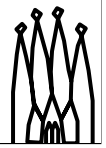
9

LHCb Detector Description

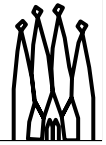


Schedule:	Timing	Topic
	20 minutes	Lecture
	0 minutes	Practice
	20 minutes	Total

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 Architecture

Sub-Architecture of Gaudi

- **Same principles**
- **Transient/Persistent representations**

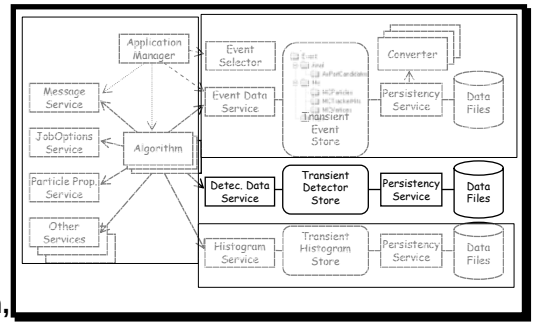
Focus on the "Physics Algorithm"

- **Access to Detector Transient Store**

Coherent access to "all" detector data

- **Geometry, Calibration, Slow Control, etc.**

Gaudi Architecture



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

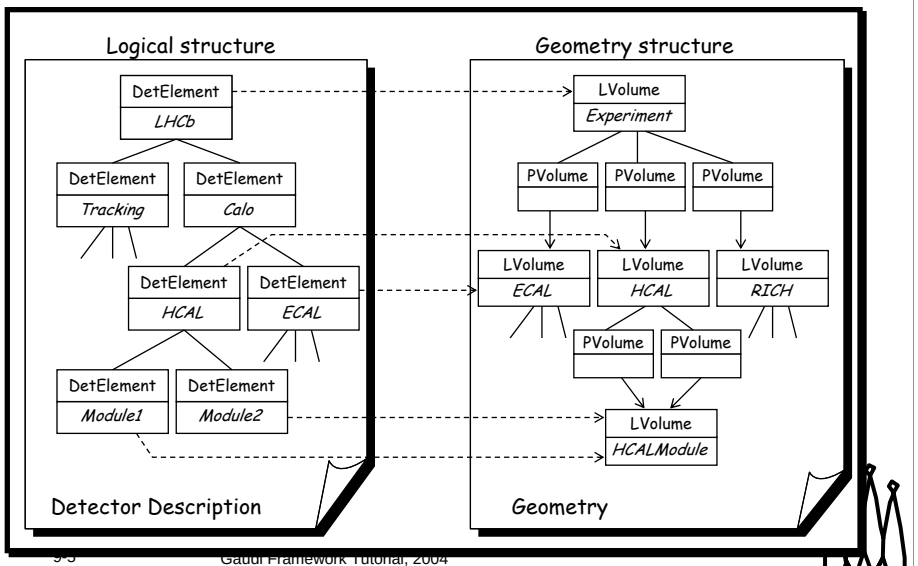
The detector description database should include the physical and a logical description of the detector. The physical description, in particular the **geometry description** covers dimensions, shape and material of the different types of elements from which the detector is constructed.

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.).

Two Hierarchies



Logical Structure

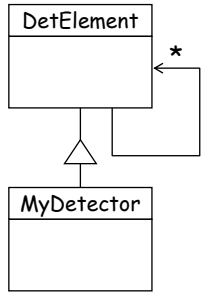
The basic object is a *Detector Element*

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

DetectorElement 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 can be coded by "Physicists"**

DetectorElement objects are shared by all Algorithms

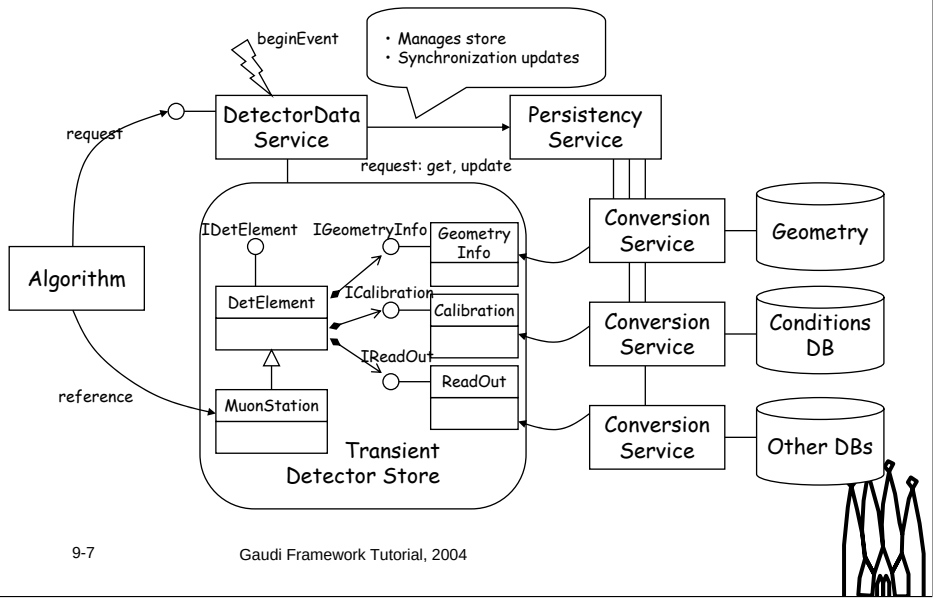


Logical Structure

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.

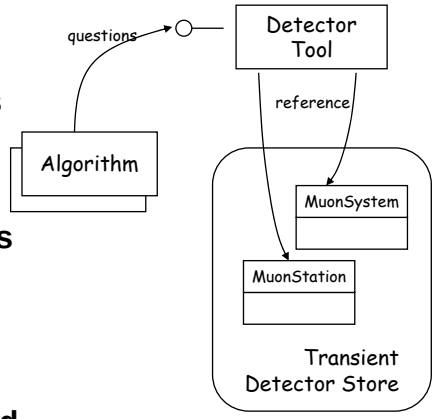
Detector Tools

Detector Tool to encapsulate the "code" to answer detector questions

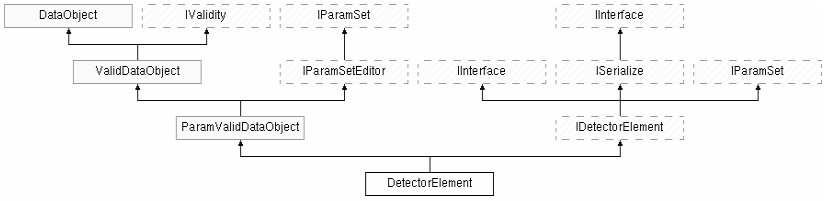
- **Keeping brainless detector elements**
- **Shared by all algorithms**
- **Caching answers**

Problems

- **Update answers when detector data invalidated**

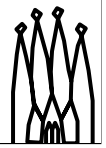


Detector Element Class



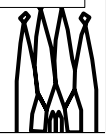
Three basic functionalities:

- **IDetectorElement: Access to other Detector information**
- **IValidity: Time validity interval management**
- **IParamSet: User parameters (key-value pairs)**



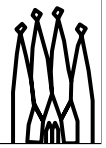
IDetectorElement

```
virtual const std::string &  name () const=0
virtual const IGeometryInfo *  geometry ()const =0
virtual const IAlignment *  alignment () const =0
virtual const ICalibration *  calibration ()const = 0
virtual const IReadOut *  readOut () const =0
virtual const ISlowControl *  slowControl () const =0
virtual const IFastControl *  fastControl () const =0

virtual IDetectorElement *  parentIDetectorElement () const=0
virtual IDetectorElement::IDEContainer &  childIDetectorElements ()
const=0
virtual IDetectorElement::IDEContainer::iterator  childBegin ()=0
virtual IDetectorElement::IDEContainer::iterator  childEnd ()=0

virtual std::ostream &  printOut (std::ostream &) const=0
virtual IDetectorElement *  reset ()=0
virtual StatusCode  initialize ()=0
```



IValidity

```
virtual bool  isValid ()=0
virtual bool  isValid (const ITime &)=0
virtual const ITime &  validSince ()=0
virtual const ITime &  validTill ()=0
virtual void  setValidity (const ITime &, const ITime
&)=0
virtual void  setValiditySince (const ITime &)=0
virtual void  setValidityTill (const ITime &)=0
virtual StatusCode  updateValidity ()=0
```



IParamSet

```
virtual IParamSet::Kind  paramKind (std::string name)=0
virtual std::string  paramType (std::string name)=0
virtual std::string  paramComment (std::string name)=0
virtual std::string  paramAsString (std::string name)=0
virtual int  paramAsInt (std::string name)=0
virtual double  paramAsDouble (std::string name)=0
virtual double  param (std::string name)=0
virtual IParamSet::Kind  paramVectorKind (std::string name)=0
virtual std::string  paramVectorType (std::string name)=0
virtual std::string  paramVectorComment (std::string name)=0
virtual std::vector<std::string>  paramVectorAsString (std::string name
virtual std::vector<int>  paramVectorAsInt (std::string name)=0
virtual std::vector<double>  paramVectorAsDouble (std::string name)=0
virtual std::vector<double>  paramVector (std::string name)=0
virtual std::vector<std::string>  params ()=0
virtual std::vector<std::string>  paramVectors ()=0
virtual std::string  printParams ()=0
virtual std::string  printParamVectors ()=0
```

Algorithm Accessing Detector Data

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

MyDetElement* mydet = getDet("Structure/LHCb/MyDet");

...

// 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 (Geant4-like)

- **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.

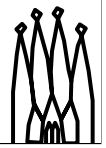
The shape of a logical volume can be constructed using basic shapes or Boolean combination of these with transformations.

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

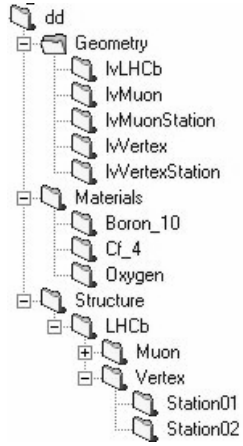
The abstract interface **IGeometryInfo** returned by the method geometry() provides the basic geometry information for a given **DetectorElement**. The basic functionality are transformations from the local system of reference to the global one and vice versa. There are also useful functions to indicate 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.

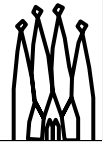
In this slide is shown a incompete list of the available methods. Please refer to the reference guide for a complete one: http://cern.ch/LHCbSoft/LHCb/v8/doc/html/class_igeometryinfo.html

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).

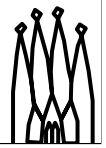
•**Material**. The Materials (Isotopes, Element, Mixtures) are also organized in catalogs. The main catalog is used for "standard" materials. Other catalogs can be used for specific materials required by sub-detectors.

Persistency based on XML files

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

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

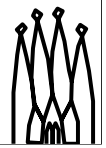
XML (eXtendable Markup Language) is a standard language which allows the definition of custom tags, unlike the fixed set of tags of HTML used for WWW. XML files are understandable by humans as well as computers. Data in XML are self-descriptive so that by looking at the XML data one can easily guess 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.

The LHCb Detector XML/DTD

Divided into 3 main parts

- **structure**
- **geometry**
- **material**

External DTDs, to be referenced in every LHCb XML files



The LHCb DTDs

There are actually three main DTDs in LHCb, each of them allowing to describe a particular type of information :

- structure.dtd : allows to describe the structure of the detector. This is mainly a way of describing a tree of detector elements.
- geometry.dtd : this is were the actual geometry is described. Each detector element of the structure part references one of the geometries described here.
- material.dtd : this is the definition of the materials used in LHCb. They can then be referenced from the geometry.

Some specificities

Expressions evaluator – units & functions known

```
12.2*mm + .17*m / tan (34*degree)
```

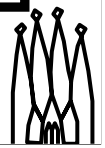
parameter : a kind of macro

```
<parameter name="InCell" value="40.6667*mm"/>
<parameter name="MidCell" value="1.5*InCell"/>
```

References : element + “ref”

```
<detelemref href="LHCb/structure.xml#LHCb"/>
```

protocol://hostname/path/file.xml#ObjectID



The LHCb DTD specificities

There are some specificities in the LHCb DTD, mainly three of them.

Every numerical value required by an attribute or an element in any of the DTDs of LHCb is an expression. This means that the value will be evaluated by the numerical expression parser. Thus, most of the current units and constants are already known. You can safely use degree, rad or pi for instance. On top of that, many mathematical functions are also known, such as sin, or exp but also arctan and many others.

A special element called parameter is defined in all DTDs from LHCb. This element allows the user to define his own parameters that can be then reused in any expression or value in the rest of the XML code. It has a name, a type, a value and a comment. The parameter element is actually a kind of macro since it will be replaced by its value everywhere it appears at parsing time. As for macros, the scope of a parameter is really uneasy to define. It is define everywhere “after” its definition. The problem is that this “after” highly depends on the way you read the XML. The basic rule is that you should always define a parameter in a place that is above every node that will use it.

Some nodes have names finishing by "ref". Each time there is a corresponding node without the ref. The "ref" nodes are actually kinds of hyperlinks on the "without ref" ones. The hyperlink is in general specified using the format : protocol://hostname/path/to/the/file.xml#ObjectID. The protocol and hostname parts can be omitted if the file resides on the local host. It is possible to write a hyperlink without the full path name in case one needs to refer to an XML object residing inside another file. In this case the relative path will be appended to the location of the currently parsed XML file. For example having the current file location /full/path/to/current.xml and inside this file a hyperlink as href="next/file.xml#NextOID" the hyperlink will be resolved as /full/path/to/next/file.xml#NextOID. If the hyperlink has the form #ObjectID this means that the referred object is located in the same file. Note that relative paths are strongly encouraged for every file except the top most one, since the whole sets of file may be copied in several different locations one day.

Structure Elements

DDDB : the root

catalog : a list

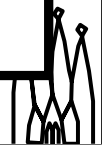
detelem : a detector element

geometryInfo : connection to the geometry

userParameter(Vector) : hook for adding parameters

specific : hook for extending the DTD

```
<DDDB>
 <catalog name="…">
  <detelem name="…">
   <geometryinfo
        lvname="…"
        npath="…"
        support="…"/>
   <userParameter
        comment="…"
        name="…"
        type="string">
    …
   </userParameter>
   <specific>
    …
   </specific>
  </detelem>
 </catalog>
</DDDB>
```



The LHCb DTD for structure

Here is a list of the main elements defined in the LHCb DTD for describing the structure of the detector :

• **DDDB** : this is to fulfill the XML basic rule that each XML document must have only one root XML element This is the root element.

• **catalog** : this is simply a list of elements, with a given name. This is a way to classify detector elements.

• **detelem** : detector elements are the essential part of the structure of the detector description. They fully describe a given part of the detector by holding data on the geometry of this part as well as on the subparts constituting it.

• **userParameter** and **userParameterVector** : this allows the user to add a parameter or a vector of parameters for a given detector element. This is intended to be used for specific parameters appearing in the subdetectors' descriptions. Their usage is described deeper in the next talk.

• **geometryInfo** : this element describes the geometry of a given detector element (logical volume, support and path from the support to the geometry).

• **specific** : this is the place where a user can extend the default detector description language and introduce new elements for his own needs. It's foreseen that the new XML elements be defined in a local DTD section of the XML data file or in a specific DTD file. Its usage is described deeper in the next next talk.

Geometry Elements (1)

DDDB : the root

catalog : a list

logvol : logical volume

physvol : physical volume

paramphysvol(**2D**)(**3D**) : replication of physical volumes

tabproperty : tabulated properties

```
<DDDB>
 <catalog name="…">
  <logvol material="…"
          name="…">
   <physvol logvol="…"
            name="…"/>
  </logvol>
  <logvol name="…">
   <paramphysvol number="5">
    <physvol logvol="…"
             name="…"/>
    <posXYZ z="20*cm"/>
   </paramphysvol>
  </logvol>
 </catalog>
</DDDB>
```

The LHCb DTD for geometry

Here is a first list of the main elements defined in the LHCb DTD for describing the geometry of the detector :

- **DDDB** : this is to fulfill the XML basic rule that each XML document must have only one root XML element This is the root element.
- **catalog** : this is simply a list of elements, with a given name. This is a way to classify the logical volumes.
- **logvol** : this defines a logical volume. See lesson 1.
- **physvol** : this defines a logical volume. See lesson 1.
- **paramphysvo**l, **paramphysvol2D**, **paramphysvol3D** : these are ways to define many physical volumes in one shot, by replicating a given volume and applying a given transformation between each replica.
- **tabproperty** : this defines a tabulated property. This is used to describe optical properties of materials and surfaces. See *"Optical properties & Surfaces*" available at *http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/Documents/Optical.pdf.*

Geometry Elements(2)

posXYZ, **posRPhiZ**, **posRThPhi** : translations

rotXYZ, **rotAxis** : rotations

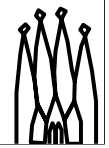
transformation : composition of transformations

box, **trd**, **trap**, **cons**, **tub**, **sphere**

union, **intersection**, **subtraction** : boolean solids

surface

```
<subtraction name="sub2">
 <box name="box3"
      sizeX="1*m"
      sizeY="1*m"
      sizeZ="15*cm"/>
 <tubs name="tub2"
       outerRadius="15*cm"
       sizeZ="25*cm"/>
</subtraction>
<posXYZ z="-40*cm"/>
<rotXYZ rotX="90*degree"/>
```



The LHCb DTD for geometry

Here is a second list of the main elements defined in the LHCb DTD for describing the geometry of the detector :

- **posXYZ** , **posRPhiZ** , **posRThPhi** : these are 3 ways of defining a translation : cartesian, cylindrical and spherical coordinate systems.
- **rotXYZ** , **rotAxis** : these are two ways of defining a rotation. Either along X, Y or Z axis, or along a user-defined axis.
- **transformation** : this defines a new transformation, by composition of several others
- **box**, **trd**, **trap**, **cons**, **tub**, **sphere** : these are all kinds of solids, namely boxes, simple trapezoids, general trapezoids, conic sections, tube sections and sphere. Please report to *"The LHCb Detector Description DTD*" available at *http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/Documents/lhcbdtd.pdf* for further details.
- **union**, **subtraction**, **intersection** : these are boolean operations on solids.
- **surface** : this defines a surface.

Material Elements

materials : the root

catalog : a list

tabproperty : tabulated properties

atom

isotope

element : a mixture of isotopes

material : mixtures of elements or materials

```
<isotope A="11*g/mole“
         name="Bor_11“ …/>
<element name="Boron“
         symbol="B“ …>
  <isotoperef href="#Bor_10“
        fractionmass="0.20"/>
  <isotoperef href="#Bor_11“
        fractionmass="0.80"/>
</element>
<element name="Oxygen“
         symbol="O“ …>
  <atom A="16*g/mole“
        Zeff="8.0000"/>
</element>
<material name="CO2“ …>
  <component name="Carbon“
             natoms="1"/>
  <component name="Oxygen“
             natoms="2"/>
</material>
```

The LHCb DTD for material

Here is a list of the main elements defined in the LHCb DTD for describing the materials of the detector :

• **materials** : this is to fulfill the XML basic rule that each XML document must have only the one root XML element This is the root element.

• **catalog** : this is simply a list of elements, with a given name. This is a way to classify materials.

• **tabproperty** : this defines a tabulated property. This is used to describe optical properties of materials and surfaces. See *"Optical properties & Surfaces*" available at *http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/Documents/Optical.pdf.*

• **atom** : this is used to define an element when it is not a mixture of isotopes.

• **isotope** : this is the definition of a given isotope of a given atom.

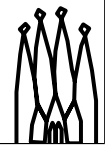
• **element** : this is a real life element, that is in general a mixture of several isotopes with given proportions.

• **component** : this is used to define mixtures. It associates a material and a proportion.

• **material** : this defines a mixture of several elements or even several other mixtures with given proportions.

Specializing Detector Elements

1. **Adding userParameter(vector)s to default DetectorElements**
2. **Extending and specializing the DetectorElement class in C++, using userParameters in XML**
3. **Extending XML DTD and writing a dedicated converter**



Specializing Detector Elements

There are mainly three ways of specializing detector elements.

- the first and less complicated one is to add userParameters to the detector element in the XML code. This will be detailed in the end of this lesson.
- the second is to extend and specialize the DetectorElement object in C++. This allows to add new members and methods. The initialization of this new object uses then the userParameters defined previously. This will be detailed in the next lesson.
- the last and most complicated way is to extend the XML DTD to allow specific XML elements and store complex information. This will need to write a dedicated converter and will be detailed in the next lesson.

Specializing by using UserParameter

Two elements :

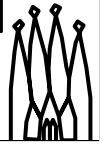
`<userParameter> and`
`<userParameterVector>`

3 string attributes : name, type and comment

One value given as text

```
<userParameter
     comment="blablabla"
     name="description"
     type="string">
  Calibration channels
</userParameter>
```

```
<userParameterVector
     name="NbChannels"
     type="int"
     comment="blabla">
   530  230
   570 270
</userParameterVector>
```

Specializing by using userParameter and userParameterVector

These are two elements of the LHCb structure DTD that allow the user to add his own parameters to a given detector element. These elements have three attributes defining the parameter :

•name : this will be the only way to retrieve the parameter in C++

•type : this has no restriction but only int, double and string are recognized. All other types are treated as strings.

•comment : you are free to put here a small explanation of the meaning and usage of the parameter

The value of the parameter is the value of the element itself, which is the text appearing between the opening and the closing tag. In the case of a vector, the different values should be separated by spaces and/or carriage return only. If the type of a parameter is int or double, the value will be computed using the expression evaluator of the XmlCnvSvc. Thus parameters (I mean the one defined via the element <parameter>, not user parameters), units and mathematical functions can be safely used.

Extending Detector Elements

Free extension of the DetectorElement class

Specific initialization using initialize()

- called after conversion
- access to userParameters

A converter is needed but very simple (4 lines)

```
#include "DetDesc/XmlUserDetElemCnv.h"
#include "MyDetElem.h"

static CnvFactory
  <XmlUserDetElemCnv<MyDetElem> > s_factory;
const ICnvFactory& XmlMyDetElemCnvFactory = s_factory;
```

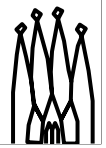

Full Customization

extension of the DTD to define new XML elements

parsing of the new XML code using the xerces parser

"real" converters to initialize C++ objects according to XML

→ Not recommended



Full Customization

Up to know, we learnt how to use userParameters and how to extend the DetectorElement object. This already allows some customization but does not allow a real extension of the default schema in the sense that you have no way to add data to the XML by using new XML elements that were not defined in the LHCb DTD.

This possibility exists but it requires some work :

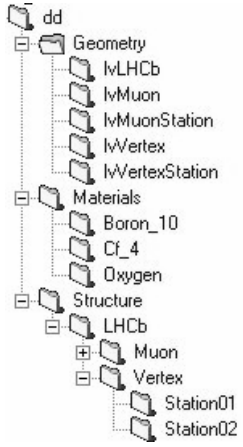
- one should first extend the LHCb DTD to define correctly the new elements
- then this extended DTD should be parsed to retrieved the data. This is done by using a free XML parser called xerces
- at last, specialized converters are needed to deal with the new data and store them into dedicated DetectorElements

We will now detail every step.

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

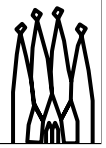
As the detector transient store is a normal Gaudi store, any DataObject can be registered on it and made available to any algorithm or detector element. Today, the information available is quite limited (structure, geometry and materials) but it is foreseen that other information will added (for example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

Det/XmlDDDB Package

Package containing all the detector description XML files (structure, geometry and materials)

- **Organized in one directory per sub-detector**
- **CVS module (versioned)**
- **Released as any other software package**
- **Managed by M. Cattaneo**



XmlEditor

Explorer-like XML viewer

No need to know XML syntax

Checks the DTD when opening a file

Allows copy, paste and drag and drop of nodes

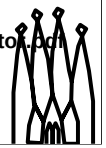
Allows view of several files at the same time

Hide references across files

⟹ **Easy XML edition**

$LHCBSOFT/Det/XmlEditor/v*/scripts/xmlEditor(.bat)

http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/Documents/XmlEditor.pdf



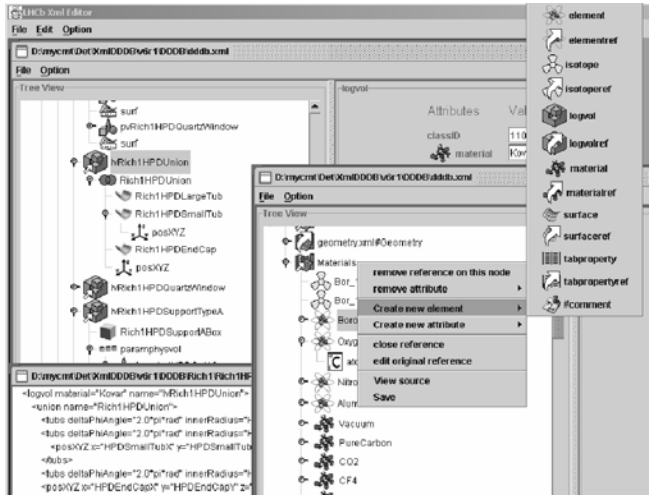
XmlEditor

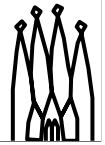
The XML editor is a tool provided to edit the XML files of the detector description database without having to learn the XML syntax.

To start XmlEditor, just type $LHCBSOFT/Det/XmlEditor/v*/scripts/xmlEditor in a UNIX shell or %LHCBSOFT%\Det\XmlEditor\v*\scripts\xmlEditor.bat at the dos prompt. v* is the number of the version you want to use (v4r1 is the latest one at this time).

A documentation dedicated to the editor is available at http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/Documents/XmlEditor.pdf.

XmlEditor screenshot





XmlEditor screenshot

You can see on this picture the different elements of the XmlEditor GUI. A typical window is divided into two parts. On the left hand side, the tree of elements is displayed. Each element has an icon, the ones with a big arrow are references, the one with a small arrow are open references. You can right click on an element to get a contextual popup menu and perform some actions (create/remove elements/attributes, edit/close reference, view XML source…). On the right hand side are displayed the attributes of the selected node.

Several window can be open at the same time, even if they reuse the same file. Drag and drop or cut and paste across windows is allowed.

Panoramix: Geometry Viewer

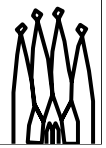
Events and Geometry viewer

Takes the LHCb specificities into account

- **references**
- **logical volumes hierarchy**
- **subDetectors**

Interactive navigation inside the geometry hierarchy

→ **Do not develop geometry without it**



Panoramix

Panoramix is the new package for visualization of both the detector geometry and the events. In our case, the event display is not available to avoid waiting too much at launch time (due to Sicb).

To start panoramix, just type :

$LHCBSOFT/Vis/Panoramix/v*/scripts/panoramix where V* is the version number under Linux

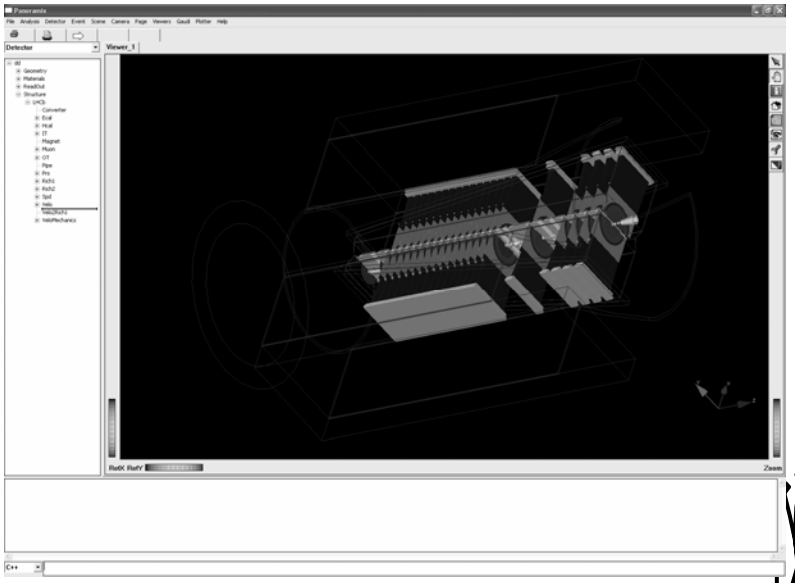
%LHCBSOFT%\Vis\Panoramix\v*\scripts\panoramix.bat where V* is the version number under Windows

or double click on the icon if you run a file manager.

Note that these scripts make take –noevent as argument. In this case, panoramix starts faster but your are not able to display events.

A documentation dedicated to panoramix is available at http://www.lal.in2p3.fr/SI/Panoramix/tutorial/tutorial.html.

Panoramix screenshot



Panoramix screenshot

You can see on this picture the different elements of the XmlEditor GUI. A tool bar on the right is missing and we won't use here the tree structure on the left. The center window displays the geometry, with different colors for different sub detectors.

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 (LCG project)

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 Database Requirements

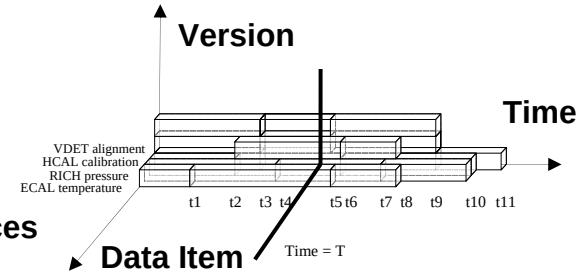
Storage/Retrieval of time depend data items

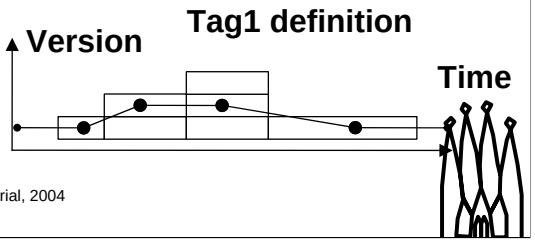
Versioning

Tagging

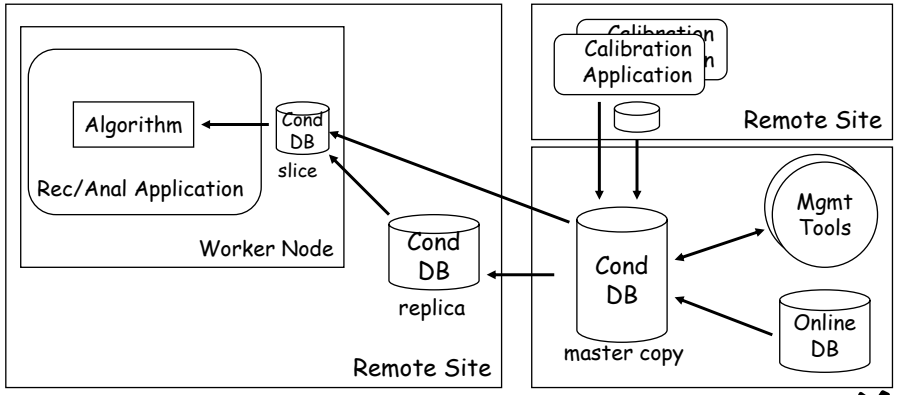
Ability to extract slices of data

Not intrusive and as much as possible transparent for the algorithms





The Big Picture



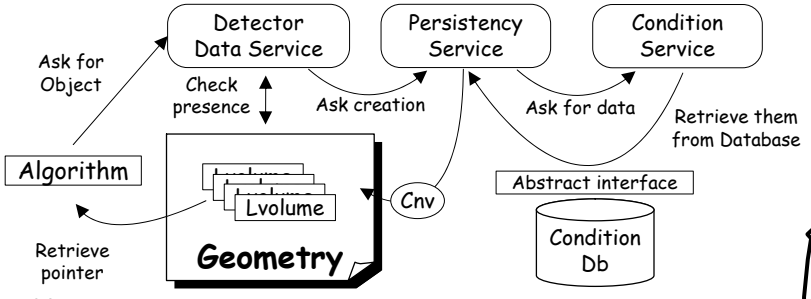
Gaudi Interface to Conditions Db

Emphasis on the data retrieval functionality

One new service was defined : *ConditionSvc*

Independent from data content, only deals with data retrieval depending on time, version and/or tag

Fully transparent for the user



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.

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