Detector Description: Basics

http://cern.ch/geant4
PART II

Describing a detector - I

• Detector geometry modeling
• The basic concepts: solids & volumes
Describe your detector

- Derive your own concrete class from `G4VUserDetectorConstruction` abstract base class.
- Implementing the method `Construct()`:
  - Modularize it according to each detector component or sub-detector:
    - Construct all necessary materials
    - Define shapes/solids required to describe the geometry
    - Construct and place volumes of your detector geometry
    - Define sensitive detectors and identify detector volumes which to associate them
    - Associate magnetic field to detector regions
    - Define visualization attributes for the detector elements
Creating a Detector Volume

- Start with its Shape & Size
  - Box 3x5x7 cm, sphere R=8m

- Add properties:
  - material, B/E field,
  - make it sensitive

- Place it in another volume
  - in one place
  - repeatedly using a function

- Solid
- Logical-Volume
- Physical-Volume
Define detector geometry

- Three conceptual layers
  - **G4VSolid** -- shape, size
  - **G4LogicalVolume** -- daughter physical volumes, material, sensitivity, user limits, etc.
  - **G4VPhysicalVolume** -- position, rotation
Define detector geometry

- Basic strategy
  ```cpp
  G4VSolid* pBoxSolid = new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);
  G4LogicalVolume* pBoxLog = new G4LogicalVolume( pBoxSolid, pBoxMaterial, "aBoxLog", 0, 0, 0);
  G4VPhysicalVolume* aBoxPhys = new G4PVPlacement( pRotation, G4ThreeVector(posX, posY, posZ), pBoxLog, "aBoxPhys", pMotherLog, 0, copyNo);
  ```

- A unique physical volume which represents the experimental area must exist and fully contains all other components
  - The world volume
PART II

Describing a detector - II

• Logical and Physical Volumes
G4LogicalVolume

```cpp
G4LogicalVolume(G4VSolid* pSolid, G4Material* pMaterial, 
    const G4String& name, G4FieldManager* pFieldMgr=0, 
    G4VSensitiveDetector* pSDetector=0, 
    G4UserLimits* pULimits=0, 
    G4bool optimise=true);
```

- Contains all information of volume except position:
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits
  - Shower parameterisation

- Physical volumes of same type can share a logical volume.
- The pointers to solid and material must be NOT null
- Once created it is automatically entered in the LV store
- It is not meant to act as a base class
Geometrical hierarchy

- Mother and daughter volumes
  - A volume is placed in its mother volume
    - Position and rotation of the daughter volume is described with respect to the local coordinate system of the mother volume
    - The origin of the mother's local coordinate system is at the center of the mother volume
    - Daughter volumes cannot protrude from the mother volume
    - Daughter volumes cannot overlap
  - One or more volumes can be placed in a mother volume
Geometrical hierarchy

- **Mother and daughter volumes (cont.)**
  - The logical volume of mother knows the daughter volumes it contains
    - It is uniquely defined to be their mother volume
    - If the logical volume of the mother is placed more than once, all daughters appear by definition in all these physical instances of the mother

- **World volume is the root volume of the hierarchy**
  - The world volume must be a unique physical volume which **fully contains with some margin** all other volumes
    - The world defines the global coordinate system
    - The origin of the global coordinate system is at the center of the world volume
    - Should not share any surface with contained geometry
One logical volume can be placed more than once. One or more volumes can be placed in a mother volume.

Note that the mother-daughter relationship is an information of `G4LogicalVolume`.

- If the mother volume is placed more than once, all daughters by definition appear in each placed physical volume.

The world volume must be a unique physical volume which fully contains with some margin all the other volumes.

- The world volume defines the global coordinate system. The origin of the global coordinate system is at the center of the world volume.
- Position of a track is given with respect to the global coordinate system.
G4VPhysicalVolume

- **G4PVPlacement**  1 Placement = One Volume
  - A volume instance positioned once in a mother volume

- **G4PVPParameterised**  1 Parameterised = Many Volumes
  - Parameterised by the copy number
    - Shape, size, material, position and rotation can be parameterised, by implementing a concrete class of G4VPVPParameterisation.
  - Reduction of memory consumption
    - Currently: parameterisation can be used only for volumes that either a) have no further daughters or b) are identical in size & shape.

- **G4PVReplica**  1 Replica = Many Volumes
  - Slicing a volume into smaller pieces (if it has a symmetry)
Physical Volumes

- **Placement**: it is one positioned volume
- **Repeated**: a volume placed many times
  - can represent any number of volumes
  - reduces use of memory.
  - Replica
    - simple repetition, similar to G3 divisions
  - Parameterised

- A **mother** volume can contain either
  - many **placement** volumes **OR**
  - one **repeated** volume
**G4PVPlacement**

\[\text{G4PVPlacement}(\text{G4RotationMatrix}* \ p\text{Rot}, \ // \ rotation \ of \ mother \ frame} \\\n\text{const \ G4ThreeVector& \ tlate, \ // \ position \ in \ rotated \ frame} \\\n\text{G4LogicalVolume* \ pCurrentLogical,} \\\n\text{const \ G4String& \ pName,} \\\n\text{G4LogicalVolume* \ pMotherLogical,} \\\n\text{G4bool \ pMany,} \ // \ not \ used. \ Set \ it \ to \ false... \\\n\text{G4int \ pCopyNo,} \ // \ unique \ arbitrary \ index \\\n\text{G4bool \ pSurfChk=false);} \ // \ optional \ overlap \ check

- Single volume positioned relatively to the mother volume
  - In a frame rotated and translated relative to the coordinate system of the mother volume

- Three additional constructors:
  - A simple variation: specifying the mother volume as a pointer to its physical volume instead of its logical volume.
  - Using G4Transform3D to represent the direct rotation and translation of the solid instead of the frame (alternative ctor)
  - The combination of the two variants above
G4PVPlacement
Rotation of mother frame ...

G4PVPlacement(G4RotationMatrix* pRot,      // rotation of mother frame
               const G4ThreeVector& tlate,  // position in mother frame
               G4LogicalVolume* pCurrentLogical,
               const G4String& pName,
               G4LogicalVolume* pMotherLogical,
               G4bool pMany,           // not used. Set it to false...
               G4int pCopyNo,         // unique arbitrary index
               G4bool pSurfChk=false ); // optional overlap check

- Single volume positioned relatively to the mother volume

Mother volume
G4PVPlacement

Rotation in mother frame ...

G4PVPlacement( G4Transform3D( G4RotationMatrix &pRot,  // rotation of daughter frame
                     const G4ThreeVector &tlate),  // position in mother frame
        G4LogicalVolume *pDaughterLogical,
        const G4String &pName,
        G4LogicalVolume *pMotherLogical,
        G4bool pMany,       // not used, set it to false...
        G4int pCopyNo,      // unique arbitrary integer
        G4bool pSurfChk=false ); // optional overlap check

Mother volume
Parameterised Physical Volumes

- User written functions define:
  - the size of the solid (dimensions)
    - Function ComputeDimensions(…)
  - where it is positioned (transformation)
    - Function ComputeTransformations(…)

- Optional:
  - the type of the solid
    - Function ComputeSolid(…)
  - the material
    - Function ComputeMaterial(…)

- Limitations:
  - Applies to a limited set of solids
  - Daughter volumes allowed only for special cases

- Very powerful
  - Consider parameterised volumes as “leaf” volumes
Uses of Parameterised Volumes

- Complex detectors
  - with large repetition of volumes
    - regular or irregular

- Medical applications
  - the material in animal tissue is measured
    - cubes with varying material
G4PVPVParameterised

G4PVPVParameterised(const G4String& pName, 
    G4LogicalVolume* pCurrentLogical, 
    G4LogicalVolume* pMotherLogical, 
    const EAxis pAxis, 
    const G4int nReplicas, 
    G4VPVParameterisation* pParam, 
    G4bool pSurfChk=false);

- Replicates the volume \( n_{\text{Replicas}} \) times using the parameterisation \( p_{\text{Param}} \), within the mother volume
- The positioning of the replicas is dominant along the specified Cartesian axis
  - If \( k_{\text{Undefined}} \) is specified as axis, 3D voxelisation for optimisation of the geometry is adopted
- Represents many touchable detector elements differing in their positioning and dimensions. Both are calculated by means of a G4VPVParameterisation object
- Alternative constructor using pointer to physical volume for the mother
G4VSolid* solidChamber = new G4Box("chamber", 100*cm, 100*cm, 10*cm);
G4LogicalVolume* logicChamber =
    new G4LogicalVolume(solidChamber, ChamberMater, "Chamber", 0, 0, 0);
G4double firstPosition = -trackerSize + 0.5*ChamberWidth;
G4double firstLength = fTrackerLength/10;
G4double lastLength  = fTrackerLength;
G4VPVParameterisation* chamberParam =
    new ChamberParameterisation( NbOfChambers, firstPosition,
                                ChamberSpacing, ChamberWidth,
                                firstLength, lastLength);
G4VPhysicalVolume* physChamber =
    new G4PVParameterised( "Chamber", logicChamber, logicTracker,
                           kZAxis, NbOfChambers, chamberParam);

Use **kUndefined** for activating 3D voxelisation for optimisation
class ChamberParameterisation : public G4VPVParameterisation
{
  public:

    ChamberParameterisation( G4int NoChambers, G4double startZ,
                             G4double spacing, G4double widthChamber,
                             G4double lenInitial, G4double lenFinal );

    ~ChamberParameterisation();

    void ComputeTransformation (const G4int copyNo, G4VPhysicalVolume* physVol) const;

    void ComputeDimensions (G4Box& trackerLayer, const G4int copyNo, const G4VPhysicalVolume* physVol) const;

};
void ChamberParameterisation::ComputeTransformation
(const G4int copyNo, G4VPhysicalVolume* physVol) const
{
    G4double Zposition = fStartZ + (copyNo+1) * fSpacing;
    G4ThreeVector origin(0, 0, Zposition);
    physVol->SetTranslation(origin);
    physVol->SetRotation(0);
}

void ChamberParameterisation::ComputeDimensions
(G4Box& trackerChamber, const G4int copyNo,
const G4VPhysicalVolume* physVol) const
{
    G4double halfLength = fHalfLengthFirst + copyNo * fHalfLengthIncr;
    trackerChamber.SetXHalfLength(halfLength);
    trackerChamber.SetYHalfLength(halfLength);
    trackerChamber.SetZHalfLength(fHalfWidth);
}
Replicated Physical Volumes

- The mother volume is sliced into replicas, all of the same size and dimensions.
- Represents many touchable detector elements differing only in their positioning.
- Replication may occur along:
  - Cartesian axes (X, Y, Z) – slices are considered perpendicular to the axis of replication
    - Coordinate system at the center of each replica
  - Radial axis (Rho) – cons/tubs sections centered on the origin and un-rotated
    - Coordinate system same as the mother
  - Phi axis (Phi) – phi sections or wedges, of cons/tubs form
    - Coordinate system rotated such as that the X axis bisects the angle made by each wedge
G4PVReplica

G4PVReplica(const G4String& pName,
G4LogicalVolume* pCurrentLogical,
G4LogicalVolume* pMotherLogical,
const EAxis pAxis,
const G4int nReplicas,
const G4double width,
const G4double offset=0);

- Alternative constructor:
  - Using pointer to physical volume for the mother
- An offset can be associated
  - Only to a mother offset along the axis of replication
- Features and restrictions:
  - Replicas can be placed inside other replicas
  - Normal placement volumes can be placed inside replicas, assuming no intersection/overlaps with the mother volume or with other replicas
  - No volume can be placed inside a radial replication
  - Parameterised volumes cannot be placed inside a replica
Cartesian axes - kXaxis, kYaxis, kZaxis
- Offset shall not be used
- Center of n-th daughter is given as
  \[-width \times (n\text{Replicas}-1) \times 0.5 + n \times width\]

Radial axis - kRaxis
- Center of n-th daughter is given as
  \[width \times (n+0.5) + \text{offset}\]

Phi axis - kPhi
- Center of n-th daughter is given as
  \[width \times (n+0.5) + \text{offset}\]
Replication example

```cpp
G4double tube_dPhi = 2.* M_PI * rad;
G4VSolid* tube =
    new G4Tubs("tube",20*cm,50*cm,30*cm,0.,tube_dPhi);
G4LogicalVolume * tube_log =
    new G4LogicalVolume(tube, Air, "tubeL", 0, 0, 0);
G4VPhysicalVolume* tube_phys =
    new G4PVPlacement(0,G4ThreeVector(-200.*cm,0.,0.),
                       "tubeP", tube_log, world_phys, false, 0);
G4double divided_tube_dPhi = tube_dPhi/6.;
G4VSolid* div_tube =
    new G4Tubs("div_tube", 20*cm, 50*cm, 30*cm,
                -divided_tube_dPhi/2., divided_tube_dPhi);
G4LogicalVolume* div_tube_log =
    new G4LogicalVolume(div_tube,Pb,"div_tubeL",0,0,0);
G4VPhysicalVolume* div_tube_phys =
    new G4PVReplica("div_tube_phys", div_tube_log,
                    tube_log, kPhi, 6, divided_tube_dPhi);
```
Divided Physical Volumes

- Implemented as “special” kind of parameterised volumes
  - Applies to CSG-like solids only (box, tubs, cons, para, trd, polycone, polyhedra)
  - Divides a volume in identical copies along one of its axis (copies are not strictly identical)
    - e.g. - a tube divided along its radial axis
    - Offsets can be specified
- The possible axes of division vary according to the supported solid type
- Represents many touchable detector elements differing only in their positioning
- G4PVDivision is the class defining the division
  - The parameterisation is calculated automatically using the values provided in input
Divided Volumes - 2

- **G4PVDivision** is a special kind of parameterised volume
  - The parameterisation is automatically generated according to the parameters given in **G4PVDivision**.
- Divided volumes are similar to replicas but ...
  - Allowing for gaps in between mother and daughter volumes
    - Planning to allow also gaps between daughters and gaps on side walls
  - Shape of all daughter volumes must be same shape as the mother volume
    - Solid (to be assigned to the daughter logical volume) must be the same type, but different object.
- Replication must be aligned along one axis
- If no gaps in the geometry, **G4PVReplica** is recommended
  - For identical geometry, navigation in pure replicas is faster
G4PVDivision(const G4String& pName, 
        G4LogicalVolume* pDaughterLogical, 
        G4LogicalVolume* pMotherLogical, 
        const EAxis pAxis, 
        const G4int nDivisions, // number of division is given 
        const G4double offset);

- The size (width) of the daughter volume is calculated as 
  ( (size of mother) - offset ) / nDivisions
G4PVDivision(const G4String& pName,
    G4LogicalVolume* pDaughterLogical,
    G4LogicalVolume* pMotherLogical,
    const EAxis pAxis,
    const G4double width,  // width of daughter volume is given
    const G4double offset);

- The number of daughter volumes is calculated as
  \[ \text{int}(\ (\text{size of mother}) - \text{offset} ) / \text{width} ) \]
- As many daughters as width and offset allow
Divided Volumes - 5

G4PVDivision(const G4String& pName,
              G4LogicalVolume* pDaughterLogical,
              G4LogicalVolume* pMotherLogical,
              const EAxis pAxis,
              const G4int nDivisions,  // both number of divisions
              const G4int width,      // and width are given
              const G4double offset);
Divisions are allowed for the following shapes / axes:

- **G4Box**: kXAxis, kYAxis, kZAxis
- **G4Tubs**: kRho, kPhi, kZAxis
- **G4Cons**: kRho, kPhi, kZAxis
- **G4Trd**: kXAxis, kYAxis, kZAxis
- **G4Para**: kXAxis, kYAxis, kZAxis
- **G4Polycone**: kRho, kPhi, kZAxis
- **G4Polyhedra**: kRho, kPhi, kZAxis
  - kPhi - the number of divisions has to be the same as solid sides, (i.e. numSides), the width will not be taken into account

In the case of division along kRho of G4Cons, G4Polycone, G4Polyhedra, if width is provided, it is taken as the width at the -z radius; the width at other radii will be scaled to this one
PART II

Describing a detector - III

- Solids & Touchables
Abstract class. All solids in Geant4 derive from it

Defines but does not implement all functions required to:
- compute distances to/from the shape
- check whether a point is inside the shape
- compute the extent of the shape
- compute the surface normal to the shape at a given point

Once constructed, each solid is automatically registered in a specific solid store
**Solids**

- **Solids defined in Geant4:**
  - CSG (Constructed Solid Geometry) solids
    - G4Box, G4Tubs, G4Cons, G4Trd, ...
    - Analogous to simple GEANT3 CSG solids
  - Specific solids (CSG like)
    - G4Polycone, G4Polyhedra, G4Hype, ...
    - G4TwistedTubs, G4TwistedTrap, ...
  - BREP (Boundary REPresented) solids
    - G4BREPSolidPolycone, G4BSplineSurface, ...
    - Any order surface
  - Boolean solids
    - G4UnionSolid, G4SubtractionSolid, ...
CSG: G4Tubs, G4Cons

```cpp
G4Tubs(const G4String& pname, // name
        G4double pRmin, // inner radius
        G4double pRmax, // outer radius
        G4double pDz,   // Z half length
        G4double pSphi, // starting Phi
        G4double pDphi); // segment angle

G4Cons(const G4String& pname, // name
        G4double pRmin1, // inner radius -pDz
        G4double pRmax1, // outer radius -pDz
        G4double pRmin2, // inner radius +pDz
        G4double pRmax2, // outer radius +pDz
        G4double pDz,   // Z half length
        G4double pSphi, // starting Phi
        G4double pDphi); // segment angle
```

Detector Description: Basics - Geant4 Course
Specific CSG Solids: G4Polycone

```cpp
G4Polycone(const G4String& pName,
           G4double phiStart,
           G4double phiTotal,
           G4int numRZ,
           const G4double r[],
           const G4double z[]);
```

- **numRZ** - numbers of corners in the \( r,z \) space
- **\( r, z \)** - coordinates of corners
- Additional constructor using planes
BREP Solids

BREP = Boundary REPresented Solid

Listing all its surfaces specifies a solid
- e.g. 6 squares for a cube

Surfaces can be
- planar, 2\textsuperscript{nd} or higher order
  - elementary BREPS
- Splines, B-Splines,
  NURBS (Non-Uniform B-Splines)
  - advanced BREPS

Few elementary BREPS pre-defined
- box, cons, tubs, sphere, torus, polycone, polyhedra

Advanced BREPS built through CAD systems
BREPS example: G4BREPSolidPolyhedra

G4BREPSolidPolyhedra(const G4String& pName,
                      G4double phiStart,
                      G4double phiTotal,
                      G4int sides,
                      G4int nZplanes,
                      G4double zStart,
                      const G4double zval[],
                      const G4double rmin[],
                      const G4double rmax[]);

- **sides** - numbers of sides of each polygon in the \( x-y \) plane
- **nZplanes** - numbers of planes perpendicular to the \( z \) axis
- **zval[]** - \( z \) coordinates of each plane
- **rmin[], rmax[]** - Radii of inner and outer polygon at each plane
Solids can be combined using boolean operations:

- **G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid**
- Requires: 2 solids, 1 boolean operation, and an (optional) transformation for the 2\textsuperscript{nd} solid
  - 2\textsuperscript{nd} solid is positioned relative to the coordinate system of the 1\textsuperscript{st} solid
  - Component solids must **not** be disjoint and must **well intersect**

```
G4Box box("Box", 20, 30, 40);
G4Tubs cylinder("Cylinder", 0, 50, 50, 0, 2*M_PI); // r: 0 -> 50
       // z: -50 -> 50
       // phi: 0 -> 2 pi
G4UnionSolid union("Box+Cylinder", &box, &cylinder);
G4IntersectionSolid intersect("Box Intersect Cylinder", &box, &cylinder);
G4SubtractionSolid subtract("Box-Cylinder", &box, &cylinder);
```

- Solids can be either CSG or other Boolean solids
- **Note**: tracking cost for the navigation in a complex Boolean solid is proportional to the number of constituent solids
Areas, volumes and masses

- Surface area and geometrical volume of a generic solid or Boolean composition can be computed from the `solid`:
  
  ```cpp
  G4double GetSurfaceArea();
  G4double GetCubicVolume();
  ```

- Overall mass of a geometry setup (sub-detector) can be computed from the `logical volume`:
  
  ```cpp
  G4double GetMass(G4Bool forced=false,
  G4Material* parameterisedMaterial=0);
  ```
How to identify a volume uniquely?

- Suppose a geometry is made of sensitive layers C which are placed in a volume B.

  - Volume B is a daughter volume of a divided volume A.

- Volume A has 24 positions in the world.

  - While in the 'logical' geometry tree, volume C is represented by just one physical volume, in the real world there are many C 'volumes'.

  - *How can we then identify these volumes C?*
A touchable for a volume serves the purpose of providing a unique identification for a detector element.

It is a geometrical entity (volume or solid) which has a unique placement in a detector description.

It can be uniquely identified by providing the copy numbers for all daughters in the geometry hierarchy.

In our case these are:
- CopyNo of C in B: 1
- CopyNo of B in A: 1, 2, 3
- CopyNo of A in the world: 1, .., 24

Example of touchable identification:
- A.3/B.2/C.1

Detector Description: Basics - Geant4 Course
Suppose a calorimeter is made of 4x5 cells
  - and is implemented by two levels of replica
In reality, there is only one physical volume object for each level. Its position is parameterized by its copy number
How to get the copy number of each level, when a step belongs to two cells?
  - Remember: geometrical information in G4Track is identical to "PostStepPoint". You cannot get the correct copy number for "PreStepPoint" if you directly access to the physical volume
  - Use touchable to get the proper copy number, transformation matrix,...
What a touchable provides?

- **G4VTouchable** - a base class for all touchable implementations – defines the following 'requests' (methods) which all touchable have to respond, where **depth** means always the number of levels up in the tree to be considered:
  - depth = 0: the bottom level (volume C in B)
  - depth = 1: the level of its mother volume (volume B in A)
  - depth = 2: the grandmother volume (volume A in world)
- **GetCopyNumber**(G4int depth = 0)
  - returns the copy number of the given level
- **GetTranslation**(G4int depth = 0); **GetRotation**(G4int depth=0)
  - return the components of the volume's transformation
- **GetSolid**(G4int depth =0)
  - returns the solid
- **GetVolume**(G4int depth =0)
  - returns the physical volume
Benefits of Touchables in track

- Full geometrical information available
  - to processes, to sensitive detectors, to hits
- All the geometrical information of the particular step should be taken from “PreStepPoint”
  - Available in G4TouchableHistory object
    - Copy-number, transformations
  - Accessible via Handles (or smart-pointers) to touchables
- Note: the geometrical information associated with G4Track is basically the same as “PostStepPoint”
Detector Description: Basics - Geant4 Course

### Touchable & Steps

- **G4Step** has two **G4StepPoint** objects as its starting and ending points. All the geometrical information of the particular step should be got from “**PreStepPoint**”
  - Geometrical information associated with **G4Track** is basically same as “**PostStepPoint**”
- Each **G4StepPoint** object provides:
  - position in world coordinate system
  - global and local time
  - Material
  - Associated **G4TouchableHistory**
    - Touchable to be used for geometrical information
    - Copy-number, transformations
- **Handles** (or **smart-pointers**) to touchables are intrinsically used. Touchables are reference counted
How to use Touchables ...

- **G4TouchableHistory** has information of the geometrical hierarchy at the point

```cpp
G4Step* aStep = ..;
G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
G4TouchableHandle theTouchable =
    preStepPoint->GetTouchableHandle();
G4int copyNo = theTouchable->GetReplicaNumber();
G4int motherCopyNo = theTouchable->GetReplicaNumber(1);
G4ThreeVector worldPos = preStepPoint->GetPosition();
G4ThreeVector localPos = theTouchable->GetHistory()->
    GetTopTransform().TransformPoint(worldPos);
```
PART II

GDML

• Importing and exporting detector descriptions
GDML components

- GDML (Geometry Description Markup Language) is defined through XML Schema (XSD)
  - XSD = XML based alternative to Document Type Definition (DTD)
  - defines document structure and the list of legal elements
  - XSD are in XML -> they are extensible

- GDML can be written by hand or generated automatically in Geant4
  - 'GDML writer' allows exporting a GDML file
  - GDML needs a “reader”, integrated in Geant4
  - 'GDML reader' imports and creates 'in-memory' the representation of the geometry description

Detector Description: Basics - Geant4 Course
GDML – Geant4 binding

- XML schema available from [http://cern.ch/gdml](http://cern.ch/gdml)
  - Also available within Geant4 distribution
    - See in `geant4/source/persistency/gdml/schema/`
  - Latest schema release GDML_3_0_0 (as from 9.2 release)
- Requires XercesC++ XML parser
  - Available from: [http://xerces.apache.org/xerces-c](http://xerces.apache.org/xerces-c)
  - Tested with versions 2.8.0 and 3.0.1
- Optional package to be linked against during build
  - `G4LIB_BUILD_GDML` and `XERCESCROOT` variables
  - Examples available: `geant4/examples/extended/persistency/gdml`
CMS detector through GDML

~19000 physical volumes

Geant4 CMS geometry imported in Root through GDML
LHCb detector through GDML

~5000 physical volumes

Geant4 LHCb geometry imported in Root through GDML
Using GDML in Geant4

to write:

```
#include "G4GDMLParser.hh"
G4GDMLParser parser;
parsed.Write("g4test.gdml", pWorld, true, "path_to_schema/gdml.xsd");
```

- instantiate GDML parser
- Concatenate or not pointers to entity names
- pass the 'top' volume to the writer
- Activate or de-activate schema validation

```
pWorld = GDMLProcessor::GetInstance()->GetWorldVolume();
```

- get pointer to 'top' world volume

to read:

```
parsed.Read( "g4test.gdml", true );
```

GDML examples in: `geant4/examples/extended/persistency/gdml`

Detector Description: Basics - Geant4 Course
Any geometry tree can be dumped to file
- ... just provide its physical volume pointer (pVol):
  ```csharp
  parser.Write("g4test.gdml", pVol);
  ```
A geometry setup can be split in modules
- ... starting from a geometry tree specified by a physical volume:
  ```csharp
  parser.AddModule(pVol);
  ```
- ... indicating the depth from which starting to modularize:
  ```csharp
  parser.AddModule(depth);
  ```
Provides facility for importing CAD geometries generated through STEP-Tools
Allows for easy extensions of the GDML schema and treatment of auxiliary information associated to volumes
Full coverage of materials, solids, volumes and simple language constructs (variables, loops, etc...)
Importing CAD geometries with GDML

- CAD geometries generated through STEP-Tools (stFile.geom, stFile.tree files) can be imported through the GDML reader:
  - parser.ParseST("stFile", WorldMaterial, GeomMaterial);

- Tool like FastRad allow for importing CAD STEP files and directly convert to GDML
GDML processing performance

- GDML reader/writer tested on
  - complete LHCb and CMS geometries
  - parts of ATLAS geometry
    • full ATLAS geometry includes custom solids
- for LHCb geometry (~5000 logical volumes)
  - writing out ~10 seconds (on P4 2.4GHz)
  - reading in ~ 5 seconds
  - file size ~2.7 Mb (~40k lines)
- for CMS geometry (~19000 logical volumes)
  - writing out ~30 seconds
  - reading in ~15 seconds
  - file size ~7.9 Mb (~120k lines)
Linear Collider Detector Description (LCDD) extends GDML with Geant4-specific information (sensitive detectors, physics cuts, etc).

GDML/LCDD is generic and flexible:
- several different full detector design concepts, including SiD, GLD, and LDC, where simulated using the same application.
GDML as primary geometry source

- Space Research @ ESA
  - Geant4 geometry models
    - component degradation studies (JWST, ConeXpress,...)
    - GRAS (Geant4 Radiation Analysis for Space)
  - enables flexible geometry configuration and changes
  - main candidate for CAD to Geant4 exchange format

Detector Description: Basics - Geant4 Course 59
GDML as primary geometry source

- Anthropomorphic Phantom
  - Modeling of the human body and anatomy for radioprotection studies
  - no hard-coded geometry, flexible configuration