pyg4ometry Documentation

Release 1.0.0

Royal Holloway

Jul 01, 2021

CONTENTS

1 Licence & Disclaimer **3**

2 Authorship **5**

3 Installation **7**
- 3.1 Requirements . . . 7
- 3.2 Installation . . . 7
- 3.3 Docker image . . . 8
- 3.4 Linux installation . . . 8
- 3.5 FreeCAD support for CAD to GDML conversion . . . 8
- 3.6 Python 3.9 . . . 9

4 Introduction **11**
- 4.1 Need for programatic geometry generation . . . 11
- 4.2 Geant4 key concepts . . . 11
- 4.3 Geometry key concepts . . . 11
- 4.4 Implementation concepts . . . 12
- 4.5 Publications . . . 12

5 Basic python geometry scripting **13**
- 5.1 Precepts . . . 13
- 5.2 Units . . . 13
- 5.3 Geant4 Python Scripting . . . 14
- 5.4 GDML Defines . . . 14
- 5.5 Solids . . . 16
- 5.6 Materials . . . 16
  - 5.6.1 NIST Materials . . . 18
- 5.7 Detector Construction . . . 18
- 5.8 Transformations & Physical Volumes . . . 18
- 5.9 Optical Surfaces . . . 19
- 5.10 Registry and GDML Output . . . 19
- 5.11 Visualisation . . . 19
- 5.12 Overlap Checking . . . 20
- 5.13 GDML Output . . . 21

6 Tutorials **23**
- 6.1 GDML loading . . . 23
- 6.2 STL loading . . . 24
- 6.3 STEP/STP loading . . . 24
- 6.4 Merging geometry . . . 25
- 6.5 Assembly conversion . . . 26
- 6.6 STL output . . . 26
- 6.7 GDML Conversion to FLUKA . . . 27
- 6.8 Conversion of FLUKA To GDML . . . 28

7 Advanced tutorials 31
7.1 Finding volumes . . . 31
7.2 Navigating the LV-PV hierarchy . . . 31
7.3 Edit existing geometry . . . 32
7.4 Complex geometry builder . . . 32
7.5 Fluka geometry scripting . . . 32
7.6 Export scene to paraview/vtk . . . 32
7.7 Export scene to unity/unreal . . . 33

8 Module Contents 35
8.1 Defines and variables . . . 35
8.2 Geant4 solids . . . 39
8.3 Geant4 module . . . 50
8.4 VTK module . . . 53
8.5 Freecad module . . . 56
8.6 STL module . . . 56
8.7 GDML module . . . 57
8.8 Fluka bodies . . . 60
8.9 Fluka module . . . 67

9 Developer 69
9.1 Unit tests . . . 69
9.2 Coverage . . . 69
9.3 Profiling . . . 69
9.4 Updating A Version . . . 69
9.5 Updating Copyright . . . 70

10 Version History 71
10.1 v 1.0.0 - 2021 / 07 / 01 . . . 71
10.2 Pre-History . . . 71

11 Indices and tables 73

Python Module Index 75

Index 77

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

CHAPTER

ONE

LICENCE & DISCLAIMER

CHAPTER

TWO

AUTHORSHIP

The following people have contributed to pyg4ometry:

- Stewart Boogert (RHUL)
- Andrey Abramov (RHUL -> CERN)
- Alistair Butcher (RHUL -> Amazon)
- Cedric Hernalsteens (ULB -> CERN)
- Laurie Nevay (RHUL)
- William Shields (RHUL)
- Ben Shellswell (former RHUL)
- Eliott Ramoisiaux (ULB)
- Stuart Walker (RHUL -> DESY)

CHAPTER

THREE

INSTALLATION

3.1 Requirements

- pyg4ometry is developed exclusively for Python 3 (Python2 is deprecated) - recommend up to Python 3.8 currently as no VTK wheels for 3.9.
- VTK (Visualisation toolkit)
- Freecad
- antlr4
- cython
- GitPython
- matplotlib
- CGAL
- pybind11

3.2 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.

3.3 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)`

3.4 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

3.5 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/0.19_pre.tar.gz
tar zxf 0.19_pre.tar.gz
mkdir build
mkdir install
```

(continues on next page)

(continued from previous page)

```
cd build
cmake ../FreeCAD-0.18.4 -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
```

3.6 Python 3.9

Generally we recommend up to Python 3.8.

At the time of writing, there are no VTK distributions for Python 3.9 on pypi. 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.

Similarly, there are fixed version requirements for networkx and antlr. These specific versions will not be available for Python 3.9 but newer versions will likely work. In this case, you can remove the fixed version requirement (leaving only the name).

> **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.

CHAPTER

FOUR

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 flies.

4.1 Need for programatic 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

4.2 Geant4 key concepts

- GMDL

4.3 Geometry key concepts

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

4.4 Implementation concepts

- Registry
- Parameter
- ParameterVector
- Pycsg

4.5 Publications

On pyg4ometry

- Pyg4ometry : A Tool To Create Geometries For Geant4, Bdsmi, G4Beamline and Fluka For Particle Loss and Energy Deposit Studies, IPAC2019, Melbourne, Australia, 2019 Google scholar cites

CHAPTER

FIVE

BASIC PYTHON GEOMETRY SCRIPTING

5.1 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

5.2 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
none	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 specifed), so mm. We can specify them:

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

5.3 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

```
# load pyg4ometry
import pyg4ometry

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

# world solid and logical
ws   = pyg4ometry.geant4.solid.Box("ws",50,50,50,reg)
wl   = pyg4ometry.geant4.LogicalVolume(ws,"G4_Galactic","wl",reg)
reg.setWorld(wl.name)

# box placed at origin
b1   = pyg4ometry.geant4.solid.Box("b1",10,10,10,reg)
b1_l = pyg4ometry.geant4.LogicalVolume(b1,"G4_Fe","b1_l",reg)
b1_p = pyg4ometry.geant4.PhysicalVolume([0,0,0],[0,0,0],b1_l,"b1_p",wl,reg)

# visualise geometry
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(wl)
v.addAxes(20)
v.view()
```

Here is the vtk visualiser output of the above example

5.4 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

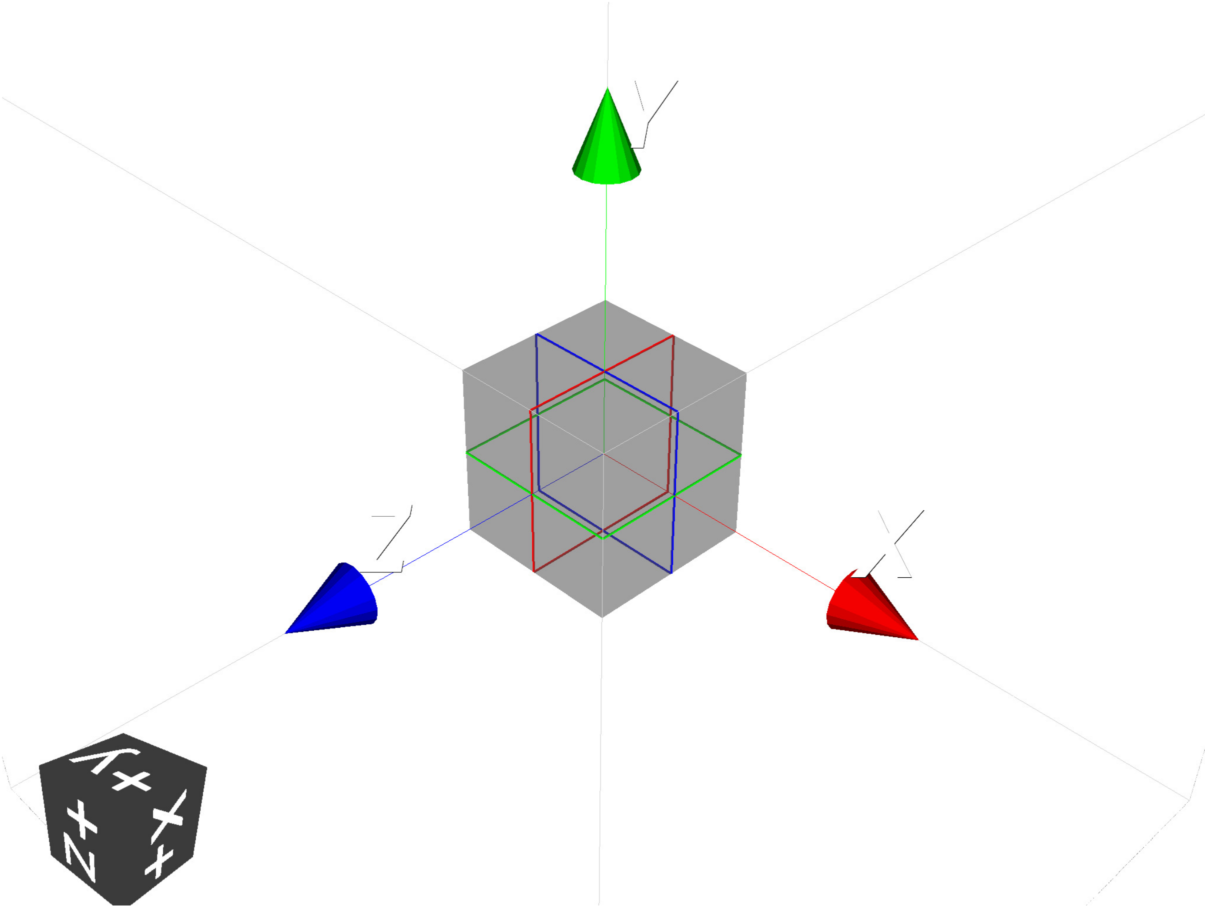
```
# registry to store gdml data
reg = pyg4ometry.geant4.Registry()

# 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

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

So the box example above can be rewritten using constants

```
# load pyg4ometry
import pyg4ometry

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

bx = pyg4ometry.gdml.Constant("bx","10",reg,True)
by = pyg4ometry.gdml.Constant("by",2*bx,reg,True)
bz = pyg4ometry.gdml.Constant("bz",2*by,reg,True)

# world solid and logical
ws   = pyg4ometry.geant4.solid.Box("ws",50,50,50,reg)
```

(continues on next page)

(continued from previous page)

```python
wl   = pyg4ometry.geant4.LogicalVolume(ws,"G4_Galactic","wl",reg)

# box placed at origin
b1   = pyg4ometry.geant4.solid.Box("b1",bx,by,bz,reg)
b1_l = pyg4ometry.geant4.LogicalVolume(b1,"G4_Fe","b1_l",reg)
b1_p = pyg4ometry.geant4.PhysicalVolume([0,0,0],[0,0,0],b1_l,"b1_p",wl,reg)

# visualise geometry
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(wl)
v.addAxes(20)
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

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

5.5 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

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

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

A full list of solids can be found in *Geant4 solids*.

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

5.6 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

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

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

```
import pyg4ometry.geant4 as _g4
wm = _g4.MaterialSingleElement("galactic",1,1.008,1e-25,reg)   # low density hydrogen
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

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

To define a compound using number of atoms

```
import pyg4ometry.geant4 as _g4
bm = _g4.MaterialCompound("plastic",1.38,3,reg)    # Generic PET C_10 H_8 O_4
he = _g4.ElementSimple("hydrogen","H",1,1.008)
ce = _g4.ElementSimple("carbon","C",6,12.0096)
oe = _g4.ElementSimple("oxygen","O",8,16.0)
bm.add_element_natoms(he,8)
bm.add_element_natoms(ce,10)
bm.add_element_natoms(oe,4)
```

Material as a mixture of materials

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

Example of elements formed by isotopes

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

5.6.1 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.

```
import pyg4ometry
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.

```
import pyg4ometry
nistHydrogenMaterial = pyg4ometry.geant4.nist_material_2geant4Material('G4_H')
nistConcreteMaterial = pyg4ometry.geant4.nist_material_2geant4Material('G4_CONCRETE')
```

5.7 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

5.8 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 classes like in Geant4, matrices and vectors used for placements are 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: *Geant4 module*, but an example is shown here.

```
import pyg4ometry
r = pyg4ometry.geant4.Registry()
vacuum = _g4.MaterialPredefined("G4_Galactic")
water = _g4.MaterialPredefined("G4_WATER")
worldSolid = pyg4ometry.geant4.solid.Box("world_solid", 100, 100, 100, reg)
```

(continues on next page)

(continued from previous page)

```python
boxSolid = pyg4ometry.geant4.solid.Box("box_solid", 10, 20, 40, reg)
worldLV = pyg4ometry.geant4.LogicalVolume(worldSolid, vacuum, "world_lv", reg)
boxLV = pyg4ometry.geant4.LogicalVolume(boxSolid, water, "box_lv", reg)

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:

```python
import numpy as np
pyg4ometry.geant4.PhysicalVolume([0,np.pi/3.0,0],
                                 [0,0,0],
                                 boxLV,
                                 "box_pv",
                                 worldLV,
                                 reg)
```

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.

5.9 Optical Surfaces

5.10 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 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 an 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

5.11 Visualisation

Any logical volume `lv` can be visualised using

```python
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(lv)
v.addAxes(20)
v.view()
```

which will open a Vtk render window. The render window now receives keyboard and mouse commands. To exit render window `q`, to restart interaction with the visualiser

```python
v.start()
```

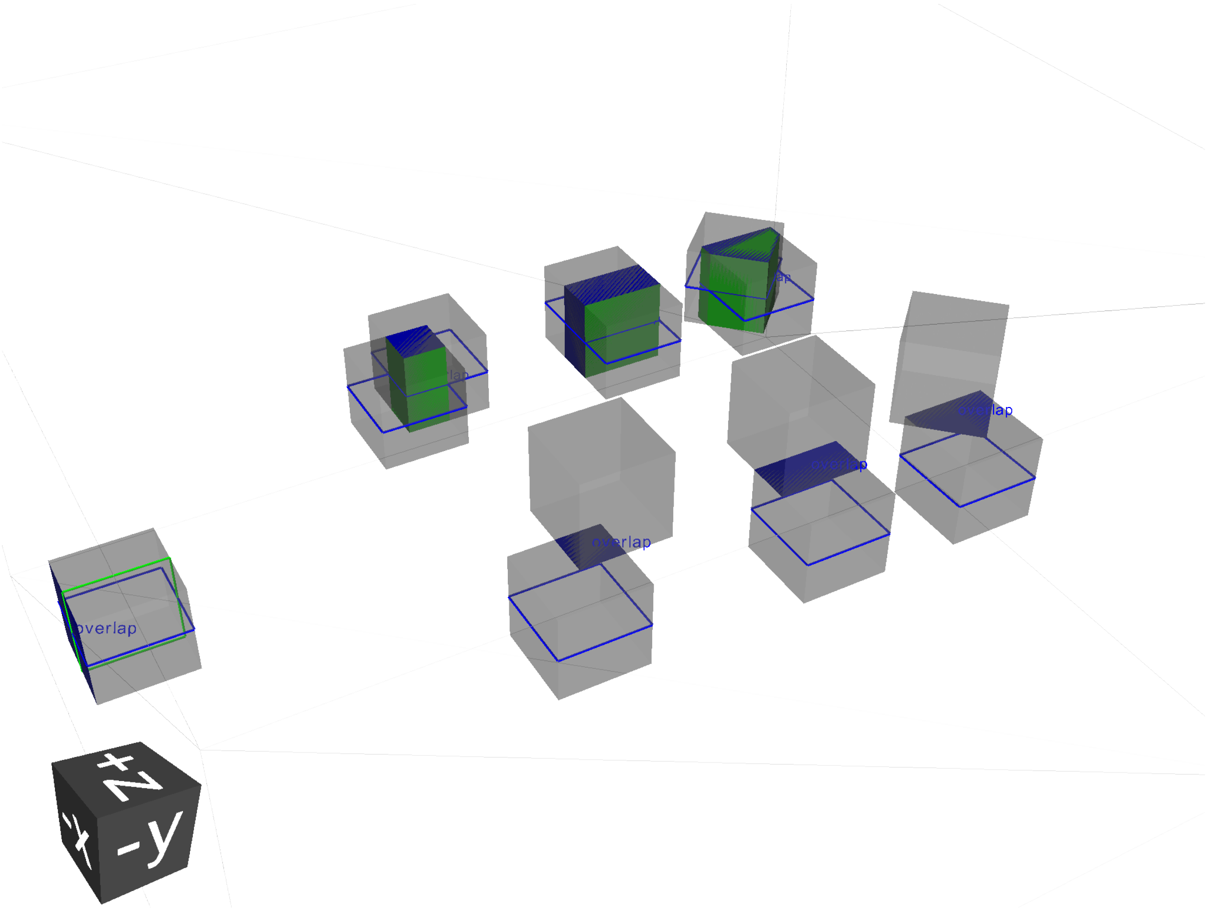
There are also convenience methods of `pyg4ometry.visualisation.VtkViewer()` the allow changing of the viewing parameters. So if the viewer is active then render window needs to be stopped `q` and then commands can be typed into the terminal, for example

```
v.setOpactity(0.1)
v.setWirefrace()
v.start()
```

5.12 Overlap Checking

Given all the PVs (daughters) of a LV (mother) should be bounded by the LV/mother solid. It is possible check between all daughter solid meshes and between daughters and the mother solid mesh. Given an `LV` this check can be performed by calling the following code.

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



There is no output when `checkOverlaps` is called but a overlap, protrusion or coplanar meshes are computed and stored in the logical volume instance and displayed by the `VtkViewer`

5.13 GDML Output

To write an GDML file file given a `pyg4ometry.geant4.registy reg`

```
import pyg4ometry
w = p4gometry.gdml.Writer()
w.addDetector(reg)
w.write('./file.gdml')
w.writeGmadTester('./file.gmad')
```

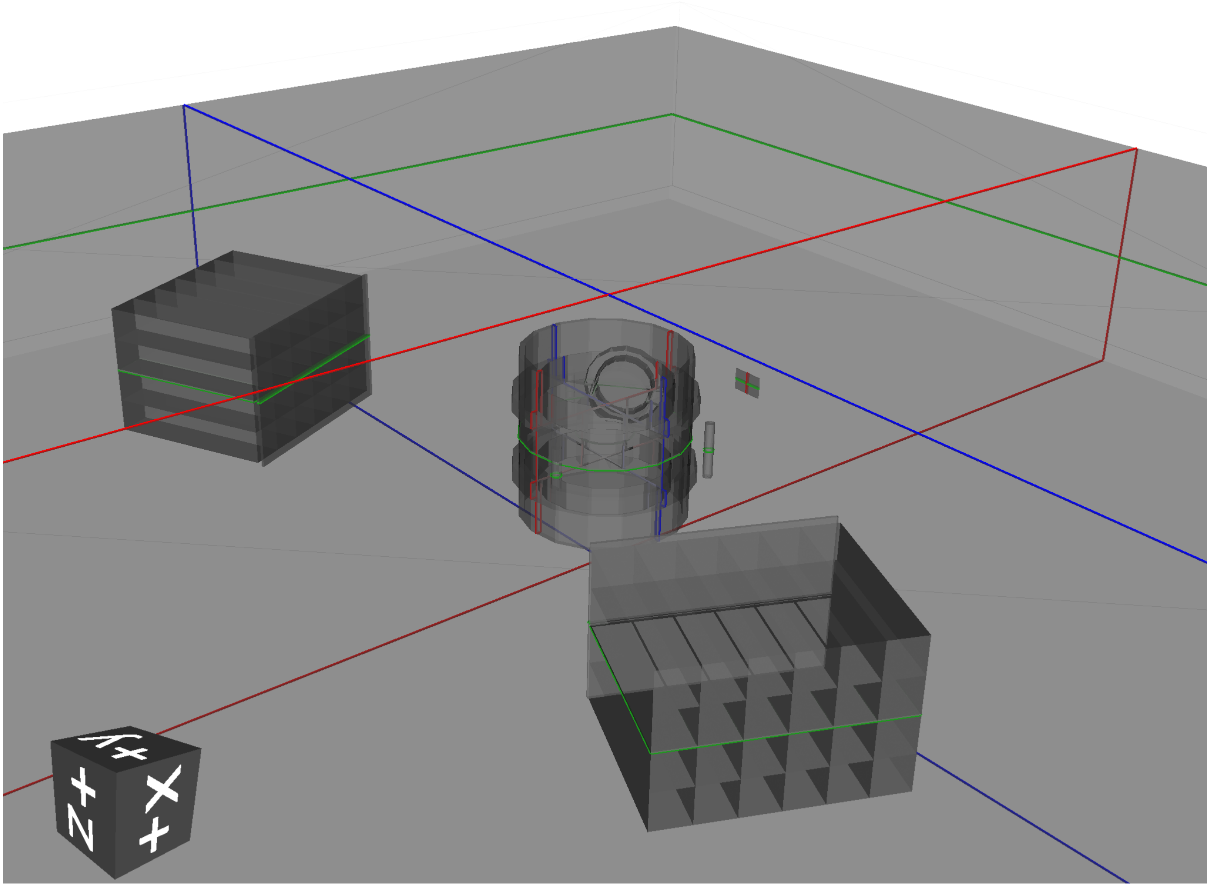
CHAPTER SIX

TUTORIALS

6.1 GDML loading

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

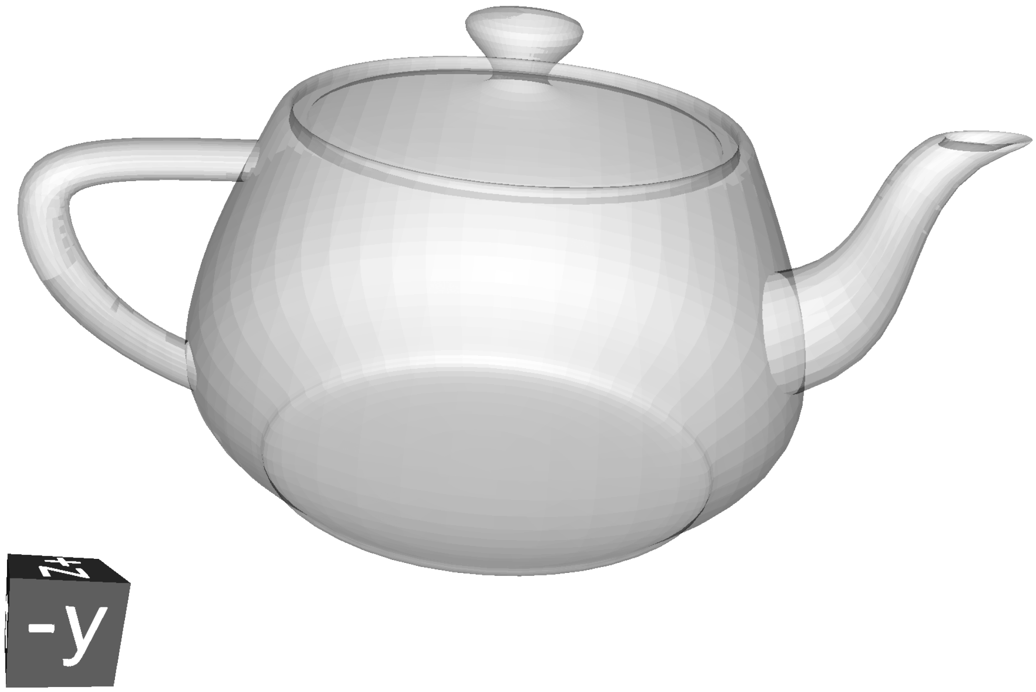
```
import pyg4ometry
r = pyg4ometry.gdml.Reader("lht.gdml")
l = r.getRegistry().getWorldVolume()
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(l)
v.view()
```



6.2 STL loading

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

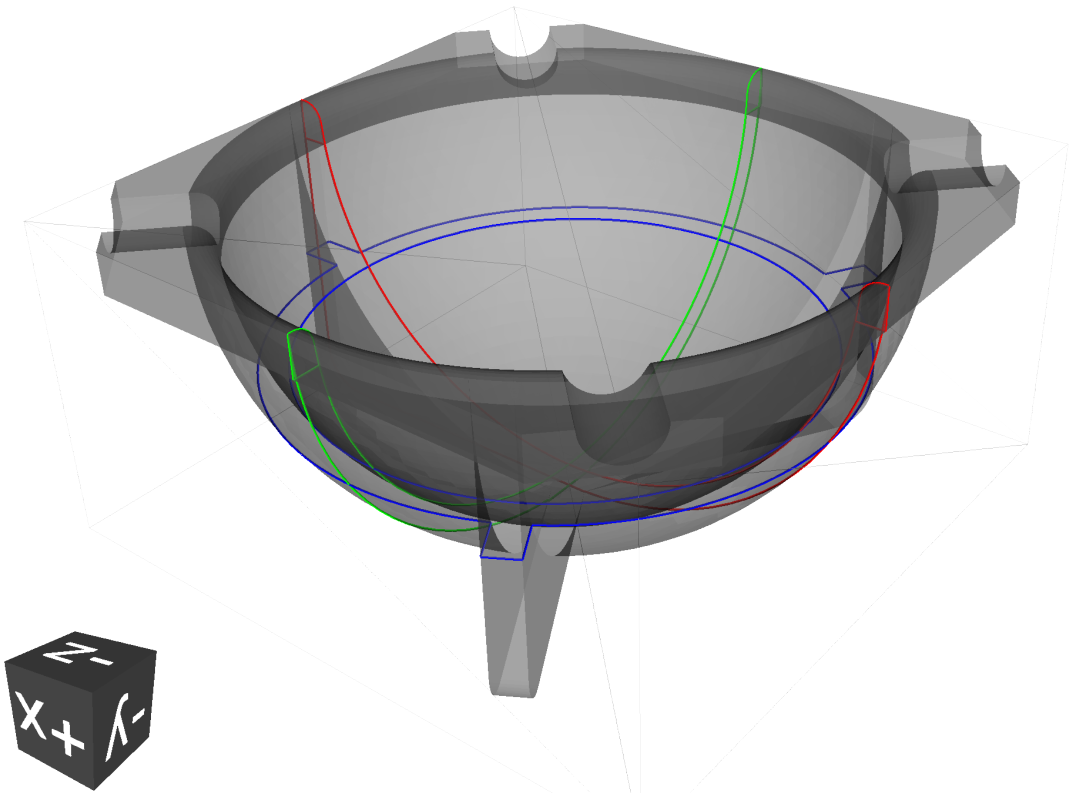
```
import pyg4ometry
reg = pyg4ometry.geant4.Registry()
r = pyg4ometry.stl.Reader("utahteapot.stl")
l = r.logicalVolume("test","G4_Cu",reg)
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(l)
v.view()
```



6.3 STEP/STP loading

In directory `pyg4ometry/pyg4ometry/test/freecad`

```
import pyg4ometry
r  = pyg4ometry.freecad.Reader("./08_AshTray.step")
r.relabelModel()
r.convertFlat()
l = r.getRegistry().getWorldVolume()
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(l)
v.view()
```



6.4 Merging 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

```
import T001_Box
T001_Box.Test(True,True)

import T002_Tubs
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

```
import pyg4ometry
r = pyg4ometry.gdml.Reader("T002_Tubs.gdml")
reg = r.getRegistry()

# printing the names of the logical volumes
print(reg.logicalVolumeDict.keys())

# printing the names of the physical volumes
print(reg.physicalVolumeDict.keys())

```

(continues on next page)

(continued from previous page)

```
lv = reg.logicalVolume["tl"]
```

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

```
import pyg4ometry
r1 = pyg4ometry.gdml.Reader("T001_Box.gdml")
reg1 = r1.getRegistry()

r2 = pyg4ometry.gdml.Reader("T002_Tubs.gdml")
reg2 = r2.getRegistry()

lv = reg2.logicalVolumeDict["tl"]

# create physical volume with placement
pv = pyg4ometry.geant4.PhysicalVolume([0,0,0],[50,0,0], lv, "tl_pv", reg1.
→getWorldVolume(), reg1)

reg1.addVolumeRecursive(pv)

# gdml output
w = pyg4ometry.gdml.Writer()
w.addDetector(reg1)
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.

6.5 Assembly conversion

Given two sources of geometry, placement of top level world logical volume solids will likely result in an overlap. To avoid these types of problems, it might required to convert one of the logical volumes to an AssemblyVolume.

6.6 STL output

To write an STL file from `m = volume.pycsgmesh()`

```
vtkConverter = vtk.Convert()
vtkPD        =  vtkConverter.MeshListToPolyData(m)
r = vtk.WriteSTL("file.stl",vtkPD)
```

6.7 GDML Conversion to FLUKA

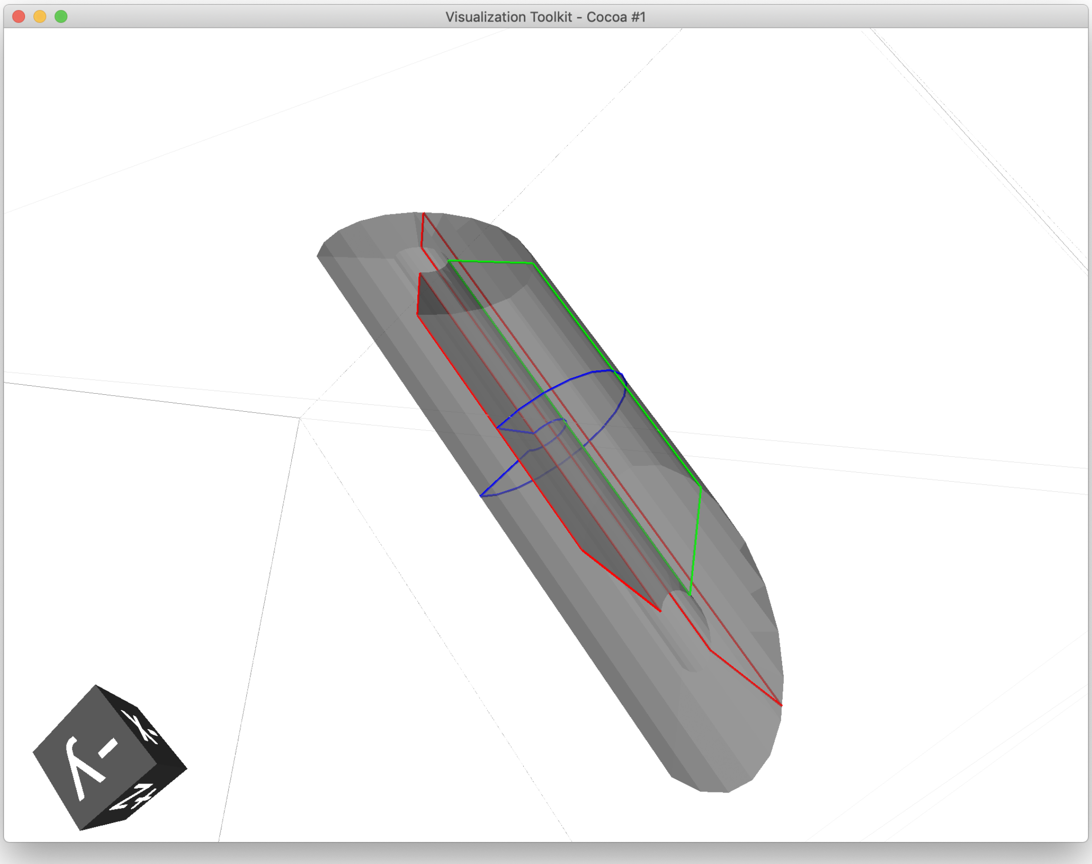
It is possible convert a pyg4ometry geometry to FLUKA. This is currently a work in progress and not all Geant4-GDML constructions are implemented, although they can be quickly added. Given a LV variable named `logical`

```
import pyg4ometry
reader = pyg4ometry.gdml.Reader("input.gdml")
logical = reader.getRegistry().getWorldVolume()
freg = pyg4ometry.convert.geant4Logical2Fluka(logical)
w = pyg4ometry.fluka.Writer()
w.addDetector(freg)
w.write("FileName.inp")
```

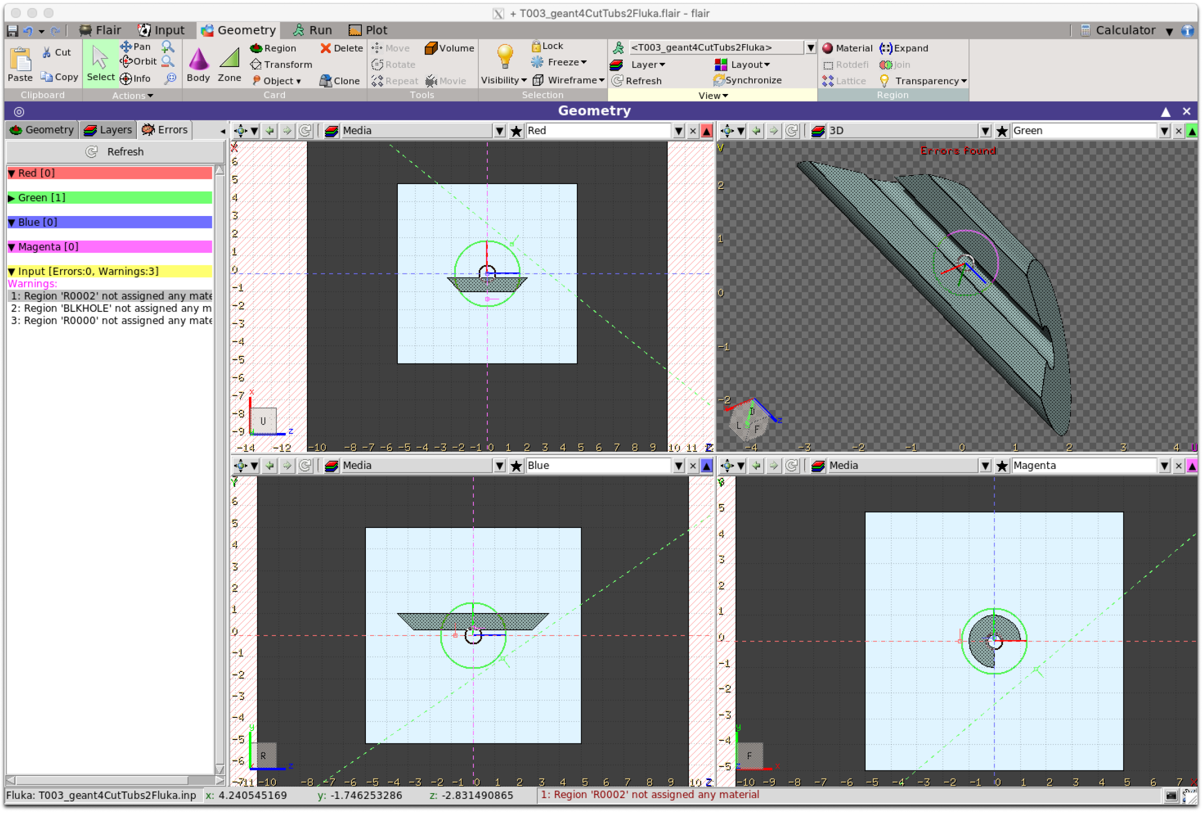
If you want to load a file into Flair then a flair file can be written based on `FileName.inp` using the following

```
extent = logical.extent(includeBoundingSolid=True)
f = pyg4ometry.fluka.Flair("FileName.inp",extent)
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.

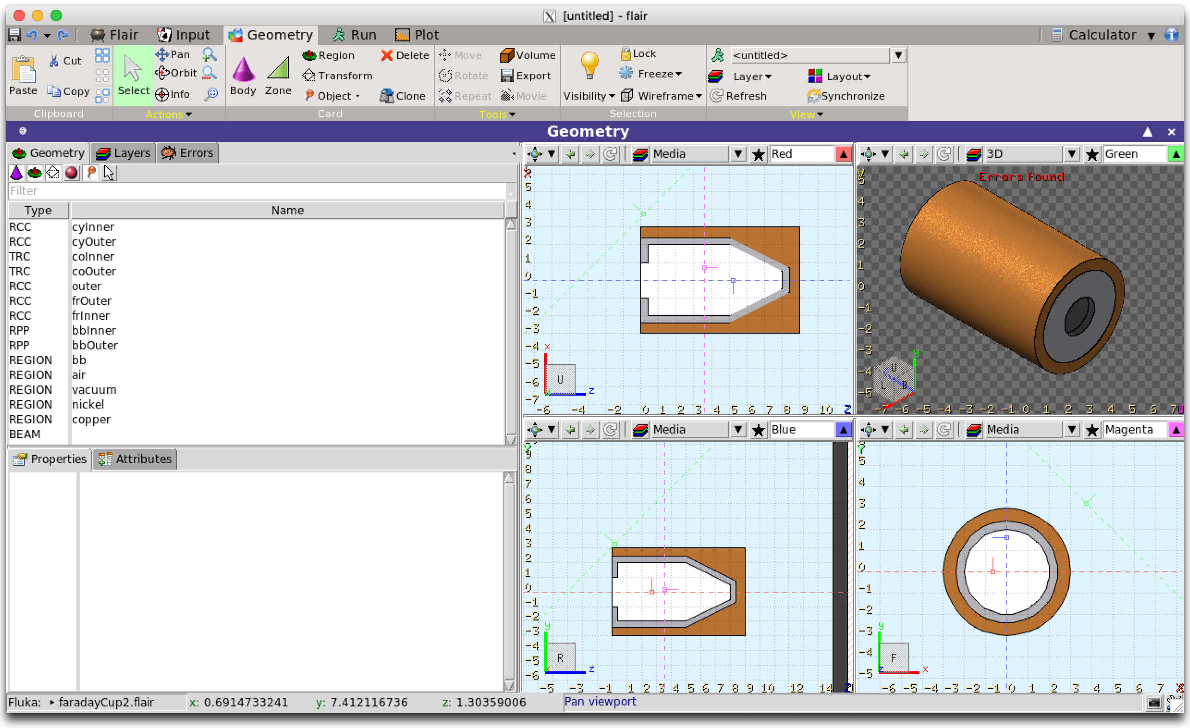
6.8 Conversion of FLUKA To GDML

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.

```
import pyg4ometry.fluka as fluka
import pyg4ometry.gdml as gdml
from pyg4ometry.convert import fluka2Geant4

# Read the FLUKA file, get the FlukaRegistry, convert the registry to a
# Geant4 Registry
reader = fluka.Reader("model.inp")
flukaregistry = reader.flukaregistry
geant4Registry = fluka2Geant4(flukaRegistry)

worldLogicalVolume = geant4Registry.getWorldVolume()
worldLogicalVolume.clipSolid()

writer = gdml.Writer()
writer.addDetector(geant4Registry)
writer.write("model.gdml")
```

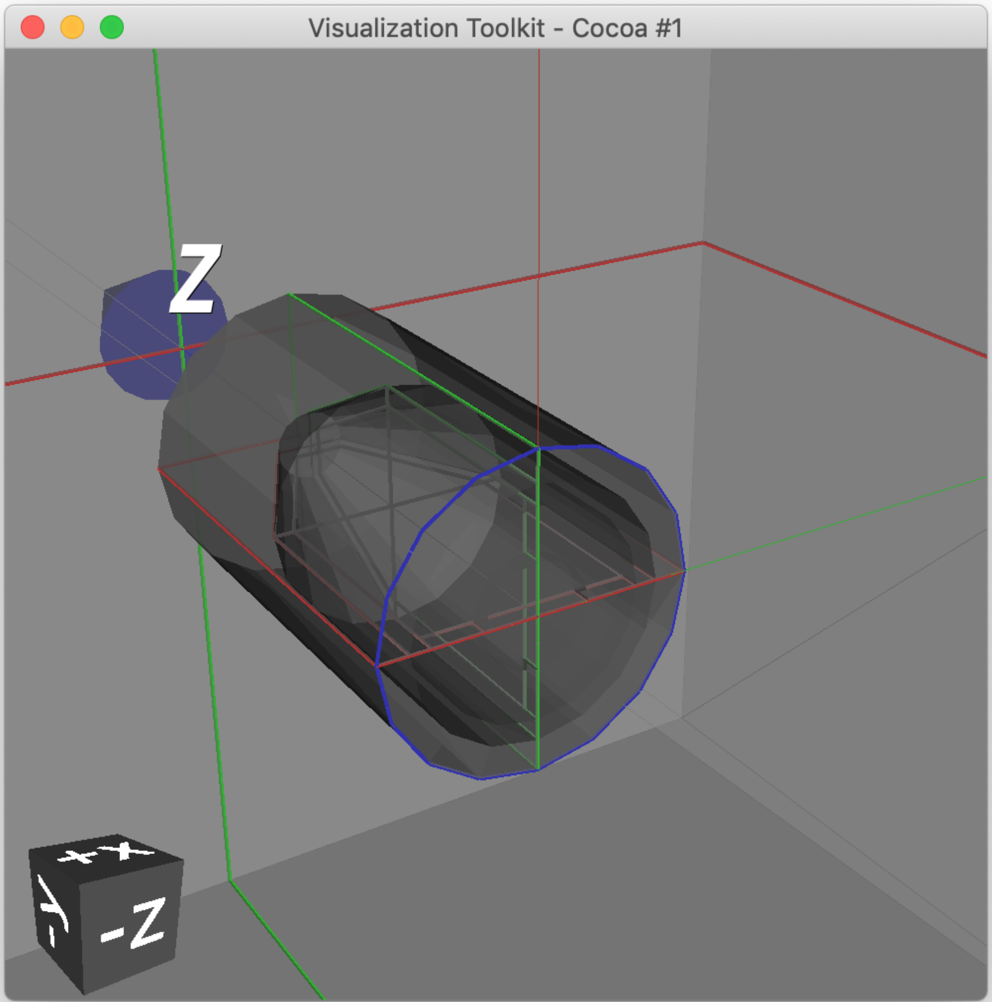
The core of this functionality is the translation of the *FlukaRegistry* instance into the equivalent *Registry* (i.e. Geant4) instance.

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/bredth 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.


Visualization Toolkit - Cocoa #1
Z
+X
-Y
-Z

CHAPTER

SEVEN

ADVANCED TUTORIALS

7.1 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/pyg4ometry/test/gdmlG4examples/ChargeExchangeMC/`

```
import pyg4ometry
r = pyg4ometry.gdml.Reader("lht.gdml")
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.

7.2 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.

7.3 Edit existing geometry

After loading some geometry it is possible to modify the memory resident geometry. This could adjusting the parameter of a given solid or PV, or replacing entirely the type of solid used for an LV. To edit geometry a LV instance is required

7.4 Complex geometry builder

Having access to geometry construction in python allows the rapid construction of geometry using functions which return an appropriate LV. Examples of this available in `pyg4ometry/pyg4ometry/test/pythonCompoundExamples`

7.5 Fluka geometry scripting

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

```
import pyg4ometry.convert as convert
import pyg4ometry.visualisation as vi
from pyg4ometry.fluka import RPP, Region, Zone, FlukaRegistry

freg = FlukaRegistry()

rpp = RPP("RPP_BODY", 0, 10, 0, 10, 0, 10, flukaregistry=freg)
z = Zone()
z.addIntersection(rpp)
region = Region("RPP_REG", material="COPPER")
region.addZone(z)
freg.addRegion(region)

greg = convert.fluka2Geant4(freg)
greg.getWorldVolume().clipSolid()

v = vi.VtkViewer()
v.addAxes(length=20)
v.addLogicalVolume(greg.getWorldVolume())
v.view()
```

7.6 Export scene to paraview/vtk

```
import pyg4ometry
```

7.7 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/pyg4ometry/test/pythonCompoundExamples/`

```
import pyg4ometry
r = pyg4ometry.gdml.Reader("./Chamber.gdml")
l = r.getRegistry().getWorldVolume()
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(l)
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.

```
import pyg4ometry
r = pyg4ometry.fluka.Reader("./Chamber.inp")
greg = pyg4ometry.convert.fluka2geant4(r.getRegistry())
l = greg.getWorldVolume()
v = pyg4ometry.visualisation.VtkViewer()
v.addLogicalVolume(l)
v.exportOBJScene("Chamber")
```

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

```
import pyg4ometry
r = pyg4ometry.gdml.Reader("./Chamber.gdml")
l = r.getRegistry().getWorldVolume()
w = pyg4ometry.visualisation.RenderWriter()
w.addLogicalVolumeRecursive(l)
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.

CHAPTER

EIGHT

MODULE CONTENTS

This documentation is automatically generated by scanning all the source code. Parts may be incomplete.

8.1 Defines and variables

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

class pyg4ometry.gdml.Defines.**ScalarBase**

Bases: object

Base class for all scalars (Constants, Quantity, Variable and Expression)

__add__(*other*)

__mul__(*other*)

__neg__()

__rsub__(*other*)

__sub__(*other*)

setExpression(*expr*)

setName(*name*)

Set name of scalar

Parameters **name** (*str*) – name of object

setRegistry(*registry*)

class pyg4ometry.gdml.Defines.**Constant**(*name*, *value*, *registry*, *addRegistry=True*)

Bases: *pyg4ometry.gdml.Defines.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

eval()
Evaluate constant

Returns numerical evaluation of Constant

Return type float

class pyg4ometry.gdml.Defines.**Quantity**(*name*, *value*, *unit*, *type*, *registry*, *addRegistry=True*)
Bases: *pyg4ometry.gdml.Defines.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

eval()
Evaluate quantity

Returns numerical evaluation of Quantity

Return type float

class pyg4ometry.gdml.Defines.**Variable**(*name*, *value*, *registry*, *addRegistry=True*)
Bases: *pyg4ometry.gdml.Defines.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

eval()
Evaluate variable

Returns numerical evaluation of Constant

Return type float

class pyg4ometry.gdml.Defines.**Expression**(*name*, *value*, *registry*, *addRegistry=False*)
Bases: *pyg4ometry.gdml.Defines.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

eval()
Evaluate expression

Returns numerical evaluation of Constant

Return type float

`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.asin(arg)`
ArcSin of a ScalarBase object, returns a Constant

Parameters **arg** (Constant, Quantity, Variable *or* Expression) – Argument of asin

`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.cos(arg)`
Cosine of a ScalarBase object, returns a Constant

Parameters **arg** (Constant, Quantity, Variable *or* Expression) – Argument of cos

`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.sin(arg)`
Sin of a ScalarBase object, returns a Constant

Parameters **arg** (Constant, Quantity, Variable *or* Expression) – Argument of sin

`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.tan(arg)`
Tangent of a ScalarBase object, returns a Constant

Parameters **arg** (Constant, Quantity, Variable *or* Expression) – Argument of tan

`class pyg4ometry.gdml.Defines.VectorBase`
Bases: `object`

`__add__(other)`

`__mul__(other)`

`__rmul__(other)`

`__sub__(other)`

`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

`setName`(*name*)
Set name of vector

Parameters **name** (*str*) – name of object

`setRegistry`(*registry*)

`class pyg4ometry.gdml.Position`(*name, x, y, z, unit='mm', registry=None, addRegistry=True*)
Bases: *pyg4ometry.gdml.Defines.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: *pyg4ometry.gdml.Defines.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: *pyg4ometry.gdml.Defines.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, addRegistry=True*)
Bases: `object`

GDML matrix define wrapper object

Parameters

- **name** (*str*) – of constant 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 constant to registry

`eval`()
Evaluate matrix

Returns numerical evaluation of matrix

Return type numpy.array

`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
- **type** (*str*) – class type of vector (position, rotation, scale)
- **addRegistry** (*bool*) – flag to add to registry

`pyg4ometry.gdml.Defines.upgradeToVector(var, reg, type='position', 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

8.2 Geant4 solids

`class pyg4ometry.geant4.solid.Plane.Plane(name, normal, dist, zlength=10000)`

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

`class pyg4ometry.geant4.solid.Wedge.Wedge(name, pRMax=1000, pSPhi=0, pDPhi=1.5, halfzlength=10000, nslice=None)`

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

`class pyg4ometry.geant4.solid.Box.Box(name, pX, pY, pZ, registry, lunit='mm', addRegistry=True)`

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

class pyg4ometry.geant4.solid.Tubs.**Tubs**(*name*, *pRMin*, *pRMax*, *pDz*, *pSPhi*, *pDPhi*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *addRegistry=True*)

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

class pyg4ometry.geant4.solid.CutTubs.**CutTubs**(*name*, *pRMin*, *pRMax*, *pDz*, *pSPhi*, *pDPhi*, *pLowNorm*, *pHighNorm*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *addRegistry=True*)

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

class pyg4ometry.geant4.solid.Cons.**Cons**(*name*, *pRmin1*, *pRmax1*, *pRmin2*, *pRmax2*, *pDz*, *pSPhi*, *pDPhi*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *addRegistry=True*)

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

`class pyg4ometry.geant4.solid.Para.Para`(*name*, *pDx*, *pDy*, *pDz*, *pAlpha*, *pTheta*, *pPhi*, *registry*, *lunit='mm'*, *aunit='rad'*, *addRegistry=True*)

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 tothe 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

`class pyg4ometry.geant4.solid.Trd.Trd`(*name*, *pDx1*, *pDx2*, *pDy1*, *pDy2*, *pDz*, *registry*, *lunit='mm'*, *addRegistry=True*)

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 postitioned 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

`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*)

`class pyg4ometry.geant4.solid.Sphere.Sphere`(*name*, *pRmin*, *pRmax*, *pSPhi*, *pDPhi*, *pSTheta*, *pDTheta*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *nstack=None*, *addRegistry=True*)

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

mesh()

working off 0 < phi < 2pi 0 < theta < pi

class pyg4ometry.geant4.solid.Orb.**Orb**(*name*, *pRMax*, *registry*, *lunit='mm'*, *nslice=None*, *nstack=None*, *addRegistry=True*)

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

class pyg4ometry.geant4.solid.Torus.**Torus**(*name*, *pRmin*, *pRmax*, *pRtor*, *pSPhi*, *pDPhi*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *nstack=None*, *addRegistry=True*)

Constructs a torus.

Parameters

- **name** (*str*) – string, name of the volume
- **pRmin** (*float*, Constant, Quantity, Variable, Expression) – innder 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

class pyg4ometry.geant4.solid.Polycone.**Polycone**(*name*, *pSPhi*, *pDPhi*, *pZpl*, *pRMin*, *pRMax*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *addRegistry=True*)

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.

class pyg4ometry.geant4.solid.GenericPolycone.**GenericPolycone**(*name*, *pSPhi*, *pDPhi*, *pR*, *pZ*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *addRegistry=True*)

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

class pyg4ometry.geant4.solid.Polyhedra.**Polyhedra**(*name*, *pSPhi*, *pDPhi*, *numSide*, *numZPlanes*, *zPlane*, *rInner*, *rOuter*, *registry*, *lunit='mm'*, *aunit='rad'*, *addRegistry=True*)

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

class pyg4ometry.geant4.solid.EllipticalTube.**EllipticalTube**(*name*, *pDx*, *pDy*, *pDz*, *registry*, *lunit='mm'*, *nstack=None*, *nslice=None*, *addRegistry=True*)

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

mesh()

new meshing based of Tubs meshing

class pyg4ometry.geant4.solid.Ellipsoid.**Ellipsoid**(*name*, *pxSemiAxis*, *pySemiAxis*, *pzSemiAxis*, *pzBottomCut*, *pzTopCut*, *registry*, *lunit='mm'*, *nslice=None*, *nstack=None*, *addRegistry=True*)

Constructs an ellipsoid optinoally 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

`class pyg4ometry.geant4.solid.Paraboloid.Paraboloid(name, pDz, pR1, pR2, registry, lunit='mm', nslice=16, nstack=8, addRegistry=True)`

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

`class pyg4ometry.geant4.solid.Hype.Hype(name, innerRadius, outerRadius, innerStereo, outerStereo, lenZ, registry, lunit='mm', aunit='rad', nslice=None, nstack=None, addRegistry=True)`

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

`class pyg4ometry.geant4.solid.Tet.Tet(name, anchor, p2, p3, p4, registry, lunit='mm', degeneracyFlag=False, addRegistry=True)`

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

class pyg4ometry.geant4.solid.ExtrudedSolid.**ExtrudedSolid**(*name*, *pPolygon*, *pZslices*, *registry*, *lunit='mm'*, *addRegistry=True*)

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,v3]] - vertices of polygon in clockwise order zSlices = [[z1,[offsx1, offsy1],scale1],[z2,[offsx2, offsy2],scale2]]

class pyg4ometry.geant4.solid.TwistedBox.**TwistedBox**(*name*, *twistedangle*, *pDx*, *pDy*, *pDz*, *registry*, *lunit='mm'*, *aunit='rad'*, *nstack=None*, *refine=0*, *addRegistry=True*)

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

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*)

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

class pyg4ometry.geant4.solid.TwistedTrd.**TwistedTrd**(*name*, *twistedangle*, *pDx1*, *pDx2*, *pDy1*, *pDy2*, *pDz*, *registry*, *lunit='mm'*, *aunit='rad'*, *nstack=None*, *refine=0*, *addRegistry=True*)

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

class pyg4ometry.geant4.solid.TwistedTubs.**TwistedTubs**(*name*, *endinnerrad*, *endouterrad*, *zlen*, *phi*, *twistedangle*, *registry*, *lunit='mm'*, *aunit='rad'*, *nslice=None*, *nstack=None*, *addRegistry=True*)

Creates a twisted tube segement. Note that only 1 constructor is supprted.

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 segement
- **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

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*)

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

class pyg4ometry.geant4.solid.TessellatedSolid.**TessellatedSolid**(*name*, *meshTess*, *registry*, *meshtype=1*, *addRegistry=True*)

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

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

class pyg4ometry.geant4.solid.Union.**Union**(*name*, *obj1*, *obj2*, *tra2*, *registry*, *addRegistry=True*)

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

class pyg4ometry.geant4.solid.Intersection.**Intersection**(*name*, *obj1*, *obj2*, *tra2*, *registry*, *addRegistry=True*)

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 tra2
- **tra2** (*list*) – [rot,tra] = [[a,b,g],[dx,dy,dz]]
- **registry** (Registry) – for storing solid

class pyg4ometry.geant4.solid.Subtraction.**Subtraction**(*name*, *obj1*, *obj2*, *tra2*, *registry*, *addRegistry=True*)

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

8.3 Geant4 module

class pyg4ometry.geant4.**Registry**

Bases: object

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.

addDefine(*define*, *namePolicy='none'*)

Parameters **define** (Constant, Quantity, Variable, Matrix) – Defintion object for starage

addLogicalVolume(*volume*, *namePolicy='none'*)

Parameters **volume** (LogicalVolume) – LogicalVolume object for starage

addMaterial(*material*, *namePolicy='reuse'*)

Parameters **material** (Material) – Material object for starage

addPhysicalVolume(*volume*, *namePolicy='increment'*)

Parameters **volume** (PhysicalVolume) – PhysicalVolume object for starage

addSolid(*solid*, *namePolicy='none'*)

Parameters **solid** (*One of the geant4 solids*) – Solid object for starage

class pyg4ometry.geant4.**Material**(***kwargs*)

Bases: pyg4ometry.geant4._Material.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

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

`add_property`(*name*, *value*)

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

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

`property state_variables`

`class pyg4ometry.geant4.LogicalVolume.LogicalVolume`(*solid*, *material*, *name*, *registry=None*, *addRegistry=True*, ***kwargs*)

Bases: `object`

LogicalVolume : G4LogicalVolume :param solid: :param material: :param name: :param registry: :param addRegistry:

`add`(*physicalVolume*)

`addAuxiliaryInfo`(*auxiliary*)

`addBDSIMObject`(*bdsimobject*)

`assemblyVolume`()

`checkOverlaps`(*recursive=False*, *coplanar=True*, *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 - Whether to print out a summary of N overlaps detected
- `nOverlapsDetected` – [int] - internal use only for recursion - ignore

`clipSolid`(*lengthSafety=1e-06*)

`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.

`extent`(*includeBoundingSolid=False*)

`findLogicalByName`(*name*)

`makeLogicalPhysicalNameSets`()

`makeMaterialNameSet`()

`makeSolidTessellated`()

`makeWorldVolume`(*worldMaterial='G4_Galactic'*)

`setSolid`(*solid*)

class `pyg4ometry.geant4.PhysicalVolume.`**`PhysicalVolume`**(*rotation*, *position*, *logicalVolume*, *name*, *motherVolume*, *registry=None*, *addRegistry=True*, *scale=None*)

Bases: `object`

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
- **addRegistry** – bool - whether to add to the registry or not

extent(*includeBoundingSolid=True*)

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

Bases: `object`

kPhi = 5

kRho = 4

kXAxis = 1

kYAxis = 2

kZAxis = 3

createReplicaMeshes()

extent(*includeBoundingSolid=True*)

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'*)
Bases: object

class **ConeDimensions**(*pRMin1*, *pRMax1*, *pRMin2*, *pRMax2*, *pDz*, *pSPhi*, *pDPhi*, *lunit='mm'*, *aunit='rad'*)
Bases: object

class **EllipsoidDimensions**(*pxSemiAxis*, *pySemiAxis*, *pzSemiAxis*, *pzBottomCut*, *pzTopCut*, *lunit='mm'*)
Bases: object

class **HypeDimensions**(*innerRadius*, *outerRadius*, *innerStereo*, *outerStereo*, *lenZ*, *lunit='mm'*, *aunit='rad'*)
Bases: object

class **OrbDimensions**(*pRMax*, *lunit='mm'*)
Bases: object

class **ParaDimensions**(*pX*, *pY*, *pZ*, *pAlpha*, *pTheta*, *pPhi*, *lunit='mm'*, *aunit='rad'*)
Bases: object

class **PolyconeDimensions**(*pSPhi*, *pDPhi*, *pZpl*, *pRMin*, *pRMax*, *lunit='mm'*, *aunit='rad'*)
Bases: object

class **PolyhedraDimensions**(*pSPhi*, *pDPhi*, *numSide*, *pZpl*, *pRMin*, *pRMax*, *lunit='mm'*, *aunit='rad'*)
Bases: object

class **SphereDimensions**(*pRMin*, *pRMax*, *pSPhi*, *pDPhi*, *pSTheta*, *pDTheta*, *lunit='mm'*, *aunit='rad'*)
Bases: object

class **TorusDimensions**(*pRMin*, *pRMax*, *pRTor*, *pSPhi*, *pDPhi*, *lunit='mm'*, *aunit='rad'*)
Bases: object

class **TrapDimensions**(*pDz*, *pTheta*, *pDPhi*, *pDy1*, *pDx1*, *pDx2*, *pAlp1*, *pDy2*, *pDx3*, *pDx4*, *pAlp2*, *lunit='mm'*, *aunit='rad'*)
Bases: object

class **TrdDimensions**(*pX1*, *pX2*, *pY1*, *pY2*, *pZ*, *lunit='mm'*)
Bases: object

class **TubeDimensions**(*pRMin*, *pRMax*, *pDz*, *pSPhi*, *pDPhi*, *lunit='mm'*, *aunit='rad'*)
Bases: object

createParameterisedMeshes()

extent(*includeBoundingSolid=True*)

8.4 VTK module

class pyg4ometry.visualisation.VtkViewer.**MouseInteractorNamePhysicalVolume**(*renderer*, *vtkviewer*)
Bases: vtkInteractionStylePython.vtkInteractorStyleTrackballCamera

rightButtonPressEvent(*obj*, *event*)

pyg4ometry.visualisation.VtkViewer.**PubViewer**
alias of *pyg4ometry.visualisation.VtkViewer.VtkViewerColoured*

class pyg4ometry.visualisation.VtkViewer.**VtkViewer**(*size=(1024, 1024)*, *interpolation='none'*, ***kwargs*)
Bases: object

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()

addBooleanSolidRecursive(*solid*, *mtra=matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])*, *tra=array([0, 0, 0])*, *first=True*)

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) poisition 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.

addLogicalVolume(*logical*, *mtra=matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])*, *tra=array([0, 0, 0])*)

addLogicalVolumeBounding(*logical*)

addLogicalVolumeRecursive(*logical*, *mtra=matrix([[1, 0, 0], [0, 1, 0], [0, 0, 1]])*, *tra=array([0, 0, 0])*)

addMaterialVisOption(*materialName*, *visOptionInstance*)
Append a visualisation option instance to the dictionary of materials.

Parameters

- **materialName** – str - material name to match
- **visOptionInstance** – `VisualisationOptions` instance

addMesh(*pv_name*, *solid_name*, *mesh*, *mtra*, *tra*, *localmeshes*, *filters*, *mappers*, *mapperMap*, *actors*, *actorMap*, *visOptions=None*, *overlap=False*, *cutters=True*, *clippers=False*)

addMeshSimple(*csgMesh, visOptions=<VisOpt: rep=surface, rgba=[0.5, 0.5, 0.5, 0.5], vis=True, linewidth=1>, clip=False*)

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)
```

`exportOBJScene`(*fileName='scene'*)

`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

`exportVRMLScene`(*fileName='scene'*)

`getMaterialVisOptions`(*pv*)

`getOverlapVisOptions`(*overlaptype*)

`printViewParameters`()

`setCameraFocusPosition`(*focalPoint=[0, 0, 0]*, *position=[100, 100, 100]*)

`setCutterNormal`(*dimension*, *normal*)

Parameters

- **dimension** – str - 'x', 'y', or 'z'
- **normal** – list([x,y,z]) - should be unit vector

`setCutterOrigin`(*dimension*, *origin*)

Parameters

- **dimension** – str - 'x', 'y', or 'z'
- **origin** – list([x,y,z])

`setMaterialVisOptions`(*materialDict*)
Replace the (by default None) dictionary for materials to colours :param materialDict: {"materialName": VisualisationOptions}

See also `VisualisationOptions`.

`setOpacity`(*v*, *iActor=- 1*)

`setOpacityOverlap`(*v*, *iActor=- 1*)

`setRandomColours`(*seed=0*)

`setSurface`(*iActor=- 1*)

`setSurfaceOverlap`(*iActor=- 1*)

`setWireframe`(*iActor=- 1*)

`setWireframeOverlap`(*iActor=- 1*)

`start`()

`view`(*interactive=True*, *resetCamera=True*)

viewSection(*dir='x'*)

class pyg4ometry.visualisation.VtkViewer.**VtkViewerColoured**(**args*, *defaultColour=None*, *materialVisOptions=None*, ***kwargs*)

Bases: *pyg4ometry.visualisation.VtkViewer.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.

class pyg4ometry.visualisation.VtkViewer.**VtkViewerColouredMaterial**(**args*, ***kwargs*)

Bases: *pyg4ometry.visualisation.VtkViewer.VtkViewerColoured*

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.

pyg4ometry.visualisation.VtkViewer.**axesFromExtents**(*extent*)

8.5 Freecad module

8.6 STL module

class pyg4ometry.stl.Reader.**Reader**(*filename*, *solidname='stl_tessellated'*, *scale=1*, *registry=None*)

Bases: object

getRegistry()

getSolid()

load()

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.

8.7 GDML module

class pyg4ometry.gdml.Reader.**Reader**(*fileName*, *registryOn=True*)

Bases: object

extractStructureNodeData(*node*)

getRegistry()

load()

parseBox(*node*)

parseCone(*node*)

parseCutTube(*node*)

parseDefines(*xmldoc*)

parseEllipsoid(*node*)

parseEllipticalCone(*node*)

parseEllipticalTube(*node*)

parseExtrudedSolid(*node*)

parseGenericPolycone(*node*)

parseGenericPolyhedra(*node*)

parseGenericTrap(*node*)

parseHype(*node*)

parseIntersection(*node*)

parseMaterials(*xmldoc*)

parseMultiUnion(*node*)

parseOpticalSurface(*node*)

parseOrb(*node*)

parsePara(*node*)

parseParaboloid(*node*)

parsePhysicalVolumeChildren(*node*, *vol*)

parsePolycone(*node*)

parsePolyhedra(*node*)

parseScaledSolid(*node*)

parseSolidLoop(*node*)

parseSolids(*xmldoc*)

parseSphere(*node*)

parseStructure(*xmldoc*)

parseSubtraction(*node*)

parseTessellatedSolid(*node*)

parseTet(*node*)

parseTorus(*node*)

parseTrap(*node*)

parseTrd(*node*)

parseTube(*node*)

parseTwistedBox(*node*)

parseTwistedTrap(*node*)

parseTwistedTrd(*node*)

parseTwistedTubs(*node*)

parseUnion(*node*)

parseUserInfo(*xmldoc*)

parseVector(*node*, *type='position'*, *addRegistry=True*)

class pyg4ometry.gdml.Writer.Writer(*prepend=''*)
Bases: object

addDetector(*registry*)

checkDefineName(*defineName*)

checkLogicalVolumeName(*logicalVolumeName*)

checkMaterialName(*materialName*)

checkPhysicalVolumeName(*physicalVolumeName*)

checkSolidName(*solidName*)

createPosition(*name*, *x*, *y*, *z*)

createQuadrangularFacet(*vertex1*, *vertex2*, *vertex3*, *vertex4*)

createSection(*zOrder*, *zPosition*, *xOffset*, *yOffset*, *scalingFactor*)

createTriangularFacet(*vertex1*, *vertex2*, *vertex3*)

createTwoDimVertex(*x*, *y*)

createrzPoint(*r*, *z*)

createzPlane(*rInner*, *rOuter*, *zplane*)

getValueOrExpr(*var*)

getValueOrExprFromInstance(*instance*, *variable*, *index=None*)

write(*filename*)

writeAssemblyVolume(*lv*)

writeAuxiliary(*aux*, *parent=None*)

writeBorderSurface(*instance*)

writeBox(*instance*)

writeCons(*instance*)

writeCutTubs(*instance*)

writeDefaultLattice(*filename='lattice.gmad'*)

writeDefine(*define*)

writeDivisionVolume(*instance*)

writeEllipsoid(*instance*)

writeEllipticalCone(*instance*)

writeEllipticalTube(*instance*)

`writeExtrudedSolid`(*instance*)

`writeGMADTesterNoBeamline`(*gmad*, *gdml*)

`writeGenericPolycone`(*instance*)

`writeGenericPolyhedra`(*instance*)

`writeGenericTrap`(*instance*)

`writeGmadTester`(*filenameGmad*, *filenameGDML*, *writeDefaultLattice=False*, *preprocessGDML=True*)

`writeHype`(*instance*)

`writeIntersection`(*instance*)

`writeLogicalVolume`(*lv*)

`writeMaterial`(*material*)

`writeMultiUnion`(*instance*)

`writeOpticalSurface`(*instance*)

`writeOrb`(*instance*)

`writePara`(*instance*)

`writeParaboloid`(*instance*)

`writeParametrisedVolume`(*instance*)

`writePhysicalVolume`(*pv*)

`writePolycone`(*instance*)

`writePolyhedra`(*instance*)

`writeReplicaVolume`(*instance*)

`writeScaled`(*instance*)

`writeSkinSurface`(*instance*)

`writeSolid`(*solid*)
Dispatch to correct member function based on type string in SolidBase.

`writeSphere`(*instance*)

`writeSubtraction`(*instance*)

`writeTessellatedSolid`(*instance*)

`writeTet`(*instance*)

`writeTorus`(*instance*)

`writeTrap`(*instance*)

`writeTrd`(*instance*)

`writeTubs`(*instance*)

`writeTwistedBox`(*instance*)

`writeTwistedTrap`(*instance*)

`writeTwistedTrd`(*instance*)

`writeTwistedTubs`(*instance*)

`writeUnion`(*instance*)

writeVectorVariable(*node*, *vector_var*, *allow_ref=True*, *suppress_trivial=True*)
Writes an XML child node for a vector variable - position, roration, 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.

8.8 Fluka bodies

Set of classes for FLUKA bodies.

class pyg4ometry.fluka.body.**ARB**(*name*, *vertices*, *facenumbers*, *transform=None*, *flukaregistry=None*, *comment=''*)
Bases: `pyg4ometry.fluka.body.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 polydedron.
- **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.

_extent()

_faceNumbersToZeroCountingIndices()

_getVerticesAndPolygons()

_toTesselatedSolid(*verticesAndPolygons*, *greg*, *addRegistry*)

_toVerticesAndPolygons(*reverse*)

_withLengthSafety(*safety*, *reg*)

centre(*aabb=None*)

flukaFreeString()

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

rotation()

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

_withLengthSafety(*safety*, *reg*)

centre(*aabb=None*)

flukaFreeString()

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

lengths()

rotation()

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

Bases: pyg4ometry.fluka.body.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.

_linearEccentricity()

_semiminor()

_withLengthSafety(*safety*, *reg*)

centre(*aabb=None*)

flukaFreeString()

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

rotation()

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

Bases: pyg4ometry.fluka.body._HalfSpaceMixin

Infinite half-space delimited by the x-y plane (pependicular 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.

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

flukaFreeString()

rotation()

toPlane()

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

Bases: pyg4ometry.fluka.body.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

static **_quadricMatrixToCoefficients**(*matrix*)

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

centre(*aabb=None*)

coefficientsMatrix()

flukaFreeString()

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

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

rotation()

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

Bases: pyg4ometry.fluka.body._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.**RCC**(*name*, *face*, *direction*, *radius*, *transform=None*, *flukaregistry=None*, *comment=''*)

Bases: pyg4ometry.fluka.body.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.

_withLengthSafety(*safety*, *reg*)

centre(*aabb=None*)

flukaFreeString()

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

`rotation`()

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

Bases: `pyg4ometry.fluka.body.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.

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

`centre`(*aabb=None*)

`flukaFreeString`()

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

`rotation`()

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

Bases: `pyg4ometry.fluka.body.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)

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

`centre`(*aabb=None*)

`flukaFreeString`()

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

`lengths`()

`rotation`()

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

Bases: `pyg4ometry.fluka.body.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.

_withLengthSafety(*safety*, *reg*)

centre(*aabb=None*)

flukaFreeString()

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

rotation()

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

Bases: pyg4ometry.fluka.body.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.

_withLengthSafety(*safety*, *reg*)

centre(*aabb=None*)

flukaFreeString()

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

rotation()

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

Bases: pyg4ometry.fluka.body._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.**XCC**(*name*, *y*, *z*, *radius*, *transform=None*, *flukaregistry=None*, *comment=''*)

Bases: pyg4ometry.fluka.body._InfiniteCylinderMixin, pyg4ometry.fluka.body._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

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

centre(*aabb=None*)

direction()

flukaFreeString()

point()

rotation()

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

Bases: `pyg4ometry.fluka.body.BodyMixin`, `pyg4ometry.fluka.body._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

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

centre(*aabb=None*)

flukaFreeString()

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

rotation()

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

Bases: `pyg4ometry.fluka.body._HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (pependicular 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.

_withLengthSafety(*safety*, *reg*)

flukaFreeString()

toPlane()

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

Bases: `pyg4ometry.fluka.body._HalfSpaceMixin`

Infinite half-space delimited by the x-y plane (pependicular 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.

_withLengthSafety(*safety*, *reg*)

flukaFreeString()

toPlane()

class pyg4ometry.fluka.body.**YCC**(*name*, *z*, *x*, *radius*, *transform=None*, *flukaregistry=None*, *comment=''*)
Bases: pyg4ometry.fluka.body._InfiniteCylinderMixin, pyg4ometry.fluka.body._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

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

centre(*aabb=None*)

direction()

flukaFreeString()

point()

rotation()

class pyg4ometry.fluka.body.**YEC**(*name*, *z*, *x*, *zsemi*, *xsemi*, *transform=None*, *flukaregistry=None*, *comment=''*)
Bases: pyg4ometry.fluka.body.BodyMixin, pyg4ometry.fluka.body._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

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

centre(*aabb=None*)

flukaFreeString()

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

rotation()

class pyg4ometry.fluka.body.**YZP**(*name*, *x*, *transform=None*, *flukaregistry=None*, *comment=''*)
Bases: pyg4ometry.fluka.body._HalfSpaceMixin

Infinite half-space delimited by the x-y plane (pependicular 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.

_withLengthSafety(*safety*, *reg*)

flukaFreeString()

toPlane()

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

Bases: `pyg4ometry.fluka.body._InfiniteCylinderMixin`, `pyg4ometry.fluka.body._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

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

centre(*aabb=None*)

direction()

flukaFreeString()

point()

rotation()

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

Bases: `pyg4ometry.fluka.body.BodyMixin`, `pyg4ometry.fluka.body._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

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

centre(*aabb=None*)

flukaFreeString()

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

rotation()

8.9 Fluka module

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.

Class to read a FLUKA file.

CHAPTER

NINE

DEVELOPER

9.1 Unit tests

```
cd pyg4ometry/pyg4ometry/test
python2.7 runTests.py
```

9.2 Coverage

```
cd pyg4ometry/coverage
./runCoverage.sh
```

9.3 Profiling

```
python2.7 -m cProfile -s tottime myscript.py > myscript.log
```

```
pycallgraph-2.7 graphviz -- ../pyg4ometry/test/python/T008_Sphere.py
```

9.4 Updating A Version

Update the version number in the following files:

- *setup.py*
- *setup.cfg*
- *pyg4ometry/docs/source/conf.py*

Make manual and commit to *pyg4ometry/docs/pyg4ometry.pdf*.

9.5 Updating Copyright

Update the year in the following files:

- *LICENCE.txt*
- *README.md*
- *docs/source/conf.py*
- *docs/source/licence.rst*

CHAPTER TEN

VERSION HISTORY

10.1 v 1.0.0 - 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*.

10.2 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

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

pyg4ometry.fluka.body, 60
pyg4ometry.fluka.FlukaRegistry, 67
pyg4ometry.fluka.Reader, 67
pyg4ometry.gdml.Defines, 38
pyg4ometry.gdml.Reader, 57
pyg4ometry.gdml.Writer, 58
pyg4ometry.geant4.LogicalVolume, 51
pyg4ometry.geant4.ParameterisedVolume, 52
pyg4ometry.geant4.PhysicalVolume, 51
pyg4ometry.geant4.ReplicaVolume, 52
pyg4ometry.geant4.solid.Box, 39
pyg4ometry.geant4.solid.Cons, 40
pyg4ometry.geant4.solid.CutTubs, 40
pyg4ometry.geant4.solid.Ellipsoid, 44
pyg4ometry.geant4.solid.EllipticalTube, 44
pyg4ometry.geant4.solid.ExtrudedSolid, 46
pyg4ometry.geant4.solid.GenericPolycone, 43
pyg4ometry.geant4.solid.GenericTrap, 48
pyg4ometry.geant4.solid.Hype, 45
pyg4ometry.geant4.solid.Intersection, 49
pyg4ometry.geant4.solid.Orb, 42
pyg4ometry.geant4.solid.Para, 41
pyg4ometry.geant4.solid.Paraboloid, 45
pyg4ometry.geant4.solid.Plane, 39
pyg4ometry.geant4.solid.Polycone, 43
pyg4ometry.geant4.solid.Polyhedra, 43
pyg4ometry.geant4.solid.Sphere, 41
pyg4ometry.geant4.solid.Subtraction, 49
pyg4ometry.geant4.solid.TessellatedSolid, 49
pyg4ometry.geant4.solid.Tet, 45
pyg4ometry.geant4.solid.Torus, 42
pyg4ometry.geant4.solid.Trap, 41
pyg4ometry.geant4.solid.Trd, 41
pyg4ometry.geant4.solid.Tubs, 40
pyg4ometry.geant4.solid.TwistedBox, 46
pyg4ometry.geant4.solid.TwistedTrap, 46
pyg4ometry.geant4.solid.TwistedTrd, 47
pyg4ometry.geant4.solid.TwistedTubs, 48
pyg4ometry.geant4.solid.Union, 49
pyg4ometry.geant4.solid.Wedge, 39
pyg4ometry.stl.Reader, 56
pyg4ometry.visualisation.VtkViewer, 53

INDEX

Symbols

__add__() (*pyg4ometry.gdml.Defines.ScalarBase method*), 35
__add__() (*pyg4ometry.gdml.Defines.VectorBase method*), 37
__mul__() (*pyg4ometry.gdml.Defines.ScalarBase method*), 35
__mul__() (*pyg4ometry.gdml.Defines.VectorBase method*), 37
__neg__() (*pyg4ometry.gdml.Defines.ScalarBase method*), 35
__rmul__() (*pyg4ometry.gdml.Defines.VectorBase method*), 37
__rsub__() (*pyg4ometry.gdml.Defines.ScalarBase method*), 35
__sub__() (*pyg4ometry.gdml.Defines.ScalarBase method*), 35
__sub__() (*pyg4ometry.gdml.Defines.VectorBase method*), 37
_extent() (*pyg4ometry.fluka.body.ARB method*), 60
_faceNumbersToZeroCountingIndices() (*pyg4ometry.fluka.body.ARB method*), 60
_getVerticesAndPolygons() (*pyg4ometry.fluka.body.ARB method*), 60
_linearEccentricity() (*pyg4ometry.fluka.body.ELL method*), 61
_quadricMatrixToCoefficients() (*pyg4ometry.fluka.body.QUA static method*), 62
_semiminor() (*pyg4ometry.fluka.body.ELL method*), 61
_toTesselatedSolid() (*pyg4ometry.fluka.body.ARB method*), 60
_toVerticesAndPolygons() (*pyg4ometry.fluka.body.ARB method*), 60
_withLengthSafety() (*pyg4ometry.fluka.body.ARB method*), 60
_withLengthSafety() (*pyg4ometry.fluka.body.BOX method*), 60
_withLengthSafety() (*pyg4ometry.fluka.body.ELL method*), 61
_withLengthSafety() (*pyg4ometry.fluka.body.PLA method*), 61
_withLengthSafety() (*pyg4ometry.fluka.body.QUA method*), 62
_withLengthSafety() (*pyg4ometry.fluka.body.RCC method*), 62
_withLengthSafety() (*pyg4ometry.fluka.body.REC method*), 63
_withLengthSafety() (*pyg4ometry.fluka.body.RPP method*), 63
_withLengthSafety() (*pyg4ometry.fluka.body.SPH method*), 64
_withLengthSafety() (*pyg4ometry.fluka.body.TRC method*), 64
_withLengthSafety() (*pyg4ometry.fluka.body.XCC method*), 64
_withLengthSafety() (*pyg4ometry.fluka.body.XEC method*), 65
_withLengthSafety() (*pyg4ometry.fluka.body.XYP method*), 65
_withLengthSafety() (*pyg4ometry.fluka.body.XZP method*), 65
_withLengthSafety() (*pyg4ometry.fluka.body.YCC method*), 66
_withLengthSafety() (*pyg4ometry.fluka.body.YEC method*), 66
_withLengthSafety() (*pyg4ometry.fluka.body.YZP method*), 66
_withLengthSafety() (*pyg4ometry.fluka.body.ZCC method*), 67
_withLengthSafety() (*pyg4ometry.fluka.body.ZEC method*), 67

A

acos() (*in module pyg4ometry.gdml.Defines*), 36
add() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
add_element_massfraction() (*pyg4ometry.geant4.Material method*), 50
add_element_natoms() (*pyg4ometry.geant4.Material method*), 50
add_material() (*pyg4ometry.geant4.Material method*), 50
add_property() (*pyg4ometry.geant4.Material method*), 51

addAuxiliaryInfo() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
addAxes() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addAxesWidget() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addBDSIMObject() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
addBooleanSolidRecursive() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addCutterPlane() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addDefine() (*pyg4ometry.geant4.Registry method*), 50
addDetector() (*pyg4ometry.gdml.Writer.Writer method*), 58
addLogicalVolume() (*pyg4ometry.geant4.Registry method*), 50
addLogicalVolume() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addLogicalVolumeBounding() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addLogicalVolumeRecursive() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addMaterial() (*pyg4ometry.geant4.Registry method*), 50
addMaterialVisOption() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addMesh() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addMeshSimple() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 54
addPhysicalVolume() (*pyg4ometry.geant4.Registry method*), 50
addSolid() (*pyg4ometry.geant4.Registry method*), 50
ARB (*class in pyg4ometry.fluka.body*), 60
asin() (*in module pyg4ometry.gdml.Defines*), 37
assemblyVolume() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
atan() (*in module pyg4ometry.gdml.Defines*), 37
axesFromExtents() (*in module pyg4ometry.visualisation.VtkViewer*), 56

B

BOX (*class in pyg4ometry.fluka.body*), 60
Box (*class in pyg4ometry.geant4.solid.Box*), 39

C

centre() (*pyg4ometry.fluka.body.ARB method*), 60
centre() (*pyg4ometry.fluka.body.BOX method*), 61
centre() (*pyg4ometry.fluka.body.ELL method*), 61
centre() (*pyg4ometry.fluka.body.QUA method*), 62
centre() (*pyg4ometry.fluka.body.RCC method*), 62
centre() (*pyg4ometry.fluka.body.REC method*), 63
centre() (*pyg4ometry.fluka.body.RPP method*), 63
centre() (*pyg4ometry.fluka.body.SPH method*), 64
centre() (*pyg4ometry.fluka.body.TRC method*), 64
centre() (*pyg4ometry.fluka.body.XCC method*), 65
centre() (*pyg4ometry.fluka.body.XEC method*), 65
centre() (*pyg4ometry.fluka.body.YCC method*), 66
centre() (*pyg4ometry.fluka.body.YEC method*), 66
centre() (*pyg4ometry.fluka.body.ZCC method*), 67
centre() (*pyg4ometry.fluka.body.ZEC method*), 67
checkDefineName() (*pyg4ometry.gdml.Writer.Writer method*), 58
checkLogicalVolumeName() (*pyg4ometry.gdml.Writer.Writer method*), 58
checkMaterialName() (*pyg4ometry.gdml.Writer.Writer method*), 58
checkOverlaps() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
checkPhysicalVolumeName() (*pyg4ometry.gdml.Writer.Writer method*), 58
checkSolidName() (*pyg4ometry.gdml.Writer.Writer method*), 58
clipSolid() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
coefficientsMatrix() (*pyg4ometry.fluka.body.QUA method*), 62
Cons (*class in pyg4ometry.geant4.solid.Cons*), 40
Constant (*class in pyg4ometry.gdml.Defines*), 35
cos() (*in module pyg4ometry.gdml.Defines*), 37
createParameterisedMeshes() (*pyg4ometry.geant4.ParameterisedVolume.ParameterisedVolum method*), 53
createPosition() (*pyg4ometry.gdml.Writer.Writer method*), 58
createQuadrangularFacet() (*pyg4ometry.gdml.Writer.Writer method*), 58
createReplicaMeshes() (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume method*), 52
createrzPoint() (*pyg4ometry.gdml.Writer.Writer method*), 58
createSection() (*pyg4ometry.gdml.Writer.Writer method*), 58
createTessellatedSolid() (*in module pyg4ometry.geant4.solid.TessellatedSolid*), 49
createTriangularFacet() (*pyg4ometry.gdml.Writer.Writer method*), 58
createTwoDimVertex() (*pyg4ometry.gdml.Writer.Writer method*), 58
createzPlane() (*pyg4ometry.gdml.Writer.Writer method*), 58
cullDaughtersOutsideSolid() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
CutTubs (*class in pyg4ometry.geant4.solid.CutTubs*),

40

D

direction() (pyg4ometry.fluka.body.XCC method), 65
direction() (pyg4ometry.fluka.body.YCC method), 66
direction() (pyg4ometry.fluka.body.ZCC method), 67

E

ELL (class in pyg4ometry.fluka.body), 61
Ellipsoid (class in pyg4ometry.geant4.solid.Ellipsoid), 44
EllipticalTube (class in pyg4ometry.geant4.solid.EllipticalTube), 44
eval() (pyg4ometry.gdml.Defines.Constant method), 35
eval() (pyg4ometry.gdml.Defines.Expression method), 36
eval() (pyg4ometry.gdml.Defines.Quantity method), 36
eval() (pyg4ometry.gdml.Defines.Variable method), 36
eval() (pyg4ometry.gdml.Defines.VectorBase method), 37
eval() (pyg4ometry.gdml.Matrix method), 38
exp() (in module pyg4ometry.gdml.Defines), 37
exportCutterSection() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 54
exportOBJScene() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 55
exportScreenShot() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 55
exportVRMLScene() (pyg4ometry.visualisation.VtkViewer.VtkViewer method), 55
Expression (class in pyg4ometry.gdml.Defines), 36
extent() (pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 51
extent() (pyg4ometry.geant4.ParameterisedVolume.ParameterisedVolume method), 53
extent() (pyg4ometry.geant4.PhysicalVolume.PhysicalVolume method), 52
extent() (pyg4ometry.geant4.ReplicaVolume.ReplicaVolume method), 52
extractStructureNodeData() (pyg4ometry.gdml.Reader.Reader method), 57
ExtrudedSolid (class in pyg4ometry.geant4.solid.ExtrudedSolid), 46

F

findLogicalByName() (pyg4ometry.geant4.LogicalVolume.LogicalVolume method), 51
flukaFreeString() (pyg4ometry.fluka.body.ARB method), 60
flukaFreeString() (pyg4ometry.fluka.body.BOX method), 61
flukaFreeString() (pyg4ometry.fluka.body.ELL method), 61
flukaFreeString() (pyg4ometry.fluka.body.PLA method), 61
flukaFreeString() (pyg4ometry.fluka.body.QUA method), 62
flukaFreeString() (pyg4ometry.fluka.body.RCC method), 62
flukaFreeString() (pyg4ometry.fluka.body.REC method), 63
flukaFreeString() (pyg4ometry.fluka.body.RPP method), 63
flukaFreeString() (pyg4ometry.fluka.body.SPH method), 64
flukaFreeString() (pyg4ometry.fluka.body.TRC method), 64
flukaFreeString() (pyg4ometry.fluka.body.XCC method), 65
flukaFreeString() (pyg4ometry.fluka.body.XEC method), 65
flukaFreeString() (pyg4ometry.fluka.body.XYP method), 65
flukaFreeString() (pyg4ometry.fluka.body.XZP method), 65
flukaFreeString() (pyg4ometry.fluka.body.YCC method), 66
flukaFreeString() (pyg4ometry.fluka.body.YEC method), 66
flukaFreeString() (pyg4ometry.fluka.body.YZP method), 66
flukaFreeString() (pyg4ometry.fluka.body.ZCC method), 67
flukaFreeString() (pyg4ometry.fluka.body.ZEC method), 67

G

geant4Solid() (pyg4ometry.fluka.body.ARB method), 60
geant4Solid() (pyg4ometry.fluka.body.BOX method), 61
geant4Solid() (pyg4ometry.fluka.body.ELL method), 61
geant4Solid() (pyg4ometry.fluka.body.QUA method), 62
geant4Solid() (pyg4ometry.fluka.body.RCC method), 62
geant4Solid() (pyg4ometry.fluka.body.REC method), 63
geant4Solid() (pyg4ometry.fluka.body.RPP method), 63
geant4Solid() (pyg4ometry.fluka.body.SPH method), 64
geant4Solid() (pyg4ometry.fluka.body.TRC method), 64

geant4Solid() (*pyg4ometry.fluka.body.XEC method*), 65
geant4Solid() (*pyg4ometry.fluka.body.YEC method*), 66
geant4Solid() (*pyg4ometry.fluka.body.ZEC method*), 67
GenericPolycone (*class in pyg4ometry.geant4.solid.GenericPolycone*), 43
GenericTrap (*class in pyg4ometry.geant4.solid.GenericTrap*), 48
getMaterialVisOptions() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
getOverlapVisOptions() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
getRegistry() (*pyg4ometry.gdml.Reader.Reader method*), 57
getRegistry() (*pyg4ometry.stl.Reader.Reader method*), 56
getSolid() (*pyg4ometry.stl.Reader.Reader method*), 56
getValueOrExpr() (*pyg4ometry.gdml.Writer.Writer method*), 58
getValueOrExprFromInstance() (*pyg4ometry.gdml.Writer.Writer method*), 58

H

Hype (*class in pyg4ometry.geant4.solid.Hype*), 45

I

Intersection (*class in pyg4ometry.geant4.solid.Intersection*), 49

K

kPhi (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute*), 52
kRho (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute*), 52
kXAxis (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute*), 52
kYAxis (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute*), 52
kZAxis (*pyg4ometry.geant4.ReplicaVolume.ReplicaVolume.Axis attribute*), 52

L

lengths() (*pyg4ometry.fluka.body.BOX method*), 61
lengths() (*pyg4ometry.fluka.body.RPP method*), 63
load() (*pyg4ometry.gdml.Reader.Reader method*), 57
load() (*pyg4ometry.stl.Reader.Reader method*), 56
log() (*in module pyg4ometry.gdml.Defines*), 37
log10() (*in module pyg4ometry.gdml.Defines*), 37
LogicalVolume (*class in pyg4ometry.geant4.LogicalVolume*), 51

M

makeLogicalPhysicalNameSets() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
makeMaterialNameSet() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
makeSolidTessellated() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
makeWorldVolume() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
Material (*class in pyg4ometry.geant4*), 50
Matrix (*class in pyg4ometry.gdml*), 38
mesh() (*pyg4ometry.fluka.body.QUA method*), 62
mesh() (*pyg4ometry.geant4.solid.EllipticalTube.EllipticalTube method*), 44
mesh() (*pyg4ometry.geant4.solid.Sphere.Sphere method*), 42
module
  pyg4ometry.fluka.body, 60
  pyg4ometry.fluka.FlukaRegistry, 67
  pyg4ometry.fluka.Reader, 67
  pyg4ometry.gdml.Defines, 35, 36, 38
  pyg4ometry.gdml.Reader, 57
  pyg4ometry.gdml.Writer, 58
  pyg4ometry.geant4.LogicalVolume, 51
  pyg4ometry.geant4.ParameterisedVolume, 52
  pyg4ometry.geant4.PhysicalVolume, 51
  pyg4ometry.geant4.ReplicaVolume, 52
  pyg4ometry.geant4.solid.Box, 39
  pyg4ometry.geant4.solid.Cons, 40
  pyg4ometry.geant4.solid.CutTubs, 40
  pyg4ometry.geant4.solid.Ellipsoid, 44
  pyg4ometry.geant4.solid.EllipticalTube, 44
  pyg4ometry.geant4.solid.ExtrudedSolid, 46
  pyg4ometry.geant4.solid.GenericPolycone, 43
  pyg4ometry.geant4.solid.GenericTrap, 48
  pyg4ometry.geant4.solid.Hype, 45
  pyg4ometry.geant4.solid.Intersection, 49
  pyg4ometry.geant4.solid.Orb, 42
  pyg4ometry.geant4.solid.Para, 41
  pyg4ometry.geant4.solid.Paraboloid, 45
  pyg4ometry.geant4.solid.Plane, 39
  pyg4ometry.geant4.solid.Polycone, 43
  pyg4ometry.geant4.solid.Polyhedra, 43
  pyg4ometry.geant4.solid.Sphere, 41
  pyg4ometry.geant4.solid.Subtraction, 49
  pyg4ometry.geant4.solid.TessellatedSolid, 49
  pyg4ometry.geant4.solid.Tet, 45
  pyg4ometry.geant4.solid.Torus, 42
  pyg4ometry.geant4.solid.Trap, 41

pyg4ometry.geant4.solid.Trd, 41
pyg4ometry.geant4.solid.Tubs, 40
pyg4ometry.geant4.solid.TwistedBox, 46
pyg4ometry.geant4.solid.TwistedTrap, 46
pyg4ometry.geant4.solid.TwistedTrd, 47
pyg4ometry.geant4.solid.TwistedTubs, 48
pyg4ometry.geant4.solid.Union, 49
pyg4ometry.geant4.solid.Wedge, 39
pyg4ometry.stl.Reader, 56
pyg4ometry.visualisation.VtkViewer, 53
MouseInteractorNamePhysicalVolume (*class in pyg4ometry.visualisation.VtkViewer*), 53

N

nonzero() (*pyg4ometry.gdml.Defines.VectorBase method*), 37

O

Orb (*class in pyg4ometry.geant4.solid.Orb*), 42

P

Para (*class in pyg4ometry.geant4.solid.Para*), 41
Paraboloid (*class in pyg4ometry.geant4.solid.Paraboloid*), 45
ParameterisedVolume (*class in pyg4ometry.geant4.ParameterisedVolume*), 52
ParameterisedVolume.BoxDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.ConeDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.EllipsoidDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.HypeDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.OrbDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.ParaDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.PolyconeDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.PolyhedraDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.SphereDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.TorusDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.TrapDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.TrdDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
ParameterisedVolume.TubeDimensions (*class in pyg4ometry.geant4.ParameterisedVolume*), 53
parseBox() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseCone() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseCutTube() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseDefines() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseEllipsoid() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseEllipticalCone() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseEllipticalTube() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseExtrudedSolid() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseGenericPolycone() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseGenericPolyhedra() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseGenericTrap() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseHype() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseIntersection() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseMaterials() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseMultiUnion() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseOpticalSurface() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseOrb() (*pyg4ometry.gdml.Reader.Reader method*), 57
parsePara() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseParaboloid() (*pyg4ometry.gdml.Reader.Reader method*), 57
parsePhysicalVolumeChildren() (*pyg4ometry.gdml.Reader.Reader method*), 57

parsePolycone() (*pyg4ometry.gdml.Reader.Reader method*), 57
parsePolyhedra() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseScaledSolid() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseSolidLoop() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseSolids() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseSphere() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseStructure() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseSubtraction() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseTessellatedSolid() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseTet() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseTorus() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseTrap() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseTrd() (*pyg4ometry.gdml.Reader.Reader method*), 57
parseTube() (*pyg4ometry.gdml.Reader.Reader method*), 58
parseTwistedBox() (*pyg4ometry.gdml.Reader.Reader method*), 58
parseTwistedTrap() (*pyg4ometry.gdml.Reader.Reader method*), 58
parseTwistedTrd() (*pyg4ometry.gdml.Reader.Reader method*), 58
parseTwistedTubs() (*pyg4ometry.gdml.Reader.Reader method*), 58
parseUnion() (*pyg4ometry.gdml.Reader.Reader method*), 58
parseUserInfo() (*pyg4ometry.gdml.Reader.Reader method*), 58
parseVector() (*pyg4ometry.gdml.Reader.Reader method*), 58
PhysicalVolume (*class in pyg4ometry.geant4.PhysicalVolume*), 51
PLA (*class in pyg4ometry.fluka.body*), 61
Plane (*class in pyg4ometry.geant4.solid.Plane*), 39
point() (*pyg4ometry.fluka.body.XCC method*), 65
point() (*pyg4ometry.fluka.body.YCC method*), 66
point() (*pyg4ometry.fluka.body.ZCC method*), 67
Polycone (*class in pyg4ometry.geant4.solid.Polycone*), 43
Polyhedra (*class in pyg4ometry.geant4.solid.Polyhedra*), 43
Position (*class in pyg4ometry.gdml*), 38
printViewParameters() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
PubViewer (*in module pyg4ometry.visualisation.VtkViewer*), 53
pyg4ometry.fluka.body
  module, 60
pyg4ometry.fluka.FlukaRegistry
  module, 67
pyg4ometry.fluka.Reader
  module, 67
pyg4ometry.gdml.Defines
  module, 35, 36, 38
pyg4ometry.gdml.Reader
  module, 57
pyg4ometry.gdml.Writer
  module, 58
pyg4ometry.geant4.LogicalVolume
  module, 51
pyg4ometry.geant4.ParameterisedVolume
  module, 52
pyg4ometry.geant4.PhysicalVolume
  module, 51
pyg4ometry.geant4.ReplicaVolume
  module, 52
pyg4ometry.geant4.solid.Box
  module, 39
pyg4ometry.geant4.solid.Cons
  module, 40
pyg4ometry.geant4.solid.CutTubs
  module, 40
pyg4ometry.geant4.solid.Ellipsoid
  module, 44
pyg4ometry.geant4.solid.EllipticalTube
  module, 44
pyg4ometry.geant4.solid.ExtrudedSolid
  module, 46
pyg4ometry.geant4.solid.GenericPolycone
  module, 43
pyg4ometry.geant4.solid.GenericTrap
  module, 48
pyg4ometry.geant4.solid.Hype
  module, 45
pyg4ometry.geant4.solid.Intersection
  module, 49
pyg4ometry.geant4.solid.Orb
  module, 42
pyg4ometry.geant4.solid.Para
  module, 41
pyg4ometry.geant4.solid.Paraboloid
  module, 45
pyg4ometry.geant4.solid.Plane
  module, 39
pyg4ometry.geant4.solid.Polycone
  module, 43
pyg4ometry.geant4.solid.Polyhedra
  module, 43

pyg4ometry.geant4.solid.Sphere
module, 41
pyg4ometry.geant4.solid.Subtraction
module, 49
pyg4ometry.geant4.solid.TessellatedSolid
module, 49
pyg4ometry.geant4.solid.Tet
module, 45
pyg4ometry.geant4.solid.Torus
module, 42
pyg4ometry.geant4.solid.Trap
module, 41
pyg4ometry.geant4.solid.Trd
module, 41
pyg4ometry.geant4.solid.Tubs
module, 40
pyg4ometry.geant4.solid.TwistedBox
module, 46
pyg4ometry.geant4.solid.TwistedTrap
module, 46
pyg4ometry.geant4.solid.TwistedTrd
module, 47
pyg4ometry.geant4.solid.TwistedTubs
module, 48
pyg4ometry.geant4.solid.Union
module, 49
pyg4ometry.geant4.solid.Wedge
module, 39
pyg4ometry.stl.Reader
module, 56
pyg4ometry.visualisation.VtkViewer
module, 53

Q

QUA (*class in pyg4ometry.fluka.body*), 61
Quantity (*class in pyg4ometry.gdml.Defines*), 36

R

RAW (*class in pyg4ometry.fluka.body*), 62
RCC (*class in pyg4ometry.fluka.body*), 62
Reader (*class in pyg4ometry.gdml.Reader*), 57
Reader (*class in pyg4ometry.stl.Reader*), 56
REC (*class in pyg4ometry.fluka.body*), 63
Registry (*class in pyg4ometry.geant4*), 50
ReplicaVolume (*class in pyg4ometry.geant4.ReplicaVolume*), 52
ReplicaVolume.Axis (*class in pyg4ometry.geant4.ReplicaVolume*), 52
rightButtonPressEvent() (*pyg4ometry.visualisation.VtkViewer.MouseInteractorNamePhysicalVolume method*), 53
Rotation (*class in pyg4ometry.gdml*), 38
rotation() (*pyg4ometry.fluka.body.ARB method*), 60
rotation() (*pyg4ometry.fluka.body.BOX method*), 61
rotation() (*pyg4ometry.fluka.body.ELL method*), 61
rotation() (*pyg4ometry.fluka.body.PLA method*), 61
rotation() (*pyg4ometry.fluka.body.QUA method*), 62
rotation() (*pyg4ometry.fluka.body.RCC method*), 63
rotation() (*pyg4ometry.fluka.body.REC method*), 63
rotation() (*pyg4ometry.fluka.body.RPP method*), 63
rotation() (*pyg4ometry.fluka.body.SPH method*), 64
rotation() (*pyg4ometry.fluka.body.TRC method*), 64
rotation() (*pyg4ometry.fluka.body.XCC method*), 65
rotation() (*pyg4ometry.fluka.body.XEC method*), 65
rotation() (*pyg4ometry.fluka.body.YCC method*), 66
rotation() (*pyg4ometry.fluka.body.YEC method*), 66
rotation() (*pyg4ometry.fluka.body.ZCC method*), 67
rotation() (*pyg4ometry.fluka.body.ZEC method*), 67
RPP (*class in pyg4ometry.fluka.body*), 63

S

ScalarBase (*class in pyg4ometry.gdml.Defines*), 35
Scale (*class in pyg4ometry.gdml*), 38
set_pressure() (*pyg4ometry.geant4.Material method*), 51
set_temperature() (*pyg4ometry.geant4.Material method*), 51
setCameraFocusPosition() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setCutterNormal() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setCutterOrigin() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setExpression() (*pyg4ometry.gdml.Defines.ScalarBase method*), 35
setMaterialVisOptions() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setName() (*pyg4ometry.gdml.Defines.ScalarBase method*), 35
setName() (*pyg4ometry.gdml.Defines.VectorBase method*), 38
setOpacity() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setOpacityOverlap() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setRandomColours() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setRegistry() (*pyg4ometry.gdml.Defines.ScalarBase method*), 35
setRegistry() (*pyg4ometry.gdml.Defines.VectorBase method*), 38
setSolid() (*pyg4ometry.geant4.LogicalVolume.LogicalVolume method*), 51
setSurface() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setSurfaceOverlap() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setWireframe() (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
setWireframeOverlap() (*pyg4ometry.visualisation.VtkViewer.VtkViewer*

method), 55
`sin()` (*in module pyg4ometry.gdml.Defines*), 37
`SPH` (*class in pyg4ometry.fluka.body*), 63
`Sphere` (*class in pyg4ometry.geant4.solid.Sphere*), 41
`sqrt()` (*in module pyg4ometry.gdml.Defines*), 37
`start()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
`state_variables` (*pyg4ometry.geant4.Material property*), 51
`Subtraction` (*class in pyg4ometry.geant4.solid.Subtraction*), 49

T

`tan()` (*in module pyg4ometry.gdml.Defines*), 37
`TessellatedSolid` (*class in pyg4ometry.geant4.solid.TessellatedSolid*), 49
`Tet` (*class in pyg4ometry.geant4.solid.Tet*), 45
`toPlane()` (*pyg4ometry.fluka.body.PLA method*), 61
`toPlane()` (*pyg4ometry.fluka.body.XYP method*), 65
`toPlane()` (*pyg4ometry.fluka.body.XZP method*), 65
`toPlane()` (*pyg4ometry.fluka.body.YZP method*), 66
`Torus` (*class in pyg4ometry.geant4.solid.Torus*), 42
`Trap` (*class in pyg4ometry.geant4.solid.Trap*), 41
`TRC` (*class in pyg4ometry.fluka.body*), 64
`Trd` (*class in pyg4ometry.geant4.solid.Trd*), 41
`Tubs` (*class in pyg4ometry.geant4.solid.Tubs*), 40
`TwistedBox` (*class in pyg4ometry.geant4.solid.TwistedBox*), 46
`TwistedTrap` (*class in pyg4ometry.geant4.solid.TwistedTrap*), 46
`TwistedTrd` (*class in pyg4ometry.geant4.solid.TwistedTrd*), 47
`TwistedTubs` (*class in pyg4ometry.geant4.solid.TwistedTubs*), 48

U

`Union` (*class in pyg4ometry.geant4.solid.Union*), 49
`upgradeToStringExpression()` (*in module pyg4ometry.gdml.Defines*), 35
`upgradeToTransformation()` (*in module pyg4ometry.gdml.Defines*), 38
`upgradeToVector()` (*in module pyg4ometry.gdml.Defines*), 39

V

`Variable` (*class in pyg4ometry.gdml.Defines*), 36
`VectorBase` (*class in pyg4ometry.gdml.Defines*), 37
`view()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
`viewSection()` (*pyg4ometry.visualisation.VtkViewer.VtkViewer method*), 55
`visualise()` (*pyg4ometry.stl.Reader.Reader method*), 56
`VtkViewer` (*class in pyg4ometry.visualisation.VtkViewer*), 53
`VtkViewerColoured` (*class in pyg4ometry.visualisation.VtkViewer*), 56
`VtkViewerColouredMaterial` (*class in pyg4ometry.visualisation.VtkViewer*), 56

W

`WED` (*class in pyg4ometry.fluka.body*), 64
`Wedge` (*class in pyg4ometry.geant4.solid.Wedge*), 39
`write()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeAssemblyVolume()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeAuxiliary()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeBorderSurface()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeBox()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeCons()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeCutTubs()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeDefaultGDML()` (*pyg4ometry.stl.Reader.Reader method*), 56
`writeDefaultLattice()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeDefine()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeDivisionVolume()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeEllipsoid()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeEllipticalCone()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeEllipticalTube()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeExtrudedSolid()` (*pyg4ometry.gdml.Writer.Writer method*), 58
`writeGenericPolycone()` (*pyg4ometry.gdml.Writer.Writer method*), 59
`writeGenericPolyhedra()` (*pyg4ometry.gdml.Writer.Writer method*), 59
`writeGenericTrap()` (*pyg4ometry.gdml.Writer.Writer method*), 59
`writeGmadTester()` (*pyg4ometry.gdml.Writer.Writer method*), 59
`writeGMADTesterNoBeamline()` (*pyg4ometry.gdml.Writer.Writer method*), 59
`writeHype()` (*pyg4ometry.gdml.Writer.Writer method*), 59
`writeIntersection()` (*pyg4ometry.gdml.Writer.Writer method*), 59
`writeLogicalVolume()` (*pyg4ometry.gdml.Writer.Writer method*), 59
`writeMaterial()` (*pyg4ometry.gdml.Writer.Writer method*), 59

writeMultiUnion() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeOpticalSurface() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeOrb() (*pyg4ometry.gdml.Writer.Writer method*), 59
writePara() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeParaboloid() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeParametrisedVolume() (*pyg4ometry.gdml.Writer.Writer method*), 59
writePhysicalVolume() (*pyg4ometry.gdml.Writer.Writer method*), 59
writePolycone() (*pyg4ometry.gdml.Writer.Writer method*), 59
writePolyhedra() (*pyg4ometry.gdml.Writer.Writer method*), 59
Writer (*class in pyg4ometry.gdml.Writer*), 58
writeReplicaVolume() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeScaled() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeSkinSurface() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeSolid() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeSphere() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeSubtraction() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTessellatedSolid() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTet() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTorus() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTrap() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTrd() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTubs() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTwistedBox() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTwistedTrap() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTwistedTrd() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeTwistedTubs() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeUnion() (*pyg4ometry.gdml.Writer.Writer method*), 59
writeVectorVariable() (*pyg4ometry.gdml.Writer.Writer method*), 59

X

XCC (*class in pyg4ometry.fluka.body*), 64
XEC (*class in pyg4ometry.fluka.body*), 65
XYP (*class in pyg4ometry.fluka.body*), 65
XZP (*class in pyg4ometry.fluka.body*), 65

Y

YCC (*class in pyg4ometry.fluka.body*), 66
YEC (*class in pyg4ometry.fluka.body*), 66
YZP (*class in pyg4ometry.fluka.body*), 66

Z

ZCC (*class in pyg4ometry.fluka.body*), 67
ZEC (*class in pyg4ometry.fluka.body*), 67