GDML User Manual

What is GDML?

GDML stands for geometry description markup language. Its purpose is to store Geant4 setups. It is XML based so there is no need to use any special software to modify GDML files.

What is new?

In Geant4 9.1 GDML became a fully integrated part of the system. The previous implementation of GDML, known as GDML 2.10.0, was an external package. Building Geant4 with GDML support is only a matter of setting an environment variable now. The setup script will offer it as an option.

A simple GDML example

A GDML file has five main sections within the root element: define, materials, solids, structure and setup. These are discussed in detail in the following chapters. The root element itself is named gdml.

```
<gdml>

    <define>
        <constant name="size" value="100.0"/>
        <rotation name="rotZ" z="30" unit="deg"/>
    </define>

    <materials>
        <material name="Air" Z="1.0">
            <D value="1e-25"/>
            <atom value="1.00794"/>
        </material>
        <material name="Aliminium" Z="13.0">
            <D value="2.7"/>
            <atom value="26.98"/>
        </material>
    </materials>

    <solids>
        <box name="TheBox" x="size"  y="size"  z="size" lunit="mm"/>
        <box name="WorldBox" x="500" y="500" z="500" lunit="mm"/>
    </solids>

    <structure>
        <volume name="lvBox">
            <materialref ref="Aluminium"/>
            <solidref ref="TheBox"/>
        </volume>
        <volume name="TOP">
            <materialref ref="Air"/>
            <solidref ref="WorldBox"/>
            <physvol>
                <volumeref ref="lvBox"/>
                <rotationref ref="rotZ"/>
            </physvol>
        </volume>
    </stucture>

    <setup name="Default" version="1.0">
        <world ref="TOP"/>
    </setup>

</gdml>
```

Definitions

The first section of a GDML file contains definitions. These defined items can be referenced by its names afterwards.

constant

Once a constant is defined, it can be used in any expression. The name is mandatory and the value is zero by default. A constant has no unit.

```
<constant name="Pi"  value="3.1415"/>
<constant name="TwoPi"  value="2.0*Pi"/>
```

variable

Once a variable is defined, it can be used in any expression and loop. The name is mandatory and the value is zero by default. A variable has no unit.

```
<variable name="x"  value="125.0"/>
```

scale

Once a scale is defined, it can be referenced where a scale is expected. The name is mandatory and every scale factor is one by default. Scale has no unit.

```
<scale name="scl"  x="2.0"  y="4.0"  z="8.0" />
<scale name="StrectchY"  y="2.0"/>
```

Constants can be used as well:

```
<constant name="ratio" value="2.0"/>
<scale name="scl"  x="ratio"  y="2.0*ratio"  z="4.0*ratio" />
```

Scale can be used to implement reflection:

```
<scale name="planar_reflection"  x="-1"  y="+1"  z="+1" />
<scale name="axial_reflection"  x="-1"  y="-1"  z="+1" />
<scale name="point_reflection"  x="-1"  y="-1"  z="-1" />
```

rotation

Performs counter-clockwise rotation in a left-handed coordinate system. Once a rotation is defined, it can be referenced where a rotation is expected. The name is mandatory, the rotations are zeros and the unit is radian by default.

```
<rotation name="rot"  x="30.0"  y="45.0"  z="60.0"  unit="deg"/>
<rotation name="RotateZ"  z="30"  unit="deg"/>
```

Constants can be used as well:

```
<constant name="Pi"  value="3.1415"/>
<rotation name="rot"  x="Pi/6.0"  y="Pi/4.0"  z="Pi/3.0"  unit="rad"/>
```

position

Once a position is defined, it can be referenced where a position is expected. The name is mandatory, the coordinates are zeros and the unit is „mm” by default.

<position name=”pos” x=”25.0” y=”50.0” z=”75.0” unit=”mm”/>
<position name=”TranslateX” x=”100.0” unit=”cm”/>

Constants can be used as well:

<constant name=”size” value=”25.0”/>
*<position name=”pos” x=”size” y=”2.0*size” z=”3.0*size” unit=”mm”/>*

matrix

Definition of a two-dimensional matrix or a row/column matrix.

coldim:	number of columns
values:	matrix elements in row-mayor order, separated with space

The matrix should be filled up correctly: the number of matrix elements should be an integer multiple of the number of columns. The elements should be specified in row-mayor order, separated with space. The matrix elements have no unit. Once a matrix is defined it can be referenced where a matrix is expected. Below you can find examples of various matrices and how they will look like in GDML. Here is an example of a square matrix:

| 1 2 3 |
| 4 5 6 |
| 7 8 9 |

<matrix name=”MSquare” coldim=”3” values=”1 2 3 4 5 6 7 8 9”/>

Here is an example of a row matrix:

| 1 2 3 |

<matrix name=”MRow” coldim=”3” values=”1 2 3”/>

Here is an example of a column-matrix:

| 1 |
| 2 |
| 3 |

<matrix name=”MCol” coldim=”1” values=”1 2 3”/>

The elements of a matrix can be accessed and used in any following expression. The indices should be specified in row-mayor oder and they are zero-based. Int he following case the value of „x” equals to 16.

*<constant name=”x” value=”4.0*MSquare[1,0]”/>*

Row matrices and column matrices have obviously only one index.

*<constant name=”z” value=”4.0*MCol[0]/”>*
*<constant name=”y” value=”4.0*MRow[0]”/>*

quantity

Definition of a quantity. In GDML quantities are constants with unit. Once a quantity is defined it can be referenced where a quantity is expected with the same type.

type:	the type of the quantity.
value:	the value of the quantity
unit:	the unit of the quantity

<quantity name="WaterDensity" type="density" value="1" unit="g/cm3"/>

Available quantities		
description of quantity	type string	Default unit
absorption length	lambda	cm
atomic mass	A	g/mole
density	density	g/cm3
pressure	pressure	pascal
Radiation length	X0	cm
temperature	temperature	K

Materials

isotope

Definition of an isotope.

Z:	atomic number
N:	number of nucleons

```
<isotope name="U235" Z="92" N="235"/>
   <atom value="235.01"  unit="g/mole"/>
</isotope>
```

element

Definition of an element.

```
<element name="Oxygen"  Z="8">
   <atom value="16.0"  unit="g/mole"/>
</element>
```

An element also can be defined from isotopes specified by fractional masses:

```
<isotope name="enriched_uranium">
   <fraction n="0.9" ref="U235"/>
   <fraction n="0.1" ref="U238"/>
</isotope>
```

material

A material can be defined directly from an element:

```
<material name="Oxygen" Z="8">
   <atom value="16.0" unit="g/mole"/>
</material>
```

A material can be defined by a combination of elements specified by fractional masses:

```
<material name="Air">
   <fraction  n="0.7" ref="Nitrogen"/>
   <fraction n="0.3" ref="Oxygen"/>
</material>
```

A material can be defined by a combination of elements specified by atom count:

```
<material name="Air">
   <composite n="2" ref="Hydrogen"/>
   <composite n="1" ref="Oxygen"/>
</material>
```

Solids

The geometry of a particular setup is described with a composition of various solids. All of them should be defined in the solids section. In GDML the following solids are available:

box

Definition of an axis aligned box. The origin is at the center of the box.

x,y,z:	dimensions of the box

<box name="TheBox" x="100" y="200" z="300" lunit="mm"/>

cone

Definition of a cone or conical section. The origin is at halfway between the base and the top of the cone. The inner radius always should be less than it's respective outter radius.

rmin1:	inner radius at the base
rmax1:	outter radius at the base
rmin2:	inner radius at the top
rmax2:	outter radius at the top
z:	distance between the base and the top
startphi:	start of angle segment
endphi:	end of angle segment

<cone name="TheCone" rmin1="1.0" rmax1="2.0" rmin2="2.0" rmax2="4.0" z="4.0" startphi="0.0" endphi="180.0" aunit="deg" lunit="mm"/>

ellipsoid

Definition of an axis-aligned ellipsoid. The origin is at the center. The clipping planes are perpendicular to the Z-axis and their position is measured along the Z-axis. The position of the first clipping plane must be less than the position of the second clipping plane.

ax,by,cz:	radiuses
zcut1:	position of first clipping plane
zcut2:	position of second clipping plane

<ellipsoid name="TheEllipsoid" ax="1.0" by="2.0" cz="3.0" zcut1="-1.5" zcut2="1.5" lunit="mm"/>

eltube

Definition of a tube with elliptical cross section. The tube is aligned to the Z-axis and the origin is at the center.

dx,dy:	radiuses of the base ellipsoid
dz:	half length of the tube

<eltube name="TheEltube" dx="1.0" dy="2.0" dz="4.0" lunit="mm"/>

hype

Definition of a tube with hyperbolic profile.

rmin:	inner radius
rmax:	outter radius
inst:	inner stereo angle
outst:	outter stereo angle
z:	length

<hype name="TheHype" rmin="1.0" rmax="4.0" inst="" inst="" outst="" aunit="deg" lunit="mm" />

orb

Definition of a sphere.

r:	radius

<sphere name="TheSphere" r="4.0" lunit="mm"/>

para

Definition of a paralelepiped.

x,y,z:	dimensions of the initial box.
alpha:	alpha
theta:	theta
phi:	phi

<para name="ThePara" x="10" y="20" z="30" alpha="" theta="" phi=""/>

polycone

Definition of extrusion of a circle.

startphi:	start of angle segment
deltaphi:	size of angle segment

```
<polycone name="ThePolycone" startphi="0" deltaphi="180" aunit="deg"
lunit="mm">
    <zplane rmin="1.0" rmax="4.0" z="0.0"/>
    <zplane rmin="0.5" rmax="2.0" z="4.0"/>
    <zplane rmin="1.0" rmax="4.0" z="8.0"/>
</polycone>
```

polyhedra

Definition of extrusion of a polyhedra.

startphi:	start angle of segment
deltaphi:	size of angle segment
numsides:	number of sides of the polyhedra

```
<polyhedra name="ThePolyhedra" startphi="0" deltaphi="180" numsides="8"
aunit="deg" lunit="mm">
    <zplane rmin="1.0" rmax="4.0" z="0.0"/>
    <zplane rmin="0.5" rmax="2.0" z="4.0"/>
    <zplane rmin="1.0" rmax="4.0" z="8.0"/>
</polyhedra>
```

reflectedSolid

Definition of a general transformation for a solid. In the previous version of GDML it was the only way of using scaling transformation or reflection. In this version it is recommended using the scale transformation instead (Please refer to section …).

solid:	the name of the solid to be transformed
sx,sy,sz:	scaling factors
rx,ry,rz:	rotational angles
dx,dy,dz:	translation

```
<reflectedSolid name="TheReflected" solid="The" sx="-1" sy="1" sz="1" rx="0"
ry="0" rz="0" dx="0" dy="0" dz="0"/>
```

sphere

Definition of a sphere or spherical shell section.

rmin:	inner radius
rmax:	outter radius
startphi:	start of azimuthal angle segment
deltaphi:	size of azimuthal angle segment
starttheta:	start of polar angle segment
deltatheta:	size of polar angle segment

```
<sphere name="TheSphere" rmin="1.0" rmax="2.0" startphi="0" deltaphi="270"
starttheta="0" deltatheta="90" aunit="deg" lunit="mm"/>
```

tessellated

Definition of a tessellated solid. A solid can be formed using either triangular or quadrangular faces. At first, all the vertices should be defined int he define section.

```
<define>
   <position name="base0" x="+20" y="+20" z="0" unit="mm"/>
   <position name="base1" x="+20" y="-20" z="0" unit="mm"/>
   <position name="base2" x="-20" y="-20" z="0" unit="mm"/>
   <position name="base3" x="-20" y="+20" z="0" unit="mm"/>
   <position name="top" x="0" y="0" z="20" unit="mm"/>
</define>
```

Once the vertices are defined, they can be referenced in the tessellated solid. Obviously the triangular face has three attributes and the quadrangular face hase four attributes. The faces should be defined in anti-clockwise order.

```
<solids>
   <tessellated name="TheTessellated">
         <triangular vertex1="base0" vertex2="base1" vertex3="top"/>
         <triangular vertex1="base1" vertex2="base2" vertex3="top"/>
         <triangular vertex1="base2" vertex2="base3" vertex3="top"/>
         <triangular vertex1="base3" vertex2="base0" vertex3="top"/>
         <quadrangular vertex1="base3" vertex2="base2" vertex3="base1"
         vertex4="base0"/>
   <tessellated>
</solids>
```

tet

Definition of a tetrahedron. A tetrahedron is formed using four vertices. At first all four vertices should be defined in the define section:

```
<define>
   <position name="base0" x="0" y="0" z="0" unit="mm"/>
   <position name="base1" x="8" y="0" z="0" unit="mm"/>
   <position name="base2" x="0" y="8" z="0" unit="mm"/>
   <position name="top" x="0" y="0" z="8" unit="mm"/>
</define>
```

Once the vertices are defined, they can be referenced in the tetrahedron:

```
<solids>
   <tet name="TheTet" vertex1="base0" vertex2="base1" vertex3="base2"
   vertex4="top"/>
</solids>
```

torus

Definition of a torus segment.

rmin:	inner radius
rmax:	outter radius
rtor:	radius of the torus
startphi:	start of angle segment
deltaphi:	size of angle segment

<torus name="TheTorus" rmin="1.0" rmax="2.0" rtor="5.0" startphi="0" endphi="90" aunit="deg" lunit="rad"/>

trap

Definition of a general trapezoid.

z:	length
theta:	polar angle of the joining centres
phi:	azimuthal angle of the joining centres
y1:	y length at the bottom
x1:	x length at the bottom left
x2:	x length at the bottom right
alpha1:	angle to the y axis from the centre of the side (lower endcap)
y2:	y length at the top
x3:	x length at the top left
x4:	x length at the top right
alpha2:	angle to the y axis from the centre of the side (upper endcap)

<trap name="TheTrap" z="4.0" theta="0" phi="0" y1="8.0" x1="6.0" x2="5.0" alpha1="0" y2="6.0" x3="3.0" x4="4.0" alpha2="0" lunit="mm" aunit="deg"/>

trd

Definition of a regular trapezoid. The base and the top rectangles are aligned to the X and Y axes therefore the height, the distance between the base and the top, is measured along the Z axis.

x1:	base rectangle size along the X axis
y1:	base rectangle size along the Y axis
x2:	top rectangle size along the X axis
y2:	top rectangle size along the Y axis
z:	height of the trapezoid

<trd name="TheTrd" x1="200" y1="400" x2="100" y2="200" z="100" lunit="mm" />

tube

Definition of a tube or a tube segment. The tube segment is aligned to the Z-axis and the origin is at the center.

rmin:	inner radius
rmax:	outter radius
z:	length of the tube
startphi:	start of angle segment
deltaphi:	size of angle segment

<tube name="TheTube" rmin="1.0" rmax="2.0" z="8.0" startphi="0" deltaphi="90" aunit="deg" lunit="mm"/>

twistedbox

Definition of a twisted box. The twisting is applied to the Z-axis.

x,y,z:	dimensions of the box
PhiTwist:	angle of the twist

<twistedbox name="Thetwistedbox" x="100" y="200" z="300" PhiTwist="45" aunit="deg" lunit="mm"/>

twistedtrap

Definition of a twisted general trapezoid.

z:	length
theta:	polar angle of the joining centres
phi:	azimuthal angle of the joining centres
y1:	y length at the bottom
x1:	x length at the bottom left
x2:	x length at the bottom right
alpha1:	angle to the y axis from the centre of the side (lower endcap)
y2:	y length at the top
x3:	x length at the top left
x4:	x length at the top right
alpha2:	angle to the y axis from the centre of the side (upper endcap)
PhiTwist:	angle of twist along the Z axis

<twistedtrap name="TheTwistedtrap" z="4.0" theta="0" phi="0" y1="8.0" x1="6.0" x2="5.0" alpha1="0" y2="6.0" x3="3.0" x4="4.0" alpha2="0" PhTwist="90" lunit="mm" aunit="deg"/>

twistedtrd

Definition of a twisted regular trapezoid. The base and the top rectangles are aligned to the X and Y axes therefore the height, the distance between the base and the top, is measured along the Z axis.

x1:	base rectangle size along the X axis
y1:	base rectangle size along the Y axis
x2:	top rectangle size along the X axis
y2:	top rectangle size along the Y axis
z:	height of the trapezoid
PhiTwist:	angle of the twist

```
<twistedtrd name="TheTwistedtrd"  x1="200"  y1="400" x2="100" y2="200"
z="100"PhiTwist="90.0" aunit="deg"  lunit="mm" />
```

twistedtubs

Definition of a twisted tube segment. . The tube segment is aligned to the Z-axis and the origin is at the center. The twisting is applied to the Z-axis.

endinnerrad:	inner radius
endouterrad:	outter radius
zlen:	length of the tube
phi:	size of angle segment
twistedangle:	angle of twist

```
<twistedtubs name="TheTwistedtubs" endinnerrad="1.0" endouterrad="2.0"
zlen="8.0" phi="90" twistedangle="45"  aunit="deg" lunit="mm"/>
```

xtru

Definition of a solid formed by polygon extrusion. The vertices of the polygon should be defined in clockwise order. The following example demonstrates a pyramid.

zPosition:	distance of the section from plane Z=0
xOffset,yOffset:	the section can be displaced with a vector paralell to the plane Z.
scalingFactor:	scale ratio of the section relative to the original polygon

```
<xtru name="TheXtru">
   <TwoDimVertex x="+10" y="+10"/>
   <TwoDimVertex x="+10" y="-10"/>
   <TwoDimVertex x="-10" y="-10"/>
   <TwoDimVertex x="-10" y="+10"/>
   <section  zPosition="0"  xOffset="0" yOffset="0" scalingFactor="1.0"/>
   <section  zPosition="10"  xOffset="0" yOffset="0" scalingFactor="0.5"/>
</xtru>
```

Structure

Once we are having all the materials and solids of our experimental setup or detector, we can compose it.

physvol

Definition of a physical volume placement with a modeling transformation.

volumeref:	name of the volume to be placed (mandatory)
scale:	scaling (optional)
rotation:	rotation (optional)
position:	translation (optional)

```
<volume name="World">
   <materialref ref="Air"/>
   <solidref ref="WorldBox"/>
   <physvol>
         <volumeref ref="AluminiumCube"/>
         <position name="ThePosition" x="100.0" y="100" z="100.0" unit="mm"/>
         <rotation name ="TheRotation" x="30.0" y=" 45.0" z="60.0" unit="deg"/>
         <scale name="TheScale" x="1.0" y="1.0" z="1.0"/>
   </physvol>
</volume>
```

In this example, using a physical volume placement, the „World" volume is becoming the mother of the daughter volume „AluminiumCube". The modeling transformation is described by three components: scaling, rotation and translation. These components can be specified either directly, as it can be seen in the above example, or by referencing. In this case they must be defined in the define section of the GDML file.

```
<physvol>
   <volumeref  ref="AluminiumCube"/>
   <positionref  ref="ThePosition" />
   <rotationre fref ="TheRotation" />
   <scaleref  ref="TheScale" />
</physvol>
```

It is also possible to refer to a volume from an other GDML file. In this case the file name must be specified and a logical volume as an option. If no logical volume is specified, the world volume of that module will be taken by default. All the names coming from the module will automatically include the file name of that module without extension. The volume name will be „module_TOP" in the example below.

name: name of the module (mandatory)
volname: name of the logical volume in that module (optional)

```
<physvol>
   <file name="module.gdml" volname="TOP"/>
   <positionref  ref="ThePosition" />
   <rotationre fref ="TheRotation" />
   <scaleref  ref="TheScale" />
</physvol>
```

volume

Definition of a logical volume, what is an association of a solid with a material

name:	the only attribute is the name with it can be referenced by
materialref:	name of the material
solidref:	name of the solid

The following example is featuring a cube made of aluminium.

```
<volume name="AluminiumCube">
    <materialref ref="Aluminium"/>
    <solidref ref="Cube"/>
</volume>
```

A logical volume itself does not appear in the geometry tree. The only exception is the world logical volume which is the root of the geometry tree. The world volume must be chosen in the setup section (Please refer to chapter: „A simple GDML example").

replicavol

If there is a sequence of physical volume placements of the same logical volume, following a regular pattern, it can be substitued with a single replica. You can

```
<replicavol>
<replicavol>
```

divisionvol

paramvol

Loops

Loops can be used in the solids and structure section of GDML files in a similiar way like in a programming language. The following example illustrates an array of cubes. Note that the loop variable can be used both for forming a name and for manipulating expressions like the dimensions of a box.

```
<define>
   <variable name=”i” value=”0”/>
</define>
<solids>
   <loop for=”i” from=”0” to=”9” step=”1”>
         <box name=”BoxArray[i]”  x=”10+2*i” y=”10+2*i” z=”10+2*i”/>
   <loop />
</solids>
```

Multiply embedded loops can be used as well. The following example illustrates a three-dimensional array of cubes.

```
<define>
   <variable name=”i” value=”0”/>
   <variable name=”j” value=”0”/>
   <variable name=”k” value=”0”/>
</define>
<solids>
   <loop for=”i” from=”0” to=”9” step=”1”>
   <loop for=’j” from=”0” to=”9” step=”1”>
   <loop for=’k” from=”0” to=”9” step=”1”>
         <box name=”BoxArray[i,j,k]”  x=”10” y=”10” z=”10”/>
   <loop />
   <loop />
   <loop />
</solids>
```

How to read and write GDML files?

In your code you only need to include a single file:

#include <G4GDMLParser.hh>

Now you can instantiate the class which is capable of both reading and writing GDML files. In the „geant4/examples/extended/gdml" directory you can find an example demonstrating both the reader and the writer functionality.

G4GDMLParser parser;

If you wish to read a GDML setup into geant4, use the following function with the filename specified as an argument.

parser.Read(„input.gdml");

After a succesful reading use the function below to get a pointer to the world/top volume. The returned volume is a physical volume, so it can be directly used in your detector construction class.

*G4VPhysicalVolume *pvWorld = parser.GetWorldVolume();*

If you wish to write out your Geant4 setup created in C++, use the following function where you should specify two arguments: the name of the intended output file and a pointer to the world volume, what should be a logical volume.

parser.Write(„output.gdml",(G4LogicalVolume)lvWorld);*

By default, the schema location is pointing to a web address. As an option, you can specify an alternative location of the schema. Also the schema is distributed with the package and it can be found in the 'geant4/source/persistency/gdml/schema' directory.

parser.Write(„output.gdml",(G4LogicalVolume)lvWorld,"GDMLSchema/gdml.xsd");*

Uniqueness of the names

Geant4 entities are linked together by their pointers, but in GDML they are linked together by their names. These names are specified by the user, therefore it is the users responsibility to specify unique names. However, there is a possibility to include pointers into the names ensuring the uniqueness. This feature increases security, but reduces readablity.

parser.SetAddPointerToName(true);

How to write modular GDML files?

GDML has the possibility of splitting up a single setup into multiple modules. This can be useful in the case of highly complex models. Modules can be created by specifying pointers to the physical volumes where we intend to split the geometry tree. The module will inherit the name of the physical volume where the splitting occurs.

```
G4GDMLParser writer;
writer.AddModule(pvFieldCoils);
writer.AddModule(pvArmature);
writer.Write(„main.gdml”,lvGeneratorl);
```

Modules can be created by specifying the level of depth or level of depths where you intend to split. In this case it is not possible to specify names for the modules because we don't know in advance how many modules will be created. The module will inherit the name of the physical volume where the splitting occurs.

```
G4GDMLParser writer;
writer.AddModule(4);
writer.AddModule(8);
writer.Write(„main.gdml”,lvGenerator);
```

Modules can be created by mixing the two approaches as well.

```
G4GDMLParser writer;
writer.AddModule(pvFieldCoils);
writer.AddModule(8);
writer.Write(„main.gdml”,lvGenerator);
```

The order of specifying the modules does not matter. However, the 'Write' function, with the root of the geometry tree, must be called after specifying the modules. An error message will be thrown if a certain modularization is requested twice. The modules are linked in the physical volume placement (see 'physvol' in chapter 'Structure').

Advices on modularization

If we intend to break up a complex experimental setup into more modules, the following advices should be taken into consideration:

- It is wise to select a module in a way that it reflects the real world. For example, we can have a separate module for a calorimeter. Properly selected modules can be easily reused at a later stage.
- A module should contain only the volumes those contribute to the real-world module.
- Volumes of a particular module should not include volumes of an other module.
- Selecting independent modules from a tree of volumes is the users responsibility, but the rest of the job will be done by the writer: only the involved materials/solids will be included into a module.

Importing CAD geometry

The package has the possibility of parsing STEP files directly into Geant4. A STEP setup consists of two files: a ’geom’ and a ’tree’ file. The ’geom’ file represents the list of solids and the ’tree’ file represents the structure. A STEP setup has no information regarding materials. The example below features parsing a STEP setup.

```
G4GDMLParser parser;
G4Material* Air = new G4Material(...);
G4Material* Aluminium = new G4Material(...);
parser.ReadSTEP(„engine”,Air,Aluminium);
```

Since a STEP setup involves two files with same name but different extension, you should specify a single filename but without extension. The example will read the ’engine.geom’ and ’engine.tree’ files. It is also necessary to specify the material of the medium and the material of the solid parts as well, since a STEP setup has no information regarding materials.