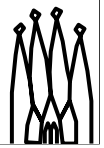
1

Detector Description Overview

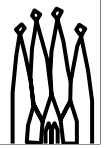


Schedule:	Timing	Topic
	20 minutes	Lecture
	0 minutes	Practice
	20 minutes	Total

Goals

Overview of Detector Description in Gaudi

- What we understand as “Detector Description”
- Understanding the Transient view
- Understanding the Persistent view
- Role of Conditions Database



Goals

The goals of this first lesson is to offer an overview to the detector description facilities existing in the Gaudi framework before diving into the various parts that will be covered in the other lessons in this tutorial.

Detector Description

Logical Structure

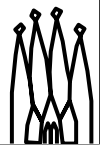
- **Breakdown of detectors**
- **Identification**

Geometry Structure

- **Hierarchy of geometrical volumes**
- **LogicalVolumes (unplaced dimensioned shape)**
- **PhysicalVolumes (placed volume)**

Other detector data

- **Calibration, Alignment, Readout maps, Slow control, etc.**



Detector Description

The detector description database should include the physical and a logical description of the detector. The physical description, in particular the **geometry description** covers dimensions, shape and material of the different types of elements from which the detector is constructed.

The **logical description** provides two main functions. The first is a simplified access to particular parts of a physical detector description. This could be a hierarchical description where a given detector setup is composed of various sub-detectors, each of which is made up of a number stations, modules or layers, etc. and there would be a simple way for a client to use this description to navigate to the information of interest. The second function of the logical description is to provide a means of detector element identification. This allows for different sets of information which are correlated to specific detector elements to be correctly associated with each other

In a detector description, the definition of the detector elements and of the data associated to their physical description may vary over time, for instance due to real or hypothetical changes to the detector. Each such change should be recorded as a different version of the detector element. Additionally, it should be possible to capture, for an entire description, a version of each of the elements and to associate a name to that set. This is similar to the way CVS allows one to tag a set of files so that one does not need to know the independent version numbers for each file in the set.

The current implementation of detector description includes only the logical description of the detector, its geometry and the description of the required materials. We are actively working in incorporating the so called **Conditions Database**, which will include the rest of the time varying detector information (calibration, alignment, slow control, etc.).

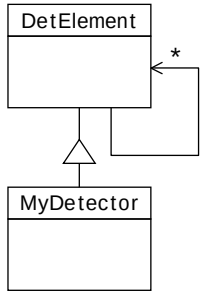
Logical Structure

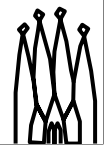
The basic object is a Detector Element

- **Identification**
- **Navigation (tree-like)**

DetElement as information center

- **Be able to answer any detector related question**
  - **E.g. global position of strip#, temperature of detector, absolute channel gain, etc.**
- **Placeholder for specific code**
  - **The specific answers will be coded by "you" ➔ Lesson 3**



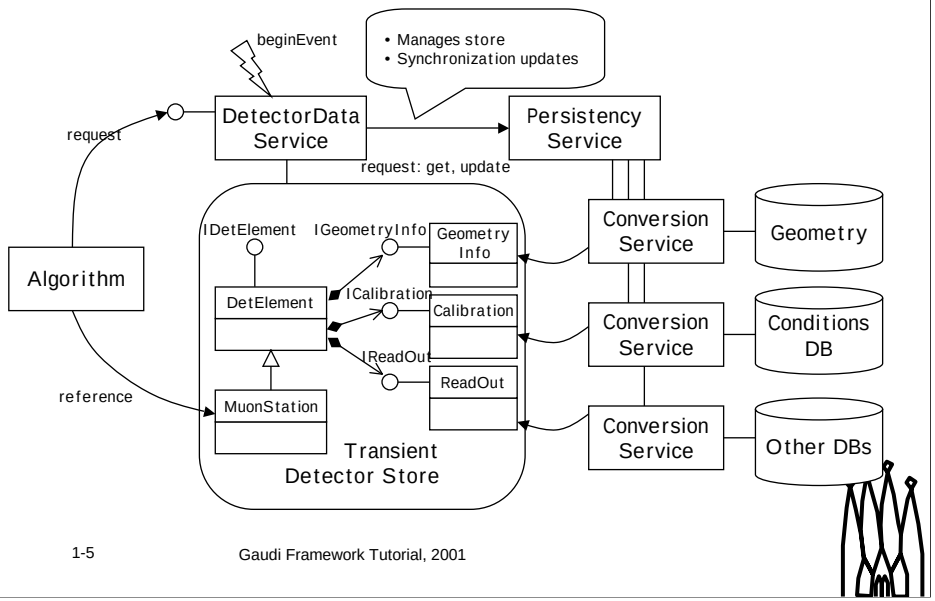


Logical Structure

The central entity used to describe the logical structure is the **DetectorElement**. It represents any detector element from the complete detector, sub-detector, station to a any module or chamber. It main role is two-fold: identification and navigability, and as an information concentrator to any kind of detector information (geometry, alignment, calibrarion, slow control, etc,).

In addition, this is the class the sub-detector developers will have to extern to add specific code to answer specific questions. For examples are: what is the position of a detector channel given its strip number, what is the corrected gain of a calorimeter cell, etc.

Algorithm Accessing Detector Data



Accessing Detector Data

An algorithm that needs to access a given detector part uses the detector data service to locate the relevant **DetectorElement**. This operation can be generally done during the initialization phase of the **algorithm**. Contrary to the Event Data, the Detector Data store is not cleared for each event and the references to detector elements remain valid and are updated automatically during the execution of the program.

Main Features

•The persistent representation of the detector data, in particular the detector description (logical structure and geometry) is different than the transient representation. This is XML files in our current solution.

•The DetectorElement will be (not yet implemented) updated with up to date information when the event being processed will have a time stamp outside the validity range.

Algorithm Accessing Detector Data

```
// Algorithm code fragment (initialize() or execute())

SmartDataPtr<MyDetElement> mydet(detSvc(),
                                 "Structure/LHCb/MyDet");
if( !mydet ) {
  log << MSG::ERROR << "Can't retrieve MyDet" << endmsg;
  return StatusCode::FAILURE;
}
...
// get the number of sub-DetectorElements
ndet = mydet->childIDetectorElements().size()
// get the material
material = mydet->geometry()->lvolume()->materialName();
```


Accessing detector description

Similarly to the event data, accessing detector data is done using the DetectorDataSvc (detSvc()) and with the help of a *SmartDataPtr()*. What is obtained is a pointer to a DetectorElement element, which is then used for obtained the required information.

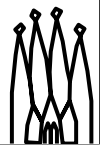
Geometry Information

Constructed using Logical Volumes and Physical Volumes (Geant 4)

- **Logical Volume: Unplaced detector described as a solid of a given material and a set of daughters (physical volumes).**
- **Physical Volume: Placement of a logical volume (rotation & translation).**

Solids

- **A number of basic shapes (boxes, tubes, cones, trds, spheres,…) with dimensions**
- **Boolean solids (unions, intersections and subtractions)**



Geometry Information

The geometry information is build using a Logical and Physical Volumes. The names of these objects comes from the Geant4 nomenclature.

•**Logical Volume**: It is an unplaced dimensioned volume of a given shape and a given material. It is also the system of reference where the sub-detector elements (daughters) will be placed.

•**Physical Volume**: It is the placement of a daughter logical volume into the mother logical volume. It is constituted of a reference to a logical volume and its transformation (rotation and translation) with respect to the mother logical volume.

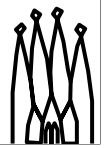
The shape of a logical volume can be constructed using basic shapes or Boolean combination of these with transformations.

Geometry Information (2)

```
IGeometryInfo* geom = mydetelem->geometry();
```

IGeometryInfo

```
HepTransform3D& matrix()       // To Local
HepTransform3D& matrixInv()    // To Global
HepPoint3D toLocal( HepPoint3D& )
HepPoint3D toGlobal( HepPoint3D& )
bool isInside( HepPoint3D& )
string belongsToPath( HepPoint3D& )
IGeometryInfo* belongsTo( HepPoint3D& )
...
fullGeoInfoForPoint( HepPoint3D&, ...)
string lVolumeName()
ILVolume* lvolume() ...
```

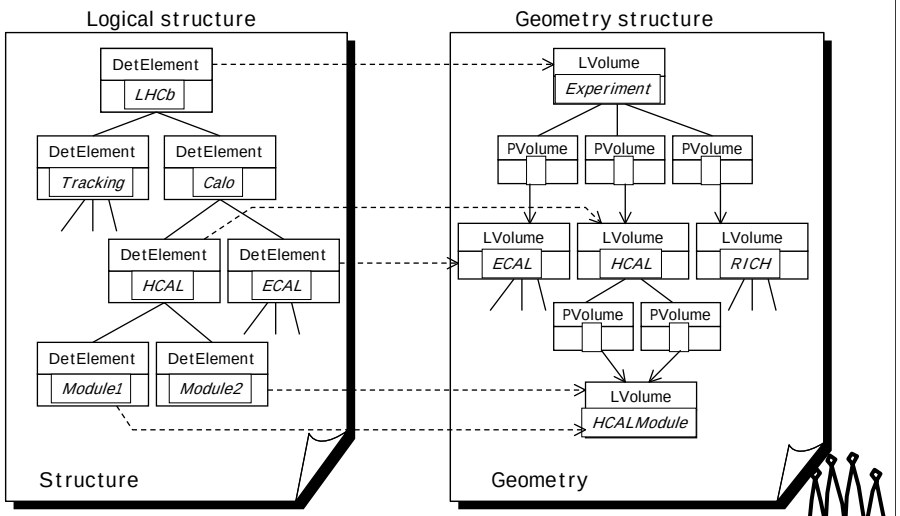



Geometry Information

The abstract interface **IGeometryInfo** returned by the method geometry() provides the basic geometry information for a given **DetectorElement**. The basic functionality are transformations from the local system of reference to the global one and vice versa. There are also useful functions to indicate if a given 3D point belongs to a given detector element or to find the complete list of volume hierarchy for a given 3D point.

In this slide is shown a incompete list of the available methods. Please refer to the reference guide for a complete one: http://cern.ch/LHCbSoft/LHCb/v8/doc/html/class_igeometryinfo.html

Two Hierarchies



The Logical Structure and the Geometry Structure hierarchies

The logical structure and geometry structure are both hierarchically organized. The Logical structure is tree-like structure of **DetectorElements**. The geometry structure is not a true tree-like, it is an acyclic graph of **LVolumes** and **PVolumes.** The global placement of a given LVolume is calculated following a given the path from the root LVolume. The main differences from both structures are:

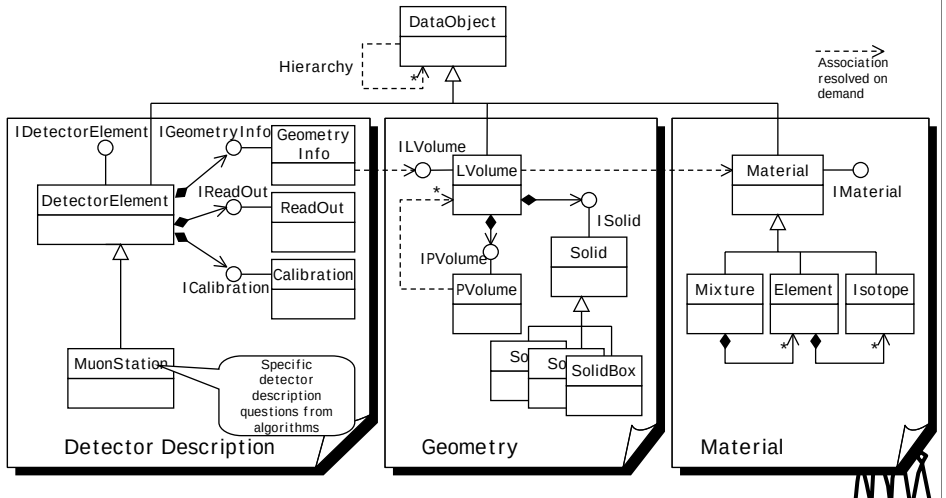
- The logical structure typically stops earlier than the geometry one. As soon the need for identification is not needed we can stop the structure. The geometry one will be as detailed as it is needed for the simulation (Geant4).
- The levels in both structures do not need to coincide. Extra levels may be added in the geometry one if this helps for optimizing the simulation. Or vise versa extra levels can be added in the logical description if this helps identification and navigability.
- Each detector element in the experiment will have a corresponding instantiated object in the transient store. On the contrary, only one logical volume will be instantiated for all volume placements of the same volume type.

Connection between both hierarchies

Having two independent hierarchies requires to have connections between them and keep them in synch. The connection is done from the logical structure to the geometry structure. The ingredients are:

- The **logical volume** the detector element is kind of (type). Different detector elements may point to the same logical volume.
- To place the detector element in the global system of reference we need to specify the overall **support** (detector element) and the **path** from the associated support logical volume to the logical volume this detector element is kind of.

Class Diagram (Simplified)



Class Diagram

This shows a simplified class diagram of the main classes involved in the transient representation of the detector description. All the main objects involved inherit from the base class **DataObject**, which allows them to be registered and retrieved to and from the detector data store using a name. The dotted lines are “links” between objects which are resolved only on demand. This means that in the transient detector store we only load what is needed at time a link is followed.

General features of the geometry tree

•The geometry tree is constructed from Logical Volumes and Physical Volumes.

•There are no "up-links" in the geometry tree. It means that each node has no information about the "up" (or "parent", "mother") node.

•Each Logical Volume has a information about its "down" ("children") nodes, represented by Physical Volumes

•Each Logical Volume has information about its shape and dimensions ("Solid") and material content

•Neither Logical Volumes nor Physical Volumes have any information about their absolute position in the space.

•Each Physical Volume has a information about its position inside the mother ("parent") Logical Volume. This is the only geometry information available in the whole tree.

•Boolean operations should be performed at the level of Solids.

Some consequences are:

•The top-level Logical Volume (presumably the experimental hall, or cave, or the whole LHCb detector) defines the absolute coordinate reference system. In other words, the null-point (0,0,0) in the so called Global Reference System is just the center of the top Logical Volume.

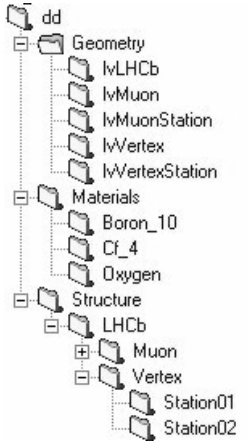
•All geometry calculations, computations, inputs and outputs, performed with the usage of a Logical Volume are in the local reference system of this Logical Volume.

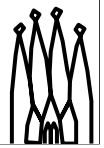
•All geometry calculations, computations, inputs and outputs, performed with the usage of a Physical Volume are in the local reference system of its parent Logical Volume.

Transient Store Organization

Standard Gaudi Transient Store

- **"Catalogs" of Logical Volumes and Materials**
- **"Structure" as a tree**
- **All elements identified with names of the form: /xxx/yyy/zzzz**





Transient Store Organization

The detector description **DataObjects** have a name and are organized in the transient store as a Unix file-system.

•**DetectorElement**. The name structure of the detector elements follow the logical structure of the detector (detector, sub-detector, subsub-detectors, …)

•**LVolumes**. The logical volumes has a unique name and are organized in "catalogs" for convenience. The organization of these catalogs do not need to reflect the geometry tree (it cannot in general) but it is convenience that we organize the logical volumes by sub-detector. Physical volumes are not directly identifiable. Their identification is done through the logical volume that contains the physical volumes (placements).

•**Material**. The Materials (Isotopes, Element, Mixtures) are also organized in catalogs. The main catalog is used for "standard" materials. Other catalogs can be used for specific materials required by sub-detectors.

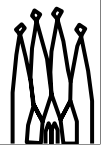
Persistency based on XML files

XML is used as persistent representation of the Structure, Geometry and Materials

➔ Lesson 2

Why XML?

- **Instead of inventing our own format use a standard one (extendible)**
- **Many available Parsers and Tools**
- **Strategic technology**



Persistency

The current persistency for the logical and geometrical information is based on text files formatted as XML. In the long term we envisage to use the Conditions database also to store the geometry and logical structure taking advantage of the time dependency and versioning available. In any case, the formatting of the geometry in the conditions DB can be continued to be XML formatted strings.

XML

XML (eXtendable Markup Language) is a standard language which allows the definition of custom tags, unlike the fixed set of tags of HTML used for WWW. XML files are understandable by humans as well as computers. Data in XML are self-descriptive so that by looking at the XML data one can easily guess what the data mean. Unlike the HTML tags, tags in XML do not define how to render or visualize the data. This is left to an application which understands the data and can visualize them if wanted. An advantage of XML is that there exists plenty of software which can be used for parsing and analysing, as it is an industry standard.

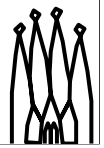
Conditions DB

Detector conditions data (calibration, slow control, alignment, etc.) are characterized by:

- **Time validity period**
- **Version**

The conditions data objects will also appear in the Detector Transient Store

The persistency of conditions data is done with the Conditions DB (IT product)



Conditions DB

The Conditions DB will be used to store "detector conditions" that are time dependent and versioned. Examples of detector conditions are: calibration constants, alignment constants, slow control parameters, etc.

The Conditions database is implemented using a DBMS (the current implementation is based on Objectivity) with a standard interface. The conditions objects will appear in the transient detector store as any other object. The sub-detector teams will define the contents and structure of the conditions data. The Gaudi framework will take care of of the synchronization of the conditions data with the time of the event being processed off-loading the sub-detector algorithm code of that task.

Condition Data Object

"Block" of data belonging to some detector element

- **E.g. channel thresholds for module 7 of ECAL**

Time (CondDBKey) validity range

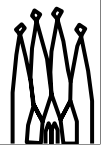
- **[since, till)**
- **CondDBKey is a 64 bit integer number. Sufficient flexibility (absolute time in ns, run number, etc.)**

Version

- **Sequence version number**

Extra information

- **Textual description, insertion time, etc.**



Condition Data Objects

Each condition data item is identified by the name of the detector element it is associated to (e.g. "/LHCb/Calo/Ecal/Module1") and to the type or nature of the data (e.g. "calib", "temperatures", etc.). The combination of both names uniquely identifies each data item.

Each condition data item has a time validity range, with a start time and an end time. Time can be expressed as real time (preferred), or run/event/beam crossing number.

In general, data items can have several versions valid at a given time (exceptions could perhaps be slow control monitoring data). Each version be identified by a version number or local tag. The default version is not necessarily the most recently added version. The most recent version could be identified by a logical tag like "head version" or similar.

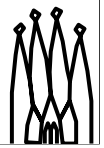
It must be possible to tag a given configuration of the whole database. With this "global tag" we should be able to select the correct version of each data item valid at each time.

Extending Detector Element

The natural place to add specific code to answer specific detector questions is in a concrete detector element

- **The answer is workout using all available detector information (geometry, alignment, calibration, parameters, etc.)**
- **Code is reused by several Algorithms**

➔ Lesson 4



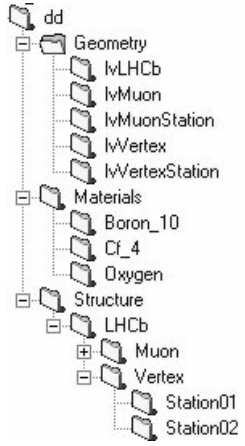
Extending the Detector Element

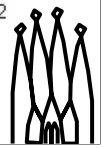
The place foreseen for add sub-detector code to answer detector description specific questions to reconstruction or analysis algorithms is an extension of the Detector Element base class. Each, sub-detector can extern the base class adding specific functionality. Typically a sub-detector can add this functionally at different levels in the hierarchy of the logical structure. For example. Questions related to the sub-detector as a whole will be implemented at the level of the sub-detector detector element, while questions to individual modules can be asked to the detector element representing those modules.

Adding More Information

The Detector Data Store may contain any other detector information

- **Any DataObject can be registered on the store**
- **Useful to not repeat many times the same parameters to DetectorElements**





Other Information

As the detector transient store is a normal Gaudi store, any DataObject can be registered on it and made available to any algorithm or detector element. Today, the information available is quite limited (structure, geometry and materials) but it is foreseen that other information will added (for example conditions data).

Is is quite usual that many parameters are same among different detector elements. In this case makes sense to define new objects of the type “detector element pattern” that can be referenced by detector elements. These new objects can also be made persistent using the same mechanisms (XML files) or can be generated at run-time based from a set of primordial parameters.

Summary

Today Detector Information consists of

- **Logical Structure (DetectorElements)**
- **Geometry (LVolume, Solids, etc.)**
- **Materials (Isotope, Mixtures, etc.)**

On the way of adding Detector Conditions

- **Conditions Data (Sub-detector specific)**

Other Information could be added if required