllipticalTube
        nslice = 20
        nstack = 10
    class GenericPolycone
        nslice = 16
    class Hype
        nslice = 16
        nstack = 16
```

```
class Orb
    nslice = 16
    nstack = 16
class Paraboloid
    nslice = 16
    nstack = 8
class Polycone
    nslice = 16
class Sphere
    nslice = 10
    nstack = 10
class Torus
    nslice = 50
    nstack = 10
class TwistedBox
    nstack = 20
class TwistedTrap
    nstack = 3
class TwistedTrd
    nstack = 20
class TwistedTubs
    nslice = 20
    nstack = 20
class Wedge
    nslice = 16
```

`pyg4ometry.exceptions`

Module Contents

exception `pyg4ometry.exceptions.NullMeshError(arg)`

Bases: `Exception`

Null Mesh exception. Should be raised when the output of `mesh.toPolygons` evaluates to the empty list. `arg` can be a user-provided message (string), or a `SolidBase`-derived instance.

exception `pyg4ometry.exceptions.IdenticalNameError(name, nametype=None)`

Bases: `Exception`

Exception for trying to add the same name to the geant4 registry.

Parameters

- **name** (*str*) – the name that has been duplicated
- **nametype** (*str*) – optional extra information regarding the nature of the duplicate (“material”, “solid”, etc.)

exception `pyg4ometry.exceptions.FLUKAError`

Bases: `Exception`

Common base class for all non-exit exceptions.

exception `pyg4ometry.exceptions.FLUKAInputError(message)`

Bases: `Exception`

Common base class for all non-exit exceptions.

`pyg4ometry.meshutils`

Module Contents

Functions

MeshShrink(*m*[, *shrinkFactor*])

`pyg4ometry.meshutils.MeshShrink(m, shrinkFactor=1e-05)`

`pyg4ometry.transformation`

Module Contents

Functions

<code>rad2deg(rad)</code>	Convert rad in radians into degrees
<code>deg2rad(deg)</code>	Convert deg in degrees into radians
<code>grad2rad(gradians)</code>	Convert rad in gradians into radians
<code>tbxyz2axisangle(rv)</code>	Tait-Bryan x-y-z rotation to axis-angle representation
<code>matrix2axisangle(matrix)</code>	Convert 3x3 transformation matrix to axis angle representation
<code>axisangle2matrix(axis, angle)</code>	Convert axis angle to transformation matrix
<code>matrix2tbxyz(matrix)</code>	Convert rotation matrix to Tait-Bryan angles.
<code>axisangle2tbxyz(axis, angle)</code>	Convert axis and angle to tait bryan angles
<code>tbxyz2matrix(angles)</code>	Convert tait bryan angles to a single passive rotation matrix.
<code>tbzyx2matrix(angles)</code>	Convert Tait-Bryan angles to a single passive rotation matrix.
<code>matrix_from(v_from, v_to)</code>	Returns the rotation matrix that rotates v_from to parallel to
<code>_rodrigues_anti_parallel(v_from, v_to)</code>	v_from = vector FROM
<code>are_parallel(vector_1, vector_2[, tolerance])</code>	Check if vector vector_1 is parallel to vector vector_2 down to
<code>are_anti_parallel(vector_1, vector_2[, tolerance])</code>	Check if vector vector_1 is parallel to vector vector_2 down to
<code>reverse(angles)</code>	Invert the rotation represented by these angles.
<code>two_fold_orientation(v1, v2, e1, e2)</code>	matrix_from will align one vector with another, but there are

`pyg4ometry.transformation.rad2deg(rad)`

Convert rad in radians into degrees

Parameters

rad (*float*) – Input in radians

Returns

rad in degrees

Return type

float

`pyg4ometry.transformation.deg2rad(deg)`

Convert deg in degrees into radians

Parameters

deg (*float*) – Input in degrees

Returns

deg in radians

Return type

float

`pyg4ometry.transformation.grad2rad(gradians)`

Convert rad in gradians into radians

Parameters

gradians (*float*) – Input in gradians

Returns

gradians in radians

Return type

float

`pyg4ometry.transformation.tbxyz2axisangle(rv)`

Tait-Bryan x-y-z rotation to axis-angle representation Algorithm from <http://www.sedris.org/wg8home/Documents/WG80485.pdf>

For converting rotation angles to an active axis/angle pair for use in pycsg. Order of rotation: x->y->z.

Parameters

rv (*float*(3)) – rotation angles

Returns

[axis,angle]

Return type

list(list(3),float)

`pyg4ometry.transformation.matrix2axisangle(matrix)`

Convert 3x3 transformation matrix to axis angle representation

Parameters

matrix (*array*(3,3)) – 3x3 rotation matrix array

Returns

[axis,angle]

Return type

list(list(3),float)

`pyg4ometry.transformation.axisangle2matrix(axis, angle)`

Convert axis angle to transformation matrix

Parameters

- **axis** (*list/array*(3)) – axis for rotation
- **angle** (*float*) – rotation angle

Returns

transformation matrix

Return type

array(3,3)

`pyg4ometry.transformation.matrix2tbxyz(matrix)`

Convert rotation matrix to Tait-Bryan angles. Order of rotation is x -> y -> z.

Parameters

matrix – active (positive angle = anti-clockwise rotation about that axis when looking at the axis) matrix.

Returns

[x, y, z] Tait-Bryan angles in a list.

Return type

list(3)

`pyg4ometry.transformation.axisangle2tbxyz(axis, angle)`

Convert axis and angle to tait bryan angles

Parameters

- **axis** (*list/array(3)*) – axis for rotation
- **angle** (*float*) – rotation angle

Returns

tait bryan angles (x-y-z)

Return type

array(3)

`pyg4ometry.transformation.tbxyz2matrix(angles)`

Convert tait bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters

angles – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

`pyg4ometry.transformation.tbzyx2matrix(angles)`

Convert Tait-Bryan angles to a single passive rotation matrix. rotation order = x -> y -> z.

Parameters

angles – list of angles: x, y, z

Returns

rotation matrix

Return type

array(3,3)

`pyg4ometry.transformation.matrix_from(v_from, v_to)`

Returns the rotation matrix that rotates *v_from* to parallel to *v_to*.

Useful for ensuring a given face points in a certain direction.

v_from and *v_to* should be array-like three-vectors.

`pyg4ometry.transformation._rodrigues_anti_parallel(v_from, v_to)`

v_from = vector FROM *v_to* = vector TO

source: http://en.citizendium.org/wiki/Rotation_matrix#Case_that_.22from.22_and_.22to.22_vectors_are_anti-parallel

`pyg4ometry.transformation.are_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector *vector_1* is parallel to vector *vector_2* down to some tolerance.

Parameters

- **vector_1** (*array*) – First input vector
- **vector_2** (*array*) – Second input vector
- **tolerance** (*float*) – Tolerance for calculation

Returns

if vectors are parallel

Return type`bool``pyg4ometry.transformation.are_anti_parallel(vector_1, vector_2, tolerance=1e-10)`

Check if vector `vector_1` is parallel to vector `vector_2` down to some tolerance.

Parameters

- **vector_1** (`array`) – First input vector
- **vector_2** (`array`) – Second input vector
- **tolerance** (`float`) – Tolerance for calculation

Returns

if vectors are antiparallel

Return type`bool``pyg4ometry.transformation.reverse(angles)`

Invert the rotation represented by these angles.

`pyg4ometry.transformation.two_fold_orientation(v1, v2, e1, e2)`

`matrix_from` will align one vector with another, but there are an infinite number of such matrices that align two vectors. This further constrains the rotation by introducing a second pair of vectors.

`v1` start `v` `v2` end `v` `e1` start `e` `e2` end `v`

pyg4ometry.utils**Module Contents****Classes**

*Samples**Timer*

A class for recording the CPU time at specific positions within code

Functions

`_write_pickle(obj, path)``_load_pickle(path)`

`pyg4ometry.utils._write_pickle(obj, path)``pyg4ometry.utils._load_pickle(path)``class pyg4ometry.utils.Samples(**metadata)`

```

add(name, time)

n_samples_description()

sample_names(exclude=None)

means(exclude=None)

stds(exclude=None)

classmethod from_existing(*samples)
    Merge two or more existing samples non-destructively to create a new Sample instance.

__str__()
    Return str(self).

write(path)

writeAppend(path, verbose=False)

__getitem__(key)

__setitem__(key, value)

__contains__(key)

```

```
class pyg4ometry.utils.Timer(**metadata)
```

A class for recording the CPU time at specific positions within code with the use of a name. Records time stamps in a Samples instance. To record multiple samples of a given event, simply call add with the same name repeatedly.

```

timer = Timer()

while True:
    ... timer.add("step1") ... timer.add("step2")

mean_step1 = timer.samples.means()["step1"] std_step1 = timer.samples.stds()["step1"]

add(name)

update()
    Update the last_time attribute to now.

__str__()
    Return str(self).

updateTotal()

```

5.1.3 Package Contents

```
pyg4ometry.__version__ = 'unknown version'
```


INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- pyg4ometry, 49
- pyg4ometry.analysis, 49
- pyg4ometry.analysis.bdsimData, 50
- pyg4ometry.analysis.Data, 49
- pyg4ometry.analysis.flukaData, 50
- pyg4ometry.analysis.Plot, 50
- pyg4ometry.bdsim, 52
- pyg4ometry.bdsim.beam, 52
- pyg4ometry.bdsim.beamline, 52
- pyg4ometry.bdsim.element, 53
- pyg4ometry.bdsim.gmad, 53
- pyg4ometry.bdsim.options, 53
- pyg4ometry.bdsim.sampler, 54
- pyg4ometry.bdsim.scoring, 54
- pyg4ometry.cli, 416
- pyg4ometry.compare, 56
- pyg4ometry.compare._Compare, 56
- pyg4ometry.config, 417
- pyg4ometry.convert, 67
- pyg4ometry.convert.fluka2g4materials, 71
- pyg4ometry.convert.fluka2Geant4, 67
- pyg4ometry.convert.freecad2Fluka, 73
- pyg4ometry.convert.gdml2stl, 73
- pyg4ometry.convert.geant42Fluka, 74
- pyg4ometry.convert.geant42FlukaBake, 75
- pyg4ometry.convert.geant42Geant4, 76
- pyg4ometry.convert.geant42Vtk, 76
- pyg4ometry.convert.oce2Geant4, 76
- pyg4ometry.convert.stl2gdml, 78
- pyg4ometry.convert.vis2oce, 78
- pyg4ometry.exceptions, 419
- pyg4ometry.features, 94
- pyg4ometry.features._accelerator, 94
- pyg4ometry.features.algos, 95
- pyg4ometry.fluka, 100
- pyg4ometry.fluka.body, 120
- pyg4ometry.fluka.boolean_algebra, 131
- pyg4ometry.fluka.card, 133
- pyg4ometry.fluka.directive, 134
- pyg4ometry.fluka.elcfield, 136
- pyg4ometry.fluka.extruder, 136
- pyg4ometry.fluka.flair, 137
- pyg4ometry.fluka.fluka_registry, 138
- pyg4ometry.fluka.lattice, 142
- pyg4ometry.fluka.material, 143
- pyg4ometry.fluka.mgnfield, 146
- pyg4ometry.fluka.preprocessor, 146
- pyg4ometry.fluka.reader, 148
- pyg4ometry.fluka.region, 150
- pyg4ometry.fluka.RegionExpression, 100
- pyg4ometry.fluka.RegionExpression.RegionLexer, 100
- pyg4ometry.fluka.RegionExpression.RegionParser, 102
- pyg4ometry.fluka.RegionExpression.RegionParserVisitor, 109
- pyg4ometry.fluka.vector, 155
- pyg4ometry.fluka.vis, 157
- pyg4ometry.fluka.Writer, 119
- pyg4ometry.freecad, 180
- pyg4ometry.freecad.Reader, 180
- pyg4ometry.gdml, 182
- pyg4ometry.gdml.Defines, 196
- pyg4ometry.gdml.GdmlExpression, 182
- pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval, 182
- pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer, 183
- pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser, 185
- pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor, 194
- pyg4ometry.gdml.Reader, 205
- pyg4ometry.gdml.Units, 207
- pyg4ometry.gdml.Writer, 207
- pyg4ometry.geant4, 225
- pyg4ometry.geant4._Material, 324
- pyg4ometry.geant4.AssemblyVolume, 309
- pyg4ometry.geant4.BorderSurface, 310
- pyg4ometry.geant4.DivisionVolume, 311
- pyg4ometry.geant4.LogicalVolume, 312
- pyg4ometry.geant4.Loop, 316
- pyg4ometry.geant4.ParameterisedVolume, 316

pyg4ometry.geant4.PhysicalVolume, 317
pyg4ometry.geant4.Registry, 318
pyg4ometry.geant4.ReplicaVolume, 322
pyg4ometry.geant4.SkinSurface, 323
pyg4ometry.geant4.solid, 225
pyg4ometry.geant4.solid.Box, 225
pyg4ometry.geant4.solid.Cons, 226
pyg4ometry.geant4.solid.CutTubs, 227
pyg4ometry.geant4.solid.Ellipsoid, 227
pyg4ometry.geant4.solid.EllipticalCone, 228
pyg4ometry.geant4.solid.EllipticalTube, 229
pyg4ometry.geant4.solid.ExtrudedSolid, 230
pyg4ometry.geant4.solid.GenericPolycone, 230
pyg4ometry.geant4.solid.GenericPolyhedra, 231
pyg4ometry.geant4.solid.GenericTrap, 232
pyg4ometry.geant4.solid.Hype, 233
pyg4ometry.geant4.solid.Intersection, 234
pyg4ometry.geant4.solid.Layer, 235
pyg4ometry.geant4.solid.MultiUnion, 235
pyg4ometry.geant4.solid.OpticalSurface, 236
pyg4ometry.geant4.solid.Orb, 237
pyg4ometry.geant4.solid.Para, 238
pyg4ometry.geant4.solid.Paraboloid, 238
pyg4ometry.geant4.solid.Plane, 239
pyg4ometry.geant4.solid.Polycone, 240
pyg4ometry.geant4.solid.Polyhedra, 241
pyg4ometry.geant4.solid.Scaled, 241
pyg4ometry.geant4.solid.SolidBase, 242
pyg4ometry.geant4.solid.Sphere, 243
pyg4ometry.geant4.solid.Subtraction, 244
pyg4ometry.geant4.solid.TessellatedSolid, 244
pyg4ometry.geant4.solid.Tet, 246
pyg4ometry.geant4.solid.Torus, 246
pyg4ometry.geant4.solid.Trap, 247
pyg4ometry.geant4.solid.Trd, 248
pyg4ometry.geant4.solid.Tubs, 249
pyg4ometry.geant4.solid.TwistedBox, 250
pyg4ometry.geant4.solid.TwistedSolid, 251
pyg4ometry.geant4.solid.TwistedTrap, 251
pyg4ometry.geant4.solid.TwistedTrd, 252
pyg4ometry.geant4.solid.TwistedTubs, 253
pyg4ometry.geant4.solid.TwoVector, 254
pyg4ometry.geant4.solid.Union, 255
pyg4ometry.geant4.solid.Wedge, 256
pyg4ometry.geant4.SurfaceBase, 323
pyg4ometry.gui, 352
pyg4ometry.gui.example1, 356
pyg4ometry.gui.GeometryModel, 352
pyg4ometry.gui.MainWindow, 352
pyg4ometry.gui.QVTKRenderWindowInteractor, 353
pyg4ometry.io, 365
pyg4ometry.io.R00TTGeo, 365
pyg4ometry.meshutils, 420
pyg4ometry.misc, 366
pyg4ometry.misc.NestedSolids, 366
pyg4ometry.misc.TestUtils, 366
pyg4ometry.montecarlo, 368
pyg4ometry.montecarlo.beam, 368
pyg4ometry.montecarlo.boundary, 369
pyg4ometry.montecarlo.config, 369
pyg4ometry.montecarlo.scoring, 369
pyg4ometry.pycgal, 370
pyg4ometry.pycgal.core, 371
pyg4ometry.pycgal.HalfPlane, 370
pyg4ometry.pycgal.pythonHelpers, 373
pyg4ometry.pycgal_old, 376
pyg4ometry.pycgal_old.cgal, 376
pyg4ometry.pycsg, 381
pyg4ometry.pyoce, 382
pyg4ometry.pyoce.pythonHelpers, 382
pyg4ometry.pyoce.Reader, 382
pyg4ometry.stl, 383
pyg4ometry.stl.Reader, 383
pyg4ometry.transformation, 420
pyg4ometry.utils, 424
pyg4ometry.visualisation, 387
pyg4ometry.visualisation.Convert, 387
pyg4ometry.visualisation.Mesh, 387
pyg4ometry.visualisation.Plot, 388
pyg4ometry.visualisation.RenderWriter, 389
pyg4ometry.visualisation.ViewerBase, 389
pyg4ometry.visualisation.VisualisationOptions, 392
pyg4ometry.visualisation.VtkExporter, 393
pyg4ometry.visualisation.VtkViewer, 395
pyg4ometry.visualisation.VtkViewerNew, 400
pyg4ometry.visualisation.Writer, 403

Symbols

- `_BODY_NAMES` (in module `pyg4ometry.fluka.reader`), 149
- `_Boolean` (class in `pyg4ometry.fluka.region`), 152
- `_CURSOR_MAP` (`pyg4ometry.gui.QVTKRenderWindowInteractor` attribute), 361, 363
- `_CURSOR_MAP` (`pyg4ometry.gui.QVTKRenderWindowInteractor` attribute), 355
- `_CURSOR_MAP` (`pyg4ometry.gui.example1.QVTKRenderWindowInteractor` attribute), 358
- `_Calc` (class in `pyg4ometry.fluka.preprocessor`), 147
- `_Element` (class in `pyg4ometry.compare`), 64
- `_Element` (class in `pyg4ometry.gdml`), 214
- `_FLUKA_ELEMENT_SYMBOLS` (in module `pyg4ometry.convert.fluka2g4materials`), 72
- `_FLUKA_PREDEFINED_FUNCTIONS` (in module `pyg4ometry.fluka.preprocessor`), 147
- `_FLUKA_PREDEFINED_IDS` (in module `pyg4ometry.fluka.preprocessor`), 147
- `_Facet` (class in `pyg4ometry.stl`), 385
- `_Facet` (class in `pyg4ometry.stl.Reader`), 384
- `_FlukaDataFile` (class in `pyg4ometry.analysis.flukaData`), 51
- `_FlukaToG4MaterialConverter` (class in `pyg4ometry.convert.fluka2g4materials`), 72
- `_GenericPolyhedra` (class in `pyg4ometry.geant4.solid`), 270, 271, 300
- `_GetCtrlShift()` (`pyg4ometry.gui.QVTKRenderWindowInteractor` method), 361, 364
- `_GetCtrlShift()` (`pyg4ometry.gui.QVTKRenderWindowInteractor` method), 355
- `_GetCtrlShift()` (`pyg4ometry.gui.example1.QVTKRenderWindowInteractor` method), 358
- `_IfInfo` (class in `pyg4ometry.fluka.preprocessor`), 147
- `_Isotope` (class in `pyg4ometry.gdml`), 215
- `_Layer` (class in `pyg4ometry.geant4.solid`), 294, 296, 298
- `_MatProp` (class in `pyg4ometry.fluka.material`), 144
- `_Material` (class in `pyg4ometry.compare`), 62
- `_Material` (class in `pyg4ometry.gdml`), 213
- `_MeshedZoneInfo` (in module `pyg4ometry.fluka.boolean_algebra`), 133
- `_OverlapType` (class in `pyg4ometry.geant4`), 332, 337
- `_PREDEFINED_COMPOUNDS` (in module `pyg4ometry.fluka.material`), 144
- `_PREDEFINED_ELEMENTS` (in module `pyg4ometry.fluka.material`), 143
- `PeriodicTable` (class in `pyg4ometry.convert.fluka2g4materials`), 72
- `PhysicalVolume` (class in `pyg4ometry.geant4`), 336, 339
- `PolygonProcessing` (class in `pyg4ometry.convert`), 81
- `_ROOTMatStateToGeant4MatState()` (`pyg4ometry.io.ROOTTGeo.Reader` method), 365
- `_ReplicaVolume` (class in `pyg4ometry.geant4`), 338
- `_ReplicaVolume.Axis` (class in `pyg4ometry.geant4`), 338
- `_SolidBase` (class in `pyg4ometry.geant4.solid`), 261–269, 272, 273, 277, 281, 285–293, 295, 297, 299–304, 308
- `_StripPointer()` (in module `pyg4ometry.gdml`), 213
- `_StripPointer()` (in module `pyg4ometry.gdml.Reader`), 207
- `_TwistedSolid` (class in `pyg4ometry.geant4.solid`), 293, 295, 298
- `_TwoVector` (class in `pyg4ometry.geant4.solid`), 294, 295, 297
- `_UpdateComplexity()` (in module `pyg4ometry.geant4`), 348
- `_UpdateComplexity()` (in module `pyg4ometry.geant4.Registry`), 321
- `VisOptions` (class in `pyg4ometry.visualisation`), 411
- `_WITH_PARAVIEW` (in module `pyg4ometry.visualisation`), 414
- `_WITH_PARAVIEW` (in module `pyg4ometry.visualisation.VtkExporter`), 393
- `__abs__()` (`pyg4ometry.gdml.Defines.ScalarBase` method), 199
- `__abs__()` (`pyg4ometry.gdml.ScalarBase` method), 219
- `__abs__()` (`pyg4ometry.geant4.solid.TwoVector` method), 261
- `__abs__()` (`pyg4ometry.geant4.solid.TwoVector.TwoVector` method), 255
- `__abs__()` (`pyg4ometry.geant4.solid._TwoVector` method), 294, 296, 297

```

__add__() (pyg4ometry.compare.ComparisonResult
method), 65
__add__() (pyg4ometry.compare._Compare.ComparisonResult
method), 59
__add__() (pyg4ometry.fluka.Three method), 173
__add__() (pyg4ometry.fluka.vector.Three method), 156
__add__() (pyg4ometry.gdml.Defines.ScalarBase
method), 199
__add__() (pyg4ometry.gdml.Defines.VectorBase
method), 202
__add__() (pyg4ometry.gdml.ScalarBase method), 219
__add__() (pyg4ometry.gdml.VectorBase method), 222
__add__() (pyg4ometry.geant4.solid.TwoVector
method), 261
__add__() (pyg4ometry.geant4.solid.TwoVector.TwoVector
method), 254
__add__() (pyg4ometry.geant4.solid._TwoVector
method), 294, 295, 297
__contains__() (pyg4ometry.fluka.FlukaBodyStoreExact
method), 172
__contains__() (pyg4ometry.fluka.fluka_registry.FlukaBodyStore
method), 59
__contains__() (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact
method), 172
__contains__() (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact
method), 142
__contains__() (pyg4ometry.utils.Samples method),
425
__delitem__() (pyg4ometry.fluka.FlukaBodyStoreExact
method), 172
__delitem__() (pyg4ometry.fluka.RecursiveRotoTranslation
method), 178
__delitem__() (pyg4ometry.fluka.directive.RecursiveRotoTranslation
method), 135
__delitem__() (pyg4ometry.fluka.directive.RecursiveRotoTranslation
method), 142
__delitem__() (pyg4ometry.fluka.fluka_registry.FlukaBodyStore
method), 140
__delitem__() (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact
method), 140
__delitem__() (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact
method), 204
__delitem__() (pyg4ometry.fluka.fluka_registry.RotoTranslationStore
method), 203
__delitem__() (pyg4ometry.fluka.fluka_registry.RotoTranslationStore
method), 140
__doc__ (pyg4ometry.convert.RAW attribute), 87
__doc__ (pyg4ometry.fluka.RAW attribute), 163
__doc__ (pyg4ometry.fluka.body.RAW attribute), 125
__eq__() (pyg4ometry.fluka.AABB method), 173
__eq__() (pyg4ometry.fluka.Three method), 173
__eq__() (pyg4ometry.fluka.vector.AABB method), 156
__eq__() (pyg4ometry.fluka.vector.Three method), 156
__eq__() (pyg4ometry.gdml.Constant method), 221
__eq__() (pyg4ometry.gdml.Defines.Constant method),
201
__float__() (pyg4ometry.gdml.BasicExpression
method), 218
__float__() (pyg4ometry.gdml.Defines.BasicExpression
method), 198
__float__() (pyg4ometry.gdml.Defines.ScalarBase
method), 199
__float__() (pyg4ometry.gdml.ScalarBase method),
219
__float__() (pyg4ometry.gdml.Constant method), 221
__ge__() (pyg4ometry.gdml.Defines.Constant method),
201
__getattr__() (pyg4ometry.gui.QVTKRenderWindowInteractor
method), 361, 363
__getattr__() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRe
method), 355
__getattr__() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor
method), 358
__getitem__() (pyg4ometry.analysis.Data.TH1
method), 49
__getitem__() (pyg4ometry.analysis.Data.TH2
method), 49
__getitem__() (pyg4ometry.analysis.Data.TH3
method), 50
__getitem__() (pyg4ometry.compare.ComparisonResult
method), 65
__getitem__() (pyg4ometry.compare._Compare.ComparisonResult
method), 59
__getitem__() (pyg4ometry.fluka.FlukaBodyStoreExact
method), 172
__getitem__() (pyg4ometry.fluka.RecursiveRotoTranslation
method), 177
__getitem__() (pyg4ometry.fluka.directive.RecursiveRotoTranslation
method), 135
__getitem__() (pyg4ometry.fluka.fluka_registry.FlukaBodyStore
method), 141
__getitem__() (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact
method), 142
__getitem__() (pyg4ometry.fluka.fluka_registry.RotoTranslationStore
method), 203
__getitem__() (pyg4ometry.gdml.Defines.Matrix
method), 224
__getitem__() (pyg4ometry.gdml.Defines.VectorBase
method), 223
__getitem__() (pyg4ometry.gdml.Matrix method), 224
__getitem__() (pyg4ometry.gdml.VectorBase method),
223
__getitem__() (pyg4ometry.geant4.solid.Layer.Layer
method), 235
__getitem__() (pyg4ometry.geant4.solid.TwoVector
method), 261
__getitem__() (pyg4ometry.geant4.solid.TwoVector.TwoVector
method), 254
__getitem__() (pyg4ometry.geant4.solid._Layer
method), 294, 296, 298
__getitem__() (pyg4ometry.geant4.solid._TwoVector
method), 294, 295, 297
__getitem__() (pyg4ometry.utils.Samples method), 425
__gt__() (pyg4ometry.gdml.Constant method), 221
__gt__() (pyg4ometry.gdml.Defines.Constant method),
201

```


<code>__iadd__()</code> (pyg4ometry.compare.ComparisonResult method), 65	<code>__mul__()</code> (pyg4ometry.gdml.Defines.ScalarBase method), 199
<code>__iadd__()</code> (pyg4ometry.compare._Compare.ComparisonResult method), 59	<code>__mul__()</code> (pyg4ometry.gdml.Defines.VectorBase method), 202
<code>__iadd__()</code> (pyg4ometry.fluka.Three method), 173	<code>__mul__()</code> (pyg4ometry.gdml.ScalarBase method), 219
<code>__iadd__()</code> (pyg4ometry.fluka.vector.Three method), 156	<code>__mul__()</code> (pyg4ometry.gdml.VectorBase method), 222
<code>__int__()</code> (pyg4ometry.gdml.Defines.Expression method), 202	<code>__mul__()</code> (pyg4ometry.geant4.solid.TwoVector method), 261
<code>__int__()</code> (pyg4ometry.gdml.Expression method), 222	<code>__mul__()</code> (pyg4ometry.geant4.solid.TwoVector.TwoVector method), 254
<code>__ior__()</code> (pyg4ometry.compare.TestResult method), 65	<code>__mul__()</code> (pyg4ometry.geant4.solid._TwoVector method), 294, 296, 297
<code>__ior__()</code> (pyg4ometry.compare._Compare.TestResult method), 58	<code>__ne__()</code> (pyg4ometry.fluka.Three method), 173
<code>__isub__()</code> (pyg4ometry.fluka.Three method), 173	<code>__ne__()</code> (pyg4ometry.fluka.vector.Three method), 156
<code>__isub__()</code> (pyg4ometry.fluka.vector.Three method), 156	<code>__ne__()</code> (pyg4ometry.gdml.Constant method), 221
<code>__iter__()</code> (pyg4ometry.features.Plane method), 99	<code>__ne__()</code> (pyg4ometry.gdml.Defines.Constant method), 201
<code>__iter__()</code> (pyg4ometry.features.algos.Line method), 96	<code>__neg__()</code> (pyg4ometry.gdml.Defines.ScalarBase method), 199
<code>__iter__()</code> (pyg4ometry.features.algos.Plane method), 96	<code>__neg__()</code> (pyg4ometry.gdml.ScalarBase method), 219
<code>__iter__()</code> (pyg4ometry.fluka.FlukaBodyStoreExact method), 172	<code>__next__()</code> (pyg4ometry.features.Plane method), 99
<code>__iter__()</code> (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method), 141	<code>__next__()</code> (pyg4ometry.features.algos.Line method), 96
<code>__iter__()</code> (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method), 142	<code>__next__()</code> (pyg4ometry.features.algos.Plane method), 96
<code>__iter__()</code> (pyg4ometry.fluka.fluka_registry.RotoTranslationStoreExact method), 140	<code>__or__()</code> (pyg4ometry.compare.TestResult method), 65
<code>__le__()</code> (pyg4ometry.gdml.Constant method), 221	<code>__or__()</code> (pyg4ometry.compare._Compare.TestResult method), 58
<code>__le__()</code> (pyg4ometry.gdml.Defines.Constant method), 201	<code>__pow__()</code> (pyg4ometry.gdml.Defines.ScalarBase method), 199
<code>__len__()</code> (pyg4ometry.compare.ComparisonResult method), 65	<code>__pow__()</code> (pyg4ometry.gdml.ScalarBase method), 219
<code>__len__()</code> (pyg4ometry.compare.Tests method), 64	<code>__radd__()</code> (pyg4ometry.gdml.Defines.ScalarBase attribute), 199
<code>__len__()</code> (pyg4ometry.compare._Compare.ComparisonResult method), 59	<code>__radd__()</code> (pyg4ometry.gdml.ScalarBase attribute), 219
<code>__len__()</code> (pyg4ometry.compare._Compare.Tests method), 58	<code>__radd__()</code> (pyg4ometry.fluka.Three method), 173
<code>__len__()</code> (pyg4ometry.fluka.FlukaBodyStoreExact method), 172	<code>__radd__()</code> (pyg4ometry.fluka.vector.Three method), 156
<code>__len__()</code> (pyg4ometry.fluka.RecursiveRotoTranslationStoreExact method), 178	<code>__repr__()</code> (pyg4ometry.compare.Tests method), 64
<code>__len__()</code> (pyg4ometry.fluka.directive.RecursiveRotoTranslationStoreExact method), 136	<code>__repr__()</code> (pyg4ometry.compare._Compare.Tests method), 58
<code>__len__()</code> (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method), 141	<code>__repr__()</code> (pyg4ometry.convert.ARB method), 87
<code>__len__()</code> (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method), 142	<code>__repr__()</code> (pyg4ometry.convert.BOX method), 84
<code>__len__()</code> (pyg4ometry.fluka.fluka_registry.RotoTranslationStoreExact method), 140	<code>__repr__()</code> (pyg4ometry.convert.ELL method), 86
<code>__lt__()</code> (pyg4ometry.gdml.Constant method), 221	<code>__repr__()</code> (pyg4ometry.convert.ELA method), 89
<code>__lt__()</code> (pyg4ometry.gdml.Defines.Constant method), 201	<code>__repr__()</code> (pyg4ometry.convert.QUA method), 92
	<code>__repr__()</code> (pyg4ometry.convert.RCC method), 85
	<code>__repr__()</code> (pyg4ometry.convert.REC method), 85
	<code>__repr__()</code> (pyg4ometry.convert.RPP method), 83
	<code>__repr__()</code> (pyg4ometry.convert.SPH method), 84
	<code>__repr__()</code> (pyg4ometry.convert.TRC method), 86
	<code>__repr__()</code> (pyg4ometry.convert.XCC method), 89
	<code>__repr__()</code> (pyg4ometry.convert.XEC method), 91
	<code>__repr__()</code> (pyg4ometry.convert.XYP method), 88
	<code>__repr__()</code> (pyg4ometry.convert.XZP method), 88

```

__repr__ () (pyg4ometry.convert.YCC method), 90
__repr__ () (pyg4ometry.convert.YEC method), 91
__repr__ () (pyg4ometry.convert.YZP method), 88
__repr__ () (pyg4ometry.convert.ZCC method), 90
__repr__ () (pyg4ometry.convert.ZEC method), 92
__repr__ () (pyg4ometry.features.CoordinateSystem
method), 99
__repr__ () (pyg4ometry.features.Plane method), 99
__repr__ () (pyg4ometry.features.algos.CoordinateSystem
method), 97
__repr__ () (pyg4ometry.features.algos.Line method),
96
__repr__ () (pyg4ometry.features.algos.Plane method),
96
__repr__ () (pyg4ometry.fluka.AABB method), 173
__repr__ () (pyg4ometry.fluka.ARB method), 164
__repr__ () (pyg4ometry.fluka.BOX method), 160
__repr__ () (pyg4ometry.fluka.BuiltIn method), 178
__repr__ () (pyg4ometry.fluka.Card method), 179
__repr__ () (pyg4ometry.fluka.Compound method), 179
__repr__ () (pyg4ometry.fluka.ELL method), 162
__repr__ () (pyg4ometry.fluka.FlukaBodyStoreExact
method), 172
__repr__ () (pyg4ometry.fluka.Material method), 179
__repr__ () (pyg4ometry.fluka.PLA method), 165
__repr__ () (pyg4ometry.fluka.QUA method), 169
__repr__ () (pyg4ometry.fluka.RCC method), 161
__repr__ () (pyg4ometry.fluka.REC method), 161
__repr__ () (pyg4ometry.fluka.RPP method), 160
__repr__ () (pyg4ometry.fluka.RecursiveRotoTranslation
method), 177
__repr__ () (pyg4ometry.fluka.Region method), 176
__repr__ () (pyg4ometry.fluka.RotoTranslation method),
177
__repr__ () (pyg4ometry.fluka.SPH method), 160
__repr__ () (pyg4ometry.fluka.TRC method), 162
__repr__ () (pyg4ometry.fluka.XCC method), 165
__repr__ () (pyg4ometry.fluka.XEC method), 167
__repr__ () (pyg4ometry.fluka.XYP method), 164
__repr__ () (pyg4ometry.fluka.XZP method), 164
__repr__ () (pyg4ometry.fluka.YCC method), 166
__repr__ () (pyg4ometry.fluka.YEC method), 167
__repr__ () (pyg4ometry.fluka.YZP method), 165
__repr__ () (pyg4ometry.fluka.ZCC method), 166
__repr__ () (pyg4ometry.fluka.ZEC method), 168
__repr__ () (pyg4ometry.fluka.body.ARB method), 126
__repr__ () (pyg4ometry.fluka.body.BOX method), 122
__repr__ () (pyg4ometry.fluka.body.ELL method), 124
__repr__ () (pyg4ometry.fluka.body.PLA method), 127
__repr__ () (pyg4ometry.fluka.body.QUA method), 131
__repr__ () (pyg4ometry.fluka.body.RCC method), 123
__repr__ () (pyg4ometry.fluka.body.REC method), 123
__repr__ () (pyg4ometry.fluka.body.RPP method), 122
__repr__ () (pyg4ometry.fluka.body.SPH method), 122
__repr__ () (pyg4ometry.fluka.body.TRC method), 124
__repr__ () (pyg4ometry.fluka.body.XCC method), 128
__repr__ () (pyg4ometry.fluka.body.XEC method), 129
__repr__ () (pyg4ometry.fluka.body.XYP method), 126
__repr__ () (pyg4ometry.fluka.body.XZP method), 126
__repr__ () (pyg4ometry.fluka.body.YCC method), 128
__repr__ () (pyg4ometry.fluka.body.YEC method), 130
__repr__ () (pyg4ometry.fluka.body.YZP method), 127
__repr__ () (pyg4ometry.fluka.body.ZCC method), 129
__repr__ () (pyg4ometry.fluka.body.ZEC method), 130
__repr__ () (pyg4ometry.fluka.card.Card method), 133
__repr__ () (pyg4ometry.fluka.directive.RecursiveRotoTranslation
method), 135
__repr__ () (pyg4ometry.fluka.directive.RotoTranslation
method), 135
__repr__ () (pyg4ometry.fluka.fluka_registry.BaseCacher
method), 141
__repr__ () (pyg4ometry.fluka.fluka_registry.FlukaBodyStore
method), 141
__repr__ () (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact
method), 142
__repr__ () (pyg4ometry.fluka.material.BuiltIn
method), 144
__repr__ () (pyg4ometry.fluka.material.Compound
method), 145
__repr__ () (pyg4ometry.fluka.material.Material
method), 144
__repr__ () (pyg4ometry.fluka.preprocessor.IfInfo
method), 147
__repr__ () (pyg4ometry.fluka.region.Region method),
155
__repr__ () (pyg4ometry.fluka.vector.AABB method),
156
__repr__ () (pyg4ometry.gdml.BasicExpression
method), 217
__repr__ () (pyg4ometry.gdml.Defines.BasicExpression
method), 197
__repr__ () (pyg4ometry.gdml.Defines.Matrix method),
204
__repr__ () (pyg4ometry.gdml.Defines.Quantity
method), 202
__repr__ () (pyg4ometry.gdml.Defines.ScalarBase
method), 199
__repr__ () (pyg4ometry.gdml.Defines.VectorBase
method), 202
__repr__ () (pyg4ometry.gdml.Matrix method), 224
__repr__ () (pyg4ometry.gdml.Quantity method), 222
__repr__ () (pyg4ometry.gdml.ScalarBase method), 219
__repr__ () (pyg4ometry.gdml.VectorBase method), 222
__repr__ () (pyg4ometry.geant4.AssemblyVolume
method), 336
__repr__ () (pyg4ometry.geant4.AssemblyVolume.AssemblyVolume
method), 310
__repr__ () (pyg4ometry.geant4.BorderSurface

```

```

        method), 345
__repr__ () (pyg4ometry.geant4.BorderSurface.BorderSurface
        method), 311
__repr__ () (pyg4ometry.geant4.DivisionVolume
        method), 341
__repr__ () (pyg4ometry.geant4.DivisionVolume.DivisionVolume
        method), 312
__repr__ () (pyg4ometry.geant4.LogicalVolume
        method), 332, 341
__repr__ () (pyg4ometry.geant4.LogicalVolume.LogicalVolume
        method), 312
__repr__ () (pyg4ometry.geant4.MaterialBase method),
        350
__repr__ () (pyg4ometry.geant4.ParameterisedVolume
        method), 339
__repr__ () (pyg4ometry.geant4.ParameterisedVolume.ParameterisedVolume
        method), 317
__repr__ () (pyg4ometry.geant4.PhysicalVolume
        method), 335, 344
__repr__ () (pyg4ometry.geant4.PhysicalVolume.PhysicalVolume
        method), 317
__repr__ () (pyg4ometry.geant4.ReplicaVolume
        method), 338
__repr__ () (pyg4ometry.geant4.ReplicaVolume.ReplicaVolume
        method), 323
__repr__ () (pyg4ometry.geant4.SkinSurface method),
        344
__repr__ () (pyg4ometry.geant4.SkinSurface.SkinSurface
        method), 323
__repr__ () (pyg4ometry.geant4._Material.MaterialBase
        method), 327
__repr__ () (pyg4ometry.geant4._PhysicalVolume
        method), 337, 340
__repr__ () (pyg4ometry.geant4._ReplicaVolume
        method), 339
__repr__ () (pyg4ometry.geant4.solid.Box method), 263
__repr__ () (pyg4ometry.geant4.solid.Box.Box method),
        225
__repr__ () (pyg4ometry.geant4.solid.Cons method),
        267
__repr__ () (pyg4ometry.geant4.solid.Cons.Cons
        method), 226
__repr__ () (pyg4ometry.geant4.solid.CutTubs method),
        265
__repr__ () (pyg4ometry.geant4.solid.CutTubs.CutTubs
        method), 227
__repr__ () (pyg4ometry.geant4.solid.Ellipsoid
        method), 268
__repr__ () (pyg4ometry.geant4.solid.Ellipsoid.Ellipsoid
        method), 228
__repr__ () (pyg4ometry.geant4.solid.EllipticalCone
        method), 290
__repr__ () (pyg4ometry.geant4.solid.EllipticalCone.EllipticalCone
        method), 229
__repr__ () (pyg4ometry.geant4.solid.EllipticalTube
        method), 289
__repr__ () (pyg4ometry.geant4.solid.EllipticalTube.EllipticalTube
        method), 229
__repr__ () (pyg4ometry.geant4.solid.ExtrudedSolid
        method), 273
__repr__ () (pyg4ometry.geant4.solid.ExtrudedSolid.ExtrudedSolid
        method), 230
__repr__ () (pyg4ometry.geant4.solid.GenericPolycone
        method), 301
__repr__ () (pyg4ometry.geant4.solid.GenericPolycone.GenericPolycone
        method), 231
__repr__ () (pyg4ometry.geant4.solid.GenericPolyhedra
        method), 302
__repr__ () (pyg4ometry.geant4.solid.GenericPolyhedra.GenericPolyhedra
        method), 232
__repr__ () (pyg4ometry.geant4.solid.GenericTrap
        method), 303
__repr__ () (pyg4ometry.geant4.solid.GenericTrap.GenericTrap
        method), 233
__repr__ () (pyg4ometry.geant4.solid.Hype method),
        292
__repr__ () (pyg4ometry.geant4.solid.Hype.Hype
        method), 234
__repr__ () (pyg4ometry.geant4.solid.Intersection
        method), 278
__repr__ () (pyg4ometry.geant4.solid.Intersection.Intersection
        method), 234
__repr__ () (pyg4ometry.geant4.solid.Layer.Layer
        method), 235
__repr__ () (pyg4ometry.geant4.solid.MultiUnion
        method), 305
__repr__ () (pyg4ometry.geant4.solid.MultiUnion.MultiUnion
        method), 236
__repr__ () (pyg4ometry.geant4.solid.OpticalSurface
        method), 285
__repr__ () (pyg4ometry.geant4.solid.OpticalSurface.OpticalSurface
        method), 236
__repr__ () (pyg4ometry.geant4.solid.Orb method), 289
__repr__ () (pyg4ometry.geant4.solid.Orb.Orb
        method), 237
__repr__ () (pyg4ometry.geant4.solid.Para method),
        287
__repr__ () (pyg4ometry.geant4.solid.Para.Para
        method), 238
__repr__ () (pyg4ometry.geant4.solid.Paraboloid
        method), 291
__repr__ () (pyg4ometry.geant4.solid.Paraboloid.Paraboloid
        method), 239
__repr__ () (pyg4ometry.geant4.solid.Plane method),
        262
__repr__ () (pyg4ometry.geant4.solid.Plane.Plane
        method), 239
__repr__ () (pyg4ometry.geant4.solid.Polycone

```

`method`), 271
`__repr__()` (`pyg4ometry.geant4.solid.Polycone.Polycone`
`method`), 240
`__repr__()` (`pyg4ometry.geant4.solid.Polyhedra`
`method`), 272
`__repr__()` (`pyg4ometry.geant4.solid.Polyhedra.Polyhedra`
`method`), 241
`__repr__()` (`pyg4ometry.geant4.solid.Scaled` `method`),
239
`__repr__()` (`pyg4ometry.geant4.solid.Scaled.Scaled`
`method`), 242
`__repr__()` (`pyg4ometry.geant4.solid.Sphere` `method`),
266
`__repr__()` (`pyg4ometry.geant4.solid.Sphere.Sphere`
`method`), 243
`__repr__()` (`pyg4ometry.geant4.solid.Subtraction`
`method`), 282
`__repr__()` (`pyg4ometry.geant4.solid.Subtraction.Subtraction`
`method`), 244
`__repr__()` (`pyg4ometry.geant4.solid.TessellatedSolid`
`method`), 304
`__repr__()` (`pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid`
`method`), 245
`__repr__()` (`pyg4ometry.geant4.solid.Tet` `method`), 293
`__repr__()` (`pyg4ometry.geant4.solid.Tet.Tet` `method`),
246
`__repr__()` (`pyg4ometry.geant4.solid.Torus` `method`),
269
`__repr__()` (`pyg4ometry.geant4.solid.Torus.Torus`
`method`), 247
`__repr__()` (`pyg4ometry.geant4.solid.Trap` `method`),
288
`__repr__()` (`pyg4ometry.geant4.solid.Trap.Trap`
`method`), 248
`__repr__()` (`pyg4ometry.geant4.solid.Tubs` `method`),
264
`__repr__()` (`pyg4ometry.geant4.solid.Tubs.Tubs`
`method`), 249
`__repr__()` (`pyg4ometry.geant4.solid.TwistedBox`
`method`), 295
`__repr__()` (`pyg4ometry.geant4.solid.TwistedBox.TwistedBox`
`method`), 250
`__repr__()` (`pyg4ometry.geant4.solid.TwistedTrap`
`method`), 297
`__repr__()` (`pyg4ometry.geant4.solid.TwistedTrap.TwistedTrap`
`method`), 252
`__repr__()` (`pyg4ometry.geant4.solid.TwistedTrd`
`method`), 298
`__repr__()` (`pyg4ometry.geant4.solid.TwistedTrd.TwistedTrd`
`method`), 253
`__repr__()` (`pyg4ometry.geant4.solid.TwistedTubs`
`method`), 300
`__repr__()` (`pyg4ometry.geant4.solid.TwistedTubs.TwistedTubs`
`method`), 254
`__repr__()` (`pyg4ometry.geant4.solid.TwoVector`
`method`), 261
`__repr__()` (`pyg4ometry.geant4.solid.TwoVector.TwoVector`
`method`), 254
`__repr__()` (`pyg4ometry.geant4.solid.Union` `method`),
274
`__repr__()` (`pyg4ometry.geant4.solid.Union.Union`
`method`), 255
`__repr__()` (`pyg4ometry.geant4.solid.Wedge` `method`),
262
`__repr__()` (`pyg4ometry.geant4.solid.Wedge.Wedge`
`method`), 256
`__repr__()` (`pyg4ometry.geant4.solid._GenericPolyhedra`
`method`), 270, 271, 300
`__repr__()` (`pyg4ometry.geant4.solid._Layer` `method`),
294, 296, 298
`__repr__()` (`pyg4ometry.geant4.solid._TwoVector`
`method`), 294, 295, 297
`__repr__()` (`pyg4ometry.visualisation.ViewerBase`
`method`), 407
`__repr__()` (`pyg4ometry.visualisation.ViewerBase.ViewerBase`
`method`), 391
`__repr__()` (`pyg4ometry.visualisation.VisualisationOptions`
`method`), 407
`__repr__()` (`pyg4ometry.visualisation.VisualisationOptions.VisualisationOptions`
`method`), 392
`__repr__()` (`pyg4ometry.visualisation.VtkViewerNew`
`method`), 412
`__repr__()` (`pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew`
`method`), 402
`__repr__()` (`pyg4ometry.visualisation._VisOptions`
`method`), 411
`__rmul__` (`pyg4ometry.gdml.Defines.ScalarBase` `at-`
`tribute`), 199
`__rmul__` (`pyg4ometry.gdml.Defines.VectorBase` `at-`
`tribute`), 202
`__rmul__` (`pyg4ometry.gdml.ScalarBase` `attribute`), 219
`__rmul__` (`pyg4ometry.gdml.VectorBase` `attribute`), 222
`__rmul__()` (`pyg4ometry.geant4.solid.TwoVector`
`method`), 261
`__rmul__()` (`pyg4ometry.geant4.solid.TwoVector.TwoVector`
`method`), 254
`__rmul__()` (`pyg4ometry.geant4.solid._TwoVector`
`method`), 294, 296, 297
`__rsub__()` (`pyg4ometry.fluka.Three` `method`), 173
`__rsub__()` (`pyg4ometry.fluka.vector.Three` `method`),
156
`__rsub__()` (`pyg4ometry.gdml.Defines.ScalarBase`
`method`), 199
`__rsub__()` (`pyg4ometry.gdml.ScalarBase` `method`), 219
`__rtruediv__()` (`pyg4ometry.gdml.Defines.ScalarBase`
`method`), 199
`__rtruediv__()` (`pyg4ometry.gdml.ScalarBase`
`method`), 219

`__setitem__()` (`pyg4ometry.compare.ComparisonResult` attribute), 190
`method`), 65
`__setitem__()` (`pyg4ometry.compare._Compare.ComparisonResult` attribute), 191
`method`), 59
`__setitem__()` (`pyg4ometry.fluka.FlukaBodyStoreExact` attribute), 189
`method`), 172
`__setitem__()` (`pyg4ometry.fluka.RecursiveRotoTranslation` attribute), 186
`method`), 178
`__setitem__()` (`pyg4ometry.fluka.directive.RecursiveRotoTranslation` attribute), 187
`method`), 135
`__setitem__()` (`pyg4ometry.fluka.fluka_registry.FlukaBodyStore` attribute), 187
`method`), 141
`__setitem__()` (`pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact` attribute), 187
`method`), 142
`__setitem__()` (`pyg4ometry.fluka.fluka_registry.RotoTranslationStore` attribute), 191
`method`), 140
`__setitem__()` (`pyg4ometry.utils.Samples` method), 425
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 115
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 112
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 105
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 102
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 102
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 107
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 107
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 104
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 103
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 112
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 116
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 117
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 114
`__slots__` (`pyg4ometry.fluka.RegionExpression.RegionParser.RegionContext` attribute), 113
`__slots__` (`pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.AtomContext` attribute), 188
`__slots__` (`pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.ConstantContext` attribute), 189
`__slots__` (`pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.EquationContext` attribute), 185
`__slots__` (`pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.ExpressionContext` attribute), 186
`__slots__` (`pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.FuncContext` attribute), 187

`__str__()` (`pyg4ometry.geant4.solid.ExtrudedSolid.ExtrudedSolid` method), 243
`method`), 230
`__str__()` (`pyg4ometry.geant4.solid.GenericPolycone` method), 301
`method`), 231
`__str__()` (`pyg4ometry.geant4.solid.GenericPolyhedra` method), 302
`method`), 232
`__str__()` (`pyg4ometry.geant4.solid.GenericTrap` method), 303
`method`), 233
`__str__()` (`pyg4ometry.geant4.solid.Hype` method), 292
`method`), 234
`__str__()` (`pyg4ometry.geant4.solid.Intersection` method), 278
`method`), 234
`__str__()` (`pyg4ometry.geant4.solid.MultiUnion` method), 305
`method`), 236
`__str__()` (`pyg4ometry.geant4.solid.OpticalSurface` method), 285
`method`), 236
`__str__()` (`pyg4ometry.geant4.solid.Orb` method), 289
`method`), 237
`__str__()` (`pyg4ometry.geant4.solid.Para` method), 287
`method`), 238
`__str__()` (`pyg4ometry.geant4.solid.Paraboloid` method), 291
`method`), 239
`__str__()` (`pyg4ometry.geant4.solid.Polycone` method), 271
`method`), 240
`__str__()` (`pyg4ometry.geant4.solid.Polyhedra` method), 272
`method`), 241
`__str__()` (`pyg4ometry.geant4.solid.Scaled` method), 309
`method`), 242
`__str__()` (`pyg4ometry.geant4.solid.Sphere` method), 266
`method`), 243
`__str__()` (`pyg4ometry.geant4.solid.Subtraction` method), 282
`method`), 244
`__str__()` (`pyg4ometry.geant4.solid.TessellatedSolid` method), 304
`method`), 245
`__str__()` (`pyg4ometry.geant4.solid.Tet` method), 293
`method`), 246
`__str__()` (`pyg4ometry.geant4.solid.Torus` method), 269
`method`), 247
`__str__()` (`pyg4ometry.geant4.solid.Trap` method), 288
`method`), 248
`__str__()` (`pyg4ometry.geant4.solid.Tubs` method), 264
`method`), 249
`__str__()` (`pyg4ometry.geant4.solid.TwistedBox` method), 295
`method`), 250
`__str__()` (`pyg4ometry.geant4.solid.TwistedTrap` method), 297
`method`), 252
`__str__()` (`pyg4ometry.geant4.solid.TwistedTrd` method), 299
`method`), 253
`__str__()` (`pyg4ometry.geant4.solid.TwistedTubs` method), 300
`method`), 254
`__str__()` (`pyg4ometry.geant4.solid.Union` method), 274
`method`), 255
`__str__()` (`pyg4ometry.geant4.solid._GenericPolyhedra` method), 270, 271, 300
`__str__()` (`pyg4ometry.utils.Samples` method), 425
`__str__()` (`pyg4ometry.utils.Timer` method), 425
`__sub__()` (`pyg4ometry.fluka.Three` method), 173
`__sub__()` (`pyg4ometry.fluka.vector.Three` method), 156
`__sub__()` (`pyg4ometry.gdml.Defines.ScalarBase` method), 199
`__sub__()` (`pyg4ometry.gdml.Defines.VectorBase` method), 202
`__sub__()` (`pyg4ometry.gdml.ScalarBase` method), 219
`__sub__()` (`pyg4ometry.gdml.VectorBase` method), 222
`__sub__()` (`pyg4ometry.geant4.solid.TwoVector`

- method*), 261
- `__sub__()` (*pyg4ometry.geant4.solid.TwoVector.TwoVector method*), 254
- `__sub__()` (*pyg4ometry.geant4.solid._TwoVector method*), 294, 296, 297
- `__truediv__()` (*pyg4ometry.gdml.Defines.ScalarBase method*), 199
- `__truediv__()` (*pyg4ometry.gdml.Defines.VectorBase method*), 202
- `__truediv__()` (*pyg4ometry.gdml.ScalarBase method*), 219
- `__truediv__()` (*pyg4ometry.gdml.VectorBase method*), 222
- `__truediv__()` (*pyg4ometry.geant4.solid.TwoVector method*), 261
- `__truediv__()` (*pyg4ometry.geant4.solid.TwoVector.TwoVector method*), 254
- `__truediv__()` (*pyg4ometry.geant4.solid._TwoVector method*), 294, 296, 297
- `__version__` (in module *pyg4ometry*), 425
- `__version__` (in module *pyg4ometry.pycsg*), 381
- `_addProperty()` (*pyg4ometry.geant4.solid.SolidBase method*), 309
- `_addProperty()` (*pyg4ometry.geant4.solid.SolidBase.SolidBase method*), 242
- `_addProperty()` (*pyg4ometry.geant4.solid._SolidBase method*), 261–270, 272, 273, 277, 281, 285–293, 295, 297, 299–302, 304, 305, 308
- `_addToRegistry()` (*pyg4ometry.geant4.MaterialBase method*), 350
- `_addToRegistry()` (*pyg4ometry.geant4._Material.MaterialBase method*), 327
- `_appendFractionPairs()` (in module *pyg4ometry.fluka.material*), 145
- `_attempt_float_coercion()` (in module *pyg4ometry.fluka.card*), 134
- `_bodies()` (*pyg4ometry.fluka.FlukaBodyStoreExact method*), 172
- `_bodies()` (*pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method*), 140
- `_bodies()` (*pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method*), 142
- `_bodyNames()` (*pyg4ometry.fluka.FlukaBodyStoreExact method*), 172
- `_bodyNames()` (*pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method*), 140
- `_bodyNames()` (*pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method*), 142
- `_checkInternalOverlaps()` (*pyg4ometry.geant4.ReplicaVolume method*), 337
- `_checkInternalOverlaps()` (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume method*), 322
- `_checkInternalOverlaps()` (*pyg4ometry.geant4._ReplicaVolume method*), 338
- `_checkPVLikeDaughters()` (in module *pyg4ometry.compare*), 66
- `_checkPVLikeDaughters()` (in module *pyg4ometry.compare._Compare*), 60
- `_checkQuadricRegionAABBs()` (in module *pyg4ometry.convert.fluka2Geant4*), 70
- `_chkType()` (*pyg4ometry.geant4.SurfaceBase method*), 344, 345
- `_chkType()` (*pyg4ometry.geant4.SurfaceBase.SurfaceBase method*), 323
- `_combine_booleans()` (*pyg4ometry.fluka.Zone method*), 174
- `_combine_booleans()` (*pyg4ometry.fluka.region.Zone method*), 153
- `_commonInit()` (*pyg4ometry.features.CoordinateSystem method*), 99
- `_commonInit()` (*pyg4ometry.features.algos.CoordinateSystem method*), 96
- `_convertLatticeCells()` (in module *pyg4ometry.convert.fluka2Geant4*), 71
- `_copyNumber()` (in module *pyg4ometry.compare*), 67
- `_copyNumber()` (in module *pyg4ometry.compare._Compare*), 60
- `_copyStructureToNewFlukaRegistry()` (in module *pyg4ometry.convert.fluka2Geant4*), 71
- `_cubeNet()` (in module *pyg4ometry.geant4.solid*), 286, 287
- `_cylinderPoint()` (*pyg4ometry.fluka.fluka_registry.InfiniteCylinderCache static method*), 142
- `_daughterSubtractedMesh()` (in module *pyg4ometry.visualisation.ViewerBase*), 390
- `_degree_input()` (in module *pyg4ometry.fluka.preprocessor*), 147
- `_degree_output()` (in module *pyg4ometry.fluka.preprocessor*), 147
- `_distanceFromPointToPlane()` (in module *pyg4ometry.convert.fluka2Geant4*), 71
- `_draw_polygon_2()` (in module *pyg4ometry.pycgal.pythonHelpers*), 374
- `_elements()` (in module *pyg4ometry.compare*), 67
- `_elements()` (in module *pyg4ometry.compare._Compare*), 60
- `_evaluateToFloat()` (in module *pyg4ometry.compare*), 62
- `_expansionFactorTo4DMatrix()` (in module *pyg4ometry.fluka.directive*), 136
- `_expansionsTo4DMatrices()` (*pyg4ometry.fluka.Transform method*), 177
- `_expansionsTo4DMatrices()` (*pyg4ometry.fluka.directive.Transform method*), 135

- `_extent()` (*pyg4ometry.convert.ARB method*), 87
- `_extent()` (*pyg4ometry.fluka.ARB method*), 163
- `_extent()` (*pyg4ometry.fluka.body.ARB method*), 125
- `_faceNumbersToZeroCountingIndices()` (*pyg4ometry.convert.ARB method*), 87
- `_faceNumbersToZeroCountingIndices()` (*pyg4ometry.fluka.ARB method*), 163
- `_faceNumbersToZeroCountingIndices()` (*pyg4ometry.fluka.body.ARB method*), 125
- `_filterBlackHoleRegions()` (in module *pyg4ometry.convert.fluka2Geant4*), 70
- `_filterHalfSpaces()` (in module *pyg4ometry.convert.fluka2Geant4*), 70
- `_filterNullAABBs()` (in module *pyg4ometry.convert.fluka2Geant4*), 70
- `_filterRedunantZonesSymbollicaly()` (in module *pyg4ometry.fluka.boolean_algebra*), 133
- `_filterRedundantZonesMetricCheck()` (in module *pyg4ometry.fluka.boolean_algebra*), 133
- `_filterRegionNullZones()` (in module *pyg4ometry.convert.fluka2Geant4*), 70
- `_filterRegistryNullZones()` (in module *pyg4ometry.convert.fluka2Geant4*), 70
- `_filteredRegions()` (in module *pyg4ometry.convert.fluka2Geant4*), 69
- `_findDictByName()` (*pyg4ometry.geant4.Registry method*), 347
- `_findDictByName()` (*pyg4ometry.geant4.Registry.Registry method*), 321
- `_findLines()` (*pyg4ometry.fluka.Reader method*), 169
- `_findLines()` (*pyg4ometry.fluka.reader.Reader method*), 149
- `_flukaFFString` (*pyg4ometry.fluka.Writer attribute*), 170
- `_flukaFFString` (*pyg4ometry.fluka.Writer.Writer attribute*), 120
- `_flukaRegistryToG4Registry()` (in module *pyg4ometry.convert.fluka2Geant4*), 69
- `_flukaTemplateFileName` (in module *pyg4ometry.fluka.flair*), 137
- `_formatFlukaMaterialPair()` (in module *pyg4ometry.fluka.material*), 145
- `_geant4MultiUnionSubtraction()` (*pyg4ometry.fluka.Zone method*), 174
- `_geant4MultiUnionSubtraction()` (*pyg4ometry.fluka.region.Zone method*), 153
- `_generateRandomColour()` (*pyg4ometry.visualisation.VisualisationOptions method*), 407
- `_generateRandomColour()` (*pyg4ometry.visualisation.VisualisationOptions.VisOptions method*), 393
- `_generateRandomColour()` (*pyg4ometry.visualisation._VisOptions method*), 412
- `_generate_name()` (in module *pyg4ometry.fluka.region*), 152
- `_getAxisAlignedBoundingBox()` (in module *pyg4ometry.fluka.region*), 155
- `_getBoundingBox()` (in module *pyg4ometry.compare*), 67
- `_getBoundingBox()` (in module *pyg4ometry.compare._Compare*), 61
- `_getBoundingBox()` (in module *pyg4ometry.visualisation*), 405
- `_getBoundingBox()` (in module *pyg4ometry.visualisation.Mesh*), 388
- `_getBoundingBoxMesh()` (in module *pyg4ometry.visualisation*), 405
- `_getBoundingBoxMesh()` (in module *pyg4ometry.visualisation.Mesh*), 388
- `_getCacherFromBody()` (*pyg4ometry.fluka.fluka_registry.FlukaBodyStore method*), 140
- `_getClassVariables()` (in module *pyg4ometry.geant4*), 348
- `_getClassVariables()` (in module *pyg4ometry.geant4._Material*), 326
- `_getConstituentMaterialNamesFromCompound()` (in module *pyg4ometry.fluka.reader*), 149
- `_getContentsOfLatticeCells()` (in module *pyg4ometry.convert.fluka2Geant4*), 70
- `_getCutterData()` (*pyg4ometry.visualisation.VtkViewer method*), 410
- `_getCutterData()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 398
- `_getDaughterMeshes()` (*pyg4ometry.geant4.AssemblyVolume method*), 336
- `_getDaughterMeshes()` (*pyg4ometry.geant4.AssemblyVolume.AssemblyVolume method*), 310
- `_getDaughterMeshesByIndex()` (*pyg4ometry.geant4.AssemblyVolume method*), 336
- `_getDaughterMeshesByIndex()` (*pyg4ometry.geant4.AssemblyVolume.AssemblyVolume method*), 310
- `_getDaughterMeshesByName()` (*pyg4ometry.geant4.AssemblyVolume method*), 336
- `_getDaughterMeshesByName()` (*pyg4ometry.geant4.AssemblyVolume.AssemblyVolume method*), 310
- `_getDefaulOpVis()` (*pyg4ometry.visualisation.VtkViewer method*), 410
- `_getDefaultVis()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 410

- method), 399
- `_getDefaultVis()` (`pyg4ometry.visualisation.VtkViewer.VtkViewer` module method), 400
- `_getDefaultVis()` (`pyg4ometry.visualisation.VtkViewerColorizer.VtkViewerColorizer` module method), 411
- `_getDefaultVis()` (`pyg4ometry.visualisation.VtkViewerColorizer.VtkViewerColorizer` module method), 413
- `_getDefaultVis()` (`pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew` module method), 402
- `_getFLUKATemplateFileName()` (in module `pyg4ometry.fluka.flair`), 137
- `_getMaximalOfTwoAABBs()` (in module `pyg4ometry.convert.fluka2Geant4`), 70
- `_getMaximalQuadricRegionAABBs()` (in module `pyg4ometry.convert.fluka2Geant4`), 71
- `_getMeshAndAABB()` (in module `pyg4ometry.fluka.boolean_algebra`), 132
- `_getOverlappingAABBs()` (in module `pyg4ometry.convert.fluka2Geant4`), 70
- `_getPVMeshes()` (`pyg4ometry.geant4.AssemblyVolume` method), 336
- `_getPVMeshes()` (`pyg4ometry.geant4.AssemblyVolume.AssemblyVolume` method), 310
- `_getPeriodicTable()` (in module `pyg4ometry.convert.fluka2g4materials`), 72
- `_getPhysicalDaughterMesh()` (`pyg4ometry.geant4.AssemblyVolume` method), 336
- `_getPhysicalDaughterMesh()` (`pyg4ometry.geant4.AssemblyVolume.AssemblyVolume` method), 310
- `_getPhysicalDaughterMesh()` (`pyg4ometry.geant4.LogicalVolume` method), 332, 341
- `_getPhysicalDaughterMesh()` (`pyg4ometry.geant4.LogicalVolume.LogicalVolume` method), 313
- `_getPixelRatio()` (`pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor` static method), 361, 364
- `_getPixelRatio()` (`pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor` static method), 355
- `_getPixelRatio()` (`pyg4ometry.gui.example1.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor` static method), 358
- `_getPredefinedMaterialVisOptions()` (in module `pyg4ometry.visualisation`), 407, 412
- `_getProperty()` (`pyg4ometry.geant4.solid.SolidBase` method), 309
- `_getProperty()` (`pyg4ometry.geant4.solid.SolidBase.SolidBase` method), 242
- `_getProperty()` (`pyg4ometry.geant4.solid.SolidBase` method), 261–270, 272–274, 277, 281, 285–293, 295, 297, 299–302, 304, 305, 308
- `_getRawMesh()` (in module `pyg4ometry.compare`), 67
- `_getRawMesh()` (in module `pyg4ometry.compare._Compare`), 61
- `_getRegionOfInterestAABBs()` (in module `pyg4ometry.convert.fluka2Geant4`), 70
- `_getRelativeTransform()` (in module `pyg4ometry.fluka.region`), 155
- `_getSelectedRegions()` (in module `pyg4ometry.convert.fluka2Geant4`), 70
- `_getSolidFromBoolean()` (`pyg4ometry.fluka.Zone` static method), 174
- `_getSolidFromBoolean()` (`pyg4ometry.fluka.region.Zone` static method), 152
- `_getTransformedCellRegionAABB()` (in module `pyg4ometry.convert.fluka2Geant4`), 70
- `_getVerticesAndPolygons()` (`pyg4ometry.convert.ARB` method), 87
- `_getVerticesAndPolygons()` (`pyg4ometry.fluka.ARB` method), 164
- `_getVerticesAndPolygons()` (`pyg4ometry.fluka.body.ARB` method), 126
- `_getWorldDimensions()` (in module `pyg4ometry.convert.fluka2Geant4`), 70
- `_get_relative_rot_matrix()` (in module `pyg4ometry.fluka.region`), 155
- `_get_relative_rotation()` (in module `pyg4ometry.fluka.region`), 155
- `_get_relative_translation()` (in module `pyg4ometry.fluka.region`), 155
- `_grouper()` (in module `pyg4ometry.fluka.material`), 145
- `_isClose()` (in module `pyg4ometry.io.ROOTTGeo`), 365
- `_isTransformedCellRegionIntersectingWithRegion()` (in module `pyg4ometry.convert.fluka2Geant4`), 70
- `_keysyms` (in module `pyg4ometry.gui`), 364
- `_keysyms` (in module `pyg4ometry.gui.QVTKRenderWindowInteractor`), 356
- `_keysyms` (in module `pyg4ometry.gui.example1`), 359
- `_linearEccentricity()` (`pyg4ometry.convert.ELL` method), 86
- `_linearEccentricity()` (`pyg4ometry.fluka.ELL` method), 162
- `_linearEccentricity()` (`pyg4ometry.fluka.body.ELL` method), 124
- `_load()` (`pyg4ometry.fluka.Reader` method), 169
- `_load()` (`pyg4ometry.fluka.reader.Reader` method), 149
- `_loadFile()` (in module `pyg4ometry.cli`), 416
- `_load_ascii()` (`pyg4ometry.stl.Reader` method), 385
- `_load_ascii()` (`pyg4ometry.stl.Reader.Reader` method), 384
- `_load_binary()` (`pyg4ometry.stl.Reader` method), 386
- `_load_binary()` (`pyg4ometry.stl.Reader.Reader` method), 384
- `_load_pickle()` (in module `pyg4ometry.utils`), 424
- `_makeBaseCompoundMaterial()`

(pyg4ometry.convert.fluka2g4materials._FlukaToG4MaterialConverter method), 72

`_makeBodyMinimumAABBMap()` (in module `pyg4ometry.convert.fluka2Geant4`), 70

`_makeG4Element()` (`pyg4ometry.io.ROOTTGeo.Reader` method), 365

`_makeG4Isotope()` (`pyg4ometry.io.ROOTTGeo.Reader` method), 365

`_makeLengthSafetyRegistry()` (in module `pyg4ometry.convert.fluka2Geant4`), 70

`_makeMaterials()` (`pyg4ometry.gdml.Reader` method), 212

`_makeMaterials()` (`pyg4ometry.gdml.Reader.Reader` method), 205

`_makeNISTCompoundList()` (in module `pyg4ometry.geant4`), 349

`_makeNISTCompoundList()` (in module `pyg4ometry.geant4._Material`), 326

`_makeQuadricRegionBodyAABBMap()` (in module `pyg4ometry.convert.fluka2Geant4`), 71

`_makeUniqueQuadricRegions()` (in module `pyg4ometry.convert.fluka2Geant4`), 71

`_makeWorldLogicalVolume()` (in module `pyg4ometry.fluka.region`), 155

`_makeWorldVolume()` (in module `pyg4ometry.convert.fluka2Geant4`), 69

`_make_body()` (in module `pyg4ometry.fluka.reader`), 149

`_mangleElementName()` (`pyg4ometry.convert.fluka2g4materials._FlukaToG4MaterialConverter` static method), 72

`_meshes()` (in module `pyg4ometry.compare`), 67

`_meshes()` (in module `pyg4ometry.compare._Compare`), 61

`_nTermsDNFZone()` (in module `pyg4ometry.fluka.boolean_algebra`), 133

`_names()` (in module `pyg4ometry.compare`), 67

`_names()` (in module `pyg4ometry.compare._Compare`), 60

`_namesIgnorePointer()` (in module `pyg4ometry.compare`), 67

`_namesIgnorePointer()` (in module `pyg4ometry.compare._Compare`), 60

`_nistElementZToName` (in module `pyg4ometry.geant4`), 348

`_nistElementZToName` (in module `pyg4ometry.geant4._Material`), 326

`_nistMaterialDict` (in module `pyg4ometry.geant4`), 348

`_nistMaterialDict` (in module `pyg4ometry.geant4._Material`), 325

`_nistMaterialList` (in module `pyg4ometry.geant4`), 348

`_nistMaterialList` (in module `pyg4ometry.geant4._Material`), 325

`_oce2Geant4_traverse()` (in module `pyg4ometry.convert`), 94

`_oce2Geant4_traverse()` (in module `pyg4ometry.convert.oce2Geant4`), 77

`_orderMaterialList()` (`pyg4ometry.geant4.Registry` method), 346

`_orderMaterialList()` (`pyg4ometry.geant4.Registry.Registry` method), 320

`_parseAuxiliary()` (`pyg4ometry.gdml.Reader` method), 212

`_parseAuxiliary()` (`pyg4ometry.gdml.Reader.Reader` method), 205

`_parseBodies()` (`pyg4ometry.fluka.Reader` method), 169

`_parseBodies()` (`pyg4ometry.fluka.reader.Reader` method), 149

`_parseCards()` (`pyg4ometry.fluka.Reader` method), 169

`_parseCards()` (`pyg4ometry.fluka.reader.Reader` method), 149

`_parseFraction()` (in module `pyg4ometry.fluka.material`), 145

`_parseGeometryDirective()` (`pyg4ometry.fluka.Reader` method), 169

`_parseGeometryDirective()` (`pyg4ometry.fluka.reader.Reader` method), 149

`_parseLattice()` (`pyg4ometry.fluka.Reader` method), 169

`_parseLattice()` (`pyg4ometry.fluka.reader.Reader` method), 149

`_parseMaterialAssignments()` (`pyg4ometry.fluka.Reader` method), 169

`_parseMaterialAssignments()` (`pyg4ometry.fluka.reader.Reader` method), 149

`_parseMaterials()` (`pyg4ometry.fluka.Reader` method), 169

`_parseMaterials()` (`pyg4ometry.fluka.reader.Reader` method), 149

`_parseRegions()` (`pyg4ometry.fluka.Reader` method), 169

`_parseRegions()` (`pyg4ometry.fluka.reader.Reader` method), 149

`_parseRotDefinis()` (`pyg4ometry.fluka.Reader` method), 169

`_parseRotDefinis()` (`pyg4ometry.fluka.reader.Reader` method), 149

`_parseStrMultipletAsFloat()` (in module `pyg4ometry.cli`), 416

`_parseStrPythonAsDict()` (in module `pyg4ometry.cli`), 416

`_parseStrPythonAsSolid()` (in module `pyg4ometry.cli`), 416

[_parse_preprocessor_conditional\(\)](#) (in module [pyg4ometry.fluka.preprocessor](#)), 147
[_parse_preprocessor_define\(\)](#) (in module [pyg4ometry.fluka.preprocessor](#)), 147
[_parse_preprocessor_include\(\)](#) (in module [pyg4ometry.fluka.preprocessor](#)), 147
[_periodicTable](#) (in module [pyg4ometry.convert.fluka2g4materials](#)), 72
[_polyDataToActor\(\)](#) ([pyg4ometry.features.algos.vtkViewer](#) method), 97
[_polydata2Actor\(\)](#) ([pyg4ometry.visualisation.VtkViewerNew](#) method), 412
[_polydata2Actor\(\)](#) ([pyg4ometry.visualisation.VtkViewerNew](#) method), 401
[_predefinedMaterialVisOptions](#) (in module [pyg4ometry.visualisation](#)), 407
[_predefinedMaterialVisOptions](#) (in module [pyg4ometry.visualisation.VisualisationOptions](#)), 392
[_printCitation\(\)](#) (in module [pyg4ometry.cli](#)), 417
[_qt_key_to_key_sym\(\)](#) (in module [pyg4ometry.gui](#)), 364
[_qt_key_to_key_sym\(\)](#) (in module [pyg4ometry.gui.QVTKRenderWindowInteractor](#)), 356
[_qt_key_to_key_sym\(\)](#) (in module [pyg4ometry.gui.example1](#)), 359
[_quadricMatrixToCoefficients\(\)](#) ([pyg4ometry.convert.QUA](#) static method), 92
[_quadricMatrixToCoefficients\(\)](#) ([pyg4ometry.fluka.QUA](#) static method), 169
[_quadricMatrixToCoefficients\(\)](#) ([pyg4ometry.fluka.body.QUA](#) static method), 131
[_raiseIfDifferentName\(\)](#) ([pyg4ometry.fluka.RecursiveRotoTranslation](#) method), 178
[_raiseIfDifferentName\(\)](#) ([pyg4ometry.fluka.directive.RecursiveRotoTranslation](#) method), 135
[_randomName\(\)](#) (in module [pyg4ometry.fluka.region](#)), 155
[_readFile\(\)](#) ([pyg4ometry.fluka.material.multiGroupNeutronCrossSections](#) method), 144
[_regionZoneAABBsToRegionAABBs\(\)](#) (in module [pyg4ometry.convert.fluka2Geant4](#)), 71
[_rightMultiplyMatrices\(\)](#) (in module [pyg4ometry.fluka.directive](#)), 136
[_rodrigues_anti_parallel\(\)](#) (in module [pyg4ometry.geant4.solid](#)), 276, 280, 284, 307
[_rodrigues_anti_parallel\(\)](#) (in module [pyg4ometry.transformation](#)), 423
[_rotoTranslationFromTra2\(\)](#) (in module [pyg4ometry.convert](#)), 82
[_rotoTranslationsTo4DMatrices\(\)](#) ([pyg4ometry.fluka.Transform](#) method), 177
[_rotoTranslationsTo4DMatrices\(\)](#) ([pyg4ometry.fluka.directive.Transform](#) method), 135
[_safeName\(\)](#) (in module [pyg4ometry.geant4](#)), 349
[_safeName\(\)](#) (in module [pyg4ometry.geant4._Material](#)), 326
[_semiminor\(\)](#) ([pyg4ometry.convert.ELL](#) method), 86
[_semiminor\(\)](#) ([pyg4ometry.fluka.ELL](#) method), 162
[_semiminor\(\)](#) ([pyg4ometry.fluka.body.ELL](#) method), 124
[_setEventInformation\(\)](#) ([pyg4ometry.gui.QVTKRenderWindowInteractor](#) method), 361, 364
[_setEventInformation\(\)](#) ([pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor](#) method), 355
[_setEventInformation\(\)](#) ([pyg4ometry.gui.example1.QVTKRenderWindowInteractor](#) method), 358
[_setProperty\(\)](#) ([pyg4ometry.geant4.solid.SolidBase](#) method), 309
[_setProperty\(\)](#) ([pyg4ometry.geant4.solid.SolidBase.SolidBase](#) method), 242
[_setProperty\(\)](#) ([pyg4ometry.geant4.solid._SolidBase](#) method), 261–270, 272–274, 277, 281, 285–293, 295, 297, 299–302, 304, 305, 308
[_solid2tessellated\(\)](#) (in module [pyg4ometry.geant4](#)), 332
[_solid2tessellated\(\)](#) (in module [pyg4ometry.geant4.LogicalVolume](#)), 312
[_splitMaterialCards\(\)](#) ([pyg4ometry.fluka.Reader](#) method), 169
[_splitMaterialCards\(\)](#) ([pyg4ometry.fluka.reader.Reader](#) method), 149
[_testDaughterNameSets\(\)](#) (in module [pyg4ometry.compare](#)), 66
[_testDaughterNameSets\(\)](#) (in module [pyg4ometry.compare._Compare](#)), 60
[_testNames\(\)](#) ([pyg4ometry.compare.Tests](#) attribute), 64
[_testNames\(\)](#) ([pyg4ometry.compare._Compare.Tests](#) attribute), 58
[_toTessellatedSolid\(\)](#) ([pyg4ometry.convert.ARB](#) method), 87
[_toTessellatedSolid\(\)](#) ([pyg4ometry.fluka.ARB](#) method), 163
[_toTessellatedSolid\(\)](#) ([pyg4ometry.fluka.body.ARB](#) method), 126
[_toVerticesAndPolygons\(\)](#) ([pyg4ometry.convert.ARB](#) method), 87

<code>_toVerticesAndPolygons()</code> (pyg4ometry.fluka.ARB method), 164	<code>_withLengthSafety()</code> (pyg4ometry.convert.TRD method), 86	<code>(pyg4ometry.convert.TRD method), 86</code>
<code>_toVerticesAndPolygons()</code> (pyg4ometry.fluka.body.ARB method), 126	<code>_withLengthSafety()</code> (pyg4ometry.convert.XCC method), 89	<code>(pyg4ometry.convert.XCC method), 89</code>
<code>_totalMassWeightedFractionOfCompound()</code> (pyg4ometry.convert.fluka2g4materials._FlukaToG4Material method), 72	<code>_withLengthSafety()</code> (pyg4ometry.convert.XEC method), 88	<code>(pyg4ometry.convert.XEC method), 88</code>
<code>_transformationIndices()</code> (pyg4ometry.fluka.RecursiveRotoTranslation method), 178	<code>_withLengthSafety()</code> (pyg4ometry.convert.XYP method), 88	<code>(pyg4ometry.convert.XYP method), 88</code>
<code>_transformationIndices()</code> (pyg4ometry.fluka.directive.RecursiveRotoTranslation method), 136	<code>_withLengthSafety()</code> (pyg4ometry.convert.XZP method), 90	<code>(pyg4ometry.convert.XZP method), 90</code>
<code>_translationTo4DMatrix()</code> (in module pyg4ometry.fluka.directive), 136	<code>_withLengthSafety()</code> (pyg4ometry.convert.YCC method), 91	<code>(pyg4ometry.convert.YCC method), 91</code>
<code>_translationsTo4DMatrices()</code> (pyg4ometry.fluka.Transform method), 177	<code>_withLengthSafety()</code> (pyg4ometry.convert.YEC method), 88	<code>(pyg4ometry.convert.YEC method), 88</code>
<code>_translationsTo4DMatrices()</code> (pyg4ometry.fluka.directive.Transform method), 135	<code>_withLengthSafety()</code> (pyg4ometry.convert.YZP method), 90	<code>(pyg4ometry.convert.YZP method), 90</code>
<code>_twoPiValueCheck()</code> (pyg4ometry.geant4.solid.SolidBase method), 309	<code>_withLengthSafety()</code> (pyg4ometry.convert.ZCC method), 92	<code>(pyg4ometry.convert.ZCC method), 92</code>
<code>_twoPiValueCheck()</code> (pyg4ometry.geant4.solid.SolidBase method), 242	<code>_withLengthSafety()</code> (pyg4ometry.fluka.ARB method), 164	<code>(pyg4ometry.fluka.ARB method), 164</code>
<code>_twoPiValueCheck()</code> (pyg4ometry.geant4.solid._SolidBase method), 261–270, 272–274, 277, 281, 285–293, 295, 297, 299–302, 304, 305, 308	<code>_withLengthSafety()</code> (pyg4ometry.fluka.BOX method), 160	<code>(pyg4ometry.fluka.BOX method), 160</code>
<code>_vector()</code> (in module pyg4ometry.compare), 67	<code>_withLengthSafety()</code> (pyg4ometry.fluka.ELL method), 162	<code>(pyg4ometry.fluka.ELL method), 162</code>
<code>_vector()</code> (in module pyg4ometry.compare._Compare), 60	<code>_withLengthSafety()</code> (pyg4ometry.fluka.PLA method), 165	<code>(pyg4ometry.fluka.PLA method), 165</code>
<code>_vtkCutterPlane()</code> (in module pyg4ometry.features.algos), 98	<code>_withLengthSafety()</code> (pyg4ometry.fluka.QUA method), 169	<code>(pyg4ometry.fluka.QUA method), 169</code>
<code>_vtkPolydataEdgeInformation()</code> (in module pyg4ometry.features.algos), 97	<code>_withLengthSafety()</code> (pyg4ometry.fluka.RCC method), 161	<code>(pyg4ometry.fluka.RCC method), 161</code>
<code>_vtkPolydataToConnectedEdges()</code> (in module pyg4ometry.features.algos), 97	<code>_withLengthSafety()</code> (pyg4ometry.fluka.REC method), 161	<code>(pyg4ometry.fluka.REC method), 161</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.ARB method), 87	<code>_withLengthSafety()</code> (pyg4ometry.fluka.RPP method), 160	<code>(pyg4ometry.fluka.RPP method), 160</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.BOX method), 84	<code>_withLengthSafety()</code> (pyg4ometry.fluka.SPH method), 161	<code>(pyg4ometry.fluka.SPH method), 161</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.ELL method), 86	<code>_withLengthSafety()</code> (pyg4ometry.fluka.TRC method), 162	<code>(pyg4ometry.fluka.TRC method), 162</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.PLA method), 89	<code>_withLengthSafety()</code> (pyg4ometry.fluka.XCC method), 165	<code>(pyg4ometry.fluka.XCC method), 165</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.QUA method), 92	<code>_withLengthSafety()</code> (pyg4ometry.fluka.XEC method), 167	<code>(pyg4ometry.fluka.XEC method), 167</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.RCC method), 85	<code>_withLengthSafety()</code> (pyg4ometry.fluka.XYP method), 164	<code>(pyg4ometry.fluka.XYP method), 164</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.REC method), 85	<code>_withLengthSafety()</code> (pyg4ometry.fluka.XZP method), 164	<code>(pyg4ometry.fluka.XZP method), 164</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.RPP method), 83	<code>_withLengthSafety()</code> (pyg4ometry.fluka.YCC method), 166	<code>(pyg4ometry.fluka.YCC method), 166</code>
<code>_withLengthSafety()</code> (pyg4ometry.convert.SPH method), 84	<code>_withLengthSafety()</code> (pyg4ometry.fluka.YEC method), 168	<code>(pyg4ometry.fluka.YEC method), 168</code>
	<code>_withLengthSafety()</code> (pyg4ometry.fluka.YZP method), 165	<code>(pyg4ometry.fluka.YZP method), 165</code>

<code>_withLengthSafety()</code> <i>method</i>), 166	(<code>pyg4ometry.fluka.ZCC</code>	<code>ABS</code> (<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpres</code> <i>attribute</i>), 184
<code>_withLengthSafety()</code> <i>method</i>), 168	(<code>pyg4ometry.fluka.ZEC</code>	<code>ABS</code> (<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpres</code> <i>attribute</i>), 193
<code>_withLengthSafety()</code> <i>method</i>), 126	(<code>pyg4ometry.fluka.body.ARB</code>	<code>abs()</code> (in module <code>pyg4ometry.gdml</code>), 220 <code>abs()</code> (in module <code>pyg4ometry.gdml.Defines</code>), 200
<code>_withLengthSafety()</code> <i>method</i>), 122	(<code>pyg4ometry.fluka.body.BOX</code>	<code>ABS()</code> (<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExp</code> <i>method</i>), 191
<code>_withLengthSafety()</code> <i>method</i>), 125	(<code>pyg4ometry.fluka.body.ELL</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.ComplexRegi</code> <i>method</i>), 113
<code>_withLengthSafety()</code> <i>method</i>), 127	(<code>pyg4ometry.fluka.body.PLA</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.MultipleUnio</code> <i>method</i>), 113
<code>_withLengthSafety()</code> <i>method</i>), 131	(<code>pyg4ometry.fluka.body.QUA</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.MultipleUnio</code> <i>method</i>), 114
<code>_withLengthSafety()</code> <i>method</i>), 123	(<code>pyg4ometry.fluka.body.RCC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.OneSubZone</code> <i>method</i>), 116
<code>_withLengthSafety()</code> <i>method</i>), 123	(<code>pyg4ometry.fluka.body.REC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 103
<code>_withLengthSafety()</code> <i>method</i>), 122	(<code>pyg4ometry.fluka.body.RPP</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 104
<code>_withLengthSafety()</code> <i>method</i>), 123	(<code>pyg4ometry.fluka.body.SPH</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 104
<code>_withLengthSafety()</code> <i>method</i>), 124	(<code>pyg4ometry.fluka.body.TRC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 106
<code>_withLengthSafety()</code> <i>method</i>), 128	(<code>pyg4ometry.fluka.body.XCC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 102
<code>_withLengthSafety()</code> <i>method</i>), 129	(<code>pyg4ometry.fluka.body.XEC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 103
<code>_withLengthSafety()</code> <i>method</i>), 126	(<code>pyg4ometry.fluka.body.XYP</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 106
<code>_withLengthSafety()</code> <i>method</i>), 126	(<code>pyg4ometry.fluka.body.XZP</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 104
<code>_withLengthSafety()</code> <i>method</i>), 128	(<code>pyg4ometry.fluka.body.YCC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 107
<code>_withLengthSafety()</code> <i>method</i>), 130	(<code>pyg4ometry.fluka.body.YEC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 106
<code>_withLengthSafety()</code> <i>method</i>), 127	(<code>pyg4ometry.fluka.body.YZP</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 106
<code>_withLengthSafety()</code> <i>method</i>), 129	(<code>pyg4ometry.fluka.body.ZCC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 107
<code>_withLengthSafety()</code> <i>method</i>), 130	(<code>pyg4ometry.fluka.body.ZEC</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 105
<code>_writeFile()</code> (in module <code>pyg4ometry.cli</code>), 416		<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 105
<code>_write_pickle()</code> (in module <code>pyg4ometry.utils</code>), 424		<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 105
<code>_zoneGraphPycgal()</code> <i>method</i>), 176	(<code>pyg4ometry.fluka.Region</code>	<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</code> <i>method</i>), 105
<code>_zoneGraphPycgal()</code> (<code>pyg4ometry.fluka.region.Region</code> <i>method</i>), 154		<code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionsCont</code> <i>method</i>), 112 <code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.SimpleRegion</code> <i>method</i>), 113 <code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.SingleUnary</code> <i>method</i>), 116 <code>accept()</code> (<code>pyg4ometry.fluka.RegionExpression.RegionParser.SingleUnion</code> <i>method</i>), 114

A

`AABB` (class in `pyg4ometry.fluka`), 173
`AABB` (class in `pyg4ometry.fluka.vector`), 156
`aabb()` (`pyg4ometry.fluka.Region` *method*), 176
`aabb()` (`pyg4ometry.fluka.region.Region` *method*), 154

`add_material()` (*pyg4ometry.geant4.Material* method), 350
`add_vertex()` (*pyg4ometry.stl._Facet* method), 385
`add_vertex()` (*pyg4ometry.stl.Reader._Facet* method), 384
`addActor()` (*pyg4ometry.visualisation.VtkViewer* method), 410
`addActor()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 399
`addAndCollapseAssemblyVolumeRecursive()` (*pyg4ometry.geant4.Registry* method), 347
`addAndCollapseAssemblyVolumeRecursive()` (*pyg4ometry.geant4.Registry.Registry* method), 320
`addAuxiliary()` (*pyg4ometry.geant4.Registry* method), 346
`addAuxiliary()` (*pyg4ometry.geant4.Registry.Registry* method), 319
`addAuxiliaryInfo()` (*pyg4ometry.geant4.LogicalVolume* method), 334, 343
`addAuxiliaryInfo()` (*pyg4ometry.geant4.LogicalVolume.LogicalVolume* method), 315
`addAxes()` (*pyg4ometry.visualisation.VtkViewer* method), 408
`addAxes()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 396
`addAxes()` (*pyg4ometry.visualisation.VtkViewerNew* method), 412
`addAxes()` (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401
`addAxesWidget()` (*pyg4ometry.visualisation.VtkViewer* method), 408
`addAxesWidget()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 396
`addAxesWidget()` (*pyg4ometry.visualisation.VtkViewerNew* method), 412
`addAxesWidget()` (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401
`addAxis()` (*pyg4ometry.features.algos.vtkViewer* method), 97
`addBDSIMObject()` (*pyg4ometry.geant4.LogicalVolume* method), 332, 341
`addBDSIMObject()` (*pyg4ometry.geant4.LogicalVolume.LogicalVolume* method), 313
`addBeam()` (*pyg4ometry.fluka.fluka_registry.FlukaRegistry* method), 139
`addBeam()` (*pyg4ometry.fluka.FlukaRegistry* method), 171
`addBeamPos()` (*pyg4ometry.fluka.fluka_registry.FlukaRegistry* method), 139
`addBeamPos()` (*pyg4ometry.fluka.FlukaRegistry* method), 171
`addBody()` (*pyg4ometry.fluka.fluka_registry.BaseCacher* method), 141
`addBody()` (*pyg4ometry.fluka.fluka_registry.FlukaBodyStore* method), 141
`addBody()` (*pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact* method), 142
`addBody()` (*pyg4ometry.fluka.fluka_registry.FlukaRegistry* method), 138
`addBody()` (*pyg4ometry.fluka.FlukaBodyStoreExact* method), 172
`addBody()` (*pyg4ometry.fluka.FlukaRegistry* method), 170
`addBooleanSolidRecursive()` (*pyg4ometry.visualisation.VtkViewer* method), 409
`addBooleanSolidRecursive()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 398
`addCard()` (*pyg4ometry.fluka.fluka_registry.FlukaRegistry* method), 139
`addCard()` (*pyg4ometry.fluka.FlukaRegistry* method), 170
`addClipper()` (*pyg4ometry.visualisation.VtkViewerNew* method), 412
`addClipper()` (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401
`addClipperWidget()` (*pyg4ometry.visualisation.VtkViewerNew* method), 412
`addClipperWidget()` (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401
`addConstProperty()` (*pyg4ometry.compare._Material* method), 64
`addConstProperty()` (*pyg4ometry.gdml._Material* method), 214
`addConstProperty()` (*pyg4ometry.geant4._Material.Material* method), 328
`addConstProperty()` (*pyg4ometry.geant4.Material* method), 351
`addConstProperty()` (*pyg4ometry.geant4.solid.OpticalSurface* method), 286
`addConstProperty()` (*pyg4ometry.geant4.solid.OpticalSurface.OpticalSurface* method), 237
`addCutter()` (*pyg4ometry.visualisation.VtkViewerNew* method), 412
`addCutter()` (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401
`addCutterDataToPlot()` (in module *pyg4ometry.visualisation.Plot*), 388
`addCutterPlane()` (*pyg4ometry.visualisation.VtkViewer* method), 410
`addCutterPlane()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 399
`addCutterWidget()` (*pyg4ometry.visualisation.VtkViewerNew* method), 412
`addCutterWidget()` (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401

addDefaults() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
 addDefaults() (pyg4ometry.fluka.FlukaRegistry method), 139
 addDefaults() (pyg4ometry.fluka.FlukaRegistry method), 170
 addDefine() (pyg4ometry.geant4.Registry method), 346
 addDefine() (pyg4ometry.geant4.Registry.Registry method), 319
 addDeltaRay() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 140
 addDeltaRay() (pyg4ometry.fluka.FlukaRegistry method), 171
 addDetector() (pyg4ometry.fluka.Writer method), 170
 addDetector() (pyg4ometry.fluka.Writer.Writer method), 120
 addDetector() (pyg4ometry.gdml.Writer method), 215
 addDetector() (pyg4ometry.gdml.Writer.Writer method), 207
 addElcField() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addElcField() (pyg4ometry.fluka.FlukaRegistry method), 171
 addElement() (pyg4ometry.bdsim.Beamline method), 55
 addElement() (pyg4ometry.bdsim.beamline.Beamline method), 53
 addFlukaRegions() (pyg4ometry.visualisation.ViewerBase method), 405
 addFlukaRegions() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 390
 addGlobal() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addGlobal() (pyg4ometry.fluka.FlukaRegistry method), 171
 addInstance() (pyg4ometry.visualisation.RenderWriter method), 414
 addInstance() (pyg4ometry.visualisation.RenderWriter.RenderWriter method), 389
 addInstance() (pyg4ometry.visualisation.ViewerBase method), 405
 addInstance() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 390
 addIntersection() (pyg4ometry.fluka.Region method), 175
 addIntersection() (pyg4ometry.fluka.region.Region method), 154
 addIntersection() (pyg4ometry.fluka.region.Zone method), 152
 addIntersection() (pyg4ometry.fluka.Zone method), 174
 addIonFluct() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 140
 addIonFluct() (pyg4ometry.fluka.FlukaRegistry method), 171
 addLattice() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
 addLattice() (pyg4ometry.fluka.FlukaRegistry method), 170
 addLine() (pyg4ometry.features.algos.vtkViewer method), 97
 addLogicalVolume() (pyg4ometry.geant4.Registry method), 345
 addLogicalVolume() (pyg4ometry.geant4.Registry.Registry method), 319
 addLogicalVolume() (pyg4ometry.visualisation.ViewerBase method), 405
 addLogicalVolume() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 390
 addLogicalVolume() (pyg4ometry.visualisation.VtkViewer method), 409
 addLogicalVolume() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 398
 addLogicalVolumeBounding() (pyg4ometry.visualisation.VtkViewer method), 409
 addLogicalVolumeBounding() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 398
 addLogicalVolumeRecursive() (pyg4ometry.visualisation.RenderWriter method), 413
 addLogicalVolumeRecursive() (pyg4ometry.visualisation.RenderWriter.RenderWriter method), 389
 addLogicalVolumeRecursive() (pyg4ometry.visualisation.VtkViewer method), 409
 addLogicalVolumeRecursive() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 398
 addLowMat() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addLowMat() (pyg4ometry.fluka.FlukaRegistry method), 171
 addLowMatAllMaterials() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addLowMatAllMaterials() (pyg4ometry.fluka.FlukaRegistry method), 171
 addLowPwxs() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addLowPwxs() (pyg4ometry.fluka.FlukaRegistry method), 171
 addMaterial() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addMaterial() (pyg4ometry.fluka.FlukaRegistry method), 170
 addMaterial() (pyg4ometry.geant4.Registry method), 346

- 345
- `addMaterial()` (`pyg4ometry.geant4.Registry.Registry` method), 318
- `addMaterialAssignments()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 139
- `addMaterialAssignments()` (`pyg4ometry.fluka.FlukaRegistry` method), 170
- `addMaterialColour()` (`pyg4ometry.fluka.Flair` method), 178
- `addMaterialColour()` (`pyg4ometry.fluka.flair.Flair` method), 137
- `addMaterialPbrOption()` (`pyg4ometry.visualisation.ViewerBase` method), 406
- `addMaterialPbrOption()` (`pyg4ometry.visualisation.ViewerBase.ViewerBase` method), 391
- `addMaterialVisOption()` (`pyg4ometry.visualisation.ViewerBase` method), 406
- `addMaterialVisOption()` (`pyg4ometry.visualisation.ViewerBase.ViewerBase` method), 391
- `addMaterialVisOption()` (`pyg4ometry.visualisation.VtkViewer` method), 408
- `addMaterialVisOption()` (`pyg4ometry.visualisation.VtkViewer.VtkViewer` method), 397
- `addMesh()` (`pyg4ometry.visualisation.RenderWriter` method), 413
- `addMesh()` (`pyg4ometry.visualisation.RenderWriter.Renderer` method), 389
- `addMesh()` (`pyg4ometry.visualisation.ViewerBase` method), 405
- `addMesh()` (`pyg4ometry.visualisation.ViewerBase.ViewerBase` method), 390
- `addMesh()` (`pyg4ometry.visualisation.VtkExporter` method), 415
- `addMesh()` (`pyg4ometry.visualisation.VtkExporter.VtkExporter` method), 395
- `addMesh()` (`pyg4ometry.visualisation.VtkViewer` method), 410
- `addMesh()` (`pyg4ometry.visualisation.VtkViewer.VtkViewer` method), 398
- `addMeshSimple()` (`pyg4ometry.visualisation.VtkViewer` method), 409
- `addMeshSimple()` (`pyg4ometry.visualisation.VtkViewer.VtkViewer` method), 398
- `addMgnCreat()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 139
- `addMgnCreat()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
- `addMgnField()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 139
- `addMgnField()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
- `addMuphoton()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 139
- `addMuphoton()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
- `addOption()` (`pyg4ometry.bdsim.Options` method), 55
- `addOption()` (`pyg4ometry.bdsim.options.Options` method), 53
- `addOverlapMesh()` (`pyg4ometry.visualisation.Mesh` method), 405
- `addOverlapMesh()` (`pyg4ometry.visualisation.Mesh.Mesh` method), 388
- `addPairbrem()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 139
- `addPairbrem()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
- `addPbrOptions()` (`pyg4ometry.visualisation.ViewerBase` method), 406
- `addPbrOptions()` (`pyg4ometry.visualisation.ViewerBase.ViewerBase` method), 390
- `addPhotonuc()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 139
- `addPhotonuc()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
- `addPhysicalVolume()` (`pyg4ometry.geant4.Registry` method), 346
- `addPhysicalVolume()` (`pyg4ometry.geant4.Registry.Registry` method), 319
- `addPlane()` (`pyg4ometry.features.algos.vtkViewer` method), 97
- `addPointToRegion()` (`pyg4ometry.fluka.Extruder` method), 180
- `addPointToRegion()` (`pyg4ometry.fluka.extruder.Extruder` method), 137
- `addPolydata()` (`pyg4ometry.features.algos.vtkViewer` method), 97
- `addPredefinedElements()` (`pyg4ometry.convert.fluka2g4materials._FlukaToG4MaterialCon` method), 72
- `addProperty()` (`pyg4ometry.compare._Material` method), 63
- `addProperty()` (`pyg4ometry.gdml._Material` method), 214
- `addProperty()` (`pyg4ometry.geant4._Material.Material` method), 328
- `addProperty()` (`pyg4ometry.geant4.Material` method), 351
- `addProperty()` (`pyg4ometry.geant4.solid.OpticalSurface` method), 285

addProperty() (pyg4ometry.geant4.solid.OpticalSurface.OpticalSurface method), 152
 method), 236
 addRandomiz() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 173
 method), 139
 addRandomiz() (pyg4ometry.fluka.FlukaRegistry method), 171
 addRegion() (pyg4ometry.fluka.Extruder method), 180
 addRegion() (pyg4ometry.fluka.extruder.Extruder method), 137
 addRegion() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
 addRegion() (pyg4ometry.fluka.FlukaRegistry method), 170
 addRegistry() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 140
 addRegistry() (pyg4ometry.fluka.FlukaRegistry method), 172
 addRotoTranslation() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
 addRotoTranslation() (pyg4ometry.fluka.fluka_registry.RotoTranslation method), 140
 addRotoTranslation() (pyg4ometry.fluka.FlukaRegistry method), 170
 addRotprBin() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addRotprBin() (pyg4ometry.fluka.FlukaRegistry method), 171
 addScoringMesh() (pyg4ometry.visualisation.VtkViewerNew method), 413
 addScoringMesh() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 402
 addSolid() (pyg4ometry.geant4.Registry method), 345
 addSolid() (pyg4ometry.geant4.Registry.Registry method), 319
 addSolid() (pyg4ometry.visualisation.VtkViewer method), 409
 addSolid() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 398
 addStart() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addStart() (pyg4ometry.fluka.FlukaRegistry method), 171
 addSubAuxiliary() (pyg4ometry.gdml.Auxiliary method), 224
 addSubAuxiliary() (pyg4ometry.gdml.Defines.Auxiliary method), 204
 addSubtraction() (pyg4ometry.fluka.Region method), 175
 addSubtraction() (pyg4ometry.fluka.region.Region method), 154
 addSubtraction() (pyg4ometry.fluka.region.Zone method), 173
 addSurface() (pyg4ometry.geant4.Registry method), 346
 addSurface() (pyg4ometry.geant4.Registry.Registry method), 319
 addTitle() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addTitle() (pyg4ometry.fluka.FlukaRegistry method), 170
 addTracks() (pyg4ometry.visualisation.VtkViewerNew method), 413
 addTracks() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 402
 addTriangle() (pyg4ometry.geant4.solid.TessellatedSolid method), 304
 addTriangle() (pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid method), 245
 addUsrBdx() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addUsrBdx() (pyg4ometry.fluka.FlukaRegistry method), 171
 addUsrBin() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addUsrBin() (pyg4ometry.fluka.FlukaRegistry method), 171
 addUsrDump() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addUsrDump() (pyg4ometry.fluka.FlukaRegistry method), 171
 addUsrircall() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addUsrircall() (pyg4ometry.fluka.FlukaRegistry method), 171
 addUsrocall() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 addUsrocall() (pyg4ometry.fluka.FlukaRegistry method), 171
 addVecProperty() (pyg4ometry.compare._Material method), 63
 addVecProperty() (pyg4ometry.gdml._Material method), 214
 addVecProperty() (pyg4ometry.geant4._Material.Material method), 328
 addVecProperty() (pyg4ometry.geant4.Material method), 351
 addVecProperty() (pyg4ometry.geant4.solid.OpticalSurface method), 285
 addVecProperty() (pyg4ometry.geant4.solid.OpticalSurface.OpticalSurface method), 236
 addVertex() (pyg4ometry.geant4.solid.TessellatedSolid method), 304
 addVertex() (pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid method), 304

- method), 245
- addVisOptions() (pyg4ometry.visualisation.ViewerBase method), 406
- addVisOptions() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 141
- addVisOptions() (pyg4ometry.visualisation.ViewerBase method), 390
- addVolumeRecursive() (pyg4ometry.geant4.Registry method), 347
- addVolumeRecursive() (pyg4ometry.geant4.Registry.Registry method), 320
- addZone() (pyg4ometry.fluka.Region method), 175
- addZone() (pyg4ometry.fluka.region.Region method), 154
- All() (pyg4ometry.compare._Compare.TestResult static method), 58
- All() (pyg4ometry.compare.TestResult static method), 65
- allBodiesToRegistry() (pyg4ometry.fluka.Region method), 176
- allBodiesToRegistry() (pyg4ometry.fluka.region.Region method), 154
- allBodiesToRegistry() (pyg4ometry.fluka.region.Zone method), 153
- allBodiesToRegistry() (pyg4ometry.fluka.Zone method), 174
- allowed_finishes (pyg4ometry.geant4.solid.OpticalSurface attribute), 285
- allowed_finishes (pyg4ometry.geant4.solid.OpticalSurface.OpticalSurface attribute), 236
- allowed_models (pyg4ometry.geant4.solid.OpticalSurface attribute), 285
- allowed_models (pyg4ometry.geant4.solid.OpticalSurface.OpticalSurface attribute), 236
- allowed_types (pyg4ometry.geant4.solid.OpticalSurface attribute), 285
- allowed_types (pyg4ometry.geant4.solid.OpticalSurface.OpticalSurface attribute), 236
- allTransformationIndices() (pyg4ometry.fluka.fluka_registry.RotoTranslation method), 140
- AnalyseGeometryComplexity() (in module pyg4ometry.geant4), 348
- AnalyseGeometryComplexity() (in module pyg4ometry.geant4.Registry), 321
- AnalyseGeometryStructure() (in module pyg4ometry.geant4), 348
- AnalyseGeometryStructure() (in module pyg4ometry.geant4.Registry), 322
- angleBetween() (pyg4ometry.features.algos.Plane method), 96
- angleBetween() (pyg4ometry.features.Plane method), 99
- append() (pyg4ometry.fluka.fluka_registry.BaseCacher method), 141
- append() (pyg4ometry.fluka.fluka_registry.HalfSpaceCacher method), 141
- append() (pyg4ometry.fluka.fluka_registry.InfiniteCylinderCacher method), 142
- appendData() (pyg4ometry.fluka.fluka_registry.BaseCacher method), 141
- ARB (class in pyg4ometry.convert), 87
- ARB (class in pyg4ometry.fluka), 163
- ARB (class in pyg4ometry.fluka.body), 125
- are_anti_parallel() (in module pyg4ometry.geant4.solid), 277, 281, 284, 308
- are_anti_parallel() (in module pyg4ometry.transformation), 424
- are_parallel() (in module pyg4ometry.geant4.solid), 276, 280, 284, 307
- are_parallel() (in module pyg4ometry.transformation), 423
- area() (pyg4ometry.pycgal.core.CSG method), 372
- area() (pyg4ometry.pycgal.CSG method), 375
- areAABBsOverlapping() (in module pyg4ometry.fluka.vector), 157
- areAllTheSameTransformationIndices() (pyg4ometry.fluka.directive.RecursiveRotoTranslation method), 136
- areAllTheSameTransformationIndices() (pyg4ometry.fluka.RecursiveRotoTranslation method), 178
- areParallelOrAntiParallel() (in module pyg4ometry.fluka.vector), 157
- ASIN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpression attribute), 183
- ASIN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpression attribute), 193
- asin() (in module pyg4ometry.gdml), 220
- asin() (in module pyg4ometry.gdml.Defines), 200
- ASIN() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpression method), 191
- AssemblyVolume (class in pyg4ometry.geant4), 336
- AssemblyVolume (class in module pyg4ometry.geant4.AssemblyVolume), 310
- assemblyVolume() (pyg4ometry.geant4.LogicalVolume method), 335, 344
- assemblyVolume() (pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 315
- assemblyVolumes() (in module pyg4ometry.compare), 66
- assemblyVolumes() (in module pyg4ometry.compare._Compare), 60
- assignma() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
- assignma() (pyg4ometry.fluka.FlukaRegistry method),

- 170
- ATAN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 183
- ATAN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- atan() (in module pyg4ometry.gdml), 220
- atan() (in module pyg4ometry.gdml.Defines), 200
- ATAN() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 191
- atn (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 118
- atn (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer method), 104
- atn (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 100
- atn (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
- atn (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 114
- atn (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 107
- atn (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 183
- atn (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- atom() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 194
- atom() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 188
- atomicMassFromZ() (pyg4ometry.convert.fluka2g4materials.PeriodicTable method), 72
- atomicNumberAndMassFromSymbol() (pyg4ometry.convert.fluka2g4materials.PeriodicTable method), 72
- Auxiliary (class in pyg4ometry.gdml), 224
- Auxiliary (class in pyg4ometry.gdml.Defines), 204
- axesFromExtents() (in module pyg4ometry.visualisation), 411
- axesFromExtents() (in module pyg4ometry.visualisation.VtkViewer), 400
- axisangle2matrix() (in module pyg4ometry.geant4.solid), 275, 279, 283, 306
- axisangle2matrix() (in module pyg4ometry.transformation), 422
- axisangle2tbxyz() (in module pyg4ometry.geant4.solid), 276, 279, 283, 307
- axisangle2tbxyz() (in module pyg4ometry.transformation), 422
- B**
- backendName() (in module pyg4ometry.config), 418
- Bar (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 119
- Bar (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
- Bar (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 113
- Bar() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 113
- Bar() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 113
- Bar() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 104
- Bar() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 104
- Bar() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 104
- Bar() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 104
- BaseCacher (class in pyg4ometry.fluka.fluka_registry),
- BasicExpression (class in pyg4ometry.gdml), 217
- BasicExpression (class in pyg4ometry.gdml.Defines), 197
- bdsimBeam() (pyg4ometry.montecarlo.Beam method), 370
- bdsimBeam() (pyg4ometry.montecarlo.Beam method), 368
- bdsimBoundary() (pyg4ometry.montecarlo.Boundary method), 370
- bdsimBoundary() (pyg4ometry.montecarlo.boundary.Boundary method), 369
- bdsimBoundary() (pyg4ometry.montecarlo.Config method), 370
- bdsimBoundary() (pyg4ometry.montecarlo.config.Config method), 369
- bdsimBoundary() (pyg4ometry.montecarlo.Scoring method), 370
- bdsimBoundary() (pyg4ometry.montecarlo.scoring.Scoring method), 369
- Beam (class in pyg4ometry.bdsim), 55
- Beam (class in pyg4ometry.bdsim.beam), 52
- Beam (class in pyg4ometry.montecarlo), 370
- Beam (class in pyg4ometry.montecarlo.beam), 368
- Beamline (class in pyg4ometry.bdsim), 55
- Beamline (class in pyg4ometry.bdsim.beamline), 52
- beamPipe() (in module pyg4ometry.features._accelerator), 94
- beamPipeCADFeature() (in module pyg4ometry.features._accelerator), 94
- binToVtkGrid() (pyg4ometry.analysis.flukaData.FlukaBinData method), 51
- bodies() (pyg4ometry.fluka.Region method), 175
- bodies() (pyg4ometry.fluka.region.Region method), 154
- bodies() (pyg4ometry.fluka.region.Zone method), 153
- bodies() (pyg4ometry.fluka.Zone method), 174
- BodyName (pyg4ometry.fluka.RegionExpression.RegionLexer

- attribute*), 118
- BodyName (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer method), 412
- attribute*), 101
- BodyName (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 401
- attribute*), 118
- buildPipelines (pyg4ometry.visualisation.VtkViewerNew method), 412
- BodyName (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 108
- buildPipelinesAppend (pyg4ometry.visualisation.VtkViewerNew method), 412
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 107
- buildPipelinesAppend (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 412
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 107
- buildPipelinesSeparate (pyg4ometry.visualisation.VtkViewerNew method), 412
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 105
- buildPipelinesSeparate (pyg4ometry.visualisation.VtkViewerNew method), 412
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 105
- buildPipelinesSeparate (pyg4ometry.visualisation.VtkViewerNew method), 412
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 105
- buildPipelinesTransformed (pyg4ometry.visualisation.VtkViewerNew method), 412
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 116
- buildPipelinesTransformed (pyg4ometry.visualisation.VtkViewerNew method), 412
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 117
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 115
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 114
- BodyName () (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 115
- Cacheable (class in pyg4ometry.fluka.fluka_registry), 141
- BorderSurface (class in pyg4ometry.geant4), 345
- BorderSurface (class in pyg4ometry.geant4.BorderSurface), 310
- Boundary (class in pyg4ometry.montecarlo), 370
- Boundary (class in pyg4ometry.montecarlo.boundary), 369
- BOX (class in pyg4ometry.convert), 83
- BOX (class in pyg4ometry.fluka), 160
- BOX (class in pyg4ometry.fluka.body), 122
- Box (class in pyg4ometry.geant4.solid), 263
- Box (class in pyg4ometry.geant4.solid.Box), 225
- bracket_depth () (in module pyg4ometry.fluka), 176
- bracket_depth () (in module pyg4ometry.fluka.region), 151
- bracket_number () (in module pyg4ometry.fluka), 177
- bracket_number () (in module pyg4ometry.fluka.region), 151
- buildCgalPolygons () (pyg4ometry.fluka.Extruder method), 180
- buildCgalPolygons () (pyg4ometry.fluka.extruder.Extruder method), 137
- buildGeant4Extrusions () (pyg4ometry.fluka.Extruder method), 180
- buildGeant4Extrusions () (pyg4ometry.fluka.extruder.Extruder method), 137
- buildPipelines () (pyg4ometry.visualisation.VtkViewerNew method), 412
- buildPipelines () (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 401
- buildPipelinesAppend () (pyg4ometry.visualisation.VtkViewerNew method), 412
- buildPipelinesSeparate () (pyg4ometry.visualisation.VtkViewerNew method), 412
- buildPipelinesTransformed () (pyg4ometry.visualisation.VtkViewerNew method), 412
- BuiltIn (class in pyg4ometry.fluka), 178
- BuildZoneExprComp (pyg4ometry.fluka.material), 144
- Card (class in pyg4ometry.fluka), 179
- Card (class in pyg4ometry.fluka.card), 133
- centre () (pyg4ometry.convert.ARB method), 87
- centre () (pyg4ometry.convert.BOX method), 84
- centre () (pyg4ometry.convert.ELL method), 86
- centre () (pyg4ometry.convert.QUA method), 92
- centre () (pyg4ometry.convert.RCC method), 85
- centre () (pyg4ometry.convert.REC method), 85
- centre () (pyg4ometry.convert.RPP method), 83
- centre () (pyg4ometry.convert.SPH method), 84
- centre () (pyg4ometry.convert.TRC method), 86
- centre () (pyg4ometry.convert.XCC method), 89
- centre () (pyg4ometry.convert.XEC method), 91
- centre () (pyg4ometry.convert.YCC method), 90
- centre () (pyg4ometry.convert.YEC method), 91
- centre () (pyg4ometry.convert.ZCC method), 90
- centre () (pyg4ometry.convert.ZEC method), 92
- centre () (pyg4ometry.fluka.ARB method), 163
- centre () (pyg4ometry.fluka.body.ARB method), 125
- centre () (pyg4ometry.fluka.body.BOX method), 122
- centre () (pyg4ometry.fluka.body.ELL method), 124
- centre () (pyg4ometry.fluka.body.QUA method), 131
- centre () (pyg4ometry.fluka.body.RCC method), 123
- centre () (pyg4ometry.fluka.body.REC method), 123
- centre () (pyg4ometry.fluka.body.RPP method), 121
- centre () (pyg4ometry.fluka.body.SPH method), 122

- centre() (pyg4ometry.fluka.body.TRC method), 124
- centre() (pyg4ometry.fluka.body.XCC method), 127
- centre() (pyg4ometry.fluka.body.XEC method), 129
- centre() (pyg4ometry.fluka.body.YCC method), 128
- centre() (pyg4ometry.fluka.body.YEC method), 129
- centre() (pyg4ometry.fluka.body.ZCC method), 128
- centre() (pyg4ometry.fluka.body.ZEC method), 130
- centre() (pyg4ometry.fluka.BOX method), 160
- centre() (pyg4ometry.fluka.ELL method), 162
- centre() (pyg4ometry.fluka.QUA method), 169
- centre() (pyg4ometry.fluka.RCC method), 161
- centre() (pyg4ometry.fluka.REC method), 161
- centre() (pyg4ometry.fluka.Region method), 175
- centre() (pyg4ometry.fluka.region.Region method), 154
- centre() (pyg4ometry.fluka.region.Zone method), 152
- centre() (pyg4ometry.fluka.RPP method), 159
- centre() (pyg4ometry.fluka.SPH method), 160
- centre() (pyg4ometry.fluka.TRC method), 162
- centre() (pyg4ometry.fluka.XCC method), 165
- centre() (pyg4ometry.fluka.XEC method), 167
- centre() (pyg4ometry.fluka.YCC method), 166
- centre() (pyg4ometry.fluka.YEC method), 167
- centre() (pyg4ometry.fluka.ZCC method), 166
- centre() (pyg4ometry.fluka.ZEC method), 168
- centre() (pyg4ometry.fluka.Zone method), 174
- cgal_np (pyg4ometry.config.meshingType attribute), 418
- cgal_sm (pyg4ometry.config.meshingType attribute), 418
- changeSolidAndTrimGeometry()
(pyg4ometry.geant4.LogicalVolume method), 334, 342
- changeSolidAndTrimGeometry()
(pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 314
- channelNames (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 119
- channelNames (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
- channelNames (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLex attribute), 184
- checkAxis() (pyg4ometry.geant4.DivisionVolume method), 340
- checkAxis() (pyg4ometry.geant4.DivisionVolume.DivisionVolume method), 311
- checkBodyName() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 140
- checkBodyName() (pyg4ometry.fluka.FlukaRegistry method), 172
- checkDefineName() (pyg4ometry.gdml.Writer method), 215
- checkDefineName() (pyg4ometry.gdml.Writer.Writer method), 208
- checkLogicalVolumeName() (pyg4ometry.gdml.Writer method), 215
- checkLogicalVolumeName()
(pyg4ometry.gdml.Writer.Writer method), 208
- checkMaterialName()
(pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 140
- checkMaterialName()
(pyg4ometry.fluka.FlukaRegistry method), 172
- checkMaterialName() (pyg4ometry.gdml.Writer method), 215
- checkMaterialName() (pyg4ometry.gdml.Writer.Writer method), 208
- checkOverlaps() (pyg4ometry.geant4.LogicalVolume method), 334, 343
- checkOverlaps() (pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 314
- checkParameters() (pyg4ometry.geant4.solid._GenericPolyhedra method), 270, 271, 301
- checkParameters() (pyg4ometry.geant4.solid.Cons method), 267
- checkParameters() (pyg4ometry.geant4.solid.Cons.Cons method), 226
- checkParameters() (pyg4ometry.geant4.solid.GenericPolycone method), 301
- checkParameters() (pyg4ometry.geant4.solid.GenericPolycone.GenericPolycone method), 231
- checkParameters() (pyg4ometry.geant4.solid.GenericPolyhedra method), 302
- checkParameters() (pyg4ometry.geant4.solid.GenericPolyhedra.GenericPolyhedra method), 232
- checkParameters() (pyg4ometry.geant4.solid.Hype method), 292
- checkParameters() (pyg4ometry.geant4.solid.Hype.Hype method), 266
- checkParameters() (pyg4ometry.geant4.solid.Sphere method), 266
- checkParameters() (pyg4ometry.geant4.solid.Sphere.Sphere method), 266
- checkParameters() (pyg4ometry.geant4.solid.TwistedBox method), 295
- checkParameters() (pyg4ometry.geant4.solid.TwistedBox.TwistedBox method), 295
- checkParameters() (pyg4ometry.geant4.solid.TwistedTrap method), 297
- checkParameters() (pyg4ometry.geant4.solid.TwistedTrap.TwistedTrap method), 252
- checkParameters() (pyg4ometry.geant4.solid.TwistedTrd method), 299
- checkParameters() (pyg4ometry.geant4.solid.TwistedTrd.TwistedTrd method), 253
- checkPhysicalVolumeName()
(pyg4ometry.gdml.Writer method), 215
- checkPhysicalVolumeName()
(pyg4ometry.gdml.Writer.Writer method),

- 208
- checkRegionName() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 140
- checkRegionName() (pyg4ometry.fluka.FlukaRegistry method), 172
- checkSolidName() (pyg4ometry.gdml.Writer method), 215
- checkSolidName() (pyg4ometry.gdml.Writer.Writer method), 208
- clear() (pyg4ometry.geant4.Registry method), 345
- clear() (pyg4ometry.geant4.Registry.Registry method), 318
- clear() (pyg4ometry.visualisation.ViewerBase method), 405
- clear() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 390
- clear() (pyg4ometry.visualisation.VtkViewerNew method), 412
- clear() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 401
- cli() (in module pyg4ometry.cli), 417
- clipGeometry() (pyg4ometry.geant4.AssemblyVolume method), 336
- clipGeometry() (pyg4ometry.geant4.AssemblyVolume.AssemblyVolume method), 310
- clipGeometry() (pyg4ometry.geant4.LogicalVolume method), 333, 342
- clipGeometry() (pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 313
- clipSolid() (pyg4ometry.geant4.LogicalVolume method), 335, 343
- clipSolid() (pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 315
- clone() (pyg4ometry.pycgal.core.CSG method), 371
- clone() (pyg4ometry.pycgal.CSG method), 374
- closeEvent() (pyg4ometry.gui.example1.QVTKRenderWindow method), 358
- closeEvent() (pyg4ometry.gui.QVTKRenderWindowInteractive method), 361, 364
- closeEvent() (pyg4ometry.gui.QVTKRenderWindowInteractive method), 355
- coefficientsMatrix() (pyg4ometry.convert.QUA method), 92
- coefficientsMatrix() (pyg4ometry.fluka.body.QUA method), 131
- coefficientsMatrix() (pyg4ometry.fluka.QUA method), 169
- COLUMNS (pyg4ometry.fluka.fluka_registry.BaseCacher attribute), 141
- COLUMNS (pyg4ometry.fluka.fluka_registry.HalfSpaceCacher attribute), 141
- COLUMNS (pyg4ometry.fluka.fluka_registry.InfiniteCylinderCacher attribute), 142
- COMMA (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 184
- COMMA (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- COMMA() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 190
- COMMA() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 189
- compareFilesWithAssert() (in module pyg4ometry.misc), 368
- compareFilesWithAssert() (in module pyg4ometry.misc.TestUtils), 367
- compareGdmlNumericallyWithAssert() (in module pyg4ometry.misc), 368
- compareGdmlNumericallyWithAssert() (in module pyg4ometry.misc.TestUtils), 367
- compareMeshInfo() (in module pyg4ometry.misc), 368
- compareMeshInfo() (in module pyg4ometry.misc.TestUtils), 367
- compareNumericallyWithAssert() (in module pyg4ometry.misc), 368
- compareNumericallyWithAssert() (in module pyg4ometry.misc.TestUtils), 367
- ComparisonResult (class in pyg4ometry.compare), 65
- CompareWilsonResult (class in pyg4ometry.compare._Compare), 58
- Compound (class in pyg4ometry.fluka), 179
- Compound (class in pyg4ometry.fluka.material), 145
- Config (class in pyg4ometry.montecarlo), 370
- Config (class in pyg4ometry.montecarlo.config), 369
- connectedZones() (pyg4ometry.fluka.Region method), 176
- connectedZones() (pyg4ometry.fluka.region.Region method), 154
- Cons (class in pyg4ometry.geant4.solid), 266
- Cons (class in pyg4ometry.geant4.solid.Cons), 226
- Constant (class in pyg4ometry.gdml), 221
- Constant (class in pyg4ometry.gdml.Defines), 201
- constant() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 194
- constant() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 188
- convert2Tessellated() (pyg4ometry.geant4.solid._SolidBase method), 261–270, 272–274, 277, 281, 285–291, 293, 295, 297, 299, 300, 302, 304, 305, 308
- convert2Tessellated() (pyg4ometry.geant4.solid.SolidBase method), 309
- convert2Tessellated() (pyg4ometry.geant4.solid.SolidBase.SolidBase method), 242
- convertAll() (pyg4ometry.convert.fluka2g4materials._FlukaToG4Materials method), 72
- convertAtomFractionToCompound() (pyg4ometry.convert.fluka2g4materials._FlukaToG4Materials method), 72

(pyg4ometry.convert.fluka2g4materials._FlukaToG4MaterialConverter
 method), 72
 convertBuiltin() (pyg4ometry.convert.fluka2g4material.coplanarToIntersection
 method), 72
 convertCompound() (pyg4ometry.convert.fluka2g4material.coplanarToIntersection
 method), 72
 convertElement() (pyg4ometry.convert.fluka2g4material.coplanarToIntersection
 method), 72
 convertFlat() (pyg4ometry.freecad.Reader.Reader
 method), 181
 convertMassFractionCompound() (pyg4ometry.convert.fluka2g4materials._FlukaToG4MaterialConverter
 method), 72
 convertMeshToPolyTriangulation() (in module pyg4ometry.convert), 94
 convertMeshToPolyTriangulation() (in module pyg4ometry.convert.vis2oce), 78
 convertMeshToShape() (in module pyg4ometry.convert), 94
 convertMeshToShape() (in module pyg4ometry.convert.vis2oce), 79
 convertMeshToShapeUsingMakeShapeOnMesh() (in module pyg4ometry.convert), 94
 convertMeshToShapeUsingMakeShapeOnMesh() (in module pyg4ometry.convert.vis2oce), 78
 convertStructure() (pyg4ometry.freecad.Reader.Reader
 method), 181
 convertToDNF() (pyg4ometry.fluka.Region method), 175
 convertToDNF() (pyg4ometry.fluka.region.Region
 method), 154
 convertToDNF() (pyg4ometry.fluka.region.Zone
 method), 152
 convertToDNF() (pyg4ometry.fluka.Zone method), 174
 convertVolumeFractionCompound() (pyg4ometry.convert.fluka2g4materials._FlukaToG4MaterialConverter
 method), 72
 convexpolyhedron_to_planes (in module pyg4ometry.pycgal_old), 380
 convexpolyhedron_to_planes (in module pyg4ometry.pycgal_old.cgal), 378
 CoordinateSystem (class in pyg4ometry.features), 99
 CoordinateSystem (class in pyg4ometry.features.algos), 96
 coordinateSystem() (pyg4ometry.features.algos.CoordinateSystem(pyg4ometry.geant4.DivisionVolume
 method), 97
 coordinateSystem() (pyg4ometry.features.CoordinateSystem
 method), 99
 coplanar (pyg4ometry.geant4._OverlapType attribute), 332, 337
 coplanar (pyg4ometry.visualisation.Mesh.OverlapType
 attribute), 388
 coplanar (pyg4ometry.visualisation.OverlapType
 attribute), 405
 coplanarIntersection() (pyg4ometry.pycgal.core.CSG method), 371
 coplanarIntersection() (pyg4ometry.pycgal.CSG
 method), 375
 coplanarToIntersection() (pyg4ometry.fluka.AABB
 method), 173
 coplanarToIntersection() (pyg4ometry.fluka.vector.AABB
 method), 157
 copyFrom() (pyg4ometry.fluka.RegionExpression.RegionParser.ExprConte
 method), 115
 copyFrom() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionCon
 method), 112
 copyFrom() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionPar
 method), 105
 copyFrom() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionPar
 method), 103
 copyFrom() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionPar
 method), 105
 copyFrom() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionPar
 method), 103
 copyFrom() (pyg4ometry.fluka.RegionExpression.RegionParser.ZoneConte
 method), 114
 copyFrom() (pyg4ometry.fluka.RegionExpression.RegionParser.ZoneUnion
 method), 113
 cornerDistance() (pyg4ometry.fluka.AABB method), 173
 cornerDistance() (pyg4ometry.fluka.vector.AABB
 method), 157
 COS (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpres
 attribute), 183
 COS (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpres
 attribute), 192
 cos() (in module pyg4ometry.gdml), 219
 cos() (in module pyg4ometry.gdml.Defines), 199
 COS() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExp
 method), 191
 countVisibleDaughters() (pyg4ometry.visualisation.VtkExporter
 method), 416
 countVisibleDaughters() (pyg4ometry.visualisation.VtkExporter.VtkExporter
 method), 395
 createDivisionMeshes() (pyg4ometry.geant4.DivisionVolume.DivisionVolume
 method), 312
 createNewRegistry() (pyg4ometry.gui.GeometryModel method), 362, 364
 createNewRegistry() (pyg4ometry.gui.GeometryModel.GeometryModel

- method), 352
- createParameterisedMeshes()
(pyg4ometry.geant4.ParameterisedVolume
method), 339
- createParameterisedMeshes()
(pyg4ometry.geant4.ParameterisedVolume.ParameterisedVolume
method), 317
- createPosition() (pyg4ometry.gdml.Writer method),
217
- createPosition() (pyg4ometry.gdml.Writer.Writer
method), 209
- createQuadrangularFacet()
(pyg4ometry.gdml.Writer method), 216
- createQuadrangularFacet()
(pyg4ometry.gdml.Writer.Writer method),
209
- createReplicaMeshes()
(pyg4ometry.geant4._ReplicaVolume method),
338
- createReplicaMeshes()
(pyg4ometry.geant4.ReplicaVolume method),
338
- createReplicaMeshes()
(pyg4ometry.geant4.ReplicaVolume.ReplicaVolume
method), 322
- createrzPoint() (pyg4ometry.gdml.Writer method),
216
- createrzPoint() (pyg4ometry.gdml.Writer.Writer
method), 209
- createSection() (pyg4ometry.gdml.Writer method),
216
- createSection() (pyg4ometry.gdml.Writer.Writer
method), 209
- createTessellatedSolid() (in module
pyg4ometry.geant4.solid), 304
- createTessellatedSolid() (in module
pyg4ometry.geant4.solid.TessellatedSolid),
245
- CreateTimer() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor
method), 358
- CreateTimer() (pyg4ometry.gui.QVTKRenderWindowInteractor
method), 361, 363
- CreateTimer() (pyg4ometry.gui.QVTKRenderWindowInteractor
method), 355
- createTriangularFacet() (pyg4ometry.gdml.Writer
method), 216
- createTriangularFacet()
(pyg4ometry.gdml.Writer.Writer method),
209
- createTwoDimVertex() (pyg4ometry.gdml.Writer
method), 216
- createTwoDimVertex()
(pyg4ometry.gdml.Writer.Writer method),
209
- createzPlane() (pyg4ometry.gdml.Writer method),
216
- createzPlane() (pyg4ometry.gdml.Writer.Writer
method), 209
- cross() (pyg4ometry.fluka.Three method), 173
- cross() (pyg4ometry.fluka.vector.Three method), 156
- cross() (pyg4ometry.geant4.solid._TwoVector method),
294, 296, 298
- cross() (pyg4ometry.geant4.solid.TwoVector method),
261
- cross() (pyg4ometry.geant4.solid.TwoVector.TwoVector
method), 255
- CSG (class in pyg4ometry.pycgal), 374
- CSG (class in pyg4ometry.pycgal.core), 371
- cube() (pyg4ometry.pycgal.core.CSG class method), 372
- cube() (pyg4ometry.pycgal.CSG class method), 375
- cubeNet() (in module pyg4ometry.geant4.solid), 263
- cubeNet() (in module pyg4ometry.geant4.solid.Box),
225
- cullDaughtersOutsideSolid()
(pyg4ometry.geant4.LogicalVolume method),
332, 341
- cullDaughtersOutsideSolid()
(pyg4ometry.geant4.LogicalVolume.LogicalVolume
method), 313
- CursorChangedEvent()
(pyg4ometry.gui.example1.QVTKRenderWindowInteractor
method), 358
- CursorChangedEvent()
(pyg4ometry.gui.QVTKRenderWindowInteractor
method), 361, 363
- CursorChangedEvent()
(pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor
method), 355
- CutTubs (class in pyg4ometry.geant4.solid), 264
- CutTubs (class in pyg4ometry.geant4.solid.CutTubs), 227
- ## D
- debugDumpFile() (in module
pyg4ometry.analysis.flukaData), 51
- decisionsToDFA (pyg4ometry.fluka.RegionExpression.RegionLexer
attribute), 118
- decisionsToDFA (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer
attribute), 100
- decisionsToDFA (pyg4ometry.fluka.RegionExpression.RegionParser
attribute), 117
- decisionsToDFA (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser
attribute), 107
- decisionsToDFA (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer
attribute), 183
- decisionsToDFA (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser
attribute), 192
- decomposePolygon2d()
(pyg4ometry.convert._PolygonProcessing

- class method*), 82
- decomposePolygon2d()
 - (pyg4ometry.pycgal.core.PolygonProcessing *class method*), 373
- decomposePolygon2d()
 - (pyg4ometry.pycgal.PolygonProcessing *class method*), 376
- decomposePolygon2dWithHoles()
 - (pyg4ometry.convert._PolygonProcessing *class method*), 82
- decomposePolygon2dWithHoles()
 - (pyg4ometry.pycgal.core.PolygonProcessing *class method*), 373
- decomposePolygon2dWithHoles()
 - (pyg4ometry.pycgal.PolygonProcessing *class method*), 376
- defaultLinDef (in module pyg4ometry.convert), 93
- defaultLinDef (in module pyg4ometry.convert.oce2Geant4), 77
- DefineBase (class in pyg4ometry.gdml), 218
- DefineBase (class in pyg4ometry.gdml.Defines), 198
- defineBuiltInFlukaMaterials() (in module pyg4ometry.fluka.material), 144
- defaulAngDef (in module pyg4ometry.convert), 93
- defaulAngDef (in module pyg4ometry.convert.oce2Geant4), 77
- deg2rad() (in module pyg4ometry.geant4.solid), 274, 278, 282, 305
- deg2rad() (in module pyg4ometry.transformation), 421
- delete_nefpolyhedron (in module pyg4ometry.pycgal_old), 381
- delete_nefpolyhedron (in module pyg4ometry.pycgal_old.cgal), 379
- delete_polygon (in module pyg4ometry.pycgal_old), 381
- delete_polygon (in module pyg4ometry.pycgal_old.cgal), 379
- delete_polygonlist (in module pyg4ometry.pycgal_old), 381
- delete_polygonlist (in module pyg4ometry.pycgal_old.cgal), 379
- delete_polyhedron (in module pyg4ometry.pycgal_old), 381
- delete_polyhedron (in module pyg4ometry.pycgal_old.cgal), 379
- delete_surfacemesh (in module pyg4ometry.pycgal_old), 381
- delete_surfacemesh (in module pyg4ometry.pycgal_old.cgal), 379
- depth() (pyg4ometry.geant4.AssemblyVolume *method*), 336
- depth() (pyg4ometry.geant4.AssemblyVolume.AssemblyVolume *method*), 310
- depth() (pyg4ometry.geant4.LogicalVolume *method*), 335, 343
- depth() (pyg4ometry.geant4.LogicalVolume.LogicalVolume *method*), 315
- DestroyTimer() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor *method*), 358
- DestroyTimer() (pyg4ometry.gui.QVTKRenderWindowInteractor *method*), 361, 363
- DestroyTimer() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor *method*), 355
- diffFiles() (in module pyg4ometry.misc), 368
- diffFiles() (in module pyg4ometry.misc.TestUtils), 367
- direction() (pyg4ometry.convert.XCC *method*), 89
- direction() (pyg4ometry.convert.YCC *method*), 90
- direction() (pyg4ometry.convert.ZCC *method*), 90
- direction() (pyg4ometry.fluka.body.XCC *method*), 128
- direction() (pyg4ometry.fluka.body.YCC *method*), 128
- direction() (pyg4ometry.fluka.body.ZCC *method*), 129
- direction() (pyg4ometry.fluka.XCC *method*), 166
- direction() (pyg4ometry.fluka.YCC *method*), 166
- direction() (pyg4ometry.fluka.ZCC *method*), 167
- DIV (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer *attribute*), 184
- DIV (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser *attribute*), 193
- DIV() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser *method*), 187
- divideBox() (pyg4ometry.geant4.DivisionVolume *method*), 340
- divideBox() (pyg4ometry.geant4.DivisionVolume.DivisionVolume *method*), 311
- divideCons() (pyg4ometry.geant4.DivisionVolume *method*), 340
- divideCons() (pyg4ometry.geant4.DivisionVolume.DivisionVolume *method*), 311
- dividePara() (pyg4ometry.geant4.DivisionVolume *method*), 341
- dividePara() (pyg4ometry.geant4.DivisionVolume.DivisionVolume *method*), 312
- dividePolycone() (pyg4ometry.geant4.DivisionVolume *method*), 341
- dividePolycone() (pyg4ometry.geant4.DivisionVolume.DivisionVolume *method*), 312
- dividePolyhedra() (pyg4ometry.geant4.DivisionVolume *method*), 341
- dividePolyhedra() (pyg4ometry.geant4.DivisionVolume.DivisionVolume *method*), 312
- divideTrd() (pyg4ometry.geant4.DivisionVolume *method*), 341
- divideTrd() (pyg4ometry.geant4.DivisionVolume.DivisionVolume *method*), 312
- divideTubs() (pyg4ometry.geant4.DivisionVolume *method*), 340
- divideTubs() (pyg4ometry.geant4.DivisionVolume.DivisionVolume *method*), 340

method), 311
 DivisionVolume (class in *pyg4ometry.geant4*), 340
 DivisionVolume (class in *pyg4ometry.geant4.DivisionVolume*), 311
 DivisionVolume.Axis (class in *pyg4ometry.geant4*), 340
 DivisionVolume.Axis (class in *pyg4ometry.geant4.DivisionVolume*), 311
 divisionVolumes() (in module *pyg4ometry.compare*), 67
 divisionVolumes() (in module *pyg4ometry.compare._Compare*), 60
 do_intersect() (in module *pyg4ometry.pycgal*), 375
 do_intersect() (in module *pyg4ometry.pycgal.core*), 372
 doMeshing (in module *pyg4ometry.config*), 418
 dot() (*pyg4ometry.fluka.Three* method), 172
 dot() (*pyg4ometry.fluka.vector.Three* method), 156
 dot() (*pyg4ometry.geant4.solid._TwoVector* method), 294, 296, 298
 dot() (*pyg4ometry.geant4.solid.TwoVector* method), 261
 dot() (*pyg4ometry.geant4.solid.TwoVector.TwoVector* method), 255
 draw_polygon_2() (in module *pyg4ometry.pycgal.pythonHelpers*), 374
 draw_polygon_2_list() (in module *pyg4ometry.pycgal.pythonHelpers*), 374
 dump() (*pyg4ometry.stl._Facet* method), 385
 dump() (*pyg4ometry.stl.Reader._Facet* method), 384
 DumpGeometryStructureTree() (in module *pyg4ometry.geant4*), 348
 DumpGeometryStructureTree() (in module *pyg4ometry.geant4.Registry*), 322
 dumpMeshQuality() (*pyg4ometry.visualisation.ViewerBase* method), 407
 dumpMeshQuality() (*pyg4ometry.visualisation.ViewerBase.ViewerBase* method), 391
 dumps() (*pyg4ometry.fluka.Region* method), 176
 dumps() (*pyg4ometry.fluka.region.Region* method), 154
 dumps() (*pyg4ometry.fluka.region.Zone* method), 153
 dumps() (*pyg4ometry.fluka.Zone* method), 174
 dumpsDebug() (*pyg4ometry.fluka.region.Zone* method), 153
 dumpsDebug() (*pyg4ometry.fluka.Zone* method), 174
 dumpStructure() (*pyg4ometry.geant4.AssemblyVolume* method), 336
 dumpStructure() (*pyg4ometry.geant4.AssemblyVolume.AssemblyVolume* method), 310
 dumpStructure() (*pyg4ometry.geant4.LogicalVolume* method), 335, 344
 dumpStructure() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume* method), 315

E

elcfield (class in *pyg4ometry.fluka.elcfield*), 136
 Element (class in *pyg4ometry.bdsim*), 55
 Element (class in *pyg4ometry.bdsim.element*), 53
 Element (class in *pyg4ometry.geant4*), 351
 Element (class in *pyg4ometry.geant4._Material*), 328
 ElementIsotopeMixture() (in module *pyg4ometry.geant4*), 349
 ElementIsotopeMixture() (in module *pyg4ometry.geant4._Material*), 327
 ElementSimple() (in module *pyg4ometry.geant4*), 349
 ElementSimple() (in module *pyg4ometry.geant4._Material*), 326
 ELL (class in *pyg4ometry.convert*), 86
 ELL (class in *pyg4ometry.fluka*), 162
 ELL (class in *pyg4ometry.fluka.body*), 124
 Ellipsoid (class in *pyg4ometry.geant4.solid*), 267
 Ellipsoid (class in *pyg4ometry.geant4.solid.Ellipsoid*), 227
 EllipticalCone (class in *pyg4ometry.geant4.solid*), 290
 EllipticalCone (class in *pyg4ometry.geant4.solid.EllipticalCone*), 228
 EllipticalTube (class in *pyg4ometry.geant4.solid*), 289
 EllipticalTube (class in *pyg4ometry.geant4.solid.EllipticalTube*), 229
 enterEvent() (*pyg4ometry.gui.example1.QVTKRenderWindowInteractor* method), 358
 enterEvent() (*pyg4ometry.gui.QVTKRenderWindowInteractor* method), 361, 364
 enterEvent() (*pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor* method), 355
 envelops() (*pyg4ometry.fluka.AABB* method), 173
 envelops() (*pyg4ometry.fluka.vector.AABB* method), 157
 EOF (*pyg4ometry.fluka.RegionExpression.RegionParser* attribute), 117
 EOF (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser* attribute), 108
 EOF (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* attribute), 192
 EQ (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer* attribute), 184
 EQ (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* attribute), 193
 EQ() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* method), 191
 equation() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* method), 193
 EULER (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer* attribute), 184

- EULER (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- EULER() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.ConstantContext method), 189
- eval() (pyg4ometry.features.algos.Line method), 95
- eval() (pyg4ometry.gdml.BasicExpression method), 217
- eval() (pyg4ometry.gdml.Defines.BasicExpression method), 197
- eval() (pyg4ometry.gdml.Defines.Matrix method), 204
- eval() (pyg4ometry.gdml.Defines.Quantity method), 202
- eval() (pyg4ometry.gdml.Defines.ScalarBase method), 199
- eval() (pyg4ometry.gdml.Defines.VectorBase method), 203
- eval() (pyg4ometry.gdml.Matrix method), 224
- eval() (pyg4ometry.gdml.Quantity method), 222
- eval() (pyg4ometry.gdml.ScalarBase method), 219
- eval() (pyg4ometry.gdml.VectorBase method), 223
- evaluate() (pyg4ometry.fluka.preprocessor._Calc method), 148
- evaluate() (pyg4ometry.gdml.GdmlExpression.ExpressionParser method), 412
- evaluate() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionEvaluator.ExpressionParser method), 183
- evaluateParameter() (pyg4ometry.geant4.solid._SolidBase method), 261–270, 272, 273, 277, 281, 285–293, 295, 297, 299–303, 305, 308
- evaluateParameter() (pyg4ometry.geant4.solid.SolidBase method), 309
- evaluateParameter() (pyg4ometry.geant4.solid.SolidBase.SolidBase method), 242
- evaluateParameterWithUnits() (pyg4ometry.geant4.solid._SolidBase method), 261–270, 272, 273, 277, 281, 285–293, 295, 297, 299–303, 305, 308
- evaluateParameterWithUnits() (pyg4ometry.geant4.solid.ExtrudedSolid method), 273
- evaluateParameterWithUnits() (pyg4ometry.geant4.solid.ExtrudedSolid.ExtrudedSolid method), 230
- evaluateParameterWithUnits() (pyg4ometry.geant4.solid.SolidBase method), 309
- evaluateParameterWithUnits() (pyg4ometry.geant4.solid.SolidBase.SolidBase method), 242
- evaluateToFloat() (in module pyg4ometry.gdml), 218
- evaluateToFloat() (in module pyg4ometry.gdml.Defines), 198
- EXP (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 184
- EXP (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.ConstantContext method), 189
- exp() (in module pyg4ometry.gdml), 220
- exp() (in module pyg4ometry.gdml.Defines), 200
- EXP() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 191
- expandLoop() (pyg4ometry.geant4.Loop.Loop method), 316
- export_to_Paraview() (pyg4ometry.visualisation.VtkExporter method), 414
- export_to_Paraview() (pyg4ometry.visualisation.VtkExporter.VtkExporter method), 393
- export_to_VTK() (pyg4ometry.visualisation.VtkExporter method), 414
- export_to_VTK() (pyg4ometry.visualisation.VtkExporter.VtkExporter method), 394
- exportCutter() (pyg4ometry.visualisation.VtkViewerNew method), 410
- exportCutter() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 410
- exportCutterSection() (pyg4ometry.visualisation.VtkViewer method), 410
- exportCutterSection() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 398
- exportGLTFAssets() (pyg4ometry.visualisation.ViewerBase method), 407
- exportGLTFAssets() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 391
- exportGLTFScene() (pyg4ometry.visualisation.ViewerBase method), 407
- exportGLTFScene() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 391
- exportGLTFScene() (pyg4ometry.visualisation.VtkViewer method), 409
- exportGLTFScene() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 397
- exportOBJScene() (pyg4ometry.visualisation.VtkViewer method), 409
- exportOBJScene() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 397
- exportScreenShot() (pyg4ometry.visualisation.VtkViewer method), 409
- exportScreenShot() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 397
- exportThreeJSScene() (pyg4ometry.visualisation.ViewerBase method), 407
- exportThreeJSScene() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 391

- method), 391
- exportVRMLScene() (pyg4ometry.visualisation.VtkViewer method), 409
- exportVRMLScene() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 397
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser method), 118
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 108
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.SubZoneContext method), 107
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.UnaryAndSubZoneContext method), 106
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.UnaryAndSubZoneContext method), 106
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.UnaryAndSubZoneContext method), 105
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.UnaryAndSubZoneContext method), 116
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.UnaryAndSubZoneContext method), 115
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.UnaryAndSubZoneContext method), 116
- expr() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.ZoneExpression method), 114
- Expression (class in pyg4ometry.gdml), 222
- Expression (class in pyg4ometry.gdml.Defines), 202
- expression() (pyg4ometry.gdml.GdmlExpression.GdmlExpression method), 194
- expression() (pyg4ometry.gdml.GdmlExpression.GdmlExpression method), 188
- expression() (pyg4ometry.gdml.GdmlExpression.GdmlExpression method), 185
- expression() (pyg4ometry.gdml.GdmlExpression.GdmlExpression method), 190
- expression() (pyg4ometry.gdml.GdmlExpression.GdmlExpression.Parser.GdmlExpressionParser.MatrixElementContext method), 189
- ExpressionParser (class in pyg4ometry.gdml.GdmlExpression), 196
- ExpressionParser (class in pyg4ometry.gdml.GdmlExpression.GdmlExpression.Parser), 183
- expressionToZone() (in module pyg4ometry.fluka.boolean_algebra), 132
- extend() (pyg4ometry.fluka.Region method), 175
- extend() (pyg4ometry.fluka.region.Region method), 154
- extensionFromPath() (in module pyg4ometry.gui), 364
- extensionFromPath() (in module pyg4ometry.gui.GeometryModel), 352
- extent() (pyg4ometry.geant4._PhysicalVolume method), 337, 340
- extent() (pyg4ometry.geant4._ReplicaVolume method), 339
- extent() (pyg4ometry.geant4.AssemblyVolume method), 336
- extent() (pyg4ometry.geant4.AssemblyVolume.AssemblyVolume method), 310
- extent() (pyg4ometry.geant4.DivisionVolume method), 341
- extent() (pyg4ometry.geant4.DivisionVolume.DivisionVolume method), 312
- extent() (pyg4ometry.geant4.LogicalVolume method), 344
- extent() (pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 345
- extent() (pyg4ometry.geant4.ParameterisedVolume method), 340
- extent() (pyg4ometry.geant4.ParameterisedVolume.ParameterisedVolume method), 340
- extent() (pyg4ometry.geant4.PhysicalVolume method), 344
- extent() (pyg4ometry.geant4.PhysicalVolume.PhysicalVolume method), 344
- extent() (pyg4ometry.geant4.ReplicaVolume method), 340
- extent() (pyg4ometry.geant4.ReplicaVolume.ReplicaVolume method), 323
- extent() (pyg4ometry.stl.Reader method), 386
- extent() (pyg4ometry.stl.Reader.Reader method), 384
- extentCentre() (pyg4ometry.stl.Reader method), 386
- extentCentre() (pyg4ometry.stl.Reader.Reader method), 384
- extract() (in module pyg4ometry.features.algos), 98
- extract() (pyg4ometry.gdml.GdmlExpression.Parser.EquationContext method), 213
- extract() (pyg4ometry.gdml.GdmlExpression.Parser.FuncContext method), 205
- extract() (pyg4ometry.gdml.GdmlExpression.Parser.MatrixElementContext method), 205
- ExtrudedSolid (class in pyg4ometry.geant4.solid), 273
- ExtrudedSolid (class in pyg4ometry.geant4.solid.ExtrudedSolid), 230
- Extruder (class in pyg4ometry.fluka), 180
- Extruder (class in pyg4ometry.fluka.extruder), 136
- ## F
- f (in module pyg4ometry.pycgal_old), 380
- f (in module pyg4ometry.pycgal_old.cgal), 378
- FacetListAxisAlignedExtent() (in module pyg4ometry.freecad.Reader), 181
- Failed (pyg4ometry.compare._Compare.TestResult attribute), 58
- Failed (pyg4ometry.compare.TestResult attribute), 65
- FeatureData (class in pyg4ometry.features), 100
- FeatureData (class in pyg4ometry.features.algos), 98

`fill_color_dico()` (`pyg4ometry.visualisation.VtkExporter` method), 414
`fill_color_dico()` (`pyg4ometry.visualisation.VtkExporter` method), 394
`filterNullZones()` (`pyg4ometry.fluka.Region` method), 176
`filterNullZones()` (`pyg4ometry.fluka.region.Region` method), 155
`Finalize()` (`pyg4ometry.gui.example1.QVTKRenderWindow` method), 358
`Finalize()` (`pyg4ometry.gui.QVTKRenderWindowInteractor` method), 361, 363
`Finalize()` (`pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor` method), 355
`findLastBodyIndex()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 140
`findLastBodyIndex()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
`findLastMaterialIndex()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 140
`findLastMaterialIndex()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
`findLastRegionIndex()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 140
`findLastRegionIndex()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
`findLastTransformationIndex()` (`pyg4ometry.fluka.fluka_registry.FlukaRegistry` method), 140
`findLastTransformationIndex()` (`pyg4ometry.fluka.FlukaRegistry` method), 171
`findLogicalByName()` (`pyg4ometry.geant4.LogicalVolume` method), 335, 344
`findLogicalByName()` (`pyg4ometry.geant4.LogicalVolume.LogicalVolume` method), 315
`findLogicalVolumeByName()` (`pyg4ometry.geant4.Registry` method), 348
`findLogicalVolumeByName()` (`pyg4ometry.geant4.Registry.Registry` method), 321
`findMaterial()` (`pyg4ometry.fluka.material.multiGroupNeutronCrossSection` method), 144
`findMaterialByName()` (`pyg4ometry.geant4.Registry` method), 348
`findMaterialByName()` (`pyg4ometry.geant4.Registry.Registry` method), 321
`findOCCShapeByName()` (in module `pyg4ometry.pyoce.pythonHelpers`), 383
`findOCCShapeByTreeNode()` (in module `pyg4ometry.pyoce.pythonHelpers`), 383
`findPhysicalVolumeByName()` (`pyg4ometry.geant4.Registry` method), 348
`findPhysicalVolumeByName()` (`pyg4ometry.geant4.Registry.Registry` method), 321
`findSolidByName()` (`pyg4ometry.geant4.Registry` method), 348
`findSolidByName()` (`pyg4ometry.geant4.Registry.Registry` method), 321
`Flair` (class in `pyg4ometry.fluka`), 178
`Flair` (class in `pyg4ometry.fluka.flair`), 137
`flatten()` (`pyg4ometry.features.algos.Line` method), 96
`flatten()` (`pyg4ometry.features.algos.Plane` method), 96
`flatten()` (`pyg4ometry.features.Plane` method), 100
`fluka2Geant4()` (in module `pyg4ometry.convert`), 81
`fluka2Geant4()` (in module `pyg4ometry.convert.fluka2Geant4`), 69
`FLUKA_BUILTIN_TO_G4_MATERIAL_MAP` (in module `pyg4ometry.convert.fluka2g4materials`), 72
`FlukaBdxData` (class in `pyg4ometry.analysis.flukaData`), 51
`FlukaBinData` (class in `pyg4ometry.analysis.flukaData`), 51
`FlukaBodyStore` (class in `pyg4ometry.fluka.fluka_registry`), 140
`FlukaBodyStoreExact` (class in `pyg4ometry.fluka`), 172
`FlukaBodyStoreExact` (class in `pyg4ometry.fluka.fluka_registry`), 142
`FLUKAError`, 420
`flukaEventDisplay()` (in module `pyg4ometry.analysis.Plot`), 50
`flukaFreeString()` (`pyg4ometry.convert.ARB` method), 87
`flukaFreeString()` (`pyg4ometry.convert.BOX` method), 84
`flukaFreeString()` (`pyg4ometry.convert.ELL` method), 86
`flukaFreeString()` (`pyg4ometry.convert.PLA` method), 89
`flukaFreeString()` (`pyg4ometry.convert.QUA` method), 93
`flukaFreeString()` (`pyg4ometry.convert.RCC` method), 85
`flukaFreeString()` (`pyg4ometry.convert.REC` method), 85
`flukaFreeString()` (`pyg4ometry.convert.RPP` method), 83

<code>flukaFreeString()</code> <i>method</i>), 84	<code>(pyg4ometry.convert.SPH</code>	<code>flukaFreeString()</code> <i>method</i>), 130	<code>(pyg4ometry.fluka.body.YEC</code>
<code>flukaFreeString()</code> <i>method</i>), 86	<code>(pyg4ometry.convert.TRC</code>	<code>flukaFreeString()</code> <i>method</i>), 127	<code>(pyg4ometry.fluka.body.YZP</code>
<code>flukaFreeString()</code> <i>method</i>), 89	<code>(pyg4ometry.convert.XCC</code>	<code>flukaFreeString()</code> <i>method</i>), 129	<code>(pyg4ometry.fluka.body.ZCC</code>
<code>flukaFreeString()</code> <i>method</i>), 91	<code>(pyg4ometry.convert.XEC</code>	<code>flukaFreeString()</code> <i>method</i>), 130	<code>(pyg4ometry.fluka.body.ZEC</code>
<code>flukaFreeString()</code> <i>method</i>), 88	<code>(pyg4ometry.convert.XYP</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.BOX</code> <i>method</i>), 160	
<code>flukaFreeString()</code> <i>method</i>), 88	<code>(pyg4ometry.convert.XZP</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.BuiltIn</code> <i>method</i>), 178	
<code>flukaFreeString()</code> <i>method</i>), 90	<code>(pyg4ometry.convert.YCC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.Compound</code> <i>method</i>), 179	
<code>flukaFreeString()</code> <i>method</i>), 91	<code>(pyg4ometry.convert.YEC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.directive.RecursiveRotoTranslation</code> <i>method</i>), 136	
<code>flukaFreeString()</code> <i>method</i>), 88	<code>(pyg4ometry.convert.YZP</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.directive.RotoTranslation</code> <i>method</i>), 135	
<code>flukaFreeString()</code> <i>method</i>), 90	<code>(pyg4ometry.convert.ZCC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.ELL</code> <i>method</i>), 163	
<code>flukaFreeString()</code> <i>method</i>), 92	<code>(pyg4ometry.convert.ZEC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.fluka_registry.RotoTranslationStor</code> <i>method</i>), 140	
<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.ARB</code> <i>method</i>), 164		<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.Lattice</code> <i>method</i>), 178	
<code>flukaFreeString()</code> <i>method</i>), 126	<code>(pyg4ometry.fluka.body.ARB</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.lattice.Lattice</code> <i>method</i>), 143	
<code>flukaFreeString()</code> <i>method</i>), 122	<code>(pyg4ometry.fluka.body.BOX</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.Material</code> <i>method</i>), 179	
<code>flukaFreeString()</code> <i>method</i>), 125	<code>(pyg4ometry.fluka.body.ELL</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.material.BuiltIn</code> <i>method</i>), 144	
<code>flukaFreeString()</code> <i>method</i>), 127	<code>(pyg4ometry.fluka.body.PLA</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.material.Compound</code> <i>method</i>), 145	
<code>flukaFreeString()</code> <i>method</i>), 131	<code>(pyg4ometry.fluka.body.QUA</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.material.Material</code> <i>method</i>), 144	
<code>flukaFreeString()</code> <i>method</i>), 123	<code>(pyg4ometry.fluka.body.RCC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.PLA</code> <i>method</i>), 165	
<code>flukaFreeString()</code> <i>method</i>), 124	<code>(pyg4ometry.fluka.body.REC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.QUA</code> <i>method</i>), 169	
<code>flukaFreeString()</code> <i>method</i>), 122	<code>(pyg4ometry.fluka.body.RPP</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.RCC</code> <i>method</i>), 161	
<code>flukaFreeString()</code> <i>method</i>), 123	<code>(pyg4ometry.fluka.body.SPH</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.REC</code> <i>method</i>), 162	
<code>flukaFreeString()</code> <i>method</i>), 124	<code>(pyg4ometry.fluka.body.TRC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.RecursiveRotoTranslation</code> <i>method</i>), 178	
<code>flukaFreeString()</code> <i>method</i>), 128	<code>(pyg4ometry.fluka.body.XCC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.Region</code> <i>method</i>), 176	
<code>flukaFreeString()</code> <i>method</i>), 129	<code>(pyg4ometry.fluka.body.XEC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.region.Region</code> <i>method</i>), 154	
<code>flukaFreeString()</code> <i>method</i>), 126	<code>(pyg4ometry.fluka.body.XYP</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.RotoTranslation</code> <i>method</i>), 177	
<code>flukaFreeString()</code> <i>method</i>), 126	<code>(pyg4ometry.fluka.body.XZP</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.RPP</code> <i>method</i>), 160	
<code>flukaFreeString()</code> <i>method</i>), 128	<code>(pyg4ometry.fluka.body.YCC</code>	<code>flukaFreeString()</code> <code>(pyg4ometry.fluka.SPH</code> <i>method</i>), 161	

- `flukaFreeString()` (*pyg4ometry.fluka.TRC method*), 162
 - `flukaFreeString()` (*pyg4ometry.fluka.XCC method*), 165
 - `flukaFreeString()` (*pyg4ometry.fluka.XEC method*), 167
 - `flukaFreeString()` (*pyg4ometry.fluka.XYP method*), 164
 - `flukaFreeString()` (*pyg4ometry.fluka.XZP method*), 164
 - `flukaFreeString()` (*pyg4ometry.fluka.YCC method*), 166
 - `flukaFreeString()` (*pyg4ometry.fluka.YEC method*), 168
 - `flukaFreeString()` (*pyg4ometry.fluka.YZP method*), 165
 - `flukaFreeString()` (*pyg4ometry.fluka.ZCC method*), 166
 - `flukaFreeString()` (*pyg4ometry.fluka.ZEC method*), 168
 - `FLUKAInputError`, 420
 - `FlukaRegistry` (*class in pyg4ometry.fluka*), 170
 - `FlukaRegistry` (*class in pyg4ometry.fluka.fluka_registry*), 138
 - `flukaString()` (*pyg4ometry.montecarlo.Beam method*), 370
 - `flukaString()` (*pyg4ometry.montecarlo.beam.Beam method*), 368
 - `flukaString()` (*pyg4ometry.montecarlo.Boundary method*), 370
 - `flukaString()` (*pyg4ometry.montecarlo.boundary.Boundary method*), 369
 - `flukaString()` (*pyg4ometry.montecarlo.Config method*), 370
 - `flukaString()` (*pyg4ometry.montecarlo.config.Config method*), 369
 - `flukaString()` (*pyg4ometry.montecarlo.Scoring method*), 370
 - `flukaString()` (*pyg4ometry.montecarlo.scoring.Scoring method*), 369
 - `fortran_read()` (*in module pyg4ometry.analysis.flukaData*), 51
 - `fortran_skip()` (*in module pyg4ometry.analysis.flukaData*), 51
 - `Freecad` (*pyg4ometry.geant4.solid.TessellatedSolid.MeshType attribute*), 304
 - `Freecad` (*pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid.MeshType attribute*), 245
 - `freecadDoc2Fluka()` (*in module pyg4ometry.convert*), 93
 - `freecadDoc2Fluka()` (*in module pyg4ometry.convert.freecad2Fluka*), 73
 - `freeFormatStringSplit()` (*in module pyg4ometry.fluka.card*), 134
 - `freeShapes()` (*pyg4ometry.pyoce.Reader method*), 383
 - `freeShapes()` (*pyg4ometry.pyoce.Reader.Reader method*), 382
 - `from_existing()` (*pyg4ometry.utils.Samples class method*), 425
 - `fromCard()` (*pyg4ometry.fluka.directive.RotoTranslation class method*), 135
 - `fromCard()` (*pyg4ometry.fluka.Material class method*), 179
 - `fromCard()` (*pyg4ometry.fluka.material.Material class method*), 145
 - `fromCard()` (*pyg4ometry.fluka.RotoTranslation class method*), 177
 - `fromCards()` (*pyg4ometry.fluka.Compound class method*), 179
 - `fromCards()` (*pyg4ometry.fluka.material.Compound class method*), 145
 - `fromFixed()` (*pyg4ometry.fluka.Card class method*), 180
 - `fromFixed()` (*pyg4ometry.fluka.card.Card class method*), 133
 - `fromFree()` (*pyg4ometry.fluka.Card class method*), 180
 - `fromFree()` (*pyg4ometry.fluka.card.Card class method*), 133
 - `fromMesh()` (*pyg4ometry.fluka.AABB class method*), 173
 - `fromMesh()` (*pyg4ometry.fluka.vector.AABB class method*), 157
 - `fromPolygons()` (*pyg4ometry.pycgal.core.CSG class method*), 371
 - `fromPolygons()` (*pyg4ometry.pycgal.CSG class method*), 374
 - `func()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpression method*), 194
 - `func()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpression method*), 188
 - `funcname()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpression method*), 194
 - `funcname()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpression method*), 190
- ## G
- `g` (*in module pyg4ometry.pycgal_old*), 380
 - `g` (*in module pyg4ometry.pycgal_old.cgal*), 378
 - `Gdml` (*pyg4ometry.geant4.solid.TessellatedSolid.MeshType attribute*), 304
 - `Gdml` (*pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid.MeshType attribute*), 245
 - `gdml2stl()` (*in module pyg4ometry.convert*), 93
 - `gdml2stl()` (*in module pyg4ometry.convert.gdml2stl*), 73
 - `GdmlExpressionEvalVisitor` (*class in pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval*), 182

GdmlExpressionLexer (class in gdmlFiles() (in module pyg4ometry.compare), 66
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer(), 183
 (in module pyg4ometry.compare._Compare), 59)

GdmlExpressionParser (class in geant4Logical2Fluka() (in module
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), pyg4ometry.convert), 82
 185
 geant4Logical2Fluka() (in module pyg4ometry.convert.geant42Fluka), 74)

GdmlExpressionParser.AtomContext (class in geant4Logical2Fluka() (in module
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 188
 pyg4ometry.convert.geant42FlukaBake), 75)

GdmlExpressionParser.ConstantContext (class in geant4Material2Fluka() (in module
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 189
 pyg4ometry.convert), 83)

GdmlExpressionParser.EquationContext (class in geant4Material2Fluka() (in module
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), pyg4ometry.convert.geant42Fluka), 74
 185
 geant4Material2Fluka() (in module pyg4ometry.convert.geant42FlukaBake), 75)

GdmlExpressionParser.ExpressionContext (class in pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 186
 geant4MaterialDict2Fluka() (in module pyg4ometry.convert), 83)

GdmlExpressionParser.FuncContext (class in geant4MaterialDict2Fluka() (in module
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 190
 pyg4ometry.convert.geant42Fluka), 74)

GdmlExpressionParser.FuncnameContext (class in geant4MaterialDict2Fluka() (in module
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), pyg4ometry.convert.geant42FlukaBake), 190
 76)

GdmlExpressionParser.MatrixElementContext (geant4PhysicalVolume2Fluka() (in module
 (class in pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 189
 pyg4ometry.convert), 82
 geant4PhysicalVolume2Fluka() (in module pyg4ometry.convert.geant42Fluka), 74)

GdmlExpressionParser.MultiplyingExpressionContext (geant4PhysicalVolume2Fluka() (in module
 (class in pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 186
 pyg4ometry.convert.geant42FlukaBake), 75)

GdmlExpressionParser.OperatorAddSubContext (geant4Reg2FlukaReg() (in module
 (class in pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 186
 pyg4ometry.convert), 82)

GdmlExpressionParser.OperatorMulDivContext (geant4Reg2FlukaReg() (in module
 (class in pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 187
 pyg4ometry.convert.geant42Fluka), 74
 geant4Reg2FlukaReg() (in module pyg4ometry.convert.geant42FlukaBake), 75)

GdmlExpressionParser.PowExpressionContext (pyg4ometry.convert.geant42FlukaBake),
 (class in pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 187
 geant4Solid() (pyg4ometry.convert.ARB method), 87)

GdmlExpressionParser.RelopContext (class in geant4Solid() (pyg4ometry.convert.BOX method), 84
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 191
 geant4Solid() (pyg4ometry.convert.ELL method), 86
 geant4Solid() (pyg4ometry.convert.QUA method), 92)

GdmlExpressionParser.ScientificContext (class in geant4Solid() (pyg4ometry.convert.RCC method), 85
 in pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 188
 geant4Solid() (pyg4ometry.convert.REC method), 85
 geant4Solid() (pyg4ometry.convert.RPP method), 83)

GdmlExpressionParser.SignedAtomContext (class in geant4Solid() (pyg4ometry.convert.SPH method), 84
 in pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 188
 geant4Solid() (pyg4ometry.convert.TRC method), 86
 geant4Solid() (pyg4ometry.convert.XEC method), 91)

GdmlExpressionParser.VariableContext (class in geant4Solid() (pyg4ometry.convert.YEC method), 91
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser), 190
 geant4Solid() (pyg4ometry.convert.ZEC method), 92
 geant4Solid() (pyg4ometry.fluka.ARB method), 163)

GdmlExpressionVisitor (class in geant4Solid() (pyg4ometry.fluka.body.ARB method),
 pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor), 194
 geant4Solid() (pyg4ometry.fluka.body.BOX method), 163)

- 122
- geant4Solid() (pyg4ometry.fluka.body.ELL method), 124
- geant4Solid() (pyg4ometry.fluka.body.QUA method), 131
- geant4Solid() (pyg4ometry.fluka.body.RCC method), 123
- geant4Solid() (pyg4ometry.fluka.body.REC method), 123
- geant4Solid() (pyg4ometry.fluka.body.RPP method), 122
- geant4Solid() (pyg4ometry.fluka.body.SPH method), 122
- geant4Solid() (pyg4ometry.fluka.body.TRC method), 124
- geant4Solid() (pyg4ometry.fluka.body.XEC method), 129
- geant4Solid() (pyg4ometry.fluka.body.YEC method), 130
- geant4Solid() (pyg4ometry.fluka.body.ZEC method), 130
- geant4Solid() (pyg4ometry.fluka.BOX method), 160
- geant4Solid() (pyg4ometry.fluka.ELL method), 162
- geant4Solid() (pyg4ometry.fluka.QUA method), 169
- geant4Solid() (pyg4ometry.fluka.RCC method), 161
- geant4Solid() (pyg4ometry.fluka.REC method), 161
- geant4Solid() (pyg4ometry.fluka.Region method), 176
- geant4Solid() (pyg4ometry.fluka.region.Region method), 154
- geant4Solid() (pyg4ometry.fluka.region.Zone method), 152
- geant4Solid() (pyg4ometry.fluka.RPP method), 160
- geant4Solid() (pyg4ometry.fluka.SPH method), 160
- geant4Solid() (pyg4ometry.fluka.TRC method), 162
- geant4Solid() (pyg4ometry.fluka.XEC method), 167
- geant4Solid() (pyg4ometry.fluka.YEC method), 167
- geant4Solid() (pyg4ometry.fluka.ZEC method), 168
- geant4Solid() (pyg4ometry.fluka.Zone method), 174
- geant4Solid2FlukaRegion() (in module pyg4ometry.convert), 83
- geant4Solid2FlukaRegion() (in module pyg4ometry.convert.geant42Fluka), 74
- geant4Solid2FlukaRegion() (in module pyg4ometry.convert.geant42FlukaBake), 75
- geant4Solid2Geant4Tessellated() (in module pyg4ometry.convert), 93
- geant4Solid2Geant4Tessellated() (in module pyg4ometry.convert.geant42Geant4), 76
- geant4Solid2Geant4Tessellated_NoVTK() (in module pyg4ometry.convert), 93
- geant4Solid2Geant4Tessellated_NoVTK() (in module pyg4ometry.convert.geant42Geant4), 76
- generate_name() (pyg4ometry.fluka.region._Boolean method), 152
- GenericPolycone (class in pyg4ometry.geant4.solid), 301
- GenericPolycone (class in pyg4ometry.geant4.solid.GenericPolycone), 230
- GenericPolyhedra (class in pyg4ometry.geant4.solid), 302
- GenericPolyhedra (class in pyg4ometry.geant4.solid.GenericPolyhedra), 231
- GenericTrap (class in pyg4ometry.geant4.solid), 302
- GenericTrap (class in pyg4ometry.geant4.solid.GenericTrap), 232
- geometry() (in module pyg4ometry.compare), 66
- geometry() (in module pyg4ometry.compare._Compare), 59
- GeometryComplexityInformation (class in pyg4ometry.geant4), 348
- GeometryComplexityInformation (class in pyg4ometry.geant4.Registry), 321
- GeometryModel (class in pyg4ometry.gui), 362, 364
- GeometryModel (class in pyg4ometry.gui.GeometryModel), 352
- get_material_oject() (pyg4ometry.geant4._Material.MaterialBase method), 327
- get_material_oject() (pyg4ometry.geant4.MaterialBase method), 350
- get_shapeTypeString() (in module pyg4ometry.pyoce.pythonHelpers), 383
- get_TDataStd_Name_From_Label() (in module pyg4ometry.pyoce.pythonHelpers), 383
- get_TDataStd_TreeNode_From_Label() (in module pyg4ometry.pyoce.pythonHelpers), 383
- get_variables() (pyg4ometry.gdml.GdmlExpression.ExpressionParser method), 196
- get_variables() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionEvaluator method), 183
- get_vertex() (pyg4ometry.geant4.solid.GenericTrap method), 303
- get_vertex() (pyg4ometry.geant4.solid.GenericTrap.GenericTrap method), 233
- get_XCAFDLocation_From_Label() (in module pyg4ometry.pyoce.pythonHelpers), 383
- getAABBMesh() (pyg4ometry.geant4._PhysicalVolume method), 337, 340
- getAABBMesh() (pyg4ometry.geant4.AssemblyVolume method), 336
- getAABBMesh() (pyg4ometry.geant4.AssemblyVolume.AssemblyVolume method), 310
- getAABBMesh() (pyg4ometry.geant4.PhysicalVolume method), 336, 344

getAABBMesh() (pyg4ometry.geant4.PhysicalVolume.PhysicalVolume method), 141
 method), 317
 GetAxisName() (pyg4ometry.geant4._ReplicaVolume method), 338
 GetAxisName() (pyg4ometry.geant4.ReplicaVolume method), 337
 GetAxisName() (pyg4ometry.geant4.ReplicaVolume.ReplicaVolume method), 322
 getBody() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
 getBody() (pyg4ometry.fluka.FlukaRegistry method), 170
 getBodyToRegionsMap() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
 getBodyToRegionsMap() (pyg4ometry.fluka.FlukaRegistry method), 170
 getBoundingBox() (pyg4ometry.visualisation.Mesh method), 405
 getBoundingBox() (pyg4ometry.visualisation.Mesh.Mesh method), 388
 getBoundingBoxMesh() (pyg4ometry.visualisation.Mesh method), 405
 getBoundingBoxMesh() (pyg4ometry.visualisation.Mesh.Mesh method), 388
 getColour() (pyg4ometry.visualisation._VisOptions method), 412
 getColour() (pyg4ometry.visualisation.VisualisationOptions method), 407
 getColour() (pyg4ometry.visualisation.VisualisationOptions method), 393
 getCutterPolydata() (pyg4ometry.visualisation.VtkViewerNew method), 412
 getCutterPolydata() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 401
 getDefaultVisOptions() (pyg4ometry.visualisation.ViewerBase method), 406
 getDefaultVisOptions() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 391
 getDegenerateBody() (pyg4ometry.fluka.fluka_registry.BaseCacher method), 141
 getDegenerateBody() (pyg4ometry.fluka.fluka_registry.Cacheable method), 141
 getDegenerateBody() (pyg4ometry.fluka.fluka_registry.FlukaBodyStore method), 142
 getDegenerateBody() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
 getDegenerateBody() (pyg4ometry.fluka.FlukaBodyStoreExact method), 172
 getDegenerateBody() (pyg4ometry.fluka.FlukaRegistry method), 170
 getElementName() (pyg4ometry.visualisation.VtkExporter method), 415
 getElementName() (pyg4ometry.visualisation.VtkExporter.VtkExporter method), 395
 getExpressionParser() (pyg4ometry.geant4.Registry method), 345
 getExpressionParser() (pyg4ometry.geant4.Registry.Registry method), 318
 getLocalMesh() (pyg4ometry.visualisation.Mesh method), 405
 getLocalMesh() (pyg4ometry.visualisation.Mesh.Mesh method), 388
 getMask() (pyg4ometry.fluka.fluka_registry.Cacheable method), 141
 getMaterial() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 getMaterial() (pyg4ometry.fluka.FlukaRegistry method), 170
 getMaterialVisOptions() (pyg4ometry.visualisation.ViewerBase method), 406
 getMaterialVisOptions() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 391
 getMaterialVisOptions() (pyg4ometry.visualisation.VtkExporter method), 415
 getMaterialVisOptions() (pyg4ometry.visualisation.VtkExporter.VtkExporter method), 395
 getMaterialVisOptions() (pyg4ometry.visualisation.VtkViewer method), 410
 getMaterialVisOptions() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 399
 getMotherSize() (pyg4ometry.geant4.DivisionVolume method), 340
 getMotherSize() (pyg4ometry.geant4.DivisionVolume.DivisionVolume method), 311

<code>getNistElementZToName()</code>	(in module <code>pyg4ometry.geant4</code>), 348	<code>getRegistry()</code>	(<code>pyg4ometry.io.ROOTTGeo.Reader</code> method), 365
<code>getNistElementZToName()</code>	(in module <code>pyg4ometry.geant4._Material</code>), 326	<code>getRegistry()</code>	(<code>pyg4ometry.stl.Reader</code> method), 386
<code>getNistMaterialDict()</code>	(in module <code>pyg4ometry.geant4</code>), 348	<code>getRegistry()</code>	(<code>pyg4ometry.stl.Reader.Reader</code> method), 385
<code>getNistMaterialDict()</code>	(in module <code>pyg4ometry.geant4._Material</code>), 326	<code>GetRenderWindow()</code>	(<code>pyg4ometry.gui.example1.QVTKRenderWindowInteractor</code> method), 359
<code>getNistMaterialList()</code>	(in module <code>pyg4ometry.geant4</code>), 348	<code>GetRenderWindow()</code>	(<code>pyg4ometry.gui.QVTKRenderWindowInteractor</code> method), 362, 364
<code>getNistMaterialList()</code>	(in module <code>pyg4ometry.geant4._Material</code>), 326	<code>GetRenderWindow()</code>	(<code>pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor</code> method), 355
<code>getNumberPolys()</code>	(<code>pyg4ometry.pycgal.core.CSG</code> method), 371	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.Expression</code> method), 115
<code>getNumberPolys()</code>	(<code>pyg4ometry.pycgal.CSG</code> method), 374	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 112
<code>getNumberVertices()</code>	(<code>pyg4ometry.pycgal.core.CSG</code> method), 371	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 105
<code>getNumberVertices()</code>	(<code>pyg4ometry.pycgal.CSG</code> method), 374	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 103
<code>getOverlapVisOptions()</code>	(<code>pyg4ometry.visualisation.ViewerBase</code> method), 406	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 102
<code>getOverlapVisOptions()</code>	(<code>pyg4ometry.visualisation.ViewerBase.ViewerBase</code> method), 391	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 107
<code>getOverlapVisOptions()</code>	(<code>pyg4ometry.visualisation.VtkViewer</code> method), 410	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 107
<code>getOverlapVisOptions()</code>	(<code>pyg4ometry.visualisation.VtkViewer.VtkViewer</code> method), 399	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 105
<code>getPhysicalVolumes()</code>	(<code>pyg4ometry.geant4._ReplicaVolume</code> method), 338	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 103
<code>getPhysicalVolumes()</code>	(<code>pyg4ometry.geant4.ReplicaVolume</code> method), 338	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 116
<code>getPhysicalVolumes()</code>	(<code>pyg4ometry.geant4.ReplicaVolume.ReplicaVolume</code> method), 323	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 112
<code>getPredefinedMaterialVisOptions()</code>	(in module <code>pyg4ometry.visualisation</code>), 407	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 114
<code>getPredefinedMaterialVisOptions()</code>	(in module <code>pyg4ometry.visualisation.VisualisationOptions</code>), 393	<code>getRuleIndex()</code>	(<code>pyg4ometry.fluka.RegionExpression.RegionParser.RegionExpression</code> method), 113
<code>getRegistry()</code>	(<code>pyg4ometry.fluka.Reader</code> method), 169	<code>getRuleIndex()</code>	(<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser</code> method), 188
<code>getRegistry()</code>	(<code>pyg4ometry.fluka.reader.Reader</code> method), 149	<code>getRuleIndex()</code>	(<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser</code> method), 190
<code>getRegistry()</code>	(<code>pyg4ometry.freecad.Reader.Reader</code> method), 181	<code>getRuleIndex()</code>	(<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser</code> method), 186
<code>getRegistry()</code>	(<code>pyg4ometry.gdml.Reader</code> method), 212	<code>getRuleIndex()</code>	(<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser</code> method), 186
<code>getRegistry()</code>	(<code>pyg4ometry.gdml.Reader.Reader</code> method), 205	<code>getRuleIndex()</code>	(<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser</code> method), 190
		<code>getRuleIndex()</code>	(<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser</code> method), 191
		<code>getRuleIndex()</code>	(<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser</code> method), 189
		<code>getRuleIndex()</code>	(<code>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser</code> method), 189

- hash() (pyg4ometry.fluka.ELL method), 163
- hash() (pyg4ometry.fluka.PLA method), 165
- hash() (pyg4ometry.fluka.QUA method), 169
- hash() (pyg4ometry.fluka.RCC method), 161
- hash() (pyg4ometry.fluka.REC method), 162
- hash() (pyg4ometry.fluka.RPP method), 160
- hash() (pyg4ometry.fluka.SPH method), 161
- hash() (pyg4ometry.fluka.TRC method), 162
- hash() (pyg4ometry.fluka.XCC method), 166
- hash() (pyg4ometry.fluka.XEC method), 167
- hash() (pyg4ometry.fluka.XYP method), 164
- hash() (pyg4ometry.fluka.XZP method), 164
- hash() (pyg4ometry.fluka.YCC method), 166
- hash() (pyg4ometry.fluka.YEC method), 168
- hash() (pyg4ometry.fluka.YZP method), 165
- hash() (pyg4ometry.fluka.ZCC method), 167
- hash() (pyg4ometry.fluka.ZEC method), 168
- hasRotation() (pyg4ometry.fluka.directive.RotoTranslation method), 135
- hasRotation() (pyg4ometry.fluka.RotoTranslation method), 177
- hasTranslation() (pyg4ometry.fluka.directive.RotoTranslation method), 135
- hasTranslation() (pyg4ometry.fluka.RotoTranslation method), 177
- hexRGBToRGBTriplet() (in module pyg4ometry.visualisation), 407
- hexRGBToRGBTriplet() (in module pyg4ometry.visualisation.VisualisationOptions), 392
- HideCursor() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor method), 358
- HideCursor() (pyg4ometry.gui.QVTKRenderWindowInteractor method), 361, 363
- HideCursor() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method), 355
- Hype (class in pyg4ometry.geant4.solid), 292
- Hype (class in pyg4ometry.geant4.solid.Hype), 233
- I (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 184
- I (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- I() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.ConstantContext method), 190
- IdenticalNameError, 419
- InfiniteCylinderCacher (class in pyg4ometry.fluka.fluka_registry), 141
- infinity() (in module pyg4ometry.fluka), 169
- info() (pyg4ometry.pycgal.core.CSG method), 372
- info() (pyg4ometry.pycgal.CSG method), 375
- initModel() (pyg4ometry.gui.MainWindow method), 362
- initModel() (pyg4ometry.gui.MainWindow.MainWindow method), 353
- initUI() (pyg4ometry.gui.MainWindow method), 362
- initUI() (pyg4ometry.gui.MainWindow.MainWindow method), 353
- initVtk() (pyg4ometry.visualisation.VtkViewerNew method), 412
- initVtk() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 401
- InLineComment (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 118
- InLineComment (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
- InLineComment (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 118
- InLineComment (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
- insert() (pyg4ometry.fluka.directive.RecursiveRotoTranslation method), 136
- insert() (pyg4ometry.fluka.RecursiveRotoTranslation method), 178
- Integer (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 118
- Integer (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
- Integer (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 118
- Integer (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
- Integer() (pyg4ometry.fluka.RegionExpression.RegionParser.ComplexRegionParser method), 112
- Integer() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 103
- Integer() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 103
- Integer() (pyg4ometry.fluka.RegionExpression.RegionParser.SimpleRegionParser method), 113
- intersect() (pyg4ometry.features.algos.Plane method), 96
- intersect() (pyg4ometry.features.Plane method), 99
- intersect() (pyg4ometry.fluka.AABB method), 173
- intersect() (pyg4ometry.fluka.vector.AABB method), 157
- intersect() (pyg4ometry.pycgal.core.CSG method), 371
- intersect() (pyg4ometry.pycgal.CSG method), 375
- intersecting_meshes() (in module pyg4ometry.pycgal), 375
- intersecting_meshes() (in module pyg4ometry.pycgal.core), 372
- Intersection (class in pyg4ometry.fluka.region), 152
- Intersection (class in pyg4ometry.geant4.solid), 277
- Intersection (class in pyg4ometry.geant4.solid.Intersection), 234

`intersectLine()` (*pyg4ometry.features.algos.Plane method*), 96
`intersectLine()` (*pyg4ometry.features.Plane method*), 99
`intersectPlane()` (*pyg4ometry.features.algos.Line method*), 95
`intersectPlane()` (*pyg4ometry.features.algos.Plane method*), 96
`intersectPlane()` (*pyg4ometry.features.Plane method*), 99
`intersects()` (*pyg4ometry.fluka.AABB method*), 173
`intersects()` (*pyg4ometry.fluka.vector.AABB method*), 157
`inverse()` (*pyg4ometry.pycgal.core.CSG method*), 371
`inverse()` (*pyg4ometry.pycgal.CSG method*), 375
`IsAReplica()` (*in module pyg4ometry.geant4*), 332
`isClosed()` (*pyg4ometry.pycgal.core.CSG method*), 372
`isClosed()` (*pyg4ometry.pycgal.CSG method*), 375
`isComment()` (*in module pyg4ometry.gdml*), 211
`isComment()` (*in module pyg4ometry.gdml.Reader*), 205
`isDNF()` (*pyg4ometry.fluka.Region method*), 176
`isDNF()` (*pyg4ometry.fluka.region.Region method*), 155
`isDNF()` (*pyg4ometry.fluka.region.Zone method*), 153
`isDNF()` (*pyg4ometry.fluka.Zone method*), 175
`isGas()` (*pyg4ometry.fluka.material._MatProp method*), 144
`isNull()` (*pyg4ometry.fluka.AABB method*), 173
`isNull()` (*pyg4ometry.fluka.Region method*), 176
`isNull()` (*pyg4ometry.fluka.region.Region method*), 155
`isNull()` (*pyg4ometry.fluka.region.Zone method*), 153
`isNull()` (*pyg4ometry.fluka.vector.AABB method*), 157
`isNull()` (*pyg4ometry.fluka.Zone method*), 175
`isNull()` (*pyg4ometry.pycgal.core.CSG method*), 372
`isNull()` (*pyg4ometry.pycgal.CSG method*), 375
`Isotope` (*class in pyg4ometry.geant4*), 351
`Isotope` (*class in pyg4ometry.geant4._Material*), 329
`isOutwardOriented()` (*pyg4ometry.pycgal.core.CSG method*), 372
`isOutwardOriented()` (*pyg4ometry.pycgal.CSG method*), 375
`isPureTranslation()` (*pyg4ometry.fluka.directive.RotoTranslation method*), 135
`isPureTranslation()` (*pyg4ometry.fluka.RotoTranslation method*), 177
`isTriangleMesh()` (*pyg4ometry.pycgal.core.CSG method*), 372
`isTriangleMesh()` (*pyg4ometry.pycgal.CSG method*), 375
`isZoneContradiction()` (*in module pyg4ometry.fluka.boolean_algebra*), 132

K

`keyPressEvent()` (*pyg4ometry.gui.example1.QVTKRenderWindowInteractor method*), 358
`keyPressEvent()` (*pyg4ometry.gui.QVTKRenderWindowInteractor method*), 362, 364
`keyPressEvent()` (*pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method*), 355
`keyReleaseEvent()` (*pyg4ometry.gui.example1.QVTKRenderWindowInteractor method*), 358
`keyReleaseEvent()` (*pyg4ometry.gui.QVTKRenderWindowInteractor method*), 362, 364
`keyReleaseEvent()` (*pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method*), 355
`keys()` (*pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method*), 142
`keys()` (*pyg4ometry.fluka.FlukaBodyStoreExact method*), 172
`kPhi` (*pyg4ometry.geant4._ReplicaVolume.Axis attribute*), 338
`kPhi` (*pyg4ometry.geant4.DivisionVolume.Axis attribute*), 340
`kPhi` (*pyg4ometry.geant4.DivisionVolume.DivisionVolume.Axis attribute*), 311
`kPhi` (*pyg4ometry.geant4.ReplicaVolume.Axis attribute*), 337
`kPhi` (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute*), 322
`kRho` (*pyg4ometry.geant4._ReplicaVolume.Axis attribute*), 338
`kRho` (*pyg4ometry.geant4.DivisionVolume.Axis attribute*), 340
`kRho` (*pyg4ometry.geant4.DivisionVolume.DivisionVolume.Axis attribute*), 311
`kRho` (*pyg4ometry.geant4.ReplicaVolume.Axis attribute*), 337
`kRho` (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute*), 322
`kXAxis` (*pyg4ometry.geant4._ReplicaVolume.Axis attribute*), 338
`kXAxis` (*pyg4ometry.geant4.DivisionVolume.Axis attribute*), 340
`kXAxis` (*pyg4ometry.geant4.DivisionVolume.DivisionVolume.Axis attribute*), 311
`kXAxis` (*pyg4ometry.geant4.ReplicaVolume.Axis attribute*), 337
`kXAxis` (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute*), 322
`kYAxis` (*pyg4ometry.geant4._ReplicaVolume.Axis attribute*), 338
`kYAxis` (*pyg4ometry.geant4.DivisionVolume.Axis attribute*), 340
`kYAxis` (*pyg4ometry.geant4.DivisionVolume.DivisionVolume.Axis attribute*), 311
`kYAxis` (*pyg4ometry.geant4.ReplicaVolume.Axis attribute*), 337

- tribute), 337
- kYAxis (pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute), 322
- kZAxis (pyg4ometry.geant4.ReplicaVolume.Axis attribute), 338
- kZAxis (pyg4ometry.geant4.DivisionVolume.DivisionVolume.Axis attribute), 340
- kZAxis (pyg4ometry.geant4.DivisionVolume.DivisionVolume.Axis attribute), 311
- kZAxis (pyg4ometry.geant4.ReplicaVolume.Axis attribute), 337
- kZAxis (pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute), 322
- L**
- Lattice (class in pyg4ometry.fluka), 178
- Lattice (class in pyg4ometry.fluka.lattice), 142
- latticeAABBs() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
- latticeAABBs() (pyg4ometry.fluka.FlukaRegistry method), 170
- Layer (class in pyg4ometry.geant4.solid.Layer), 235
- LBRACKET (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer attribute), 184
- LBRACKET (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser attribute), 193
- LBRACKET() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser method), 189
- leafCount() (pyg4ometry.fluka.Region method), 176
- leafCount() (pyg4ometry.fluka.region.Region method), 155
- leafCount() (pyg4ometry.fluka.region.Zone method), 153
- leafCount() (pyg4ometry.fluka.Zone method), 175
- leaveEvent() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor method), 358
- leaveEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor method), 361, 364
- leaveEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor method), 355
- leftMultiplyRotation() (pyg4ometry.fluka.directive.MatrixConvertibleMixin method), 134
- leftMultiplyVector() (pyg4ometry.fluka.directive.MatrixConvertibleMixin method), 134
- length() (pyg4ometry.fluka.Three method), 172
- length() (pyg4ometry.fluka.vector.Three method), 156
- lengths() (pyg4ometry.convert.BOX method), 84
- lengths() (pyg4ometry.convert.RPP method), 83
- lengths() (pyg4ometry.fluka.body.BOX method), 122
- lengths() (pyg4ometry.fluka.body.RPP method), 121
- lengths() (pyg4ometry.fluka.BOX method), 160
- lengths() (pyg4ometry.fluka.RPP method), 159
- Line (class in pyg4ometry.features.algos), 95
- LineComment (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 118
- LineComment (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionParser attribute), 101
- LineComment (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 118
- LineComment (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
- LineComment_sempred() (pyg4ometry.fluka.RegionExpression.RegionLexer method), 119
- LineComment_sempred() (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer method), 101
- literalNames (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 119
- literalNames (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionParser attribute), 101
- literalNames (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
- literalNames (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 107
- literalNames (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer attribute), 184
- literalNames (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser attribute), 192
- LN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionParser attribute), 184
- LN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- LN() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 191
- load() (pyg4ometry.freecad.Reader.Reader method), 181
- load() (pyg4ometry.gdml.Reader method), 211
- load() (pyg4ometry.gdml.Reader.Reader method), 205
- load() (pyg4ometry.io.ROOTTGeo.Reader method), 365
- loadAuxiliaryData() (pyg4ometry.freecad.Reader.Reader method), 181
- loadMaterials() (pyg4ometry.io.ROOTTGeo.Reader method), 365
- loadNewRegistry() (pyg4ometry.gui.GeometryModel method), 362, 364
- loadNewRegistry() (pyg4ometry.gui.GeometryModel.GeometryModel method), 352
- loadNISTMaterialDict() (in module pyg4ometry.geant4), 349
- loadNISTMaterialDict() (in module pyg4ometry.geant4._Material), 326
- loadPredefined() (in module pyg4ometry.visualisation), 407
- loadPredefined() (in module pyg4ometry.visualisation.VisualisationOptions),

393

- LOG (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 184
- LOG (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- log() (in module pyg4ometry.gdml), 220
- log() (in module pyg4ometry.gdml.Defines), 200
- LOG() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 191
- log10() (in module pyg4ometry.gdml), 220
- log10() (in module pyg4ometry.gdml.Defines), 200
- logger (in module pyg4ometry.convert.fluka2Geant4), 69
- logger (in module pyg4ometry.fluka.fluka_registry), 138
- logger (in module pyg4ometry.fluka.region), 151
- LogicalVolume (class in pyg4ometry.geant4), 332, 341
- LogicalVolume (class in pyg4ometry.geant4.LogicalVolume), 312
- logicalVolume() (pyg4ometry.geant4.AssemblyVolume method), 336
- logicalVolume() (pyg4ometry.geant4.AssemblyVolume.AssemblyVolume method), 310
- logicalVolumes() (in module pyg4ometry.compare), 66
- logicalVolumes() (in module pyg4ometry.compare._Compare), 59
- Loop (class in pyg4ometry.geant4.Loop), 316
- LParen (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 119
- LParen (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
- LParen (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 118
- LParen (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
- LPAREN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 184
- LPAREN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- LParen() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.SubZoneContext method), 107
- LParen() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.SubZoneContext method), 116
- LPAREN() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.AtomContext method), 188
- LPAREN() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.FuncContext method), 190
- LT (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 184
- LT (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- LT() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.RelopContext method), 191
- M
- main() (in module pyg4ometry.cli), 417
- main() (in module pyg4ometry.fluka.card), 134
- main() (in module pyg4ometry.fluka.reader), 150
- main() (in module pyg4ometry.gui), 362
- main() (in module pyg4ometry.gui.MainWindow), 353
- MainWindow (class in pyg4ometry.gui), 362
- MainWindow (class in pyg4ometry.gui.MainWindow), 353
- make() (pyg4ometry.fluka.fluka_registry.BaseCacher method), 141
- make() (pyg4ometry.fluka.fluka_registry.FlukaBodyStore method), 141
- make() (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method), 142
- make() (pyg4ometry.fluka.FlukaBodyStoreExact method), 172
- makeBody() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
- makeBody() (pyg4ometry.fluka.FlukaRegistry method), 170
- makeFaceFromLayer() (pyg4ometry.geant4.solid._TwistedSolid method), 293, 295, 298
- makeFaceFromLayer() (pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid method), 251
- makeFlukaToG4MaterialsMap() (in module pyg4ometry.convert.fluka2g4materials), 72
- makeFromPlanes() (pyg4ometry.features.algos.CoordinateSystem method), 96
- makeFromPlanes() (pyg4ometry.features.CoordinateSystem method), 99
- makeLayerEdge() (pyg4ometry.geant4.solid.TwistedBox method), 295
- makeLayerEdge() (pyg4ometry.geant4.solid.TwistedBox.TwistedBox method), 250
- makeLayers() (pyg4ometry.geant4.solid.GenericTrap method), 293
- makeLayers() (pyg4ometry.geant4.solid.GenericTrap.GenericTrap method), 233
- makeLayers() (pyg4ometry.geant4.solid.TwistedBox method), 295
- makeLayers() (pyg4ometry.geant4.solid.TwistedBox.TwistedBox method), 250
- makeLayers() (pyg4ometry.geant4.solid.TwistedTrap method), 293
- makeLayers() (pyg4ometry.geant4.solid.TwistedTrap.TwistedTrap method), 232
- makeLayers() (pyg4ometry.geant4.solid.TwistedTrd method), 293
- makeLayers() (pyg4ometry.geant4.solid.TwistedTrd.TwistedTrd method), 235
- makeLayers() (pyg4ometry.geant4.solid.TwistedTubs method), 300

makeLayers() (*pyg4ometry.geant4.solid.TwistedTubs.TwistedTubs* method), 254
makeListFromPolygon() (*pyg4ometry.convert._PolygonProcessing* class method), 82
makeListFromPolygon() (*pyg4ometry.pycgal.core.PolygonProcessing* class method), 372
makeListFromPolygon() (*pyg4ometry.pycgal.PolygonProcessing* class method), 376
makeLogicalPhysicalNameSets() (*pyg4ometry.geant4.LogicalVolume* method), 335, 343
makeLogicalPhysicalNameSets() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume* method), 315
makeMaterialNameSet() (*pyg4ometry.geant4.LogicalVolume* method), 335, 344
makeMaterialNameSet() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume* method), 315
makeMatPropCard() (*pyg4ometry.fluka.material._MatPropCard* method), 144
makePolygonFromList() (*pyg4ometry.convert._PolygonProcessing* class method), 81
makePolygonFromList() (*pyg4ometry.pycgal.core.PolygonProcessing* class method), 372
makePolygonFromList() (*pyg4ometry.pycgal.PolygonProcessing* class method), 375
makeRegionsDNF() (*pyg4ometry.fluka.fluka_registry.FluxaRegistry* method), 138
makeRegionsDNF() (*pyg4ometry.fluka.FluxaRegistry* method), 170
makeShortName() (*in module pyg4ometry.convert*), 83
makeShortName() (*in module pyg4ometry.convert.geant42Fluka*), 75
makeShortName() (*in module pyg4ometry.convert.geant42FlukaBake*), 76
makeSide() (*pyg4ometry.geant4.solid._TwistedSolid* method), 293, 295, 298
makeSide() (*pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid* method), 251
makeSolidTessellated() (*pyg4ometry.geant4.LogicalVolume* method), 334, 343
makeSolidTessellated() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume* method), 314
makeStripName() (*in module pyg4ometry.convert*), 83
makeStripName() (*in module pyg4ometry.convert.geant42Fluka*), 75
makeStripName() (*in module pyg4ometry.convert.geant42FlukaBake*), 76
makeUnique() (*pyg4ometry.fluka.Region* method), 176
makeUnique() (*pyg4ometry.fluka.region.Region* method), 154
makeUnique() (*pyg4ometry.fluka.region.Zone* method), 153
makeUnique() (*pyg4ometry.fluka.Zone* method), 175
makeVisualisationOptionsDictFromPredefined() (*in module pyg4ometry.visualisation*), 407
makeVisualisationOptionsDictFromPredefined() (*in module pyg4ometry.visualisation.VisualisationOptions*), 393
makeWorldVolume() (*pyg4ometry.geant4.AssemblyVolume* method), 336
makeWorldVolume() (*pyg4ometry.geant4.AssemblyVolume.AssemblyVolume* method), 310
makeWorldVolume() (*pyg4ometry.geant4.LogicalVolume* method), 335, 344
makeWorldVolume() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume* method), 315
mask() (*pyg4ometry.fluka.fluka_registry.HalfSpaceCacher* method), 141
mask() (*pyg4ometry.fluka.fluka_registry.InfiniteCylinderCacher* method), 142
massNumberFromZ() (*pyg4ometry.convert.fluka2g4materials._PeriodicTable* method), 72
Material (class in *pyg4ometry.fluka*), 178
Material (class in *pyg4ometry.fluka.material*), 144
Material (class in *pyg4ometry.geant4*), 350
Material (class in *pyg4ometry.geant4._Material*), 327
MaterialArbitrary() (*in module pyg4ometry.geant4*), 349
MaterialArbitrary() (*in module pyg4ometry.geant4._Material*), 326
MaterialBase (class in *pyg4ometry.geant4*), 350
MaterialBase (class in *pyg4ometry.geant4._Material*), 327
MaterialCompound() (*in module pyg4ometry.geant4*), 349
MaterialCompound() (*in module pyg4ometry.geant4._Material*), 326
MaterialPredefined() (*in module pyg4ometry.geant4*), 349
MaterialPredefined() (*in module pyg4ometry.geant4._Material*), 326
materials() (*in module pyg4ometry.compare*), 67
materials() (*in module pyg4ometry.compare._Compare*), 60
MaterialSingleElement() (*in module*

- pyg4ometry.geant4*), 349
- MaterialSingleElement()** (in module *pyg4ometry.geant4._Material*), 326
- Matrix** (class in *pyg4ometry.gdml*), 224
- Matrix** (class in *pyg4ometry.gdml.Defines*), 204
- matrix2axisangle()** (in module *pyg4ometry.geant4.solid*), 275, 279, 283, 306
- matrix2axisangle()** (in module *pyg4ometry.transformation*), 422
- matrix2tbxyz()** (in module *pyg4ometry.geant4.solid*), 275, 279, 283, 306
- matrix2tbxyz()** (in module *pyg4ometry.transformation*), 422
- matrix_from()** (in module *pyg4ometry.geant4.solid*), 276, 280, 284, 307
- matrix_from()** (in module *pyg4ometry.transformation*), 423
- MatrixConvertibleMixin** (class in module *pyg4ometry.fluka.directive*), 134
- matrixElement()** (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexeme* method), 194
- matrixElement()** (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParameter* method), 188
- MatrixFromVectors()** (in module *pyg4ometry.gdml*), 224
- MatrixFromVectors()** (in module *pyg4ometry.gdml.Defines*), 204
- MAX** (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexeme* attribute), 184
- MAX** (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParameter* attribute), 193
- max()** (in module *pyg4ometry.gdml*), 221
- max()** (in module *pyg4ometry.gdml.Defines*), 201
- MAX()** (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexeme* method), 191
- md5_file()** (in module *pyg4ometry.misc*), 368
- md5_file()** (in module *pyg4ometry.misc.TestUtils*), 367
- means()** (*pyg4ometry.utils.Samples* method), 425
- Mesh** (class in *pyg4ometry.visualisation*), 404
- Mesh** (class in *pyg4ometry.visualisation.Mesh*), 388
- mesh()** (*pyg4ometry.convert.QUA* method), 92
- mesh()** (*pyg4ometry.fluka.body.QUA* method), 131
- mesh()** (*pyg4ometry.fluka.Extruder* method), 180
- mesh()** (*pyg4ometry.fluka.extruder.Extruder* method), 137
- mesh()** (*pyg4ometry.fluka.QUA* method), 169
- mesh()** (*pyg4ometry.fluka.Region* method), 175
- mesh()** (*pyg4ometry.fluka.region.Region* method), 154
- mesh()** (*pyg4ometry.fluka.region.Zone* method), 152
- mesh()** (*pyg4ometry.fluka.Zone* method), 174
- mesh()** (*pyg4ometry.geant4.solid._GenericPolyhedra* method), 270, 271, 301
- mesh()** (*pyg4ometry.geant4.solid.Box* method), 263
- mesh()** (*pyg4ometry.geant4.solid.Box.Box* method), 225
- mesh()** (*pyg4ometry.geant4.solid.Cons* method), 267
- mesh()** (*pyg4ometry.geant4.solid.Cons.Cons* method), 226
- mesh()** (*pyg4ometry.geant4.solid.CutTubs* method), 265
- mesh()** (*pyg4ometry.geant4.solid.CutTubs.CutTubs* method), 227
- mesh()** (*pyg4ometry.geant4.solid.Ellipsoid* method), 268
- mesh()** (*pyg4ometry.geant4.solid.Ellipsoid.Ellipsoid* method), 228
- mesh()** (*pyg4ometry.geant4.solid.EllipticalCone* method), 290
- mesh()** (*pyg4ometry.geant4.solid.EllipticalCone.EllipticalCone* method), 229
- mesh()** (*pyg4ometry.geant4.solid.EllipticalTube* method), 289
- mesh()** (*pyg4ometry.geant4.solid.EllipticalTube.EllipticalTube* method), 229
- mesh()** (*pyg4ometry.geant4.solid.ExtrudedSolid* method), 273
- mesh()** (*pyg4ometry.geant4.solid.ExtrudedSolid.ExtrudedSolid* method), 230
- mesh()** (*pyg4ometry.geant4.solid.GenericPolycone* method), 301
- mesh()** (*pyg4ometry.geant4.solid.GenericPolycone.GenericPolycone* method), 231
- mesh()** (*pyg4ometry.geant4.solid.GenericPolyhedra* method), 302
- mesh()** (*pyg4ometry.geant4.solid.GenericPolyhedra.GenericPolyhedra* method), 232
- mesh()** (*pyg4ometry.geant4.solid.GenericTrap* method), 303
- mesh()** (*pyg4ometry.geant4.solid.GenericTrap.GenericTrap* method), 233
- mesh()** (*pyg4ometry.geant4.solid.Hype* method), 292
- mesh()** (*pyg4ometry.geant4.solid.Hype.Hype* method), 234
- mesh()** (*pyg4ometry.geant4.solid.Intersection* method), 278
- mesh()** (*pyg4ometry.geant4.solid.Intersection.Intersection* method), 235
- mesh()** (*pyg4ometry.geant4.solid.MultiUnion* method), 305
- mesh()** (*pyg4ometry.geant4.solid.MultiUnion.MultiUnion* method), 236
- mesh()** (*pyg4ometry.geant4.solid.Orb* method), 289
- mesh()** (*pyg4ometry.geant4.solid.Orb.Orb* method), 237
- mesh()** (*pyg4ometry.geant4.solid.Para* method), 287
- mesh()** (*pyg4ometry.geant4.solid.Para.Para* method), 238
- mesh()** (*pyg4ometry.geant4.solid.Paraboloid* method), 291
- mesh()** (*pyg4ometry.geant4.solid.Paraboloid.Paraboloid* method), 239

- `mesh()` (*pyg4ometry.geant4.solid.Polycone* method), 271
- `mesh()` (*pyg4ometry.geant4.solid.Polycone.Polycone* method), 240
- `mesh()` (*pyg4ometry.geant4.solid.Polyhedra* method), 272
- `mesh()` (*pyg4ometry.geant4.solid.Polyhedra.Polyhedra* method), 241
- `mesh()` (*pyg4ometry.geant4.solid.Scaled* method), 309
- `mesh()` (*pyg4ometry.geant4.solid.Scaled.Scaled* method), 242
- `mesh()` (*pyg4ometry.geant4.solid.Sphere* method), 266
- `mesh()` (*pyg4ometry.geant4.solid.Sphere.Sphere* method), 243
- `mesh()` (*pyg4ometry.geant4.solid.Subtraction* method), 282
- `mesh()` (*pyg4ometry.geant4.solid.Subtraction.Subtraction* method), 244
- `mesh()` (*pyg4ometry.geant4.solid.TessellatedSolid* method), 304
- `mesh()` (*pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid* method), 245
- `mesh()` (*pyg4ometry.geant4.solid.Tet* method), 293
- `mesh()` (*pyg4ometry.geant4.solid.Tet.Tet* method), 246
- `mesh()` (*pyg4ometry.geant4.solid.Torus* method), 269
- `mesh()` (*pyg4ometry.geant4.solid.Torus.Torus* method), 247
- `mesh()` (*pyg4ometry.geant4.solid.Trap* method), 288
- `mesh()` (*pyg4ometry.geant4.solid.Trap.Trap* method), 248
- `mesh()` (*pyg4ometry.geant4.solid.Trd* method), 268
- `mesh()` (*pyg4ometry.geant4.solid.Trd.Trd* method), 249
- `mesh()` (*pyg4ometry.geant4.solid.Tubs* method), 264
- `mesh()` (*pyg4ometry.geant4.solid.Tubs.Tubs* method), 249
- `mesh()` (*pyg4ometry.geant4.solid.TwistedBox* method), 295
- `mesh()` (*pyg4ometry.geant4.solid.TwistedBox.TwistedBox* method), 250
- `mesh()` (*pyg4ometry.geant4.solid.TwistedTrap* method), 297
- `mesh()` (*pyg4ometry.geant4.solid.TwistedTrap.TwistedTrap* method), 252
- `mesh()` (*pyg4ometry.geant4.solid.TwistedTrd* method), 299
- `mesh()` (*pyg4ometry.geant4.solid.TwistedTrd.TwistedTrd* method), 253
- `mesh()` (*pyg4ometry.geant4.solid.TwistedTubs* method), 300
- `mesh()` (*pyg4ometry.geant4.solid.TwistedTubs.TwistedTubs* method), 254
- `mesh()` (*pyg4ometry.geant4.solid.Union* method), 274
- `mesh()` (*pyg4ometry.geant4.solid.Union.Union* method), 255
- `mesh_old()` (*pyg4ometry.geant4.solid.TwistedBox* method), 295
- `mesh_old()` (*pyg4ometry.geant4.solid.TwistedBox.TwistedBox* method), 250
- `MeshAnalysis()` (in module *pyg4ometry.freecad.Reader*), 181
- `MeshCleaning()` (in module *pyg4ometry.freecad.Reader*), 181
- `meshFromLayers()` (*pyg4ometry.geant4.solid._TwistedSolid* method), 294, 295, 298
- `meshFromLayers()` (*pyg4ometry.geant4.solid.TwistedSolid.TwistedSolid* method), 251
- `meshing` (in module *pyg4ometry.config*), 418
- `meshingNullException` (in module *pyg4ometry.config*), 418
- `meshingType` (class in *pyg4ometry.config*), 418
- `MeshShrink()` (in module *pyg4ometry.meshutils*), 420
- `MeshToFacetList()` (in module *pyg4ometry.freecad.Reader*), 181
- `mgncreat` (class in *pyg4ometry.fluka.mgnfield*), 146
- `mgnfield_array` (class in *pyg4ometry.fluka.mgnfield*), 146
- `mgnfield` (class in *pyg4ometry.fluka.mgnfield*), 146
- `mgnfield_mgncreat` (class in *pyg4ometry.fluka.mgnfield*), 146
- `MIN` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer* attribute), 184
- `MIN` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* attribute), 193
- `min()` (in module *pyg4ometry.gdml*), 221
- `min()` (in module *pyg4ometry.gdml.Defines*), 201
- `MIN()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* method), 191
- `Minus` (*pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer* attribute), 119
- `Minus` (*pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer* attribute), 101
- `Minus` (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser* attribute), 118
- `Minus` (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser* attribute), 108
- `MINUS` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer* attribute), 184
- `MINUS` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* attribute), 193
- `Minus()` (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser* method), 107
- `Minus()` (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser* method), 107
- `Minus()` (*pyg4ometry.fluka.RegionExpression.RegionParser.SubZoneContentParser.SubZoneContentParser* method), 116
- `Minus()` (*pyg4ometry.fluka.RegionExpression.RegionParser.UnaryExpressionParser.UnaryExpressionParser* method), 117
- `MINUS()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* method), 187

MINUS() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser* method), 188

mkVtkIdList() (in module *pyg4ometry.convert*), 93

mkVtkIdList() (in module *pyg4ometry.visualisation*), 414

mkVtkIdList() (in module *pyg4ometry.visualisation.Convert*), 387

modeNames (*pyg4ometry.fluka.RegionExpression.RegionLexer* attribute), 119

modeNames (*pyg4ometry.fluka.RegionExpression.RegionLexer.RegionExpression* attribute), 101

modeNames (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer* attribute), 184

module

- pyg4ometry*, 49
- pyg4ometry.analysis*, 49
- pyg4ometry.analysis.bdsimData*, 50
- pyg4ometry.analysis.Data*, 49
- pyg4ometry.analysis.flukaData*, 50
- pyg4ometry.analysis.Plot*, 50
- pyg4ometry.bdsim*, 52
- pyg4ometry.bdsim.beam*, 52
- pyg4ometry.bdsim.beamline*, 52
- pyg4ometry.bdsim.element*, 53
- pyg4ometry.bdsim.gmad*, 53
- pyg4ometry.bdsim.options*, 53
- pyg4ometry.bdsim.sampler*, 54
- pyg4ometry.bdsim.scoring*, 54
- pyg4ometry.cli*, 416
- pyg4ometry.compare*, 56
- pyg4ometry.compare._Compare*, 56
- pyg4ometry.config*, 417
- pyg4ometry.convert*, 67
- pyg4ometry.convert.fluka2g4materials*, 71
- pyg4ometry.convert.fluka2Geant4*, 67
- pyg4ometry.convert.freecad2Fluka*, 73
- pyg4ometry.convert.gdml2stl*, 73
- pyg4ometry.convert.geant42Fluka*, 74
- pyg4ometry.convert.geant42FlukaBake*, 75
- pyg4ometry.convert.geant42Geant4*, 76
- pyg4ometry.convert.geant42Vtk*, 76
- pyg4ometry.convert.oce2Geant4*, 76
- pyg4ometry.convert.stl2gdml*, 78
- pyg4ometry.convert.vis2oce*, 78
- pyg4ometry.exceptions*, 419
- pyg4ometry.features*, 94
- pyg4ometry.features._accelerator*, 94
- pyg4ometry.features.algos*, 95
- pyg4ometry.fluka*, 100
- pyg4ometry.fluka.body*, 120
- pyg4ometry.fluka.boolean_algebra*, 131
- pyg4ometry.fluka.card*, 133
- pyg4ometry.fluka.directive*, 134
- pyg4ometry.fluka.elcfield*, 136
- pyg4ometry.fluka.ExtrudedArea*, 136
- pyg4ometry.fluka.flair*, 137
- pyg4ometry.fluka.fluka_registry*, 138
- pyg4ometry.fluka.lattice*, 142
- pyg4ometry.fluka.material*, 143
- pyg4ometry.fluka.mgnfield*, 146
- pyg4ometry.fluka.preprocessor*, 146
- pyg4ometry.fluka.reader*, 148
- pyg4ometry.fluka.region*, 150
- pyg4ometry.fluka.RegionExpression*, 100
- pyg4ometry.fluka.RegionExpression.RegionLexer*, 100
- pyg4ometry.fluka.RegionExpression.RegionParser*, 102
- pyg4ometry.fluka.RegionExpression.RegionParserVisitor*, 109
- pyg4ometry.fluka.vector*, 155
- pyg4ometry.fluka.vis*, 157
- pyg4ometry.fluka.Writer*, 119
- pyg4ometry.freecad*, 180
- pyg4ometry.freecad.Reader*, 180
- pyg4ometry.gdml*, 182
- pyg4ometry.gdml.Defines*, 196
- pyg4ometry.gdml.GdmlExpression*, 182
- pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval*, 182
- pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer*, 183
- pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser*, 185
- pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor*, 194
- pyg4ometry.gdml.Reader*, 205
- pyg4ometry.gdml.Units*, 207
- pyg4ometry.gdml.Writer*, 207
- pyg4ometry.geant4*, 225
- pyg4ometry.geant4._Material*, 324
- pyg4ometry.geant4.AssemblyVolume*, 309
- pyg4ometry.geant4.BorderSurface*, 310
- pyg4ometry.geant4.DivisionVolume*, 311
- pyg4ometry.geant4.LogicalVolume*, 312
- pyg4ometry.geant4.Loop*, 316
- pyg4ometry.geant4.ParameterisedVolume*, 316
- pyg4ometry.geant4.PhysicalVolume*, 317
- pyg4ometry.geant4.Registry*, 318
- pyg4ometry.geant4.ReplicaVolume*, 322
- pyg4ometry.geant4.SkinSurface*, 323
- pyg4ometry.geant4.solid*, 225
- pyg4ometry.geant4.solid.Box*, 225
- pyg4ometry.geant4.solid.Cons*, 226
- pyg4ometry.geant4.solid.CutTubs*, 227
- pyg4ometry.geant4.solid.Ellipsoid*, 227

- pyg4ometry.geant4.solid.EllipticalCone, 228
- pyg4ometry.geant4.solid.EllipticalTube, 229
- pyg4ometry.geant4.solid.ExtrudedSolid, 230
- pyg4ometry.geant4.solid.GenericPolycone, 230
- pyg4ometry.geant4.solid.GenericPolyhedra, 231
- pyg4ometry.geant4.solid.GenericTrap, 232
- pyg4ometry.geant4.solid.Hype, 233
- pyg4ometry.geant4.solid.Intersection, 234
- pyg4ometry.geant4.solid.Layer, 235
- pyg4ometry.geant4.solid.MultiUnion, 235
- pyg4ometry.geant4.solid.OpticalSurface, 236
- pyg4ometry.geant4.solid.Orb, 237
- pyg4ometry.geant4.solid.Para, 238
- pyg4ometry.geant4.solid.Paraboloid, 238
- pyg4ometry.geant4.solid.Plane, 239
- pyg4ometry.geant4.solid.Polycone, 240
- pyg4ometry.geant4.solid.Polyhedra, 241
- pyg4ometry.geant4.solid.Scaled, 241
- pyg4ometry.geant4.solid.SolidBase, 242
- pyg4ometry.geant4.solid.Sphere, 243
- pyg4ometry.geant4.solid.Subtraction, 244
- pyg4ometry.geant4.solid.TessellatedSolid, 244
- pyg4ometry.geant4.solid.Tet, 246
- pyg4ometry.geant4.solid.Torus, 246
- pyg4ometry.geant4.solid.Trap, 247
- pyg4ometry.geant4.solid.Trd, 248
- pyg4ometry.geant4.solid.Tubs, 249
- pyg4ometry.geant4.solid.TwistedBox, 250
- pyg4ometry.geant4.solid.TwistedSolid, 251
- pyg4ometry.geant4.solid.TwistedTrap, 251
- pyg4ometry.geant4.solid.TwistedTrd, 252
- pyg4ometry.geant4.solid.TwistedTubs, 253
- pyg4ometry.geant4.solid.TwoVector, 254
- pyg4ometry.geant4.solid.Union, 255
- pyg4ometry.geant4.solid.Wedge, 256
- pyg4ometry.geant4.SurfaceBase, 323
- pyg4ometry.gui, 352
- pyg4ometry.gui.example1, 356
- pyg4ometry.gui.GeometryModel, 352
- pyg4ometry.gui.MainWindow, 352
- pyg4ometry.gui.QVTKRenderWindowInteractor, 353
- pyg4ometry.io, 365
- pyg4ometry.io.R00TTGeo, 365
- pyg4ometry.meshutils, 420
- pyg4ometry.misc, 366
- pyg4ometry.misc.NestedSolids, 366
- pyg4ometry.misc.TestUtils, 366
- pyg4ometry.montecarlo, 368
- pyg4ometry.montecarlo.beam, 368
- pyg4ometry.montecarlo.boundary, 369
- pyg4ometry.montecarlo.config, 369
- pyg4ometry.montecarlo.scoring, 369
- pyg4ometry.pycgal, 370
- pyg4ometry.pycgal.core, 371
- pyg4ometry.pycgal.HalfPlane, 370
- pyg4ometry.pycgal.pythonHelpers, 373
- pyg4ometry.pycgal_old, 376
- pyg4ometry.pycgal_old.cgal, 376
- pyg4ometry.pycsg, 381
- pyg4ometry.pyoce, 382
- pyg4ometry.pyoce.pythonHelpers, 382
- pyg4ometry.pyoce.Reader, 382
- pyg4ometry.stl, 383
- pyg4ometry.stl.Reader, 383
- pyg4ometry.transformation, 420
- pyg4ometry.utils, 424
- pyg4ometry.visualisation, 387
- pyg4ometry.visualisation.Convert, 387
- pyg4ometry.visualisation.Mesh, 387
- pyg4ometry.visualisation.Plot, 388
- pyg4ometry.visualisation.RenderWriter, 389
- pyg4ometry.visualisation.ViewerBase, 389
- pyg4ometry.visualisation.VisualisationOptions, 392
- pyg4ometry.visualisation.VtkExporter, 393
- pyg4ometry.visualisation.VtkViewer, 395
- pyg4ometry.visualisation.VtkViewerNew, 400
- pyg4ometry.visualisation.Writer, 403
- MouseInteractorNamePhysicalVolume (class in *pyg4ometry.visualisation*), 411, 413
- MouseInteractorNamePhysicalVolume (class in *pyg4ometry.visualisation.VtkViewer*), 400
- MouseInteractorNamePhysicalVolume (class in *pyg4ometry.visualisation.VtkViewerNew*), 402
- mouseMoveEvent() (*pyg4ometry.gui.example1.QVTKRenderWindowInteractor* method), 358
- mouseMoveEvent() (*pyg4ometry.gui.QVTKRenderWindowInteractor* method), 361, 364
- mouseMoveEvent() (*pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor* method), 355
- mousePressEvent() (*pyg4ometry.gui.example1.QVTKRenderWindowInteractor* method), 358
- mousePressEvent() (*pyg4ometry.gui.QVTKRenderWindowInteractor* method), 361, 364
- mousePressEvent() (*pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor* method), 355
- mouseReleaseEvent() (*pyg4ometry.gui.example1.QVTKRenderWindowInteractor* method), 358

- method), 358
- mouseReleaseEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor method), 361, 364
- mouseReleaseEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method), 355
- multiGroupNeutronCrossSections (class in pyg4ometry.fluka.material), 144
- multiplyingExpression() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 194
- multiplyingExpression() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 186
- MultiUnion (class in pyg4ometry.geant4.solid), 305
- MultiUnion (class in pyg4ometry.geant4.solid.MultiUnion), 235
- ## N
- n_samples_description() (pyg4ometry.utils.Samples method), 425
- name (pyg4ometry.geant4.solid._SolidBase property), 261–269, 272, 273, 277, 281, 285–293, 295, 297, 299–304, 308
- name (pyg4ometry.geant4.solid.SolidBase property), 309
- name (pyg4ometry.geant4.solid.SolidBase.SolidBase property), 242
- nameFromPath() (in module pyg4ometry.gui), 364
- nameFromPath() (in module pyg4ometry.gui.GeometryModel), 352
- nefpolyhedron_print (in module pyg4ometry.pycgal_old), 381
- nefpolyhedron_print (in module pyg4ometry.pycgal_old.cgal), 379
- nefpolyhedron_to_convexpolyhedra (in module pyg4ometry.pycgal_old), 381
- nefpolyhedron_to_convexpolyhedra (in module pyg4ometry.pycgal_old.cgal), 379
- nefPolyhedron_to_convexPolyhedra() (pyg4ometry.pycgal.core.PolyhedronProcessing class method), 373
- nefPolyhedron_to_convexPolyhedra() (pyg4ometry.pycgal.PolyhedronProcessing class method), 376
- nefpolyhedron_to_polyhedron (in module pyg4ometry.pycgal_old), 381
- nefpolyhedron_to_polyhedron (in module pyg4ometry.pycgal_old.cgal), 379
- nefpolyhedron_to_surfacemesh (in module pyg4ometry.pycgal_old), 381
- nefpolyhedron_to_surfacemesh (in module pyg4ometry.pycgal_old.cgal), 379
- nefpolyhedron_write (in module pyg4ometry.pycgal_old), 381
- nefpolyhedron_write (in module pyg4ometry.pycgal_old.cgal), 379
- NestedBoxes() (in module pyg4ometry.misc), 367
- NestedBoxes() (pyg4ometry.misc.NestedSolids), 366
- netExpansion() (pyg4ometry.fluka.directive.MatrixConvertibleMixin method), 134
- netTranslation() (pyg4ometry.fluka.directive.MatrixConvertibleMixin method), 134
- Newline (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 118
- Newline (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
- Newline (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 118
- Newline (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
- nist_element_2geant4Element() (in module pyg4ometry.geant4), 349
- nist_element_2geant4Element() (in module pyg4ometry.geant4._Material), 326
- nist_material_2geant4Material() (in module pyg4ometry.geant4), 349
- nist_material_2geant4Material() (in module pyg4ometry.geant4._Material), 326
- nist_materials_name_lookup() (in module pyg4ometry.geant4), 349
- nist_materials_name_lookup() (in module pyg4ometry.geant4._Material), 326
- nist_materials_z_lookup() (in module pyg4ometry.geant4), 349
- nist_materials_z_lookup() (in module pyg4ometry.geant4._Material), 326
- nonesToZero() (pyg4ometry.fluka.Card method), 180
- nonesToZero() (pyg4ometry.fluka.card.Card method), 133
- nonzero() (pyg4ometry.gdml.Defines.VectorBase method), 203
- nonzero() (pyg4ometry.gdml.VectorBase method), 223
- NotTested (pyg4ometry.compare._Compare.TestResult attribute), 58
- NotTested (pyg4ometry.compare.TestResult attribute), 65
- nslice (pyg4ometry.config.SolidDefaults.Cons attribute), 418
- nslice (pyg4ometry.config.SolidDefaults.CutTubs attribute), 418
- nslice (pyg4ometry.config.SolidDefaults.Ellipsoid attribute), 418
- nslice (pyg4ometry.config.SolidDefaults.EllipticalCone attribute), 418
- nslice (pyg4ometry.config.SolidDefaults.EllipticalTube attribute), 418

- attribute), 418
- nslice (pyg4ometry.config.SolidDefaults.GenericPolycone attribute), 418
- nslice (pyg4ometry.config.SolidDefaults.Hype attribute), 418
- nslice (pyg4ometry.config.SolidDefaults.Orb attribute), 419
- nslice (pyg4ometry.config.SolidDefaults.Paraboloid attribute), 419
- nslice (pyg4ometry.config.SolidDefaults.Polycone attribute), 419
- nslice (pyg4ometry.config.SolidDefaults.Sphere attribute), 419
- nslice (pyg4ometry.config.SolidDefaults.Torus attribute), 419
- nslice (pyg4ometry.config.SolidDefaults.Tubs attribute), 418
- nslice (pyg4ometry.config.SolidDefaults.TwistedTubs attribute), 419
- nslice (pyg4ometry.config.SolidDefaults.Wedge attribute), 419
- nstack (pyg4ometry.config.SolidDefaults.Ellipsoid attribute), 418
- nstack (pyg4ometry.config.SolidDefaults.EllipticalCone attribute), 418
- nstack (pyg4ometry.config.SolidDefaults.EllipticalTube attribute), 418
- nstack (pyg4ometry.config.SolidDefaults.Hype attribute), 418
- nstack (pyg4ometry.config.SolidDefaults.Orb attribute), 419
- nstack (pyg4ometry.config.SolidDefaults.Paraboloid attribute), 419
- nstack (pyg4ometry.config.SolidDefaults.Sphere attribute), 419
- nstack (pyg4ometry.config.SolidDefaults.Torus attribute), 419
- nstack (pyg4ometry.config.SolidDefaults.TwistedBox attribute), 419
- nstack (pyg4ometry.config.SolidDefaults.TwistedTrap attribute), 419
- nstack (pyg4ometry.config.SolidDefaults.TwistedTrd attribute), 419
- nstack (pyg4ometry.config.SolidDefaults.TwistedTubs attribute), 419
- nTermsDNF() (in module pyg4ometry.fluka.boolean_algebra), 133
- NullMeshError, 419
- NullModel, 69
- numpyPolygonConvex() (in module pyg4ometry.pycgal_old), 381
- numpyPolygonConvex() (in module pyg4ometry.pycgal_old.cgal), 379
- O
- object1() (pyg4ometry.geant4.solid.Intersection method), 278
- object1() (pyg4ometry.geant4.solid.Intersection.Intersection method), 235
- object1() (pyg4ometry.geant4.solid.Subtraction method), 282
- object1() (pyg4ometry.geant4.solid.Subtraction.Subtraction method), 244
- object1() (pyg4ometry.geant4.solid.Union method), 274
- object1() (pyg4ometry.geant4.solid.Union.Union method), 255
- object2() (pyg4ometry.geant4.solid.Intersection method), 278
- object2() (pyg4ometry.geant4.solid.Intersection.Intersection method), 235
- object2() (pyg4ometry.geant4.solid.Subtraction method), 282
- object2() (pyg4ometry.geant4.solid.Subtraction.Subtraction method), 244
- object2() (pyg4ometry.geant4.solid.Union method), 274
- object2() (pyg4ometry.geant4.solid.Union.Union method), 255
- oce2Geant4() (in module pyg4ometry.convert), 94
- oce2Geant4() (in module pyg4ometry.convert.oce2Geant4), 78
- oceShape_Geant4_Assembly() (in module pyg4ometry.convert), 93
- oceShape_Geant4_Assembly() (in module pyg4ometry.convert.oce2Geant4), 77
- oceShape_Geant4_LogicalVolume() (in module pyg4ometry.convert), 93
- oceShape_Geant4_LogicalVolume() (in module pyg4ometry.convert.oce2Geant4), 77
- oceShape_Geant4_Tessellated() (in module pyg4ometry.convert), 93
- oceShape_Geant4_Tessellated() (in module pyg4ometry.convert.oce2Geant4), 77
- op_map (pyg4ometry.fluka.preprocessor._Calc attribute), 148
- openFileDialog() (pyg4ometry.gui.MainWindow method), 362
- openFileDialog() (pyg4ometry.gui.MainWindow.MainWindow method), 353
- operationReturnType() (in module pyg4ometry.gdml), 218
- operationReturnType() (in module pyg4ometry.gdml.Defines), 198
- operatorAddSub() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionP method), 194
- operatorAddSub() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionP

- method), 186
- operatorMulDiv() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 194
- operatorMulDiv() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 186
- OpticalSurface (class in pyg4ometry.geant4.solid), 285
- OpticalSurface (class in pyg4ometry.geant4.solid.OpticalSurface), 236
- Options (class in pyg4ometry.bdsim), 55
- Options (class in pyg4ometry.bdsim.options), 53
- Orb (class in pyg4ometry.geant4.solid), 288
- Orb (class in pyg4ometry.geant4.solid.Orb), 237
- orderLogicalVolumes() (pyg4ometry.geant4.Registry method), 347
- orderLogicalVolumes() (pyg4ometry.geant4.Registry.Registry method), 320
- orderMaterials() (pyg4ometry.geant4.Registry method), 346
- orderMaterials() (pyg4ometry.geant4.Registry.Registry method), 320
- overlap (pyg4ometry.geant4._OverlapType attribute), 332, 337
- overlap (pyg4ometry.visualisation.Mesh.OverlapType attribute), 388
- overlap (pyg4ometry.visualisation.OverlapType attribute), 405
- OverlapType (class in pyg4ometry.visualisation), 405
- OverlapType (class in pyg4ometry.visualisation.Mesh), 388
- P**
- paintEngine() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor method), 358
- paintEngine() (pyg4ometry.gui.QVTKRenderWindowInteractor method), 361, 364
- paintEngine() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method), 355
- paintEvent() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor method), 358
- paintEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor method), 361, 364
- paintEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method), 355
- Para (class in pyg4ometry.geant4.solid), 286
- Para (class in pyg4ometry.geant4.solid.Para), 238
- Paraboloid (class in pyg4ometry.geant4.solid), 291
- Paraboloid (class in pyg4ometry.geant4.solid.Paraboloid), 238
- parallel_to() (pyg4ometry.fluka.Three method), 172
- parallel_to() (pyg4ometry.fluka.vector.Three method), 156
- ParameterisedVolume (class in pyg4ometry.geant4), 339
- ParameterisedVolume (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.BoxDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.BoxDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.ConeDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.ConeDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.EllipsoidDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.EllipsoidDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 317
- ParameterisedVolume.HypeDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.HypeDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.OrbDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.OrbDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.ParaDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.ParaDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.PolyconeDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.PolyconeDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 317
- ParameterisedVolume.PolyhedraDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.PolyhedraDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 317
- ParameterisedVolume.SphereDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.SphereDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.TorusDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.TorusDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.TrapDimensions (class in pyg4ometry.geant4), 339
- ParameterisedVolume.TrapDimensions (class in pyg4ometry.geant4.ParameterisedVolume), 316
- ParameterisedVolume.TrdDimensions (class in pyg4ometry.geant4), 339

ParameterisedVolume.TrdDimensions (class in *pyg4ometry.geant4.ParameterisedVolume*), 316
 ParameterisedVolume.TubeDimensions (class in *pyg4ometry.geant4*), 339
 ParameterisedVolume.TubeDimensions (class in *pyg4ometry.geant4.ParameterisedVolume*), 316
 parameterisedVolumes() (in module *pyg4ometry.compare*), 67
 parameterisedVolumes() (in module *pyg4ometry.compare._Compare*), 60
 parse() (*pyg4ometry.gdml.GdmlExpression.ExpressionParser* method), 196
 parse() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser* method), 183
 parseBox() (*pyg4ometry.gdml.Reader* method), 212
 parseBox() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseCone() (*pyg4ometry.gdml.Reader* method), 212
 parseCone() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseCutTube() (*pyg4ometry.gdml.Reader* method), 212
 parseCutTube() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseDefines() (*pyg4ometry.gdml.Reader* method), 212
 parseDefines() (*pyg4ometry.gdml.Reader.Reader* method), 205
 parseEllipsoid() (*pyg4ometry.gdml.Reader* method), 212
 parseEllipsoid() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseEllipticalCone() (*pyg4ometry.gdml.Reader* method), 212
 parseEllipticalCone() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseEllipticalTube() (*pyg4ometry.gdml.Reader* method), 212
 parseEllipticalTube() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseExtrudedSolid() (*pyg4ometry.gdml.Reader* method), 212
 parseExtrudedSolid() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseGenericPolycone() (*pyg4ometry.gdml.Reader* method), 212
 parseGenericPolycone() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseGenericPolyhedra() (*pyg4ometry.gdml.Reader* method), 212
 parseGenericPolyhedra() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseGenericTrap() (*pyg4ometry.gdml.Reader* method), 213
 parseGenericTrap() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseHype() (*pyg4ometry.gdml.Reader* method), 212
 parseHype() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseIntersection() (*pyg4ometry.gdml.Reader* method), 213
 parseIntersection() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseMaterials() (*pyg4ometry.gdml.Reader* method), 212
 parseMaterials() (*pyg4ometry.gdml.Reader.Reader* method), 205
 parseMultiUnion() (*pyg4ometry.gdml.Reader* method), 213
 parseMultiUnion() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseOpticalSurface() (*pyg4ometry.gdml.Reader* method), 213
 parseOpticalSurface() (*pyg4ometry.gdml.Reader.Reader* method), 207
 parseOrb() (*pyg4ometry.gdml.Reader* method), 212
 parseOrb() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parsePara() (*pyg4ometry.gdml.Reader* method), 212
 parsePara() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseParaboloid() (*pyg4ometry.gdml.Reader* method), 212
 parseParaboloid() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parsePhysicalVolumeChildren() (*pyg4ometry.gdml.Reader* method), 213
 parsePhysicalVolumeChildren() (*pyg4ometry.gdml.Reader.Reader* method), 207
 parsePolycone() (*pyg4ometry.gdml.Reader* method), 212
 parsePolycone() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parsePolyhedra() (*pyg4ometry.gdml.Reader* method), 212
 parsePolyhedra() (*pyg4ometry.gdml.Reader.Reader* method), 206
 parseScaledSolid() (*pyg4ometry.gdml.Reader* method), 213
 parseScaledSolid() (*pyg4ometry.gdml.Reader.Reader* method), 206

- method), 207
- parseSolidLoop() (pyg4ometry.gdml.Reader method), 213
- parseSolidLoop() (pyg4ometry.gdml.Reader.Reader method), 207
- parseSolids() (pyg4ometry.gdml.Reader method), 212
- parseSolids() (pyg4ometry.gdml.Reader.Reader method), 205
- parseSphere() (pyg4ometry.gdml.Reader method), 212
- parseSphere() (pyg4ometry.gdml.Reader.Reader method), 206
- parseStructure() (pyg4ometry.gdml.Reader method), 213
- parseStructure() (pyg4ometry.gdml.Reader.Reader method), 207
- parseSubtraction() (pyg4ometry.gdml.Reader method), 213
- parseSubtraction() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTessellatedSolid() (pyg4ometry.gdml.Reader method), 213
- parseTessellatedSolid() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTet() (pyg4ometry.gdml.Reader method), 212
- parseTet() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTorus() (pyg4ometry.gdml.Reader method), 212
- parseTorus() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTrap() (pyg4ometry.gdml.Reader method), 212
- parseTrap() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTrd() (pyg4ometry.gdml.Reader method), 212
- parseTrd() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTube() (pyg4ometry.gdml.Reader method), 212
- parseTube() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTwistedBox() (pyg4ometry.gdml.Reader method), 212
- parseTwistedBox() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTwistedTrap() (pyg4ometry.gdml.Reader method), 213
- parseTwistedTrap() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTwistedTrd() (pyg4ometry.gdml.Reader method), 213
- parseTwistedTrd() (pyg4ometry.gdml.Reader.Reader method), 206
- parseTwistedTubs() (pyg4ometry.gdml.Reader method), 213
- parseTwistedTubs() (pyg4ometry.gdml.Reader.Reader method), 206
- parseUnion() (pyg4ometry.gdml.Reader method), 213
- parseUnion() (pyg4ometry.gdml.Reader.Reader method), 206
- parseUserInfo() (pyg4ometry.gdml.Reader method), 212
- parseUserInfo() (pyg4ometry.gdml.Reader.Reader method), 205
- parseVector() (pyg4ometry.gdml.Reader method), 212
- parseVector() (pyg4ometry.gdml.Reader.Reader method), 205
- part2Region() (in module pyg4ometry.convert), 93
- part2Region() (in module pyg4ometry.convert.freecad2Fluka), 73
- PartFeatureGlobalPlacement() (in module pyg4ometry.freecad.Reader), 181
- Passed (pyg4ometry.compare._Compare.TestResult attribute), 58
- Passed (pyg4ometry.compare.TestResult attribute), 65
- PhysicalVolume (class in pyg4ometry.geant4), 335, 344
- PhysicalVolume (class in pyg4ometry.geant4.PhysicalVolume), 317
- physicalVolumes() (in module pyg4ometry.compare), 66
- physicalVolumes() (in module pyg4ometry.compare._Compare), 60
- PI (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpression attribute), 184
- PI (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpression attribute), 193
- PI() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpression method), 189
- PLA (class in pyg4ometry.convert), 88
- PLA (class in pyg4ometry.fluka), 165
- PLA (class in pyg4ometry.fluka.body), 127
- Plane (class in pyg4ometry.features), 99
- Plane (class in pyg4ometry.features.algos), 96
- Plane (class in pyg4ometry.geant4.solid), 261
- Plane (class in pyg4ometry.geant4.solid.Plane), 239
- plane() (pyg4ometry.features.algos.CoordinateSystem method), 97
- plane() (pyg4ometry.features.CoordinateSystem method), 99
- plot() (pyg4ometry.analysis.Data.TH1 method), 49
- plot() (pyg4ometry.analysis.Data.TH2 method), 49
- plot() (pyg4ometry.analysis.Data.TH3 method), 50
- plot() (pyg4ometry.fluka.Extruder method), 180
- plot() (pyg4ometry.fluka.extruder.Extruder method), 137
- plot3Projections() (pyg4ometry.analysis.Data.TH3 method), 50
- plotCutter() (pyg4ometry.features.algos.FeatureData method), 98
- plotCutter() (pyg4ometry.features.FeatureData method), 98

<i>method</i>), 100			240
plotEventSection() (in module <i>pyg4ometry.analysis.Plot</i>), 50		<i>polyhedron</i> (class in <i>pyg4ometry.geant4.solid</i>), 272	
plotEventSections() (in module <i>pyg4ometry.analysis.Plot</i>), 50		<i>Polyhedra</i> (class in <i>pyg4ometry.geant4.solid.Polyhedra</i>), 272	
plotFeature() (<i>pyg4ometry.features.algos.FeatureData</i> <i>method</i>), 98		<i>polyhedron_print</i> (in module <i>pyg4ometry.pycgal_old</i>), 381	
plotFeature() (<i>pyg4ometry.features.FeatureData</i> <i>method</i>), 100		<i>polyhedron_print</i> (in module <i>pyg4ometry.pycgal_old.cgal</i>), 379	
plotScoringMeshSection() (in module <i>pyg4ometry.analysis.Plot</i>), 50		<i>polyhedron_to_nefpolyhedron</i> (in module <i>pyg4ometry.pycgal_old.cgal</i>), 379	
Plus (<i>pyg4ometry.fluka.RegionExpression.RegionLexer</i> <i>attribute</i>), 119		<i>polyhedron_to_numpyArrayPlanes</i> (in module <i>pyg4ometry.pycgal.PolyhedronProcessing</i>), 376	
Plus (<i>pyg4ometry.fluka.RegionExpression.RegionLexer.RegionParser</i> <i>attribute</i>), 101		<i>Position</i> (class in <i>pyg4ometry.gdml</i>), 223	
Plus (<i>pyg4ometry.fluka.RegionExpression.RegionParser</i> <i>attribute</i>), 118		<i>Position</i> (class in <i>pyg4ometry.gdml.Defines</i>), 203	
Plus (<i>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</i> <i>attribute</i>), 108		<i>POW</i> (<i>pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionParser</i> <i>attribute</i>), 184	
PLUS (<i>pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionParser</i> <i>attribute</i>), 184			
PLUS (<i>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser</i> <i>attribute</i>), 193			
Plus() (<i>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</i> <i>method</i>), 107			
Plus() (<i>pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser</i> <i>method</i>), 107			
Plus() (<i>pyg4ometry.fluka.RegionExpression.RegionParser.SubZoneContext</i> <i>method</i>), 116			
Plus() (<i>pyg4ometry.fluka.RegionExpression.RegionParser.UnaryExpressionContext</i> <i>method</i>), 117			
PLUS() (<i>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser</i> <i>method</i>), 187			
PLUS() (<i>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser</i> <i>method</i>), 188			
POINT (<i>pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionParser</i> <i>attribute</i>), 184			
POINT (<i>pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser</i> <i>attribute</i>), 193			
point() (<i>pyg4ometry.convert.XCC</i> <i>method</i>), 89			
point() (<i>pyg4ometry.convert.YCC</i> <i>method</i>), 90			
point() (<i>pyg4ometry.convert.ZCC</i> <i>method</i>), 90			
point() (<i>pyg4ometry.fluka.body.XCC</i> <i>method</i>), 128			
point() (<i>pyg4ometry.fluka.body.YCC</i> <i>method</i>), 128			
point() (<i>pyg4ometry.fluka.body.ZCC</i> <i>method</i>), 129			
point() (<i>pyg4ometry.fluka.XCC</i> <i>method</i>), 166			
point() (<i>pyg4ometry.fluka.YCC</i> <i>method</i>), 166			
point() (<i>pyg4ometry.fluka.ZCC</i> <i>method</i>), 167			
pointOnLineClosestToPoint() (in module <i>pyg4ometry.fluka.vector</i>), 157			
pointOnPlaneClosestToPoint() (in module <i>pyg4ometry.fluka.vector</i>), 157			
Polycone (class in <i>pyg4ometry.geant4.solid</i>), 270			
Polycone (class in <i>pyg4ometry.geant4.solid.Polycone</i>), 270			

POW (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser module attribute), 193
 pow() (in module pyg4ometry.gdml), 220
 pow() (in module pyg4ometry.gdml.Defines), 200
 POW() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser module attribute), 187
 POWER (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser module attribute), 184
 POWER (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser module attribute), 193
 POWER() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser module attribute), 191
 powExpression() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser module attribute), 194
 powExpression() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser module attribute), 186
 predefinedMaterialNames() (in module pyg4ometry.fluka.material), 144
 preprocess() (in module pyg4ometry.fluka.preprocessor), 147
 print() (pyg4ometry.compare._Compare.ComparisonResult method), 59
 print() (pyg4ometry.compare.ComparisonResult method), 65
 print_header() (pyg4ometry.analysis.flukaData._FlukaData method), 51
 print_header() (pyg4ometry.analysis.flukaData.Usrbin method), 52
 printAllTestNames() (pyg4ometry.compare._Compare.Tests class method), 58
 printAllTestNames() (pyg4ometry.compare.Tests class method), 64
 printDefinitions() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 138
 printDefinitions() (pyg4ometry.fluka.FlukaRegistry method), 170
 printDumps() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 140
 printDumps() (pyg4ometry.fluka.FlukaRegistry method), 171
 printPartFeatures() (pyg4ometry.freecad.Reader.Reader method), 181
 printStats() (pyg4ometry.geant4.Registry method), 347
 printStats() (pyg4ometry.geant4.Registry.Registry method), 321
 printStructure() (pyg4ometry.freecad.Reader.Reader method), 181
 printSummary() (pyg4ometry.geant4.GeometryComplexityInformation module method), 348
 printSummary() (pyg4ometry.geant4.Registry.GeometryComplexityInformation module method), 321
 printViewParameters() (pyg4ometry.visualisation.VtkViewer method), 410
 printViewParameters() (pyg4ometry.visualisation.VtkViewer method), 399
 printViewParameters() (pyg4ometry.visualisation.Mesh.OverlapType attribute), 388
 printViewParameters() (pyg4ometry.visualisation.OverlapType attribute), 405
 printViewParameters() (pyg4ometry.visualisation.OverlapType attribute), 411
 printViewParameters() (in module pyg4ometry.visualisation.VtkViewer), 400
 pycsg (pyg4ometry.config.meshingType attribute), 418
 pycsgmesh() (pyg4ometry.geant4.solid.Plane method), 262
 pycsgmesh() (pyg4ometry.geant4.solid.Plane.Plane method), 239
 pycsgmesh() (pyg4ometry.geant4.solid.Wedge method), 262
 pycsgmesh() (pyg4ometry.geant4.solid.Wedge.Wedge method), 256
 pycsgmesh2FlukaRegion() (in module pyg4ometry.convert), 83
 pycsgmesh2FlukaRegion() (in module pyg4ometry.convert.geant42Fluka), 75
 pycsgmesh2FlukaRegion() (in module pyg4ometry.convert.geant42FlukaBake), 76
 pycsgmesh2NefPolyhedron() (in module pyg4ometry.pycgal_old), 381
 pycsgmesh2NefPolyhedron() (in module pyg4ometry.pycgal_old.cgal), 379
 pycsgMeshToObj() (in module pyg4ometry.convert), 93
 pycsgMeshToObj() (in module pyg4ometry.visualisation), 414
 pycsgMeshToObj() (in module pyg4ometry.visualisation.Convert), 387
 pycsgMeshToStl() (in module pyg4ometry.convert), 93
 pycsgMeshToStl() (in module pyg4ometry.visualisation), 414
 pycsgMeshToStl() (in module pyg4ometry.visualisation.Convert), 387
 pycsgMeshToVtkPolyData() (in module pyg4ometry.convert), 93
 pycsgMeshToVtkPolyData() (in module pyg4ometry.visualisation), 414
 pycsgMeshToVtkPolyData() (in module pyg4ometry.visualisation.Convert), 387

pyg4ometry.visualisation.Convert), 387
 pycsgmeshWritePolygon() (in module *pyg4ometry.pycgal_old*), 381
 pycsgmeshWritePolygon() (in module *pyg4ometry.pycgal_old.cgal*), 379
 pyg42VtkTransformation() (in module *pyg4ometry.convert*), 93
 pyg42VtkTransformation() (in module *pyg4ometry.visualisation*), 414
 pyg42VtkTransformation() (in module *pyg4ometry.visualisation.Convert*), 387
 pyg4ArrayToVtkPolydataLine() (in module *pyg4ometry.features.algos*), 98
 pyg4ometry
 module, 49
 pyg4ometry.analysis
 module, 49
 pyg4ometry.analysis.bdsimData
 module, 50
 pyg4ometry.analysis.Data
 module, 49
 pyg4ometry.analysis.flukaData
 module, 50
 pyg4ometry.analysis.Plot
 module, 50
 pyg4ometry.bdsim
 module, 52
 pyg4ometry.bdsim.beam
 module, 52
 pyg4ometry.bdsim.beamline
 module, 52
 pyg4ometry.bdsim.element
 module, 53
 pyg4ometry.bdsim.gmad
 module, 53
 pyg4ometry.bdsim.options
 module, 53
 pyg4ometry.bdsim.sampler
 module, 54
 pyg4ometry.bdsim.scoring
 module, 54
 pyg4ometry.cli
 module, 416
 pyg4ometry.compare
 module, 56
 pyg4ometry.compare._Compare
 module, 56
 pyg4ometry.config
 module, 417
 pyg4ometry.convert
 module, 67
 pyg4ometry.convert.fluka2g4materials
 module, 71
 pyg4ometry.convert.fluka2Geant4
 module, 67
 pyg4ometry.convert.freecad2Fluka
 module, 73
 pyg4ometry.convert.gdml2stl
 module, 73
 pyg4ometry.convert.geant42Fluka
 module, 74
 pyg4ometry.convert.geant42FlukaBake
 module, 75
 pyg4ometry.convert.geant42Geant4
 module, 76
 pyg4ometry.convert.geant42Vtk
 module, 76
 pyg4ometry.convert.oce2Geant4
 module, 76
 pyg4ometry.convert.stl2gdml
 module, 78
 pyg4ometry.convert.vis2oce
 module, 78
 pyg4ometry.exceptions
 module, 419
 pyg4ometry.features
 module, 94
 pyg4ometry.features._accelerator
 module, 94
 pyg4ometry.features.algos
 module, 95
 pyg4ometry.fluka
 module, 100
 pyg4ometry.fluka.body
 module, 120
 pyg4ometry.fluka.boolean_algebra
 module, 131
 pyg4ometry.fluka.card
 module, 133
 pyg4ometry.fluka.directive
 module, 134
 pyg4ometry.fluka.elcfield
 module, 136
 pyg4ometry.fluka.extruder
 module, 136
 pyg4ometry.fluka.flair
 module, 137
 pyg4ometry.fluka.fluka_registry
 module, 138
 pyg4ometry.fluka.lattice
 module, 142
 pyg4ometry.fluka.material
 module, 143
 pyg4ometry.fluka.mgnfield
 module, 146
 pyg4ometry.fluka.preprocessor
 module, 146
 pyg4ometry.fluka.reader

module, 148	module, 316
pyg4ometry.fluka.region	pyg4ometry.geant4.ParameterisedVolume
module, 150	module, 316
pyg4ometry.fluka.RegionExpression	pyg4ometry.geant4.PhysicalVolume
module, 100	module, 317
pyg4ometry.fluka.RegionExpression.RegionLexer	pyg4ometry.geant4.Registry
module, 100	module, 318
pyg4ometry.fluka.RegionExpression.RegionParser	pyg4ometry.geant4.ReplicaVolume
module, 102	module, 322
pyg4ometry.fluka.RegionExpression.RegionParserVisitor	pyg4ometry.geant4.SkinSurface
module, 109	module, 323
pyg4ometry.fluka.vector	pyg4ometry.geant4.solid
module, 155	module, 225
pyg4ometry.fluka.vis	pyg4ometry.geant4.solid.Box
module, 157	module, 225
pyg4ometry.fluka.Writer	pyg4ometry.geant4.solid.Cons
module, 119	module, 226
pyg4ometry.freecad	pyg4ometry.geant4.solid.CutTubs
module, 180	module, 227
pyg4ometry.freecad.Reader	pyg4ometry.geant4.solid.Ellipsoid
module, 180	module, 227
pyg4ometry.gdml	pyg4ometry.geant4.solid.EllipticalCone
module, 182	module, 228
pyg4ometry.gdml.Defines	pyg4ometry.geant4.solid.EllipticalTube
module, 196	module, 229
pyg4ometry.gdml.GdmlExpression	pyg4ometry.geant4.solid.ExtrudedSolid
module, 182	module, 230
pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer	pyg4ometry.geant4.solid.GenericPolycone
module, 182	module, 230
pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser	pyg4ometry.geant4.solid.GenericPolyhedra
module, 183	module, 231
pyg4ometry.gdml.GdmlExpression.GdmlExpressionParserVisitor	pyg4ometry.geant4.solid.GenericTrap
module, 185	module, 232
pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor	pyg4ometry.geant4.solid.Hype
module, 194	module, 233
pyg4ometry.gdml.Reader	pyg4ometry.geant4.solid.Intersection
module, 205	module, 234
pyg4ometry.gdml.Units	pyg4ometry.geant4.solid.Layer
module, 207	module, 235
pyg4ometry.gdml.Writer	pyg4ometry.geant4.solid.MultiUnion
module, 207	module, 235
pyg4ometry.geant4	pyg4ometry.geant4.solid.OpticalSurface
module, 225	module, 236
pyg4ometry.geant4._Material	pyg4ometry.geant4.solid.Orb
module, 324	module, 237
pyg4ometry.geant4.AssemblyVolume	pyg4ometry.geant4.solid.Para
module, 309	module, 238
pyg4ometry.geant4.BorderSurface	pyg4ometry.geant4.solid.Paraboloid
module, 310	module, 238
pyg4ometry.geant4.DivisionVolume	pyg4ometry.geant4.solid.Plane
module, 311	module, 239
pyg4ometry.geant4.LogicalVolume	pyg4ometry.geant4.solid.Polycone
module, 312	module, 240
pyg4ometry.geant4.Loop	pyg4ometry.geant4.solid.Polyhedra

- module, 241
- pyg4ometry.geant4.solid.Scaled
 - module, 241
- pyg4ometry.geant4.solid.SolidBase
 - module, 242
- pyg4ometry.geant4.solid.Sphere
 - module, 243
- pyg4ometry.geant4.solid.Subtraction
 - module, 244
- pyg4ometry.geant4.solid.TessellatedSolid
 - module, 244
- pyg4ometry.geant4.solid.Tet
 - module, 246
- pyg4ometry.geant4.solid.Torus
 - module, 246
- pyg4ometry.geant4.solid.Trap
 - module, 247
- pyg4ometry.geant4.solid.Trd
 - module, 248
- pyg4ometry.geant4.solid.Tubs
 - module, 249
- pyg4ometry.geant4.solid.TwistedBox
 - module, 250
- pyg4ometry.geant4.solid.TwistedSolid
 - module, 251
- pyg4ometry.geant4.solid.TwistedTrap
 - module, 251
- pyg4ometry.geant4.solid.TwistedTrd
 - module, 252
- pyg4ometry.geant4.solid.TwistedTubs
 - module, 253
- pyg4ometry.geant4.solid.TwoVector
 - module, 254
- pyg4ometry.geant4.solid.Union
 - module, 255
- pyg4ometry.geant4.solid.Wedge
 - module, 256
- pyg4ometry.geant4.SurfaceBase
 - module, 323
- pyg4ometry.gui
 - module, 352
- pyg4ometry.gui.example1
 - module, 356
- pyg4ometry.gui.GeometryModel
 - module, 352
- pyg4ometry.gui.MainWindow
 - module, 352
- pyg4ometry.gui.QVTKRenderWindowInteractor
 - module, 353
- pyg4ometry.io
 - module, 365
- pyg4ometry.io.ROOTTGeo
 - module, 365
- pyg4ometry.meshutils
 - module, 420
- pyg4ometry.misc
 - module, 366
- pyg4ometry.misc.NestedSolids
 - module, 366
- pyg4ometry.misc.TestUtils
 - module, 366
- pyg4ometry.montecarlo
 - module, 368
- pyg4ometry.montecarlo.beam
 - module, 368
- pyg4ometry.montecarlo.boundary
 - module, 369
- pyg4ometry.montecarlo.config
 - module, 369
- pyg4ometry.montecarlo.scoring
 - module, 369
- pyg4ometry.pycgal
 - module, 370
- pyg4ometry.pycgal.core
 - module, 371
- pyg4ometry.pycgal.HalfPlane
 - module, 370
- pyg4ometry.pycgal.pythonHelpers
 - module, 373
- pyg4ometry.pycgal_old
 - module, 376
- pyg4ometry.pycgal_old.cgal
 - module, 376
- pyg4ometry.pycsg
 - module, 381
- pyg4ometry.pyoce
 - module, 382
- pyg4ometry.pyoce.pythonHelpers
 - module, 382
- pyg4ometry.pyoce.Reader
 - module, 382
- pyg4ometry.stl
 - module, 383
- pyg4ometry.stl.Reader
 - module, 383
- pyg4ometry.transformation
 - module, 420
- pyg4ometry.utils
 - module, 424
- pyg4ometry.visualisation
 - module, 387
- pyg4ometry.visualisation.Convert
 - module, 387
- pyg4ometry.visualisation.Mesh
 - module, 387
- pyg4ometry.visualisation.Plot
 - module, 388
- pyg4ometry.visualisation.RenderWriter
 - module, 420

- module, 389
 - pyg4ometry.visualisation.ViewerBase
 - module, 389
 - pyg4ometry.visualisation.VisualisationOptions
 - module, 392
 - pyg4ometry.visualisation.VtkExporter
 - module, 393
 - pyg4ometry.visualisation.VtkViewer
 - module, 395
 - pyg4ometry.visualisation.VtkViewerNew
 - module, 400
 - pyg4ometry.visualisation.Writer
 - module, 403
 - pyg4PlaneToVtkPlane() (in module
 - pyg4ometry.features.algos), 97
 - PyQtImpl (in module pyg4ometry.gui), 362
 - PyQtImpl (in module pyg4ometry.gui.example1), 356, 357
 - PyQtImpl (in module pyg4ometry.gui.QVTKRenderWindowInteractor), 353
- ## Q
- QUA (class in pyg4ometry.convert), 92
 - QUA (class in pyg4ometry.fluka), 168
 - QUA (class in pyg4ometry.fluka.body), 130
 - Quantity (class in pyg4ometry.gdml), 221
 - Quantity (class in pyg4ometry.gdml.Defines), 201
 - QVTKRenderWindowConeExample() (in module
 - pyg4ometry.gui), 364
 - QVTKRenderWindowConeExample() (in module
 - pyg4ometry.gui.example1), 359
 - QVTKRenderWindowConeExample() (in module
 - pyg4ometry.gui.QVTKRenderWindowInteractor), 356
 - QVTKRenderWindowInteractor (class in
 - pyg4ometry.gui), 360, 362
 - QVTKRenderWindowInteractor (class in
 - pyg4ometry.gui.example1), 357
 - QVTKRenderWindowInteractor (class in
 - pyg4ometry.gui.QVTKRenderWindowInteractor), 354
 - QVTKRWIBase (in module pyg4ometry.gui), 362
 - QVTKRWIBase (in module pyg4ometry.gui.example1), 356, 357
 - QVTKRWIBase (in module
 - pyg4ometry.gui.QVTKRenderWindowInteractor), 354
 - QVTKRWIBaseClass (in module
 - pyg4ometry.gui.example1), 357
 - randomColour() (in module pyg4ometry.visualisation), 407
 - randomColour() (in module
 - pyg4ometry.visualisation.VisualisationOptions), 392
 - RAW (class in pyg4ometry.convert), 87
 - RAW (class in pyg4ometry.fluka), 163
 - RAW (class in pyg4ometry.fluka.body), 125
 - RBRACKET (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.Gdml
 - attribute), 184
 - RBRACKET (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.Gdml
 - attribute), 193
 - RBRACKET() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.Gdml
 - method), 189
 - RCC (class in pyg4ometry.convert), 84
 - RCC (class in pyg4ometry.fluka), 161
 - RCC (class in pyg4ometry.fluka.body), 123
 - read_data() (pyg4ometry.analysis.flukaData._FlukaDataFile
 - method), 51
 - read_data() (pyg4ometry.analysis.flukaData.Usrbdx
 - method), 52
 - read_data() (pyg4ometry.analysis.flukaData.Usrbin
 - method), 51
 - read_event() (pyg4ometry.analysis.flukaData.Usrdump
 - method), 52
 - read_file() (pyg4ometry.analysis.flukaData.Usrbdx
 - method), 52
 - read_file() (pyg4ometry.analysis.flukaData.Usrbin
 - method), 51
 - read_header() (pyg4ometry.analysis.flukaData._FlukaDataFile
 - method), 51
 - read_header() (pyg4ometry.analysis.flukaData.Usrbdx
 - method), 52
 - read_header() (pyg4ometry.analysis.flukaData.Usrbin
 - method), 51
 - read_stats() (pyg4ometry.analysis.flukaData.Usrbdx
 - method), 52
 - read_stats() (pyg4ometry.analysis.flukaData.Usrbin
 - method), 51
 - read_structure() (pyg4ometry.analysis.flukaData.Usrdump
 - method), 52
 - Reader (class in pyg4ometry.fluka), 169
 - Reader (class in pyg4ometry.fluka.reader), 149
 - Reader (class in pyg4ometry.freecad.Reader), 181
 - Reader (class in pyg4ometry.gdml), 211
 - Reader (class in pyg4ometry.gdml.Reader), 205
 - Reader (class in pyg4ometry.io.ROOTTGeo), 365
 - Reader (class in pyg4ometry.pyoce), 383
 - Reader (class in pyg4ometry.pyoce.Reader), 382
 - Reader (class in pyg4ometry.stl), 385
 - Reader (class in pyg4ometry.stl.Reader), 384
 - readFile() (pyg4ometry.features.algos.FeatureData
 - method), 98
 - readFile() (pyg4ometry.features.FeatureData method),

100
 readStepFile() (pyg4ometry.pyoce.Reader method), 383
 readStepFile() (pyg4ometry.pyoce.Reader.Reader method), 382
 REC (class in pyg4ometry.convert), 85
 REC (class in pyg4ometry.fluka), 161
 REC (class in pyg4ometry.fluka.body), 123
 recurseObjectTree() (pyg4ometry.freecad.Reader.Reader method), 181
 recursePrintObjectTree() (pyg4ometry.freecad.Reader.Reader method), 181
 recurseVolumeTree() (pyg4ometry.io.ROOTTGeo.Reader method), 365
 RecursiveRotoTranslation (class in pyg4ometry.fluka), 177
 RecursiveRotoTranslation (class in pyg4ometry.fluka.directive), 135
 Region (class in pyg4ometry.fluka), 175
 Region (class in pyg4ometry.fluka.region), 153
 region() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 118
 region() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 108
 region() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 102
 region() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 112
 region_to_sympy() (in module pyg4ometry.fluka), 177
 region_to_sympy() (in module pyg4ometry.fluka.region), 151
 regionAABBs() (pyg4ometry.fluka.fluka_registry.FlukaRegistry method), 139
 regionAABBs() (pyg4ometry.fluka.FlukaRegistry method), 170
 RegionLexer (class in pyg4ometry.fluka.RegionExpression), 118
 RegionLexer (class in pyg4ometry.fluka.RegionExpression.RegionLexer), 100
 RegionName (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 118
 RegionName (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
 RegionName (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 118
 RegionName (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
 RegionName() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 112
 RegionName() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 103
 RegionName() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 103
 RegionName() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 113
 RegionName_sempred() (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer method), 119
 RegionName_sempred() (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer method), 101
 RegionParser (class in pyg4ometry.fluka.RegionExpression), 112
 RegionParser (class in pyg4ometry.fluka.RegionExpression.RegionParser), 102
 RegionParser.ComplexRegionContext (class in pyg4ometry.fluka.RegionExpression), 112
 RegionParser.ComplexRegionContext (class in pyg4ometry.fluka.RegionExpression.RegionParser), 103
 RegionParser.ExprContext (class in pyg4ometry.fluka.RegionExpression), 115
 RegionParser.ExprContext (class in pyg4ometry.fluka.RegionExpression.RegionParser), 105
 RegionParser.MultipleUnion2Context (class in pyg4ometry.fluka.RegionExpression), 113
 RegionParser.MultipleUnion2Context (class in pyg4ometry.fluka.RegionExpression.RegionParser), 103
 RegionParser.MultipleUnionContext (class in pyg4ometry.fluka.RegionExpression), 113
 RegionParser.MultipleUnionContext (class in pyg4ometry.fluka.RegionExpression.RegionParser), 104
 RegionParser.OneSubZoneContext (class in pyg4ometry.fluka.RegionExpression), 115
 RegionParser.OneSubZoneContext (class in pyg4ometry.fluka.RegionExpression.RegionParser), 106
 RegionParser.RegionContext (class in pyg4ometry.fluka.RegionExpression), 112
 RegionParser.RegionContext (class in pyg4ometry.fluka.RegionExpression.RegionParser), 112
 RegionParser.RegionsContext (class in pyg4ometry.fluka.RegionExpression), 112
 RegionParser.RegionsContext (class in pyg4ometry.fluka.RegionExpression.RegionParser), 102
 RegionParser.SimpleRegionContext (class in pyg4ometry.fluka.RegionExpression), 113
 RegionParser.SimpleRegionContext (class in pyg4ometry.fluka.RegionExpression.RegionParser), 102

[pyg4ometry.fluka.RegionExpression.RegionParser](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 103
[RegionParser.SingleUnaryContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 116
[RegionParser.SingleUnaryContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 106
[RegionParser.SingleUnionContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 114
[RegionParser.SingleUnionContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 104
[RegionParser.SubZoneContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 116
[RegionParser.SubZoneContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 106
[RegionParser.UnaryAndBooleanContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 115
[RegionParser.UnaryAndBooleanContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 106
[RegionParser.UnaryAndSubZoneContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 116
[RegionParser.UnaryAndSubZoneContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 106
[RegionParser.UnaryExpressionContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 117
[RegionParser.UnaryExpressionContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 107
[RegionParser.ZoneBodyContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 115
[RegionParser.ZoneBodyContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 105
[RegionParser.ZoneContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 114
[RegionParser.ZoneContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 104
[RegionParser.ZoneExprContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 114
[RegionParser.ZoneExprContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 105
[RegionParser.ZoneSubZoneContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 115
[RegionParser.ZoneSubZoneContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 105
[RegionParser.ZoneUnionContext](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 113
[RegionParser.ZoneUnionContext](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParser](#)), 103
[RegionParserVisitor](#) (class in [pyg4ometry.fluka.RegionExpression](#)), 110
[RegionParserVisitor](#) (class in [pyg4ometry.fluka.RegionExpression.RegionParserVisitor](#)), 109
[regions\(\)](#) ([pyg4ometry.fluka.RegionExpression.RegionParser](#) method), 118
[regions\(\)](#) ([pyg4ometry.fluka.RegionExpression.RegionParser.RegionParserVisitor](#) method), 108
[regionToAlgebraicExpression\(\)](#) (in module [pyg4ometry.fluka.boolean_algebra](#)), 132
[RegionVisitor](#) (class in [pyg4ometry.fluka.reader](#)), 149
[registerSolidEdit\(\)](#) ([pyg4ometry.geant4.Registry](#) method), 345
[registerSolidEdit\(\)](#) ([pyg4ometry.geant4.Registry.Registry](#) method), 318
[Registry](#) (class in [pyg4ometry.geant4](#)), 345
[Registry](#) (class in [pyg4ometry.geant4.Registry](#)), 318
[relabelModel\(\)](#) ([pyg4ometry.freecad.Reader.Reader](#) method), 181
[relop\(\)](#) ([pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser](#) method), 194
[relop\(\)](#) ([pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser](#) method), 185
[reMesh\(\)](#) ([pyg4ometry.geant4.LogicalVolume](#) method), 332, 341
[reMesh\(\)](#) ([pyg4ometry.geant4.LogicalVolume.LogicalVolume](#) method), 312
[remesh\(\)](#) ([pyg4ometry.visualisation.Mesh](#) method), 405
[remesh\(\)](#) ([pyg4ometry.visualisation.Mesh.Mesh](#) method), 388
[remove\(\)](#) ([pyg4ometry.fluka.fluka_registry.BaseCacher](#) method), 141
[removeBody\(\)](#) ([pyg4ometry.fluka.Region](#) method), 176
[removeBody\(\)](#) ([pyg4ometry.fluka.region.Region](#) method), 154
[removeBody\(\)](#) ([pyg4ometry.fluka.region.Zone](#) method), 153
[removeBody\(\)](#) ([pyg4ometry.fluka.Zone](#) method), 174
[removeDuplicateVertices\(\)](#) ([pyg4ometry.geant4.solid.TessellatedSolid](#) method), 304
[removeDuplicateVertices\(\)](#) ([pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid](#) method), 245
[removeHighLight\(\)](#) ([pyg4ometry.visualisation.MouseInteractorNamePhysics](#) method), 413
[removeHighLight\(\)](#) ([pyg4ometry.visualisation.VtkViewerNew.MouseInteractorNamePhysics](#) method), 402
[removeHighLightText\(\)](#)

(pyg4ometry.visualisation.MouseInteractorNamePhysicalVolume method), 358
 method), 413
 removeHighlightText() (pyg4ometry.visualisation.VtkViewerNew.MouseInteractorNamePhysicalVolume method), 402
 removeInvisible() (pyg4ometry.visualisation.ViewerBase.removeInvisible() method), 406
 removeInvisible() (pyg4ometry.visualisation.ViewerBase.removeInvisible() method), 391
 removeNullZones() (pyg4ometry.fluka.Region method), 175
 removeNullZones() (pyg4ometry.fluka.region.Region method), 154
 removeprefix() (in module pyg4ometry.geant4), 345
 removeprefix() (in module pyg4ometry.geant4.Registry), 318
 removeZones() (pyg4ometry.fluka.Region method), 176
 removeZones() (pyg4ometry.fluka.region.Region method), 155
 rename() (pyg4ometry.fluka.Compound method), 179
 rename() (pyg4ometry.fluka.Material method), 179
 rename() (pyg4ometry.fluka.material.Compound method), 145
 rename() (pyg4ometry.fluka.material.Material method), 145
 Render() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor method), 359
 Render() (pyg4ometry.gui.QVTKRenderWindowInteractor method), 362, 364
 Render() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method), 356
 render() (pyg4ometry.visualisation.VtkViewerNew method), 412
 render() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 401
 RenderWriter (class in pyg4ometry.visualisation), 413
 RenderWriter (class in pyg4ometry.visualisation.RenderWriter), 389
 replaceSolid() (pyg4ometry.geant4.LogicalVolume method), 333, 342
 replaceSolid() (pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 313
 ReplicaVolume (class in pyg4ometry.geant4), 337
 ReplicaVolume (class in pyg4ometry.geant4.ReplicaVolume), 322
 ReplicaVolume.Axis (class in pyg4ometry.geant4), 337
 ReplicaVolume.Axis (class in pyg4ometry.geant4.ReplicaVolume), 322
 replicaVolumes() (in module pyg4ometry.compare), 67
 replicaVolumes() (in module pyg4ometry.compare._Compare), 60
 resizeEvent() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor method), 361, 364
 resizeEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method), 355
 reverse() (in module pyg4ometry.geant4.solid), 277, 281, 284, 308
 reverse() (in module pyg4ometry.transformation), 424
 reversePolygon() (pyg4ometry.convert._PolygonProcessing class method), 81
 reversePolygon() (pyg4ometry.pycgal.core.PolygonProcessing class method), 372
 reversePolygon() (pyg4ometry.pycgal.PolygonProcessing class method), 375
 rightButtonPressEvent() (pyg4ometry.visualisation.MouseInteractorNamePhysicalVolume method), 411, 413
 rightButtonPressEvent() (pyg4ometry.visualisation.VtkViewer.MouseInteractorNamePhysicalVolume method), 400
 rightButtonPressEvent() (pyg4ometry.visualisation.VtkViewerNew.MouseInteractorNamePhysicalVolume method), 402
 rootMatrix2pyg4ometry() (in module pyg4ometry.io.ROOTTGeo), 365
 rootShape2pyg4ometry() (in module pyg4ometry.io.ROOTTGeo), 365
 rotate() (pyg4ometry.pycgal.core.CSG method), 371
 rotate() (pyg4ometry.pycgal.CSG method), 374
 Rotate() (pyg4ometry.geant4.solid._Layer method), 294, 296, 298
 Rotated() (pyg4ometry.geant4.solid._TwoVector method), 294, 295, 297
 Rotated() (pyg4ometry.geant4.solid.Layer.Layer method), 235
 Rotated() (pyg4ometry.geant4.solid.TwoVector method), 261
 Rotated() (pyg4ometry.geant4.solid.TwoVector.TwoVector method), 254
 Rotation (class in pyg4ometry.gdml), 223
 Rotation (class in pyg4ometry.gdml.Defines), 203
 rotation() (pyg4ometry.convert.ARB method), 87
 rotation() (pyg4ometry.convert.BOX method), 84
 rotation() (pyg4ometry.convert.ELL method), 86
 rotation() (pyg4ometry.convert.PLA method), 89
 rotation() (pyg4ometry.convert.QUA method), 92
 rotation() (pyg4ometry.convert.RCC method), 85
 rotation() (pyg4ometry.convert.REC method), 85
 rotation() (pyg4ometry.convert.RPP method), 83
 rotation() (pyg4ometry.convert.SPH method), 84
 rotation() (pyg4ometry.convert.TRC method), 86
 rotation() (pyg4ometry.convert.XCC method), 89
 rotation() (pyg4ometry.convert.XEC method), 91
 rotation() (pyg4ometry.convert.YCC method), 90

- rotation() (pyg4ometry.convert.YEC method), 91
- rotation() (pyg4ometry.convert.ZCC method), 90
- rotation() (pyg4ometry.convert.ZEC method), 92
- rotation() (pyg4ometry.fluka.ARB method), 163
- rotation() (pyg4ometry.fluka.body.ARB method), 125
- rotation() (pyg4ometry.fluka.body.BOX method), 122
- rotation() (pyg4ometry.fluka.body.ELL method), 124
- rotation() (pyg4ometry.fluka.body.PLA method), 127
- rotation() (pyg4ometry.fluka.body.QUA method), 131
- rotation() (pyg4ometry.fluka.body.RCC method), 123
- rotation() (pyg4ometry.fluka.body.REC method), 123
- rotation() (pyg4ometry.fluka.body.RPP method), 121
- rotation() (pyg4ometry.fluka.body.SPH method), 122
- rotation() (pyg4ometry.fluka.body.TRC method), 124
- rotation() (pyg4ometry.fluka.body.XCC method), 127
- rotation() (pyg4ometry.fluka.body.XEC method), 129
- rotation() (pyg4ometry.fluka.body.YCC method), 128
- rotation() (pyg4ometry.fluka.body.YEC method), 130
- rotation() (pyg4ometry.fluka.body.ZCC method), 128
- rotation() (pyg4ometry.fluka.body.ZEC method), 130
- rotation() (pyg4ometry.fluka.BOX method), 160
- rotation() (pyg4ometry.fluka.ELL method), 162
- rotation() (pyg4ometry.fluka.PLA method), 165
- rotation() (pyg4ometry.fluka.QUA method), 169
- rotation() (pyg4ometry.fluka.RCC method), 161
- rotation() (pyg4ometry.fluka.REC method), 161
- rotation() (pyg4ometry.fluka.Region method), 175
- rotation() (pyg4ometry.fluka.region.Region method), 154
- rotation() (pyg4ometry.fluka.region.Zone method), 152
- rotation() (pyg4ometry.fluka.RPP method), 159
- rotation() (pyg4ometry.fluka.SPH method), 160
- rotation() (pyg4ometry.fluka.TRC method), 162
- rotation() (pyg4ometry.fluka.XCC method), 165
- rotation() (pyg4ometry.fluka.XEC method), 167
- rotation() (pyg4ometry.fluka.YCC method), 166
- rotation() (pyg4ometry.fluka.YEC method), 167
- rotation() (pyg4ometry.fluka.ZCC method), 166
- rotation() (pyg4ometry.fluka.ZEC method), 168
- rotation() (pyg4ometry.fluka.Zone method), 174
- rotation() (pyg4ometry.geant4.solid.Intersection method), 278
- rotation() (pyg4ometry.geant4.solid.Intersection.Intersection method), 235
- rotation() (pyg4ometry.geant4.solid.Subtraction method), 282
- rotation() (pyg4ometry.geant4.solid.Subtraction.Subtraction method), 244
- rotation() (pyg4ometry.geant4.solid.Union method), 274
- rotation() (pyg4ometry.geant4.solid.Union.Union method), 255
- RotoTranslation (class in pyg4ometry.fluka), 177
- RotoTranslation (class in pyg4ometry.fluka.directive), 135
- rotoTranslationFromTBxyz() (in module pyg4ometry.fluka.directive), 136
- rotoTranslationFromTra2() (in module pyg4ometry.fluka.directive), 136
- RotoTranslationStore (class in pyg4ometry.fluka.fluka_registry), 140
- RParen (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 119
- RParen (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
- RParen (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 118
- RParen (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
- RPAREN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 184
- RPAREN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
- RParen() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 107
- RParen() (pyg4ometry.fluka.RegionExpression.RegionParser.SubZoneCombinator method), 116
- RPAREN() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 188
- RPAREN() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 190
- RPP (class in pyg4ometry.convert), 83
- RPP (class in pyg4ometry.fluka), 159
- RPP (class in pyg4ometry.fluka.body), 121
- RULE_atom (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- RULE_constant (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- RULE_equation (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- RULE_expr (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
- RULE_expr (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
- RULE_expression (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- RULE_func (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- RULE_funcname (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- RULE_matrixElement (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- RULE_multiplyingExpression (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- RULE_operatorAddSub (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192

attribute), 192
 RULE_operatorMulDiv (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
 RULE_powExpression (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
 RULE_region (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
 RULE_region (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
 RULE_regions (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
 RULE_regions (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 107
 RULE_relop (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
 RULE_scientific (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
 RULE_signedAtom (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
 RULE_subZone (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
 RULE_subZone (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
 RULE_unaryExpression (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
 RULE_unaryExpression (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
 RULE_variable (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
 RULE_zone (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
 RULE_zone (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
 RULE_zoneUnion (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
 RULE_zoneUnion (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
 ruleNames (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 119
 ruleNames (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
 ruleNames (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
 ruleNames (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
 ruleNames (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 185
 ruleNames (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
 sample_names() (pyg4ometry.utils.Samples method), 423
 Sampler (class in pyg4ometry.bdsim), 55
 sampler (class in pyg4ometry.bdsim.sampler), 54
 Samples (class in pyg4ometry.utils), 424
 saveFileDialog() (pyg4ometry.gui.MainWindow method), 362
 saveFileDialog() (pyg4ometry.gui.MainWindow.MainWindow method), 353
 ScalarBase (class in pyg4ometry.gdml), 219
 ScalarBase (class in pyg4ometry.gdml.Defines), 199
 Scale (class in pyg4ometry.gdml), 223
 Scale (class in pyg4ometry.gdml.Defines), 203
 scale() (pyg4ometry.pygal.CSG method), 371
 scale() (pyg4ometry.pygal.CSG method), 374
 scaled (class in pyg4ometry.geant4.solid), 308
 Scaled (class in pyg4ometry.geant4.solid.Scaled), 241
 ScaleScene() (pyg4ometry.visualisation.ViewerBase method), 406
 ScaleScene() (pyg4ometry.visualisation.ViewerBase.ViewerBase method), 391
 scientific() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser method), 194
 scientific() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser method), 188
 SCIENTIFIC_NUMBER (pyg4ometry.gdml.GdmlExpression.GdmlExpression attribute), 184
 SCIENTIFIC_NUMBER (pyg4ometry.gdml.GdmlExpression.GdmlExpression attribute), 193
 SCIENTIFIC_NUMBER (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser attribute), 189
 Scorer (class in pyg4ometry.bdsim), 55
 Scorer (class in pyg4ometry.bdsim.scoring), 54
 ScorerMesh (class in pyg4ometry.bdsim), 55
 ScorerMesh (class in pyg4ometry.bdsim.scoring), 54
 Scoring (class in pyg4ometry.montecarlo), 370
 Scoring (class in pyg4ometry.montecarlo.scoring), 369
 sempred() (pyg4ometry.fluka.RegionExpression.RegionLexer method), 119
 sempred() (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer method), 101
 SensitiveErrorListener (class in pyg4ometry.fluka.reader), 150
 serializedATN() (in module pyg4ometry.fluka.RegionExpression.RegionLexer), 100
 serializedATN() (in module pyg4ometry.fluka.RegionExpression.RegionParser), 102
 serializedATN() (in module pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer), 183

serializedATN() (in module *pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser*), method, 401
 185
 set_pressure() (*pyg4ometry.compare._Material* method), 63
 set_pressure() (*pyg4ometry.gdml._Material* method), 214
 set_pressure() (*pyg4ometry.geant4._Material.Material* method), 328
 set_pressure() (*pyg4ometry.geant4.Material* method), 350
 set_registry() (*pyg4ometry.geant4._Material.MaterialBase* method), 327
 set_registry() (*pyg4ometry.geant4.MaterialBase* method), 350
 set_state() (*pyg4ometry.geant4._Material.MaterialBase* method), 327
 set_state() (*pyg4ometry.geant4.MaterialBase* method), 350
 set_temperature() (*pyg4ometry.compare._Material* method), 63
 set_temperature() (*pyg4ometry.gdml._Material* method), 214
 set_temperature() (*pyg4ometry.geant4._Material.MaterialBase* method), 328
 set_temperature() (*pyg4ometry.geant4.Material* method), 350
 setAllFalse() (*pyg4ometry.compare._Compare.Tests* method), 58
 setAllFalse() (*pyg4ometry.compare.Tests* method), 64
 setAxes() (*pyg4ometry.visualisation.VtkViewerNew* method), 412
 setAxes() (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401
 setBody() (*pyg4ometry.fluka.fluka_registry.BaseCacher* method), 141
 setCameraFocusPosition() (*pyg4ometry.visualisation.VtkViewer* method), 408
 setCameraFocusPosition() (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 397
 setClipper() (*pyg4ometry.visualisation.VtkViewerNew* method), 412
 setClipper() (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401
 setCoplanarVisOption() (*pyg4ometry.visualisation.ViewerBase* method), 406
 setCoplanarVisOption() (*pyg4ometry.visualisation.ViewerBase.ViewerBase* method), 391
 setCutter() (*pyg4ometry.visualisation.VtkViewerNew* method), 412
 setCutter() (*pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew* method), 401
 setCutterNormal() (*pyg4ometry.visualisation.VtkViewer* method), 408
 setCutterNormal() (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 397
 setCutterOrigin() (*pyg4ometry.visualisation.VtkViewer* method), 408
 setCutterOrigin() (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 397
 setDefaultVisOptions() (*pyg4ometry.visualisation.ViewerBase* method), 406
 setDefaultVisOptions() (*pyg4ometry.visualisation.ViewerBase.ViewerBase* method), 391
 setDisplayRenderer() (*pyg4ometry.gui.MainWindow* method), 362
 setDisplayRenderer() (*pyg4ometry.gui.MainWindow.MainWindow* method), 353
 setExpression() (*pyg4ometry.gdml.Defines.ScalarBase* method), 199
 setExpression() (*pyg4ometry.gdml.ScalarBase* method), 219
 setFalse() (*pyg4ometry.compare._Compare.Tests* method), 58
 setFalse() (*pyg4ometry.compare.Tests* method), 64
 setGlobalMeshSliceAndStack() (in module *pyg4ometry.config*), 418
 setLogicalVolumeMaterial() (*pyg4ometry.freecad.Reader.Reader* method), 181
 setMaterialVisOptions() (*pyg4ometry.visualisation.VtkViewer* method), 408
 setMaterialVisOptions() (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 397
 setName() (*pyg4ometry.gdml.DefineBase* method), 218
 setName() (*pyg4ometry.gdml.Defines.DefineBase* method), 198
 setName() (*pyg4ometry.gdml.Defines.ScalarBase* method), 199
 setName() (*pyg4ometry.gdml.Defines.VectorBase* method), 202
 setName() (*pyg4ometry.gdml.ScalarBase* method), 219
 setName() (*pyg4ometry.gdml.VectorBase* method), 222
 setOpacity() (*pyg4ometry.visualisation.VtkViewer* method), 408
 setOpacity() (*pyg4ometry.visualisation.VtkViewer.VtkViewer* method), 397
 setOpacityOverlap() (*pyg4ometry.visualisation.VtkViewer* method),

- 408
- `setOpacityOverlap()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 397
- `setOverlapVisOption()` (*pyg4ometry.visualisation.ViewerBase method*), 406
- `setOverlapVisOption()` (*pyg4ometry.visualisation.ViewerBase.ViewerBase method*), 391
- `setProtusionVisOption()` (*pyg4ometry.visualisation.ViewerBase method*), 406
- `setProtusionVisOption()` (*pyg4ometry.visualisation.ViewerBase.ViewerBase method*), 391
- `setRandomColours()` (*pyg4ometry.visualisation.VtkViewer method*), 408
- `setRandomColours()` (*pyg4ometry.visualisation.VtkViewer method*), 397
- `setRegionToOuterBoundary()` (*pyg4ometry.fluka.Extruder method*), 180
- `setRegionToOuterBoundary()` (*pyg4ometry.fluka.extruder.Extruder method*), 137
- `setRegistry()` (*pyg4ometry.gdml.DefineBase method*), 218
- `setRegistry()` (*pyg4ometry.gdml.Defines.DefineBase method*), 198
- `setRegistry()` (*pyg4ometry.gdml.Defines.ScalarBase method*), 199
- `setRegistry()` (*pyg4ometry.gdml.Defines.VectorBase method*), 203
- `setRegistry()` (*pyg4ometry.gdml.ScalarBase method*), 219
- `setRegistry()` (*pyg4ometry.gdml.VectorBase method*), 223
- `setSolid()` (*pyg4ometry.geant4.LogicalVolume method*), 334, 343
- `setSolid()` (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 314
- `setSubtractDaughters()` (*pyg4ometry.visualisation.ViewerBase method*), 405
- `setSubtractDaughters()` (*pyg4ometry.visualisation.ViewerBase.ViewerBase method*), 390
- `setSurface()` (*pyg4ometry.visualisation.VtkViewer method*), 408
- `setSurface()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 397
- `setSurfaceOverlap()` (*pyg4ometry.visualisation.VtkViewer method*), 408
- `setSurfaceOverlap()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 397
- `setWireframe()` (*pyg4ometry.visualisation.VtkViewer method*), 408
- `setWireframe()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 397
- `setWireframeOverlap()` (*pyg4ometry.visualisation.VtkViewer method*), 408
- `setWireframeOverlap()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 397
- `setWorld()` (*pyg4ometry.geant4.Registry method*), 346
- `setWorld()` (*pyg4ometry.geant4.Registry.Registry method*), 320
- `setWorldVolume()` (*pyg4ometry.geant4.Registry method*), 346
- `setWorldVolume()` (*pyg4ometry.geant4.Registry.Registry method*), 320
- `shapeTopology()` (in module *pyg4ometry.pyoce.pythonHelpers*), 383
- `shapeTopologyCount()` (in module *pyg4ometry.pyoce.pythonHelpers*), 382
- `sharedContextCache` (*pyg4ometry.fluka.RegionExpression.RegionParser attribute*), 117
- `sharedContextCache` (*pyg4ometry.fluka.RegionExpression.RegionParser attribute*), 107
- `sharedContextCache` (*pyg4ometry.gdml.GdmlExpression.GdmlExpression attribute*), 192
- `ShowCursor()` (*pyg4ometry.gui.example1.QVTKRenderWindowInteractor method*), 358
- `ShowCursor()` (*pyg4ometry.gui.QVTKRenderWindowInteractor method*), 361, 363
- `ShowCursor()` (*pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method*), 355
- `signedAtom()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser method*), 194
- `signedAtom()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser method*), 187
- `signedAtom()` (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser method*), 188
- `simp()` (*pyg4ometry.gdml.BasicExpression method*), 217
- `simp()` (*pyg4ometry.gdml.Defines.BasicExpression method*), 197
- `simplify()` (*pyg4ometry.fluka.Region method*), 176
- `simplify()` (*pyg4ometry.fluka.region.Region method*), 155
- `simplify_region()` (in module *pyg4ometry.fluka.region*), 151
- `simplifyModel()` (*pyg4ometry.freecad.Reader.Reader method*), 181
- `simplifyRegion()` (in module *pyg4ometry.fluka.boolean_algebra*), 133

- SIN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer.config), 419
 attribute), 183
 SolidDefaults.Wedge (class in pyg4ometry.config), 419
- SIN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser
 attribute), 192
 solidName() (in module pyg4ometry.geant4), 345
 solidName() (in module pyg4ometry.geant4.Registry), 318
- sin() (in module pyg4ometry.gdml), 219
 sin() (in module pyg4ometry.gdml.Defines), 199
 SIN() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.config), 419
 method), 191
 solids() (in module pyg4ometry.compare._Compare), 60
- sizeHint() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor), 358
 sizeHint() (pyg4ometry.gui.QVTKRenderWindowInteractor), 361, 364
 sizeHint() (pyg4ometry.gui.QVTKRenderWindowInteractor), 355
 SPH (class in pyg4ometry.fluka), 160
 SPH (class in pyg4ometry.fluka.body), 122
- SkinSurface (class in pyg4ometry.geant4), 344
 SkinSurface (class in pyg4ometry.geant4.SkinSurface), 323
 Sphere (class in pyg4ometry.geant4.solid), 265
 Sphere (class in pyg4ometry.geant4.solid.Sphere), 243
- slotModelChanged() (pyg4ometry.gui.MainWindow), 362
 slotModelChanged() (pyg4ometry.gui.MainWindow.MainWindow), 353
 Sqrt (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer.config), 419
 attribute), 184
 Sqrt (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.config), 419
 attribute), 193
- SolidBase (class in pyg4ometry.geant4.solid), 309
 SolidBase (class in pyg4ometry.geant4.solid.SolidBase), 242
 SolidDefaults (class in pyg4ometry.config), 418
 SolidDefaults.Cons (class in pyg4ometry.config), 418
 SolidDefaults.CutTubs (class in pyg4ometry.config), 418
- SolidDefaults.Ellipsoid (class in pyg4ometry.config), 418
 SolidDefaults.EllipticalCone (class in pyg4ometry.config), 418
 SolidDefaults.EllipticalTube (class in pyg4ometry.config), 418
 SolidDefaults.GenericPolycone (class in pyg4ometry.config), 418
 SolidDefaults.Hype (class in pyg4ometry.config), 418
 SolidDefaults.Orb (class in pyg4ometry.config), 418
 SolidDefaults.Paraboloid (class in pyg4ometry.config), 419
 SolidDefaults.Polycone (class in pyg4ometry.config), 419
 SolidDefaults.Sphere (class in pyg4ometry.config), 419
 SolidDefaults.Torus (class in pyg4ometry.config), 419
 SolidDefaults.Tubs (class in pyg4ometry.config), 418
 SolidDefaults.TwistedBox (class in pyg4ometry.config), 419
 SolidDefaults.TwistedTrap (class in pyg4ometry.config), 419
 SolidDefaults.TwistedTrd (class in pyg4ometry.config), 419
 SolidDefaults.TwistedTubs (class in pyg4ometry.config), 419
- sqrt() (in module pyg4ometry.gdml), 220
 sqrt() (in module pyg4ometry.gdml.Defines), 200
 Sqrt() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser.config), 419
 method), 191
 squashDegenerateBodies() (in module pyg4ometry.fluka.boolean_algebra), 132
 start() (pyg4ometry.visualisation.VtkViewer method), 408
 start() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 397
 state_variables (pyg4ometry.compare._Material property), 63
 state_variables (pyg4ometry.gdml._Material property), 213
 state_variables (pyg4ometry.geant4._Material.Material property), 327
 state_variables (pyg4ometry.geant4.Material property), 350
 stds() (pyg4ometry.utils.Samples method), 425
 Stl (pyg4ometry.geant4.solid.TessellatedSolid.MeshType attribute), 304
 Stl (pyg4ometry.geant4.solid.TessellatedSolid.TessellatedSolid.MeshType attribute), 245
 stl2gdml() (in module pyg4ometry.convert), 93
 stl2gdml() (in module pyg4ometry.convert.stl2gdml), 78
 str() (pyg4ometry.gdml.BasicExpression method), 218
 str() (pyg4ometry.gdml.Defines.BasicExpression method), 198
 structureAnalysis() (pyg4ometry.geant4.Registry method), 347
 structureAnalysis() (pyg4ometry.geant4.Registry.Registry method), 347

- 321
- subtract() (pyg4ometry.pycgal.core.CSG method), 371
- subtract() (pyg4ometry.pycgal.CSG method), 375
- Subtraction (class in pyg4ometry.fluka.region), 152
- Subtraction (class in pyg4ometry.geant4.solid), 281
- Subtraction (class in pyg4ometry.geant4.solid.Subtraction), 244
- subZone() (pyg4ometry.fluka.RegionExpression.RegionParser method), 118
- subZone() (pyg4ometry.fluka.RegionExpression.RegionParser.OneSubZoneContext method), 116
- subZone() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser method), 108
- subZone() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.OneSubZoneContext method), 106
- subZone() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.UnaryAndSubZoneContext method), 106
- subZone() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.ZoneSubZoneContext method), 105
- subZone() (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.UnaryAndSubZoneContext method), 116
- subZone() (pyg4ometry.fluka.RegionExpression.RegionParser.ZoneSubZoneContext method), 115
- SurfaceBase (class in pyg4ometry.geant4), 344, 345
- SurfaceBase (class in pyg4ometry.geant4.SurfaceBase), 323
- surfacemesh_print (in module pyg4ometry.pycgal_old), 381
- surfacemesh_print (in module pyg4ometry.pycgal_old.cgal), 379
- surfaceMesh_to_Polyhedron() (pyg4ometry.pycgal.core.PolyhedronProcessing class method), 373
- surfaceMesh_to_Polyhedron() (pyg4ometry.pycgal.PolyhedronProcessing class method), 376
- surfacemesh_write (in module pyg4ometry.pycgal_old), 381
- surfacemesh_write (in module pyg4ometry.pycgal_old.cgal), 379
- symbolicNames (pyg4ometry.fluka.RegionExpression.RegionLexer attribute), 119
- symbolicNames (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 101
- symbolicNames (pyg4ometry.fluka.RegionExpression.RegionParser attribute), 117
- symbolicNames (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 107
- symbolicNames (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer attribute), 185
- symbolicNames (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser attribute), 192
- sympy_to_region() (in module pyg4ometry.fluka), 177
- sympy_to_region() (in module pyg4ometry.fluka.region), 151
- sympy_to_zone() (in module pyg4ometry.fluka.region), 151
- syntaxError() (pyg4ometry.fluka.reader.SensitiveErrorListener method), 150
- ## T
- TAN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 183
- TAN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 192
- tan() (in module pyg4ometry.gdml), 220
- tan() (in module pyg4ometry.gdml.Defines), 200
- TAN (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 191
- tbxyz() (pyg4ometry.fluka.Region method), 175
- tbxyz() (pyg4ometry.fluka.region.Region method), 154
- tbxyz() (pyg4ometry.fluka.region.RegionZone method), 152
- tbxyz() (pyg4ometry.fluka.Zone method), 174
- tbxyz2axisangle() (in module pyg4ometry.geant4.solid), 275, 279, 282, 286
- tbxyz2matrix() (in module pyg4ometry.transformation), 422
- tbxyz2matrix() (in module pyg4ometry.geant4.solid), 276, 280, 283, 307
- tbxyz2matrix() (in module pyg4ometry.transformation), 423
- tbzyx2matrix() (in module pyg4ometry.geant4.solid), 276, 280, 284, 307
- tbzyx2matrix() (in module pyg4ometry.transformation), 423
- TessellatedSolid (class in pyg4ometry.geant4.solid), 304
- TessellatedSolid (class in pyg4ometry.geant4.solid.TessellatedSolid), 245
- TessellatedSolid.MeshType (class in pyg4ometry.geant4.solid), 304
- TessellatedSolid.MeshType (class in pyg4ometry.geant4.solid.TessellatedSolid), 245
- test() (in module pyg4ometry.features.algos), 98
- test() (in module pyg4ometry.pyoce.pythonHelpers), 383
- testNames() (pyg4ometry.compare._Compare.ComparisonResult method), 65
- testNames() (pyg4ometry.compare.ComparisonResult method), 65
- TestResult (class in pyg4ometry.compare), 65
- TestResult (class in pyg4ometry.compare._Compare), 58
- TestResultNamed (class in pyg4ometry.compare), 65

TestResultNamed (class *pyg4ometry.compare._Compare*), 58
 Tests (class in *pyg4ometry.compare*), 64
 Tests (class in *pyg4ometry.compare._Compare*), 57
 Tet (class in *pyg4ometry.geant4.solid*), 293
 Tet (class in *pyg4ometry.geant4.solid.Tet*), 246
 TH1 (class in *pyg4ometry.analysis.Data*), 49
 TH2 (class in *pyg4ometry.analysis.Data*), 49
 TH3 (class in *pyg4ometry.analysis.Data*), 49
 Three (class in *pyg4ometry.fluka*), 172
 Three (class in *pyg4ometry.fluka.vector*), 156
 Timer (class in *pyg4ometry.utils*), 425
 TimerEvent() (*pyg4ometry.gui.example1.QVTKRenderWindowInteractor* method), 358
 TimerEvent() (*pyg4ometry.gui.QVTKRenderWindowInteractor* method), 361, 363
 TimerEvent() (*pyg4ometry.gui.QVTKRenderWindowInteractor* method), 355
 TIMES (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionNode* attribute), 184
 TIMES (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionNode* attribute), 193
 TIMES() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionNode* method), 187
 to4DMatrix() (*pyg4ometry.fluka.directive.RecursiveRotoTranslation* method), 136
 to4DMatrix() (*pyg4ometry.fluka.directive.RotoTranslation* method), 135
 to4DMatrix() (*pyg4ometry.fluka.directive.Transform* method), 135
 to4DMatrix() (*pyg4ometry.fluka.RecursiveRotoTranslation* method), 178
 to4DMatrix() (*pyg4ometry.fluka.RotoTranslation* method), 177
 to4DMatrix() (*pyg4ometry.fluka.Transform* method), 177
 toCard() (*pyg4ometry.fluka.directive.RotoTranslation* method), 135
 toCard() (*pyg4ometry.fluka.RotoTranslation* method), 177
 toCards() (*pyg4ometry.fluka.Compound* method), 179
 toCards() (*pyg4ometry.fluka.Material* method), 179
 toCards() (*pyg4ometry.fluka.material.Compound* method), 145
 toCards() (*pyg4ometry.fluka.material.Material* method), 144
 toDNF() (*pyg4ometry.fluka.Region* method), 176
 toDNF() (*pyg4ometry.fluka.region.Region* method), 155
 toDNF() (*pyg4ometry.fluka.region.Zone* method), 153
 toDNF() (*pyg4ometry.fluka.Zone* method), 175
 toFixedString() (*pyg4ometry.fluka.Card* method), 180
 toFixedString() (*pyg4ometry.fluka.card.Card* method), 133
 toFreeString() (*pyg4ometry.fluka.Card* method), 180
 toFreeString() (*pyg4ometry.fluka.card.Card* method), 133
 toList() (*pyg4ometry.fluka.Card* method), 179
 toList() (*pyg4ometry.fluka.card.Card* method), 133
 toPlane() (*pyg4ometry.convert.PLA* method), 89
 toPlane() (*pyg4ometry.convert.XYP* method), 88
 toPlane() (*pyg4ometry.convert.XZP* method), 88
 toPlane() (*pyg4ometry.convert.YZP* method), 88
 toPlane() (*pyg4ometry.fluka.body.PLA* method), 127
 toPlane() (*pyg4ometry.fluka.body.XYP* method), 126
 toPlane() (*pyg4ometry.fluka.body.XZP* method), 126
 toPlane() (*pyg4ometry.fluka.body.YZP* method), 127
 toPlane() (*pyg4ometry.fluka.PLA* method), 165
 toPlane() (*pyg4ometry.fluka.XYP* method), 164
 toPlane() (*pyg4ometry.fluka.XZP* method), 164
 toPlane() (*pyg4ometry.fluka.YZP* method), 165
 toRotoTranslationMatrix() (*pyg4ometry.fluka.directive.MatrixConvertibleMixin* method), 134
 toRotoTranslationMatrix() (*pyg4ometry.geant4.solid*), 269
 Torus (class in *pyg4ometry.geant4.solid.Torus*), 246
 toScaleMatrix() (*pyg4ometry.fluka.directive.MatrixConvertibleMixin* method), 134
 toScaleMatrix() (*pyg4ometry.fluka.directive.MatrixConvertibleMixin* method), 179
 totalWeighting() (*pyg4ometry.fluka.material.Compound* method), 145
 toTranslationMatrix() (*pyg4ometry.fluka.directive.MatrixConvertibleMixin* method), 134
 toVerticesAndPolygons() (*pyg4ometry.pycgal.core.CSG* method), 371
 toVerticesAndPolygons() (*pyg4ometry.pycgal.CSG* method), 374
 trackDataToPolydata() (*pyg4ometry.analysis.flukaData.Usrdump* method), 52
 transferDefine() (*pyg4ometry.geant4.Registry* method), 346
 transferDefine() (*pyg4ometry.geant4.Registry.Registry* method), 319
 transferDefines() (*pyg4ometry.geant4.Registry* method), 346
 transferDefines() (*pyg4ometry.geant4.Registry.Registry* method), 319
 transferLogicalVolume() (*pyg4ometry.geant4.Registry* method), 346
 transferLogicalVolume() (*pyg4ometry.geant4.Registry.Registry* method), 319
 transferMaterial() (*pyg4ometry.geant4.Registry* method), 345
 transferMaterial() (*pyg4ometry.geant4.Registry.Registry* method), 319
 transferPhysicalVolume()

- (*pyg4ometry.geant4.Registry* method), 346
- `transferPhysicalVolume()` (*pyg4ometry.geant4.Registry.Registry* method), 319
- `transferSolid()` (*pyg4ometry.geant4.Registry* method), 345
- `transferSolid()` (*pyg4ometry.geant4.Registry.Registry* method), 319
- `transferSolidDefines()` (*pyg4ometry.geant4.Registry* method), 347
- `transferSolidDefines()` (*pyg4ometry.geant4.Registry.Registry* method), 321
- `transferSurface()` (*pyg4ometry.geant4.Registry* method), 346
- `transferSurface()` (*pyg4ometry.geant4.Registry.Registry* method), 319
- `Transform` (class in *pyg4ometry.fluka*), 177
- `Transform` (class in *pyg4ometry.fluka.directive*), 135
- `transform()` (*pyg4ometry.features.algos.CoordinateSystem* method), 97
- `transform()` (*pyg4ometry.features.CoordinateSystem* method), 99
- `transformationIndex` (*pyg4ometry.fluka.directive.RecursiveRotoTranslation* property), 135
- `transformationIndex` (*pyg4ometry.fluka.RecursiveRotoTranslation* property), 177
- `transformationIndex()` (*pyg4ometry.fluka.directive.RecursiveRotoTranslation* method), 136
- `transformationIndex()` (*pyg4ometry.fluka.RecursiveRotoTranslation* method), 178
- `transformDaughters()` (*pyg4ometry.geant4.LogicalVolume* method), 333, 341
- `transformDaughters()` (*pyg4ometry.geant4.LogicalVolume.LogicalVolume* method), 313
- `translate()` (*pyg4ometry.pycgal.core.CSG* method), 371
- `translate()` (*pyg4ometry.pycgal.CSG* method), 374
- `translate()` (*pyg4ometry.stl.Reader* method), 386
- `translate()` (*pyg4ometry.stl.Reader.Reader* method), 384
- `translation()` (*pyg4ometry.geant4.solid.Intersection* method), 278
- `translation()` (*pyg4ometry.geant4.solid.Intersection.Intersection* method), 235
- `translation()` (*pyg4ometry.geant4.solid.Subtraction* method), 282
- `translation()` (*pyg4ometry.geant4.solid.Subtraction.Subtraction* method), 118
- method), 244
- `translation()` (*pyg4ometry.geant4.solid.Union* method), 274
- `translation()` (*pyg4ometry.geant4.solid.Union.Union* method), 255
- `Trap` (class in *pyg4ometry.geant4.solid*), 287
- `Trap` (class in *pyg4ometry.geant4.solid.Trap*), 247
- `traverse()` (*pyg4ometry.pyoce.Reader* method), 383
- `traverse()` (*pyg4ometry.pyoce.Reader.Reader* method), 382
- `TRC` (class in *pyg4ometry.convert*), 85
- `TRC` (class in *pyg4ometry.fluka*), 162
- `TRC` (class in *pyg4ometry.fluka.body*), 124
- `Trd` (class in *pyg4ometry.geant4.solid*), 268
- `Trd` (class in *pyg4ometry.geant4.solid.Trd*), 248
- `triangulatePolygon2d()` (*pyg4ometry.convert._PolygonProcessing* class method), 82
- `triangulatePolygon2d()` (*pyg4ometry.pycgal.core.PolygonProcessing* class method), 373
- `triangulatePolygon2d()` (*pyg4ometry.pycgal.PolygonProcessing* class method), 376
- `Tubs` (class in *pyg4ometry.geant4.solid*), 263
- `Tubs` (class in *pyg4ometry.geant4.solid.Tubs*), 249
- `TwistedBox` (class in *pyg4ometry.geant4.solid*), 294
- `TwistedBox` (class in *pyg4ometry.geant4.solid.TwistedBox*), 250
- `TwistedSolid` (class in *pyg4ometry.geant4.solid.TwistedSolid*), 251
- `TwistedTrap` (class in *pyg4ometry.geant4.solid*), 296
- `TwistedTrap` (class in *pyg4ometry.geant4.solid.TwistedTrap*), 251
- `TwistedTrd` (class in *pyg4ometry.geant4.solid*), 298
- `TwistedTrd` (class in *pyg4ometry.geant4.solid.TwistedTrd*), 252
- `TwistedTubs` (class in *pyg4ometry.geant4.solid*), 299
- `TwistedTubs` (class in *pyg4ometry.geant4.solid.TwistedTubs*), 253
- `two_fold_orientation()` (in module *pyg4ometry.geant4.solid*), 277, 281, 285, 308
- `two_fold_orientation()` (in module *pyg4ometry.transformation*), 424
- `twoPiComparisonTolerance` (in module *pyg4ometry.config*), 417
- `TwoVector` (class in *pyg4ometry.geant4.solid*), 261
- `TwoVector` (class in *pyg4ometry.geant4.solid.TwoVector*), 254

U

`unaryExpression()` (*pyg4ometry.fluka.RegionExpression.RegionParser*

- unaryExpression() (pyg4ometry.fluka.RegionExpression.RegionExpression method), 108
 - unaryExpression() (pyg4ometry.fluka.RegionExpression.RegionExpression method), 106
 - unaryExpression() (pyg4ometry.fluka.RegionExpression.RegionExpression method), 106
 - unaryExpression() (pyg4ometry.fluka.RegionExpression.RegionExpression method), 116
 - unaryExpression() (pyg4ometry.fluka.RegionExpression.RegionExpression method), 115
 - Union (class in pyg4ometry.fluka.region), 152
 - Union (class in pyg4ometry.geant4.solid), 274
 - Union (class in pyg4ometry.geant4.solid.Union), 255
 - union() (pyg4ometry.fluka.AABB method), 173
 - union() (pyg4ometry.fluka.vector.AABB method), 157
 - union() (pyg4ometry.pycgal.core.CSG method), 371
 - union() (pyg4ometry.pycgal.CSG method), 375
 - unit() (in module pyg4ometry.gdml.Units), 207
 - unit() (pyg4ometry.fluka.Three method), 172
 - unit() (pyg4ometry.fluka.vector.Three method), 156
 - units (in module pyg4ometry.gdml.Units), 207
 - update() (pyg4ometry.utils.Timer method), 425
 - updateClipperPlaneCallback() (pyg4ometry.visualisation.VtkViewerNew method), 412
 - updateClipperPlaneCallback() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 401
 - updateTotal() (pyg4ometry.utils.Timer method), 425
 - upgradeToExpression() (in module pyg4ometry.gdml), 218
 - upgradeToExpression() (in module pyg4ometry.gdml.Defines), 198
 - upgradeToStringExpression() (in module pyg4ometry.gdml), 218
 - upgradeToStringExpression() (in module pyg4ometry.gdml.Defines), 198
 - upgradeToTransformation() (in module pyg4ometry.gdml), 218
 - upgradeToTransformation() (in module pyg4ometry.gdml.Defines), 198
 - upgradeToVector() (in module pyg4ometry.gdml), 218
 - upgradeToVector() (in module pyg4ometry.gdml.Defines), 198
 - useFreeCAD (in module pyg4ometry.freecad), 182
 - Usrbdx (class in pyg4ometry.analysis.flukaData), 52
 - Usrbin (class in pyg4ometry.analysis.flukaData), 51
 - Usrdump (class in pyg4ometry.analysis.flukaData), 52
- ## V
- values() (pyg4ometry.fluka.fluka_registry.FlukaBodyStoreExact method), 142
 - values() (pyg4ometry.fluka.FlukaBodyStoreExact method), 172
 - Variable (class in pyg4ometry.gdml.Defines), 202
 - VARIABLE() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpressionLexer attribute), 184
 - VARIABLE() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser attribute), 193
 - variable() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 188
 - variable() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 189
 - VARIABLE() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser method), 190
 - variables() (pyg4ometry.gdml.BasicExpression method), 217
 - variables() (pyg4ometry.gdml.Defines.BasicExpression method), 197
 - VectorBase (class in pyg4ometry.gdml), 222
 - VectorBase (class in pyg4ometry.gdml.Defines), 202
 - vertex_to_polygon (in module pyg4ometry.pycgal_old), 381
 - vertex_to_polygon (in module pyg4ometry.pycgal_old.cgal), 379
 - vertexCount() (pyg4ometry.pycgal.core.CSG method), 371
 - vertexCount() (pyg4ometry.pycgal.CSG method), 374
 - vertexfacet_to_polyhedron (in module pyg4ometry.pycgal_old), 380
 - vertexfacet_to_polyhedron (in module pyg4ometry.pycgal_old.cgal), 378
 - view() (pyg4ometry.features.algos.vtkViewer method), 97
 - view() (pyg4ometry.fluka.vis.ViewableMixin method), 157
 - view() (pyg4ometry.visualisation.VtkViewer method), 410
 - view() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 398
 - view() (pyg4ometry.visualisation.VtkViewerNew method), 412
 - view() (pyg4ometry.visualisation.VtkViewerNew.VtkViewerNew method), 402
 - ViewableMixin (class in pyg4ometry.fluka.vis), 157
 - ViewerBase (class in pyg4ometry.visualisation), 405
 - ViewerBase (class in pyg4ometry.visualisation.ViewerBase), 390
 - viewLogicalVolumeDifference() (in module pyg4ometry.visualisation), 411
 - viewLogicalVolumeDifference() (in module pyg4ometry.visualisation.VtkViewer), 400
 - viewSection() (pyg4ometry.visualisation.VtkViewer method), 410
 - viewSection() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 410

- method), 399
- vis2oce() (in module *pyg4ometry.convert*), 94
- vis2oce() (in module *pyg4ometry.convert.vis2oce*), 79
- visit_BinOp() (*pyg4ometry.fluka.preprocessor._Calc* method), 148
- visit_Call() (*pyg4ometry.fluka.preprocessor._Calc* method), 148
- visit_Constant() (*pyg4ometry.fluka.preprocessor._Calc* method), 148
- visit_Expr() (*pyg4ometry.fluka.preprocessor._Calc* method), 148
- visit_Name() (*pyg4ometry.fluka.preprocessor._Calc* method), 148
- visit_UnaryOp() (*pyg4ometry.fluka.preprocessor._Calc* method), 148
- visitAtom() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 182
- visitAtom() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 195
- visitComplexRegion() (*pyg4ometry.fluka.reader.RegionVisitor* method), 149
- visitComplexRegion() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor* method), 110
- visitComplexRegion() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor.RegionParserVisitor* method), 109
- visitConstant() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 183
- visitConstant() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 195
- visitEquation() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 194
- visitExpression() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 182
- visitExpression() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 194
- visitFunc() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 182
- visitFunc() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 195
- visitFuncname() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 183
- visitFuncname() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 195
- visitMatrixElement() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 182
- visitMatrixElement() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 195
- visitMaxExpression() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 182
- visitMinExpression() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 182
- visitMultipleUnion() (*pyg4ometry.fluka.reader.RegionVisitor* method), 150
- visitMultipleUnion() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor* method), 111
- visitMultipleUnion() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor.RegionParserVisitor* method), 109
- visitMultipleUnion2() (*pyg4ometry.fluka.reader.RegionVisitor* method), 150
- visitMultipleUnion2() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor* method), 111
- visitMultipleUnion2() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor.RegionParserVisitor* method), 109
- visitMultiplyingExpression() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 182
- visitMultiplyingExpression() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 194
- visitOneSubZone() (*pyg4ometry.fluka.reader.RegionVisitor* method), 111
- visitOneSubZone() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor* method), 110
- visitOperatorAddSub() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 195
- visitOperatorMulDiv() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 195
- visitOperatorPow() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor.GdmlExpressionVisitor* method), 195
- visitPrintExpr() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionEval.GdmlExpressionEvalVisitor* method), 182
- visitRegions() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor* method), 110
- visitRegions() (*pyg4ometry.fluka.RegionExpression.RegionParserVisitor* method), 109
- visitRelop() (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor* method), 195

visitScientific() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionEvalVisitor
 method), 182
 visitScientific() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor
 method), 195
 visitSignedAtom() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionVisitor
 method), 182
 visitSignedAtom() (pyg4ometry.gdml.GdmlExpression.GdmlExpressionEvalVisitor
 method), 195
 visitSimpleRegion() (pyg4ometry.fluka.reader.RegionVisitor
 method), 149
 visitSimpleRegion() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 110
 visitSimpleRegion() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 109
 visitSingleUnary() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitSingleUnary() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 109
 visitSingleUnion() (pyg4ometry.fluka.reader.RegionVisitor
 method), 150
 visitSingleUnion() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitSingleUnion() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 109
 visitSubZone() (pyg4ometry.fluka.reader.RegionVisitor
 method), 150
 visitSubZone() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitSubZone() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 110
 visitUnaryAndBoolean() (pyg4ometry.fluka.reader.RegionVisitor
 method), 149
 visitUnaryAndBoolean() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitUnaryAndBoolean() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 110
 visitUnaryAndSubZone() (pyg4ometry.fluka.reader.RegionVisitor
 method), 149
 visitUnaryAndSubZone() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitUnaryAndSubZone() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 110
 visitUnaryExpression() (pyg4ometry.fluka.reader.RegionVisitor
 method), 149
 visitUnaryExpression() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitUnaryExpression() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitZoneBody() (pyg4ometry.fluka.reader.RegionVisitor
 method), 150
 visitZoneBody() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitZoneBody() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitZoneExpr() (pyg4ometry.fluka.reader.RegionVisitor
 method), 150
 visitZoneExpr() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitZoneExpr() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 109
 visitZoneSubZone() (pyg4ometry.fluka.reader.RegionVisitor
 method), 150
 visitZoneSubZone() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 111
 visitZoneSubZone() (pyg4ometry.fluka.RegionExpression.RegionParserVisitor
 method), 109
 VisualisationOptions (class in
 pyg4ometry.visualisation), 407
 VisualisationOptions (class in
 pyg4ometry.RegionParserVisitorVisualisationOptions),
 392
 visualise() (pyg4ometry.stl.Reader method), 386
 visualise() (pyg4ometry.stl.Reader.Reader method),
 385
 volume() (pyg4ometry.pycgal.core.CSG method), 372
 volume() (pyg4ometry.pycgal.CSG method), 375
 volumeTree() (pyg4ometry.geant4.Registry method),
 347
 volumeTree() (pyg4ometry.geant4.Registry.Registry
 method), 321
 vtkCutterPlane() (in module
 pyg4ometry.features.algos), 98
 VtkExporter (class in pyg4ometry.visualisation), 414
 VtkExporter (class in
 pyg4ometry.visualisation.VtkExporter), 393
 vtkLoadStl() (in module pyg4ometry.features.algos),
 97
 vtkPolydataEdgesInformation() (in module
 pyg4ometry.features.algos), 97
 vtkPolydataToActor() (in module
 pyg4ometry.features.algos), 97
 vtkPolydataToConnectedEdges() (in module

- pyg4ometry.features.algos), 97
 vtkPolyDataToNumpy() (in module pyg4ometry.convert), 93
 vtkPolyDataToNumpy() (in module pyg4ometry.visualisation), 414
 vtkPolyDataToNumpy() (in module pyg4ometry.visualisation.Convert), 387
 vtkTransformation2PyG4() (in module pyg4ometry.convert), 93
 vtkTransformation2PyG4() (in module pyg4ometry.visualisation), 414
 vtkTransformation2PyG4() (in module pyg4ometry.visualisation.Convert), 387
 vtkViewer (class in pyg4ometry.features.algos), 97
 VtkViewer (class in pyg4ometry.visualisation), 407
 VtkViewer (class in pyg4ometry.visualisation.VtkViewer), 396
 VtkViewerColoured (class in pyg4ometry.visualisation), 410
 VtkViewerColoured (class in pyg4ometry.visualisation.VtkViewer), 399
 VtkViewerColouredMaterial (class in pyg4ometry.visualisation), 411
 VtkViewerColouredMaterial (class in pyg4ometry.visualisation.VtkViewer), 400
 VtkViewerColouredMaterialNew (class in pyg4ometry.visualisation), 413
 VtkViewerColouredMaterialNew (class in pyg4ometry.visualisation.VtkViewerNew), 402
 VtkViewerColouredNew (class in pyg4ometry.visualisation), 413
 VtkViewerColouredNew (class in pyg4ometry.visualisation.VtkViewerNew), 402
 VtkViewerNew (class in pyg4ometry.visualisation), 412
 VtkViewerNew (class in pyg4ometry.visualisation.VtkViewerNew), 401
- ## W
- WED (class in pyg4ometry.convert), 86
 WED (class in pyg4ometry.fluka), 163
 WED (class in pyg4ometry.fluka.body), 125
 Wedge (class in pyg4ometry.geant4.solid), 262
 Wedge (class in pyg4ometry.geant4.solid.Wedge), 256
 wheelEvent() (pyg4ometry.gui.example1.QVTKRenderWindowInteractor method), 358
 wheelEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor method), 362, 364
 wheelEvent() (pyg4ometry.gui.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor method), 355
 Whitespace (pyg4ometry.fluka.RegionExpression.RegionLexer.RegionLexer attribute), 118
 Whitespace (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 118
 Whitespace (pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser attribute), 108
 windingNumber() (pyg4ometry.convert._PolygonProcessing class method), 81
 windingNumber() (pyg4ometry.pycgal.core.PolygonProcessing class method), 372
 windingNumber() (pyg4ometry.pycgal.PolygonProcessing class method), 375
 withLengthSafety() (pyg4ometry.fluka.Region method), 176
 withLengthSafety() (pyg4ometry.fluka.region.Region method), 154
 withLengthSafety() (pyg4ometry.fluka.region.Zone method), 153
 withLengthSafety() (pyg4ometry.fluka.Zone method), 174
 WORLD_DIMENSIONS (in module pyg4ometry.convert.fluka2Geant4), 69
 write() (pyg4ometry.bdsim.Beam method), 55
 write() (pyg4ometry.bdsim.beam.Beam method), 52
 write() (pyg4ometry.bdsim.Beamline method), 55
 write() (pyg4ometry.bdsim.beamline.Beamline method), 53
 write() (pyg4ometry.bdsim.Element method), 55
 write() (pyg4ometry.bdsim.element.Element method), 53
 write() (pyg4ometry.bdsim.Gmad method), 55
 write() (pyg4ometry.bdsim.gmad.Gmad method), 53
 write() (pyg4ometry.bdsim.Options method), 55
 write() (pyg4ometry.bdsim.options.Options method), 54
 write() (pyg4ometry.bdsim.Sampler method), 55
 write() (pyg4ometry.bdsim.sampler.Sampler method), 54
 write() (pyg4ometry.bdsim.Scorer method), 55
 write() (pyg4ometry.bdsim.ScorerMesh method), 55
 write() (pyg4ometry.bdsim.scoring.Scorer method), 54
 write() (pyg4ometry.bdsim.scoring.ScorerMesh method), 54
 write() (pyg4ometry.fluka.Flair method), 178
 write() (pyg4ometry.fluka.flair.Flair method), 137
 write() (pyg4ometry.fluka.Writer method), 170
 write() (pyg4ometry.fluka.Writer.Writer method), 120
 write() (pyg4ometry.gdml.Writer method), 215
 write() (pyg4ometry.gdml.Writer.Writer method), 207
 write() (pyg4ometry.utils.Samples method), 425
 write() (pyg4ometry.visualisation.RenderWriter method), 389
 write() (pyg4ometry.visualisation.RenderWriter.RenderWriter method), 389
 writeAppend() (pyg4ometry.utils.Samples method), 425

writeAssemblyVolume() (pyg4ometry.gdml.Writer method), 215
 writeAssemblyVolume() (pyg4ometry.gdml.Writer.Writer method), 208
 writeAuxiliary() (pyg4ometry.gdml.Writer method), 215
 writeAuxiliary() (pyg4ometry.gdml.Writer.Writer method), 208
 writeBorderSurface() (pyg4ometry.gdml.Writer method), 216
 writeBorderSurface() (pyg4ometry.gdml.Writer.Writer method), 208
 writeBox() (pyg4ometry.gdml.Writer method), 216
 writeBox() (pyg4ometry.gdml.Writer.Writer method), 208
 writeCons() (pyg4ometry.gdml.Writer method), 216
 writeCons() (pyg4ometry.gdml.Writer.Writer method), 208
 writeCutTubs() (pyg4ometry.gdml.Writer method), 216
 writeCutTubs() (pyg4ometry.gdml.Writer.Writer method), 208
 writeDefaultGDML() (pyg4ometry.stl.Reader method), 386
 writeDefaultGDML() (pyg4ometry.stl.Reader.Reader method), 385
 writeDefaultLattice() (pyg4ometry.gdml.Writer method), 215
 writeDefaultLattice() (pyg4ometry.gdml.Writer.Writer method), 208
 writeDefine() (pyg4ometry.gdml.Writer method), 215
 writeDefine() (pyg4ometry.gdml.Writer.Writer method), 208
 writeDivisionVolume() (pyg4ometry.gdml.Writer method), 216
 writeDivisionVolume() (pyg4ometry.gdml.Writer.Writer method), 208
 writeEllipsoid() (pyg4ometry.gdml.Writer method), 216
 writeEllipsoid() (pyg4ometry.gdml.Writer.Writer method), 208
 writeEllipticalCone() (pyg4ometry.gdml.Writer method), 216
 writeEllipticalCone() (pyg4ometry.gdml.Writer.Writer method), 209
 writeEllipticalTube() (pyg4ometry.gdml.Writer method), 216
 writeEllipticalTube() (pyg4ometry.gdml.Writer.Writer method),
 writeExtrudedSolid() (pyg4ometry.gdml.Writer method), 216
 writeExtrudedSolid() (pyg4ometry.gdml.Writer.Writer method), 209
 writeGenericPolycone() (pyg4ometry.gdml.Writer method), 216
 writeGenericPolycone() (pyg4ometry.gdml.Writer.Writer method), 209
 writeGenericPolyhedra() (pyg4ometry.gdml.Writer method), 216
 writeGenericPolyhedra() (pyg4ometry.gdml.Writer.Writer method), 209
 writeGenericTrap() (pyg4ometry.gdml.Writer method), 217
 writeGenericTrap() (pyg4ometry.gdml.Writer.Writer method), 209
 writeGmadTester() (pyg4ometry.gdml.Writer method), 215
 writeGmadTester() (pyg4ometry.gdml.Writer.Writer method), 208
 writeGMADTesterNoBeamline() (pyg4ometry.gdml.Writer method), 215
 writeGMADTesterNoBeamline() (pyg4ometry.gdml.Writer.Writer method), 208
 writeHype() (pyg4ometry.gdml.Writer method), 216
 writeHype() (pyg4ometry.gdml.Writer.Writer method), 209
 writeIntersection() (pyg4ometry.gdml.Writer method), 216, 217
 writeIntersection() (pyg4ometry.gdml.Writer.Writer method), 209, 210
 writeLogicalVolume() (pyg4ometry.gdml.Writer method), 215
 writeLogicalVolume() (pyg4ometry.gdml.Writer.Writer method), 208
 writeMaterial() (pyg4ometry.gdml.Writer method), 215
 writeMaterial() (pyg4ometry.gdml.Writer.Writer method), 208
 writeMaterialProps() (pyg4ometry.gdml.Writer method), 215
 writeMaterialProps() (pyg4ometry.gdml.Writer.Writer method), 208
 writeMultiUnion() (pyg4ometry.gdml.Writer method), 217
 writeMultiUnion() (pyg4ometry.gdml.Writer.Writer method), 210

- writeOpticalSurface() (pyg4ometry.gdml.Writer method), 216
 writeOpticalSurface() (pyg4ometry.gdml.Writer.Writer method), 209
 writeOrb() (pyg4ometry.gdml.Writer method), 216
 writeOrb() (pyg4ometry.gdml.Writer.Writer method), 209
 writePara() (pyg4ometry.gdml.Writer method), 216
 writePara() (pyg4ometry.gdml.Writer.Writer method), 209
 writeParaboloid() (pyg4ometry.gdml.Writer method), 216
 writeParaboloid() (pyg4ometry.gdml.Writer.Writer method), 209
 writeParametrisedVolume() (pyg4ometry.gdml.Writer method), 216
 writeParametrisedVolume() (pyg4ometry.gdml.Writer.Writer method), 208
 writePhysicalVolume() (pyg4ometry.gdml.Writer method), 215
 writePhysicalVolume() (pyg4ometry.gdml.Writer.Writer method), 208
 writePolycone() (pyg4ometry.gdml.Writer method), 216
 writePolycone() (pyg4ometry.gdml.Writer.Writer method), 209
 writePolyhedra() (pyg4ometry.gdml.Writer method), 217
 writePolyhedra() (pyg4ometry.gdml.Writer.Writer method), 209
 Writer (class in pyg4ometry.fluka), 170
 Writer (class in pyg4ometry.fluka.Writer), 120
 Writer (class in pyg4ometry.gdml), 215
 Writer (class in pyg4ometry.gdml.Writer), 207
 writeReplicaVolume() (pyg4ometry.gdml.Writer method), 215
 writeReplicaVolume() (pyg4ometry.gdml.Writer.Writer method), 208
 writeScaled() (pyg4ometry.gdml.Writer method), 217
 writeScaled() (pyg4ometry.gdml.Writer.Writer method), 210
 writeSkinSurface() (pyg4ometry.gdml.Writer method), 216
 writeSkinSurface() (pyg4ometry.gdml.Writer.Writer method), 208
 WriteSMeshFile() (in module pyg4ometry.freecad.Reader), 181
 writeSolid() (pyg4ometry.gdml.Writer method), 216
 writeSolid() (pyg4ometry.gdml.Writer.Writer method), 208
 writeSphere() (pyg4ometry.gdml.Writer method), 217
 writeSphere() (pyg4ometry.gdml.Writer.Writer method), 209
 writeSubtraction() (pyg4ometry.gdml.Writer method), 217
 writeSubtraction() (pyg4ometry.gdml.Writer.Writer method), 210
 writeTessellatedSolid() (pyg4ometry.gdml.Writer method), 216
 writeTessellatedSolid() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTet() (pyg4ometry.gdml.Writer method), 217
 writeTet() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTorus() (pyg4ometry.gdml.Writer method), 217
 writeTorus() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTrap() (pyg4ometry.gdml.Writer method), 217
 writeTrap() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTrd() (pyg4ometry.gdml.Writer method), 217
 writeTrd() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTubs() (pyg4ometry.gdml.Writer method), 217
 writeTubs() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTwistedBox() (pyg4ometry.gdml.Writer method), 217
 writeTwistedBox() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTwistedTrap() (pyg4ometry.gdml.Writer method), 217
 writeTwistedTrap() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTwistedTrd() (pyg4ometry.gdml.Writer method), 217
 writeTwistedTrd() (pyg4ometry.gdml.Writer.Writer method), 209
 writeTwistedTubs() (pyg4ometry.gdml.Writer method), 217
 writeTwistedTubs() (pyg4ometry.gdml.Writer.Writer method), 210
 writeUnion() (pyg4ometry.gdml.Writer method), 217
 writeUnion() (pyg4ometry.gdml.Writer.Writer method), 210
 writeVectorVariable() (pyg4ometry.gdml.Writer method), 215
 writeVectorVariable() (pyg4ometry.gdml.Writer.Writer method), 208
 writeVtkPolyDataAsSTLFile() (in module pyg4ometry.visualisation.Writer), 403
 WS (pyg4ometry.gdml.GdmlExpression.GdmlExpressionLexer.GdmlExpression

attribute), 184

WS (*pyg4ometry.gdml.GdmlExpression.GdmlExpressionParser.GdmlExpressionParser* methods), 193

attribute), 193

X

x (*pyg4ometry.fluka.Three* property), 172

x (*pyg4ometry.fluka.vector.Three* property), 156

XCC (class in *pyg4ometry.convert*), 89

XCC (class in *pyg4ometry.fluka*), 165

XCC (class in *pyg4ometry.fluka.body*), 127

XEC (class in *pyg4ometry.convert*), 90

XEC (class in *pyg4ometry.fluka*), 167

XEC (class in *pyg4ometry.fluka.body*), 129

XYP (class in *pyg4ometry.convert*), 87

XYP (class in *pyg4ometry.fluka*), 164

XYP (class in *pyg4ometry.fluka.body*), 126

XZP (class in *pyg4ometry.convert*), 88

XZP (class in *pyg4ometry.fluka*), 164

XZP (class in *pyg4ometry.fluka.body*), 126

Y

y (*pyg4ometry.fluka.Three* property), 172

y (*pyg4ometry.fluka.vector.Three* property), 156

YCC (class in *pyg4ometry.convert*), 89

YCC (class in *pyg4ometry.fluka*), 166

YCC (class in *pyg4ometry.fluka.body*), 128

YEC (class in *pyg4ometry.convert*), 91

YEC (class in *pyg4ometry.fluka*), 167

YEC (class in *pyg4ometry.fluka.body*), 129

YZP (class in *pyg4ometry.convert*), 88

YZP (class in *pyg4ometry.fluka*), 164

YZP (class in *pyg4ometry.fluka.body*), 126

Z

z (*pyg4ometry.fluka.Three* property), 172

z (*pyg4ometry.fluka.vector.Three* property), 156

ZCC (class in *pyg4ometry.convert*), 90

ZCC (class in *pyg4ometry.fluka*), 166

ZCC (class in *pyg4ometry.fluka.body*), 128

ZEC (class in *pyg4ometry.convert*), 91

ZEC (class in *pyg4ometry.fluka*), 168

ZEC (class in *pyg4ometry.fluka.body*), 130

Zone (class in *pyg4ometry.fluka*), 173

Zone (class in *pyg4ometry.fluka.region*), 152

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser* method), 118

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.MultipleUnion2Context* method), 113

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.MultipleUnionContext* method), 114

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser* method), 108

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.MultipleUnion2Context* method), 104

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.MultipleUnionContext* method), 103

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.SimpleRegionContext* method), 103

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser.RegionParser* method), 104

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.SimpleRegionContext* method), 113

zone() (*pyg4ometry.fluka.RegionExpression.RegionParser.SingleUnionContext* method), 114

zone_to_sympy() (in module *pyg4ometry.fluka*), 177

zone_to_sympy() (in module *pyg4ometry.fluka.region*), 151

zoneAABBs() (*pyg4ometry.fluka.Region* method), 176

zoneAABBs() (*pyg4ometry.fluka.region.Region* method), 154

zoneGraph() (*pyg4ometry.fluka.Region* method), 176

zoneGraph() (*pyg4ometry.fluka.region.Region* method), 154

zoneToAlgebraicExpression() (in module *pyg4ometry.fluka.boolean_algebra*), 132

zoneToDNFZones() (in module *pyg4ometry.fluka.boolean_algebra*), 132

zoneUnion() (*pyg4ometry.fluka.RegionExpression.RegionParser* method), 118

zoneUnion() (*pyg4ometry.fluka.RegionExpression.RegionParser.ComplexRegionParser* method), 112

zoneUnion() (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser* method), 108

zoneUnion() (*pyg4ometry.fluka.RegionExpression.RegionParser.RegionParser* method), 103