Table of Contents

Introduction

This page refers to the geometry and alignment section of the LHCb detector description framework. If you are not familiar with the Gaudi detector description framework read about it in the Detector Description chapter of the Gaudi user guide. The use case can be summarised as follows:

- Provide users of the LHCb detector data service with geometrical information obtained from two places:
  - **Nominal** or *ideal* positions of all the sub-components of LHCb
  - **Deviations** from the nominal positions. Thought of as *deltas*, *misalignemts* or *off-nominal* deviations to the above. These should be obtainable from the Conditions Database, and could be modified and propagated at runtime.
- Provide a mechanism for users to modify the off-nominal deviations for a given sub-component in a coherent, consistent way. Users should be able to change the alignment of a sub-detector and see the changes automatically propagated to all the dependent sub-components.
- Provide a mechanism to write a new set of deviations to the Conditions Database.

Design

The introduction of an **off-nominal** alignment to the LHCb detector description is accomplished by mirroring the **nominal** positioning of detector sub-components in the geometry description, which is in turn achieved through a hierarchical tree-like structure of DetectorElements, as described in the Detector Description chapter of the Gaudi manual, or in this LHCb contribution to the LHC Alignment Workshop, 2006. This mirroring is achieved by giving each node in the DetectorElement hierarchy the necessary *off-nominal* alignment information, which simply is a correction to its nominal position within the structure. This, coupled with the links between DetectorElement and higher and lower members of the hierarchy, allows for a coherent propagation of the off-nominal deviations from one DetectorElement to those which depend on it (in practice, those which are positioned within it). At all times it is possible to get the *nominal* geometry information, be it in a local frame or in the global reference frame. We thus have two geometries within the DetectorElement hierarchy tree. These can be thought of as

- **Nominal** geometry: Defined by the positioning information in the Geometry DB (which package is this steered from these days?), it should reflect either the design positioning of the elements of LHCb, or the best knowledge of their position, following survey measurements. At the moment of writing it is still not clear which of the two shall define the nominal geometry. An important fact is that this informaiton is loaded by the Gaudi framework at initialisation, and cannot be changed consistently after that. Therefore this geometry is to be considered as fixed in the course of a software job, and shall be refered to in what follows as **static**.
- **Off-nominal geometry**: Describes the corrections or deviations from the nominal geometry, both locally at the level of each node in the hierarchy tree, and globally. Combined with the nominal geometry, it defines the local and global position of each DetectorElement in the hierarchy. The corrections or deviations are 3D transformations that can be modified during the course of a job. This can be done manually by direct manipulation in software, or automatically if the data being analised has been associated to different alignment constants. All the changes are propagated throughout the hierarchy automatically. Since the off-nominal geometry can be modified at will, it shall be referred to in what follows as **dynamic**.

Implementation

The basic principle behind the implementation of the alignment functionality is that off-nominal information is handled, together with nominal geometrical information, in an implementation of the IDetectorElement interface, via an implementaiton of the IGeometryInfo interface. In practice this means that any LHCb sub-component that needs to use this functionality must be a DetectorElement. Furthermore, its IGeometryInfo must be initialised with an address in the data service which points to an AlignmentCondition, be it in a local file or in the conditions database. The functionality is handled in the Det/DetDesc, Det/DetDescCnv and Det/DescSvc packages for LHCb versions higher or equal to v18r8. Here are a few key points in getting this functionality to work:

The AlignmentCondition class

AlignmentCondition is a very simple class which essentially contains a 3D transformation object and the nine double precision parameters required to generate one following an LHCb convention, and has awareness of time-validity. The LHCb convention to combine the parameters to get a 3D transformation is described below. It is possible to update the 3D transformation directly, or feed the AlignmentCondition new parameters. Internally, it either calculates the parameters by decomposing the new 3D transformation, or calculates the transformation from a new set of parameters. In this way, parameters and 3D transform are always synchronised. This is important since the conditions database stores an XML string which encodes the nine parameters, but all numerical calculations are performed using the transformation object. The XML string encoding the parameters is obtainable on-demand from the AlignmentCondition, and must always reflect the state of the 3D transformation. The calculation of the 3D transformatin, together with its decomposition into nine parameters, are implemented in the DetDescFunctions set of helper functions.

The LHCb recipe to create a 3D transformation

In the LHCb detector description framework, the convention for a 3D transformation is combine a 3D translation and a 3D rotation by applying the rotation first and the translation second (see the LHCb XML detector descripton DTD note. For the purposes of off-nominal corrections, the possibility to provide a pivot point for the rotation has been added. This is, at the time of writing, not accessible to the static geometry description. There are many ways to express a rotation in terms of a set of parameters (different 3D coordinate systems, different Euler angles representations, Quaternions, Axis angles) and the detector description geometry XML DTD allows for a number of these. These are being reviewed at the time of writing so I will swiftly skip on to what is currently essential in the context of off-nominal deviations. For a discussion of proposed changes and extensions to the DTD see slides 1 to 7 of this T-Rec talk. However, the off-nominal geometry is limited to a cartesian representation of a rotation. A dynamic geometry 3D transformation is defined by three sets of three double precision parameters, as can be seen here in the XML definition of an AlignmentCondition:

```
<condition classID="6" name="VeloSystem">
  <paramVector name="dPosXYZ" type="double">0*mm 0*mm 0*mm</paramVector>
  <paramVector name="dRotXYZ" type="double">0*rad 0*rad 0*rad</paramVector>
  <paramVector name="pivotXYZ" type="double">0*mm 0*mm 0*mm</paramVector>
</condition>
```

Each set represents a translation about cartesian axes, a rotation about cartesian axes, and a cartesian pivot point about which the rotation is performed. The rotation is described by the Euler 321 co-moving axes representation, that is, a rotation about the Z axis, followed by one about the Y' axis, followed by one about the X'' axis. The rotation is performed about the pivot point, and the translation is applied last. The calculation of the 3D transform is performed inside the AlignmentCondition implementation via a call to a publically available free function from DetDesc/3DTransformationFunctions.h:

```
const Gaudi::Transform3D localToGlobalTransformation(const std::vector<double>& translationParams
```

```
                                                const std::vector<double>& rotationParams,
                                                const std::vector<double>& pivotParams )
```

It is expected that the future changes to the DTD will do away with the use of simple but unsafe std::vectors of doubles and unify the description of 3D transformations for static and dynamic geometries.

IGeometryInfo implementation

This implementation, class GeometryInfoPlus, contains a pointer to an AlignmentCondition, which it obtains from the data service via an address passed to it upon construction. Each detector element contains one of these, and each one of these contains one, and only one, AlignmentCondition. The task of the GeometryInfo is to calculate the transformation matrices relating it to the global LHCb frame. For this it needs to navigate through the hierarchy of detector elements, picking up the nominal and off-nominal transformations of its parents, and calculating the total transformation matrix. It also must ensure that if the transformation in its own AlignmentCondition changes, the changes are propagated to its daughters, as their relation to the global frame depends on their parents. The new implementation also makes use of the CondDB data store interface, allowing for retrieval of AlignmentConditions from the conditions database if necessary, and of the UpdateManager, which establishes a network of dependencies between objects and conditions to allow for the coherent propagation of changes. The combined **nominal** and **off-nominal** transformation defines the local and global position of each IGeometryInfo, meaning that all **off-nominal** corrections are made accessible to the user automatically when they query the detector element. The IGeometryInfo also keeps the nominal information, such that, at each level in the hierarchy, it is possible to know where the nominal global position of a DetectorElement is.

Off-nominal geometry in the XML detector description

The LHCb XmlDDDB DTD allows an optional condition path in the definition of an IGeometryInfo. A specialised convertor in Det/DetDescCnv to construct an AlignmentCondition object when an XML string like the one used above to describe a 3D transformation is found by the parser. To enable off-nominal geometry in a detector element, it is necessary to give the detector element's geometryinfo definition a condition path, and modify the XML catalogue such that this path resolves to an XML string defining an AlignmentCondition.

This is an example of how to define a detector element with an AlignmentCondition path:

```
<detelem name="MUB_Up_Left_In_Plank13;">
  <author> Juan Palacios </author>
  <version> 1.0 </version>
  <geometryinfo lvname ="/dd/Geometry/Delphi/MUB/UpLeft/lvMUB_Inner_13;"
                support="/dd/Structure/Delphi/MUB/UpperLeft/Inner/Plank13"
                condition="/dd/Conditions/Alignment/MUB/UpperLeft/Inner/Plank13"
                npath  ="pvInnerPlank13"/>

</detelem>
```

For a more full and realistic example check the VELO description in versions of XmlDDDB v26r0 or higher, Velo/v200507, for a good example on how to do this. For instructions on how to write this information into a copty of CondDB, or how to create a local copy, bypass CondDB for specific conditions and more, see the Conditions database How-to

Accessing the information

This is done by default. GeometryInfoPlus uses the combined **nominal** and **off-nominal** transformation for all internal calculations. It is possible to access the nominal geometry, if necessary, via calls to the methods IGeometryInfo::toLocalMatrixNominal(), IGeometryInfo::toGlobalMatrixNominal(), etc. Note that in order to

obtain the full alignment information of a sub-component of LHCb, its detector element must be queried via the detector data service, since this triggers the initialization of the off-nominal transformations. For example, accessing /dd/Structure/LHCb/Velo will not trigger the creation of the off-nominal geometry for the daughters of the Velo system, but will initialise that of the LHCb system. This has implications for the simulation of misaligned and visualization of detectors, since typically Gauss and Panoramix load the top element, and this initialises the static geometry of all its daughters. Therefore, special care has to be taken in order to access the full (nominal + off-nominal) geometry in these domains.

Using misalignments in sub-detector descriptions

This geometry framework has been designed in order to make its incorporation into the detector descriptions of the LHCb subsystems and the use of misalignment information in all detector description client code as straight forward as possible. The use of the detector element guarantees that wherever the detector detcription data service is used, the misalignment information is automatically available. This has advantages, but also places constraints on the XML detector description. These are the requirements on a detector description:

- The structure hierarchy must be such that there is a detector element associated to each element that needs to have misalignment information. This means that the leaves of the structure tree should correspond at least to the smallest alignable object. The detector elements do not need to be specialised. It will suffice to have a default one, since all the functionality is handled via the IGeometryInfo held by the base class, which is constructed when a default detector element is defined in the XMLDDDB description.
- The IGeometryInfo in the detector element must have a path pointing to an AlignmentCondition in the data service. In practice, this is achieved through a correct XML definition of the detector element, as shown in the DELPHI example above.
- The caching of geometrical information that might depend on the dynamic geometry must be done in a controlled manner. Caching should be steered from a minumal number of methods, and each of these should be registered appropriately to the update manager service. This step requires careful attention and thought from the subsystem experts, as there is no automatic recipe to apply it. This is already implemented in the DetDesc classes, but specialized, derived classes should take care to implement it correctly. For an example look at the GeometryInfoPlus class source code.

This is all that's needed in order to incorporate the dynamic geometry functionality into the detector description of any given subsystem. This will allow users of the data service to move detectors, be it at initialisation via information stored in the conditions database, or in the course of a job via explicit modifications to the dynamic geometry information of differenc detector element objects. The framework takes care of coherently propagating the corrections such that all volumes dependent on the these also get moved.

Status (LHCb v18r8 or higher)

Current functionality

Detector elements can be freely moved around with respect to their nominal positions during the course of a job. The changes are propagated correctly via the UpdateManagerSvc, and can be visualised using the Gauss visualisation tools or Panoramix. AlignmentCondition objects can be created by hand, or existing ones can be modified. These can be turned into XML strings and stored in the conditions database. A full working example with the VELO has been developed. It is possible to simulate a misaligned VELO, but **NOT** to an arbitrary depth in the detector element hierarchy. It is only possible down to the module level (R-Phi sensor pair plus hybrid and support). There is at the moment no automatic way of accessing all the misalignment information up to the leaf level in the detector element hierarchy while guaranteeing that all the components which are not detector elements are also simulated.

Work in progress

- Simulating misaligned detectors: This is not as trivial as it seems. The problem is explained here.
- AlignmentCondition interface: Waiting for feedback. Does this simple class satisfy the requirements of users?
- Tools for AlignmentCondition creation. Would be nice to generate AlignmentConditions in a more general ways. Axes of rotation, pivot points, etc. Tools should allow the AlignmentCondition interface to remain largely unchanged.

Simulating misalignments

To simulate misalignments at the moment GaussEnv v20r1 or higher is needed. This uses LHCb v18r8, which has the required Det/!DetDesc functionality.

- Set the environment: GaussEnv v20r1 (or higher)
- Get the Det/XmlConditions package as described above
- From your CMT build directory: getpack Sim/Gauss v20r1 (or higher, must match GaussEnv)
- modify Sim/Gauss/v20r1/options/SimGeometry.opts:
    - comment out Geo.StreamItems += {"/dd/Structure/LHCb/Velo"};
    - uncomment #include "$GAUSSOPTS/SimVeloGeometry.opts"
- Modify $GAUSSROOT/cmt/requirements to use XmlDDDB v26r1 and your XmlConditions
- go to $GAUSSROOT/cmt and source setup.csh. Check that the $CONDITIONS_PATH points to your XmlCondiitons directory.
- Modify the alignment parameters in $XMLCONDITIONSROOT/DDDB/Local/Velo/alignment.xml

---

- JuanPalacios - 29 Jul 2005

Presentations

- New DetDesc features, LHCb software week, CERN, 01-05 October, 2007.
- Alignment constant and 3D transformations, Tracking & Alignment Workshop presentation, NIKHEF, 29-31 August, 2007.
- Alignment helper functions, T-Rec meeting, July 9th 2007.
- Proposed improvements to DetDesc, T-Rec meeting, July 9th 2007.
- LHC Alignment Workshop: LHCb detector description given at the LHC alignment workshop, Aug. 2006.
- CHEP '06 presentation: The LHCb Alignment framework.
- VELO meeting, 08-Jul-2005 (PDF): Alignment framework and VELO detector geometry status and consequences for VELO SW.
- LHCb SW week, 25-May-2005: Alignment framework concepts and status
- Orsay_19-05-2005.pdf: Orsay tracking workshop, 19-May-2005
- AlignmentStatus9-05-2005.pdf: LHCb T-Rec meeting, 09-May-2005
- Alignment25-04-2005.pdf: First talk to core SW group., 25/04/2005. Outlines problem, possible solutions.

-- JuanPalacios - 02 Oct 2007

Questions and Answers

-- JuanPalacios - 10 Oct 2005

- cond.txt:
- detElem.txt:
- LHCb_Palacios_v1.1.pdf: LHC Alignment Workshop, Aug 2006: LHCb Detector Description

This topic: LHCb > GeometryFramework
Topic revision: r28 - 2007-10-15 - JuanPalacios