



The LHCb Detector description DTD

Sebastien Ponce

31 August 2001

1 Overview

The LHCb detector description DTD is not unique but actually divided into three main parts :

- the DTD for the structure of the detector
- the DTD for the geometry of the detector
- the DTD for the description of materials

On top of these, three other "subDTDs" are used :

- the DTD defining catalogs
- the DTD defining surfaces
- the DTD defining tabulated properties

This document describes in deep details all these DTD and the usage of every element they define. For each file, we provide the DTD itself, fully commented, examples of XML using it and the UML schema representing it.

2 Some prerequisites

You don't have to be very familiar with the dtd definition to read this page. You could even know nothing about it. However, it could be useful to have some basic knowledge concerning XML. You also have to be a little bit familiar with the GAUDI way of dealing with data and especially what is called the transient store. If it is not the case, I advise you to read the "Detector description" chapter in the Gaudi Reference Guide, at <http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/Gaudi/>.

3 Parameters

Parameters are kind of macros that you can define and reuse in your xml files. Since they appear in different parts of the DTD, they are described here.

3.1 The DTD

```
<!ELEMENT parameter EMPTY>
```

```
<!ATTLIST parameter
  name CDATA #REQUIRED
  value CDATA #REQUIRED>
```

< >

parameter

name : CDATA
value : CDATA



This element allows the user to define his own parameters (with a given name and a given value) that can be then reused in any expression or value in the rest of the XML code.

This parameter is actually a kind of macro since it will be replaced by its value at parsing time. Its definition scope is (at this time) fully global. This means that the parser has a set of parameters it updates every time a new one is parsed and that this set is global. Thus, a given parameter can be used in files referenced by the file where the parameter is defined and recursively. However, this is not encouraged since you could imagine a situation where a file is referenced in two places and the parameters used have different values depending on the file the parser is coming from (or maybe this is a nice feature ?).

Note that the evaluation of the parameters' value is done by the CLHEP expression evaluator. Thus, most of the current units and constants are already known. You can safely use degree, rad or pi for angles for instance. On top of that, many mathematical functions are also known, such as sin, exp, arctan and many others. Please go to the CLHEP documentation, at <http://wwwinfo.cern.ch/asd/lhc++/clhep/manual/UserGuide/index.html> for further details.

3.2 Example

Here is an example of the use of parameters in XML, taken from the Rich1 description.

Here are the parameter's definitions :

```
<parameter name="MirrorInnerR" value="1700*mm" />
<parameter name="MirrorOuterR" value="1706*mm" />
<parameter name="MirrorSectorXSize" value="600/pow(2.0,0.5)*mm"/>
<parameter name="MirrorSectorYSize" value="590/pow(2.0,0.5)*mm"/>
<parameter name="MirrorSectorXGap" value="20/pow(2.0,0.5)*mm"/>
<parameter name="MirrorSectorYGap" value="20/pow(2.0,0.5)*mm"/>
<parameter name="MirrorSectorDeltaTheta"
    value="(2.0*asin(MirrorSectorXSize/(2.0*MirrorInnerR)))*rad"/>
<parameter name="MirrorSectorDeltaPhi"
    value="(2.0*asin(MirrorSectorYSize/(2.0*MirrorInnerR)))*rad"/>
<parameter name="MirrorSectorGapDeltaPhi"
    value="(2.0*asin(MirrorSectorYGap/(2.0*MirrorInnerR)))*rad"/>
```

Here is their usage :

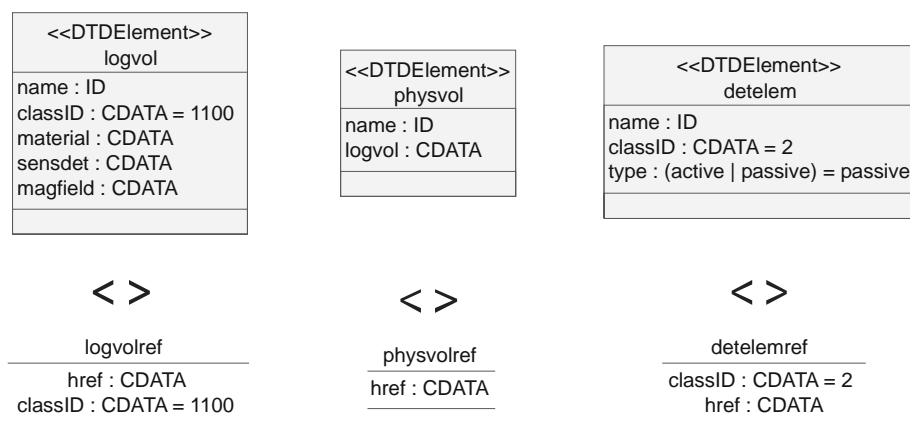
```
<sphere name="Rich1MirrorSector0"
    outerRadius = "MirrorOuterR"
    innerRadius = "MirrorInnerR"
    startPhiAngle = "MirrorSectorDeltaPhi+MirrorSectorGapDeltaPhi"
    deltaPhiAngle = "MirrorSectorDeltaPhi"
    startThetaAngle = "pi/2.0*rad·MirrorSectorDeltaTheta/2.0"
    deltaThetaAngle = "MirrorSectorDeltaTheta" />
```



4 The reference mechanism

Along the description of the LHCb DTD, you will see a lot of nodes with name finishing by "ref". Each time there is a corresponding node without the ref. The "ref" nodes are actually references on the "without ref" ones. The point of this part is to describe this reference mechanism.

4.1 UML schema



4.2 The Dtd

```

<!ELEMENT detelemref EMPTY>
<!ATTLIST detelemref classID CDATA "2"
    href CDATA #REQUIRED>
<!ELEMENT logvolref EMPTY>
<!ATTLIST logvolref href CDATA #REQUIRED
    classID CDATA "1100">
<!ELEMENT physvolref EMPTY>
<!ATTLIST physvolref href CDATA #REQUIRED>
  
```

References are very simple tags. It is just a kind of pointer to a given tag. The name of the pointer tag is the name of the pointed one plus "ref". The pointer contains two things : the type of the tag it is pointing to (its classID) and a reference to it, ie it's location in the detector description.

The classID field should only be used in the case of detector elements, in case they were extended, using the specific tag. In every other case, the default value is ok and you don't have to change it.



The reference can either refer to outside or inside the file where it is located. In the first case, the syntax is :

- `protocol://HOSTNAME/path/to/file.xml#name`

However, you can use a more simple syntax by giving a path relative to the current file :

- `relative/path/to/file#name"`

If the reference refers a tag inside the current file, the syntax is simply :

- `#name`

Note that the name used in the syntax is the string contained by the "name" attribute of the pointed tag (it always has one when a reference tag is defined). On top of that, this name attribute is of type ID, which means that it is guaranteed to be unique in the xml file where it is located (this is the XML specification).

Note also that the usage of relative paths for the outside references is strongly encouraged since files may be (and actually are) installed in different directories for different users.

4.3 Example

Here is an example of outside and inside references :

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

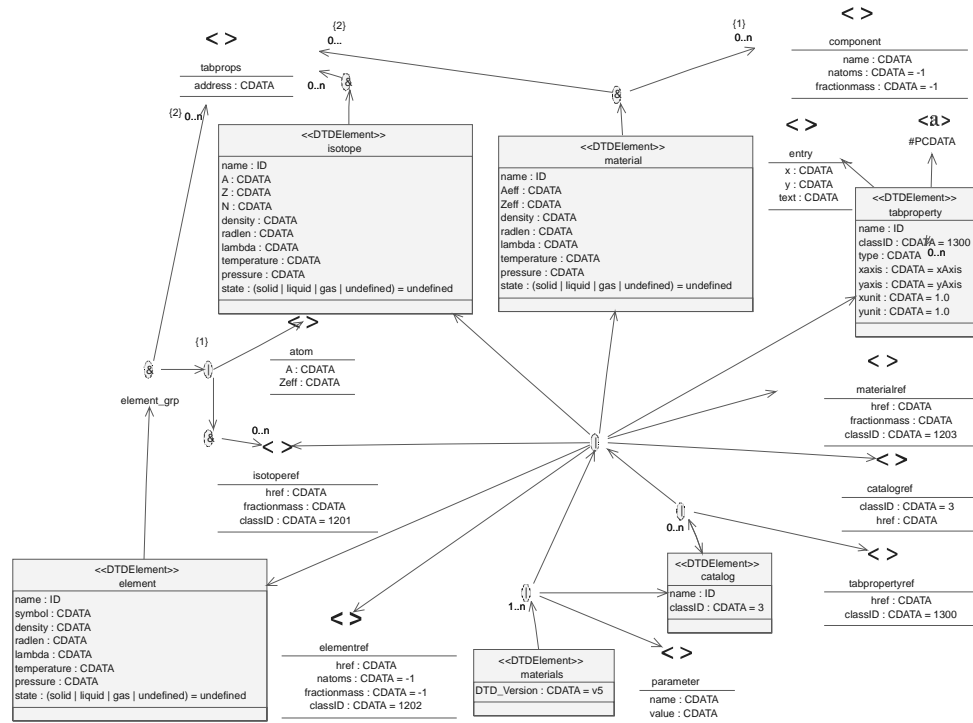
```
<logvolref href="#lvRich1" />
```

5 Structure

The structure of the detector is described here. What we call structure is mostly everything except the geometry itself and the definition of materials. This includes mostly catalogs and detector elements but also some helpful facilities like the definition of parameters or the geometryinfo, version and author tags.



5.1 UML schema



5.2 The Dtd

The Dtd is in blue, examples are in green (or red for the important stuff) and comments are in black.

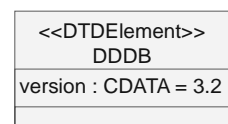
5.2.1 Xml header

```
<?xml version="1.0" encoding="UTF-8"?>
```

This DTD is using the default encoding for the whole LHCb XML. It uses version 1.0 of the XML specification.

5.2.2 DDDB Element

```
<!ELEMENT DDDB (parameter | catalog | catalogref |
detelem)+>
```



```
<!ATTLIST DDDB version CDATA "3.2" >
```

This is supposed to be the top element of an XML file using this DTD. It has a version (currently 3.2) and the only children it can have are parameters, catalogs (or references on it) and detector elements. Note that references on detector elements are not allowed here.



Here is an example of the use of this tag in a XML file :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DDDDB SYSTEM "DTD/structure.dtd">
<DDDB>
  <catalog name="dd">
    <catalogref href = "structure.xml#Structure" />
    ...
  </catalog>
</DDDB>
```

5.2.3 Catalog DTD

```
<!ENTITY % dtdForCatalog SYSTEM "catalog.dtd" >
%dtdForCatalog;
```

This includes the DTD for catalogs. This DTD is detailed in Section 8 on page 38. It only defines two tags : `<catalog>` and `<catalogref>`. The first one can contain any number of any other tag and the second one is just a reference on the first one (See Section 4 on page 4).

5.2.4 Parameters

```
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter name CDATA #REQUIRED
                    value CDATA #REQUIRED>
```

< >

parameter
name : CDATA
value : CDATA

Parameters are described in Section 3 on page 2.

5.2.5 Detector Elements

```
<!ELEMENT detelem ((author | version |
  geometryinfo | detelemref | userParameter |
  userParameterVector | specific)*)>
<!ATTLIST detelem name ID #REQUIRED
                  classID CDATA "2"
                  type (active | passive) #FIXED "passive">
```

<<DTDElement>> detelem
name : ID
classID : CDATA = 2
type : (active passive) = passive

Detector elements are the essential part 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 and potentially any other kind of data concerning this part.

One of the specificities of the detector element is that you can extend it and define your own detector element. This is achieved by using the `<specific>` tag (see Section 5.2.11 on



page 11). It can also content any kind of user specific parameters via the `<userParameter>` and `<userParameterVector>` tags (see Section 5.2.9 on page 10).

A detector element has 3 attributes :

- `name` : it's name. This is a special kind of attribute since it's type is ID. This only means that this attribute's value must be unique in the whole XML file, among all other IDs.
- `classID` : this is a unique number that equals 2 by default and has to be redefined if you want to define your own, more precise type of detector elements. This has to do with the usage of the `<specific>` tag, see Section 5.2.9 on page 10.
- `type` : this tells whether this part of the detector is active or passive. Default is passive.

The children of the detector element tag are numerous, they are detailed further in this document but here is a quick view of them :

- `author` : this defines the author or authors of this part of the detector description
- `version` : this gives a version number to this description. A given part of the detector could actually change a bit in time and may need to be described again. This version number helps to differentiate both descriptions.
- `geometryinfo` : this holds the geometry of this detector element seen as a whole. This means that this geometry may be an approximation of the actual geometry at the level of this detector element. A more precise geometry may be contained in the description of each subelement of this element.
- `detelemref` : this points to subelements of this detector element. These parts are defined in a `<detelem>` tag, like this one.
- `userParameter` : this allows the user to define new parameters for this precise detector element. These parameters are either string, int or double.
- `userParameterVector` : this allows the user to define new parameters for this precise detector element. These parameters are vectors of either string, int or double
- `specific` : this is a special tag in which the user could add its own new tags and new way of describing this detector element.

Here is an example of the use of detector elements, taken from the Ecal description :

```
<detelem name="Ecal" classID="8900">
  <author> Olivier Callot </author>
  <version> 1.0 </version>
  <geometryinfo lvname="/dd/Geometry/Ecal/lvEcal"
                support="/dd/Structure/LHCb"
                npath="EcalSubsystem" />
  <detelemref classID="8901" href = "#EcalOuter"/>
  <detelemref classID="8901" href = "#EcalMiddle"/>
  <detelemref classID="8901" href = "#EcalInner"/>
  <userParameter name="CodingBit" type="int"> 6 </userParameter>
  <userParameter name="EtSlope"
```

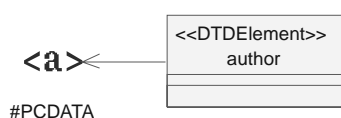



```

        type="double"
        comment="10 + .3*7 = 12.1 at 300 mrad">
    7.*GeV
</userParameter>
</detelem>

```

5.2.6 Author

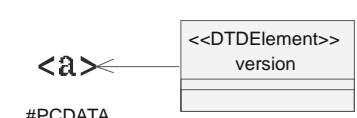

 <!ELEMENT author (#PCDATA)>

This is a very simple tag containing the name of a given person that wrote a given part of the XML description.

Here is an example of its use :

```
<author> Gonzalo Gracia Abril </author>
```

5.2.7 Version


 <!ELEMENT version (#PCDATA)>

This is a very simple tag that gives a version number to the description of the current part of the detector. This version number helps to differentiate several descriptions of the same elements, built at different times. Note that it is a string and may content more than a number.

Here is an example of its use :

```
<version> 0.0 </version>
```

5.2.8 References on detector elements

```

<!ELEMENT detelemref EMPTY>
<!ATTLIST detelemref classID CDATA "2"
                    href CDATA #REQUIRED>

```

```

< >
-----
detelemref
classID : CDATA = 2
href : CDATA
-----

```

The reference mechanism is explained in detail in Section 4 on page 4. Here is an example of a reference for a detector element :

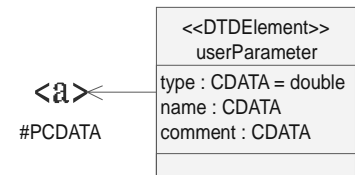
```
<detelemref href="LHCb/structure.xml#LHCb"/>
```



5.2.9 UserParameter(Vector)s

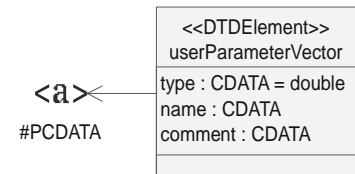
```
<!ELEMENT userParameter (#PCDATA)>
```

```
<!ATTLIST userParameter
      type CDATA "double"
      name CDATA #REQUIRED
      comment CDATA "">
```



```
<!ELEMENT userParameterVector (#PCDATA)>
```

```
<!ATTLIST userParameterVector
      type CDATA "double"
      name CDATA #REQUIRED
      comment CDATA "">
```



This allows the user to add a parameter or a vector of parameters to a given detector element. This is intended to be used for specific parameters appearing in the subdetectors' descriptions.

A parameter has three attributes on top of its value :

- a name : nothing special here. It is used to be retrieved in the C++ code after conversion.
- a type : this is the way the parameter should be handled in a converter. The value of the parameter itself is always a string but the converters will try to convert it into a double or an integer if this attributes equals "double" or "int". Otherwise, it will be taken as a string and won't be converted at all.
- a comment : this is supposed to keep a quick description of the meaning of the parameter and of its use.

To be complete, one should add that the values of a vector should be separated by spaces and/or carriage returns. No comma, semi column or any other character is accepted as separator.

Here is an example of the use of these userParameters (mostly taken from Ecal) :

```
<userParameter name="CodingBit" type="int"> 6 </userParameter>
<userParameter name="EtInCenter" type="double"
      comment="10 GeV Et Max in center">
  10.*GeV
</userParameter>
<userParameterVector name="aVector">
  10 20 30
  40 50 60
</userParameterVector>
```



5.2.10 geometry info

<pre><!ELEMENT geometryinfo EMPTY></pre> <pre><!ATTLIST geometryinfo lvname CDATA #REQUIRED</pre> <pre> support CDATA #IMPLIED</pre> <pre> npath CDATA #IMPLIED</pre> <pre> rpath CDATA #IMPLIED></pre>	<pre><></pre> <table border="1"> <tr> <td>geometryinfo</td> </tr> <tr> <td>lvname : CDATA</td> </tr> <tr> <td>support : CDATA</td> </tr> <tr> <td>npath : CDATA</td> </tr> <tr> <td>rpath : CDATA</td> </tr> </table>	geometryinfo	lvname : CDATA	support : CDATA	npath : CDATA	rpath : CDATA
geometryinfo						
lvname : CDATA						
support : CDATA						
npath : CDATA						
rpath : CDATA						

This tag describes the geometry of a given detector element. It has no child but several attributes :

- **lvname** : the logical volume name. This logical volume keeps the actual geometry description (See "Logical volumes" on page 16.). Note that the name given here is actually a full path to the logical volume in the GAUDI transient store, i.e. something like `"/dd/Geometry/SubDetectorName/..."`
- **support** : name of the supporting detector element. Once again, this name is the full path to the detector element in the GAUDI transient store. Thus something like `"/dd/Structure/SubDetectorName/..."`
- **npath** : path to follow to connect the support volume to the current volume e.g a sequence like `"calo/left/inner"`
- **rpath** : the same as **npath** but using numbers. Deprecated.

Here is an example of the use of `geometryinfo`, taken from the Rich1 description :

```
<geometryinfo lvname="/dd/Geometry/Ecal/lvEcal"
              support="/dd/Structure/LHCb"
              npath="EcalSubsystem" />
```

5.2.11 Specific fields

<pre><!ELEMENT specific ANY></pre>	<pre><...></pre>
--	------------------------

This is the place where a user can extend the default detector description language for detector elements and introduce new tags for his own needs. It's foreseen that the user defined tags have their corresponding DTD records defined in a local DTD section of the XML data file where specific tag is used or in their own, private, DTD file.

The use of user defined tags supposes that these are parsed and mapped to C++ objects or members by the XML converters of GAUDI. This can only be achieved by the user who has to provide the corresponding converters. This is described in a dedicated document : "Extending detector elements and implications", by Sebastien Ponce, available at <http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/Documents/detElemExtension.pdf>. However, here is a quick overview of what needs to be done :

- a **classID** for this new type of detector elements should be found. It has to be unique
- this **classID** must be used in the attribute **classID** of the detector element in order that the right converter be called to convert this element.



- a new C++ object has to be defined (inheriting from DetectorElement) that holds the data of this new detector element
- a converter from the XML code to the C++ object has to be written. This is facilitated by the GAUDI architecture and the user has only few lines to add to the default converter.

Here is an example of use, taken from the description of the Velo :

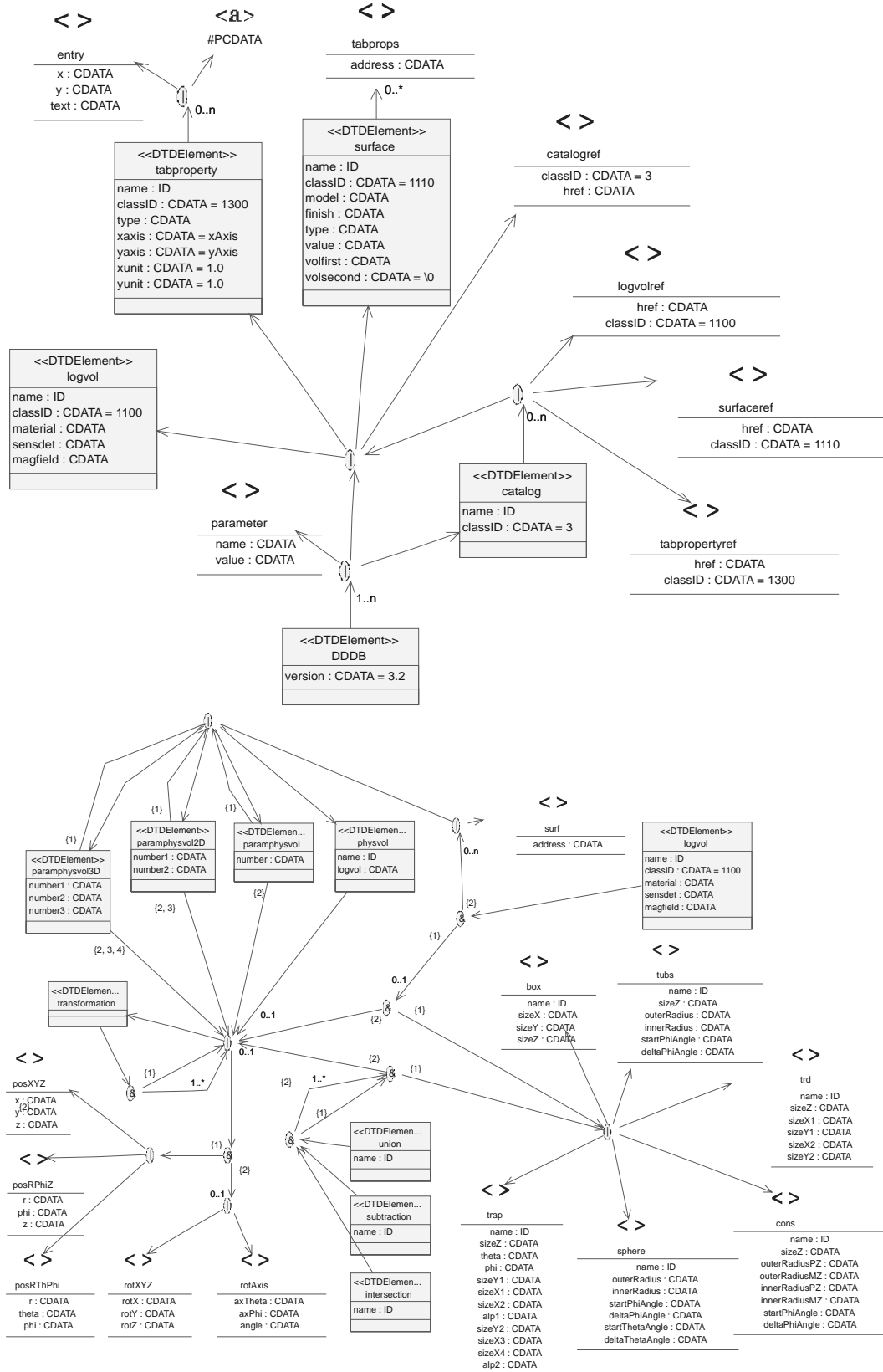
```
<detelem classID="9990" name="MStation04">
  <author>Radovan Chytrcek</author>
  <version>0.1</version>
  <geometryinfo lvname="/dd/Geometry/Muon/lvMStation04"
    rpath="3"
    support="/dd/Structure/LHCb/Muon"/>
  <specific>
    <Al_plate_thickness value="1.4444*mm"/>
    <pad_dimensions padX="2" padY="4"/>
  </specific>
</detelem>
```

6 Geometry

This DTD allows to describe the geometry of a detector. This includes the pure geometry as well as the positioning of it and the internal structure of the geometry. One of the main point is that this "geometrical" structure is fully independent of the actual structure described before.



6.1 UML schema



6.2 The Dtd

The Dtd is in blue, examples are in green (or red for the important stuff) and comments are in black.

6.2.1 Xml header

```
<?xml version="1.0" encoding="UTF-8"?>
```

This DTD is using the default encoding for the whole LHCb XML. It uses version 1.0 of the XML specification.

6.2.2 DDDB Element

```
<!ELEMENT DDDDB
  (parameter | catalog | catalogref | logvol | surface |
  tabproperty)+>
```

<<DTDElement>> DDDB
version : CDATA = 3.2

```
<!ATTLIST DDDDB version CDATA "3.2" >
```

This is supposed to be the top element of an XML file using this DTD. It has a version (currently 3.2) and the only children it can have are parameters, catalogs (and references on it), logical volumes, surfaces and tabulated properties. Note that there are no physical volumes at this level.

6.2.3 Catalog DTD

```
<!ENTITY % dtdForCatalog SYSTEM "catalog.dtd">
%dtdForCatalog;
```

This includes the DTD for catalogs. This DTD is detailed in Section 8 on page 38. It only defines two tags : <catalog> and <catalogref>. The first one can contain any number of any other tag and the second one is just a reference on the first one.

6.2.4 Tabulated Properties DTD

```
<!ENTITY % dtdForTabproperty SYSTEM "tabproperty.dtd" >
%dtdForTabproperty;
```

This includes the DTD for tabulated properties. This DTD is detailed in Section 10 on page 40. It only defines two tags : <tabproperty> and <tabpropertyref>. The first one can contain a table of figures while the second one is just a reference on the first one.

6.2.5 Surfaces DTD

```
<!ENTITY % dtdForSurface SYSTEM "surface.dtd" >
%dtdForSurface;
```



This includes the DTD for surfaces. This DTD is detailed in Section 9 on page 39. It only defines two tags : <surface> and <surfaceref>. The first one describes a surface while the second one is just a reference on the first one.

6.2.6 Some entities to simplify the DTD

```
<!ENTITY % solid "(box | cons | sphere | tubs | trd | trap |
                    union | subtraction | intersection)">
<!ENTITY % simpleTransformation
          "((posXYZ | posRPhiZ | posRThPhi), (rotXYZ | rotAxis?))">
<!ENTITY % transformation "(%simpleTransformation; | transformation)">
<!ENTITY % booleanchildren
          "((%solid;, %transformation;?), (%solid;, %transformation;?)+)">
<!ENTITY % paramphysvol
          "(paramphysvol | paramphysvol2D | paramphysvol3D)">
```

These are some entity definitions that are useful to keep the rest of the DTD readable. It defines the following concepts :

- **solid** : this represents any solid. This includes simple solids as well as boolean solids.
- **simpleTransformation** : this represents a simple transformation, i.e. a translation and possibly a rotation. Note that the rotation is always applied first but defined after the translation. This is very important in some tags where you define several transformations one after the other. If you want to define a "rotation only" transformation, you have to put an empty translation first (something like <posXYZ/>). Otherwise, it could be added to the previous transformation (provided this one is a "translation only" one) and applied before it.
- **transformation** : this represents a transformation whatever it is, simple or composed.
- **booleanchildren** : this is the definition of the children of a boolean operation. It basically says that a boolean operation must have two or more children, each of which is a solid plus (optionally) a transformation that will be applied to it before processing the boolean operation. Note that at this time, the transformation of the first child is not processed by the converters even if no error occurs.
- **paramphysvol** : this defines a parameterized physical volume, whatever its dimension is.

6.2.7 Transformation composition

```
<!ELEMENT transformation
          (%transformation;, (%transformation;)+)>
```

<<DTDElemen... transformation

This defines the composition of two or more transformations. The transformations will be applied in the order they are given but take care that one transformation is defined using two tags if it has a rotation part and that this rotation is applied before the associated translation, even if it is the second tag.

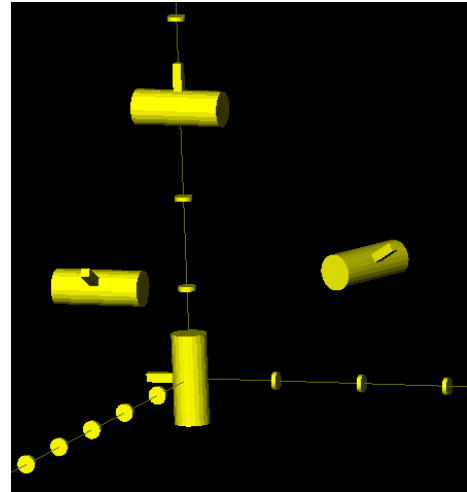
As an example, here is part of the testing files of the detector description package :



```

<transformation>
  <transformation>
    <posXYZ z="3*m"/>
    <rotXYZ rotX="-90*degree"/>
    <posXYZ/>
    <rotXYZ rotY="60*degree"/>
  </transformation>
  <posXYZ/>
  <rotXYZ rotZ="70*degree"/>
</transformation>

```



In this example, the order in which the translations and rotations are applied is the following :

- <rotXYZ rotX="-90*degree"/>
- <posXYZ z="3*m"/>
- <rotXYZ rotY="60*degree"/>
- <rotXYZ rotZ="70*degree"/>

Note that the inner transformation tag could be removed without changing the behavior. Note also that the two <posXYZ/> tags are needed even if they do not define any translation. More than that, if you want to insert a new translation in the middle of the first transformation, you have to write :

```

<transformation>
  <posXYZ z="3*m"/>
  <rotXYZ rotX="-90*degree"/>
  <posXYZ y="1*m"/>
  <posXYZ/>
  <rotXYZ rotY="60*degree"/>
</transformation>

```

If you forget the <posXYZ/> tag, the new translation and the last rotation will be taken as one transformation and the rotation will be applied first !

6.2.8 Logical volumes

```

<!ELEMENT logvol
  ((%solid; (%transformation:))?,
   (physvol | %paramphysvol; | surf)*)>
<!ATTLIST logvol name ID #REQUIRED
                 classID CDATA "1100"
                 material CDATA #IMPLIED

```

<<DTDElement>> logvol
name : ID
classID : CDATA = 1100
material : CDATA
sensdet : CDATA
magfield : CDATA




```
sensdet CDATA #IMPLIED
magfield CDATA #IMPLIED>
```

The *logvol* tag is the heart of the geometry description. A logical volume is a geometrical volume with a given shape and potentially subvolumes. It has a material but it is not positioned in any referential. It kinds of hangs in outer space.

The children of logical volumes are divided into two groups. The first one may be empty or may contain the shape of this volume. This shape is a solid that may be followed by a transformation to be applied to it. Note that the current implementation of the XML converters ignores it.

The second group of children is a set of physical volumes, surfaces and parameterized physical volumes. In any case, they are subvolumes (or surfaces) that were placed inside the logical volume. See Section 6.2.10 on page 18 for a detailed description of the *physvol* tag.

Of course, each of the subvolumes may also have a shape. Actually, the shape of a given logical volume is a kind of approximation of the actual shape of the detector at the level of this volume. The actual shape is defined in the subvolumes of the logical volume.

Concerning the attributes of the *logvol* tag, here are their descriptions :

- **name** : this is the name of the logical volume. It is of type ID, which means that this is a string and that this string must be unique in the whole XML file.
- **classID** : this has to do with the conversion mechanism of the XML. This defines which converter will convert this tag. It has a default value and this one should never be changed.
- **material** : this is a pointer to the material of this volume in the GAUDI transient store. It should either be the full path of it, i.e. something like `"/dd/materials/..."` or the part after `"/dd/materials"`. In this case, it may not begin with a `"/"`.
- **sensdet** : this tells whether this part is sensible or not
- **magfield** : this defines the magnetic field for this part

Here is an example of use of the *logvol* tag, taken from the Rich1 description :

```
<logvol name="lvRich1GasWindowAnnex" material="C4F10">
  <trd name = "GasWindowAnnexTrd"
    sizeX1 = "Rich1GasWindowAnnexXStartSize"
    sizeY1 = "Rich1GasWindowAnnexYStartSize"
    sizeX2 = "Rich1GasWindowAnnexXEndSize"
    sizeY2 = "Rich1GasWindowAnnexYEndSize"
    sizeZ = "Rich1GasWindowAnnexZSize" />
  <physvol name="pvQuartzWindow"
    logvol="/dd/Geometry/Rich1/lvRich1QuartzWindow">
    <posXYZ x ="Rich1QuartzWindowX"
      y ="Rich1QuartzWindowY"
      z ="Rich1QuartzWindowZ" />
```



```

</physvol>
<surf address=
  "/dd/Geometry/Rich1/Rich1Surfaces/Rich1QuartzWindowSurface"/>
</logvol>

```

6.2.9 Reference on logical volume

```

<!ELEMENT logvolref EMPTY>
<!ATTLIST logvolref href CDATA #REQUIRED
              classID CDATA "1100">

```

< >
logvolref
href : CDATA
classID : CDATA = 1100

The reference mechanism is explained in detail in Section 4 on page 4. Here is an example of a reference for a logical volume :

```
<logvolref href="#lvRich1" />
```

6.2.10 Physical Volume

```

<!ELEMENT physvol (%transformation;)?>
<!ATTLIST physvol name ID #REQUIRED
                  logvol CDATA #REQUIRED>

```

<<DTDElemen... physvol
name : ID
logvol : CDATA

The notion of physical volume is very simple. It is a logical volume that is positioned inside its mother logical volume. Nothing more.

Thus, a physical volume contains a transformation that positions the inner logical volume. It also has two attributes :

- **name** : this is the name of the physical volume. It is of type ID, thus it must be unique in the whole XML file.
- **logvol** : this is a pointer to the corresponding logical volume. This pointer is just the name of this volume in the GAUDI transient data store, thus something like "/dd/Geometry/SubDetectorName/..."

Here is an example of the use of the *physvol* tag, taken from the Rich1 description :

```

<physvol name="pvRich1GasWindowAnnex0"
  logvol="/dd/Geometry/Rich1/lvRich1GasWindowAnnex" >
  <posXYZ x = "Rich1GasWindowAnnexX"
    y = "Rich1GasWindowAnnexY"
    z = "Rich1GasWindowAnnexZ" />
  <rotXYZ rotX ="-1.0*pi/2.0-AnnexAngle"
    rotY ="0.0*rad"
    rotZ ="pi/2.0*rad-TrapezAngle"/>
</physvol>

```



6.2.11 Reference on physical volume

```
<!ELEMENT physvolref EMPTY>
```

```
<!ATTLIST physvolref href CDATA #REQUIRED>
```

The reference mechanism is explained in detail in Section 4 on page 4. Here is an example of a reference for a physical volume :

```
<physvolref href="main.xml#MainPhysVol"/>
```

6.2.12 Parameterized physical volumes

```
<!ELEMENT paramphysvol
  ((physvol | %paramphysvol;), %transformation;)>
```

<<DTDElemen... paramphysvol
number : CDATA

```
<!ATTLIST paramphysvol number CDATA #REQUIRED>
```

The goal of this tag is to avoid typing too much XML in case you have to define several physical volumes that are identical except for their positioning. It allows to copy *n* times a given physical volume (or a set of them defined by another parametrised physical volume). Each new volume is positioned by applying the given transformation to the preceding one.

This tag has thus two children. The first child must be the volume (or parameterized volume) to be replicated and the second one must be the transformation to be applied for placing each new instance of the volume.

It has one attribute which gives the number of replication of the volume.

Here are three examples of the use of this tag. It also illustrates the combination of several *paramphysvol* tags.

```
<logvol material="Carbon" name="paramphysvol1D">
  <paramphysvol number="4">
    <physvol name="base1"
      logvol="/dd/Geometry/paramphysvol_catalog/base_vol"/>
    <posXYZ y="1*m"/>
    <rotXYZ rotZ="45*degree"/>
  </paramphysvol>
</logvol>

<logvol material="Carbon" name="paramphysvol1Dshift">
  <physvol name="ppvc1Dshift"
    logvol="/dd/Geometry/paramphysvol_catalog/paramphysvol1D">
    <posXYZ y="1*m"/>
  </physvol>
</logvol>

<logvol material="Carbon" name="composition_vol">
```



```

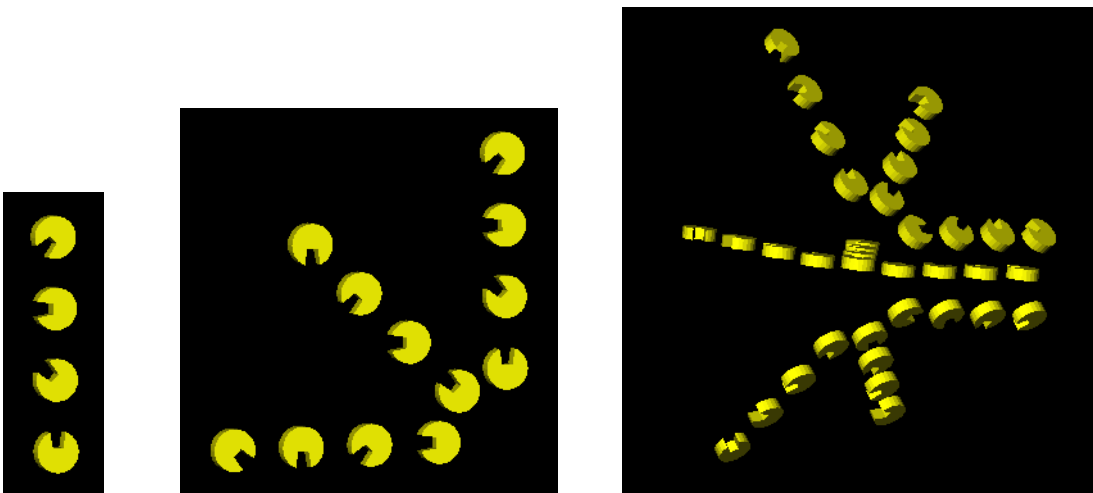
<paramphysvol number="3">
  <physvol name="ppvc2D"
    logvol="/dd/Geometry/paramphysvol_catalog/paramphysvol1Dshift"/>
  <posXYZ/>
  <rotXYZ rotZ="45*degree"/>
</paramphysvol>
</logvol>

<logvol material="Carbon" name="composition_volshift">
  <physvol name="ppvccompvolshift"
    logvol="/dd/Geometry/paramphysvol_catalog/composition_vol">
    <posXYZ x="-1*m"/>
  </physvol>
</logvol>

<logvol material="Carbon" name="composition2_vol">
  <paramphysvol number="3">
    <physvol name="ppvc3D"
      logvol="/dd/Geometry/paramphysvol_catalog/composition_volshift"/>
    <posXYZ/>
    <rotXYZ rotY="45*degree"/>
  </paramphysvol>
</logvol>

```

And here are the pictures obtained for these geometries :



6.2.13 Parameterized physical volumes 2D and 3D

```
<!ELEMENT paramphysvol2D
  ((physvol | %paramphysvol;), %transformation;
   %transformation;)>
<!ATTLIST paramphysvol2D
  number1 CDATA #REQUIRED
  number2 CDATA #REQUIRED>
```

<<DTDElement>> paramphysvol2D
number1 : CDATA
number2 : CDATA

```
<!ELEMENT paramphysvol3D
  ((physvol | %paramphysvol;), %transformation;
   %transformation;, %transformation;)>
<!ATTLIST paramphysvol3D
  number1 CDATA #REQUIRED
  number2 CDATA #REQUIRED
  number3 CDATA #REQUIRED>
```

<<DTDElement>> paramphysvol3D
number1 : CDATA
number2 : CDATA
number3 : CDATA

The `<paramphysvol>` tag allows to replicate volumes in one dimension only (even if there is the possibility of combining several of them). The `<paramphysvol2D>` and `<paramphysvol3D>` tag allow to do the same in 2 or 3 dimensions. Their syntax is very close to the one of `<paramphysvol>`. They just take more transformations and more numbers, according to the dimension.

The important point here is to understand that using a 2 or 3 dimensional parameterization is rather different from using twice or three times a one dimensional parametrization. The difference appears in the order the transformations are processed.

This comes from the fact that when you use a parameterization, be it in 1, 2 or 3 dimensions, the transformations are applied in two steps : first the volume is rotated (a given number of times for each dimension, according to its location) and then it is translated. This means that the resulting volumes are always disposed on a grid (1, 2 or 3D). See the two figures below.

On the contrary, when you compose 1D parameterization, the result of the first parameterization is seen from the second parametrization as a whole. Thus, the rotation coming from the second parameterization is applied to this whole and not to every subpart. See the pictures above and the ones in Section 6.2.12 on page 19 to see the difference.

By the way, you could also compose 2 or even 3D transformations ! But I think it will become a bit tricky !

Here are two examples. Compare them to the example given above, with the composition of 1D parameterizations.

```
<logvol material="Carbon" name="paramphysvol2D">
  <paramphysvol2D number1="4" number2="4">
    <physvol name="base2"
      logvol="/dd/Geometry/paramphysvol_catalog/base_vol"/>
    <posXYZ y="1*m"/>
```

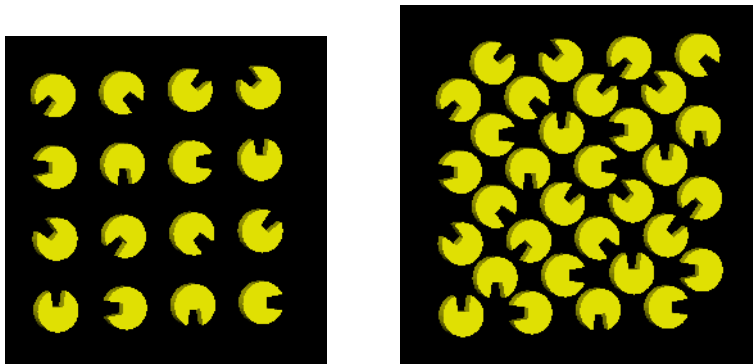


```

    <rotXYZ rotZ="45*degree"/>
    <posXYZ x="1*m"/>
    <rotXYZ rotZ="90*degree"/>
  </paramphysvol2D>
</logvol>
<logvol material="Carbon" name="paramphysvol3D">
  <paramphysvol3D number1="4" number2="4" number3="2">
    <physvol name="base3"
      logvol="/dd/Geometry/paramphysvol_catalog/base_vol"/>
    <posXYZ y="1*m"/>
    <rotXYZ rotZ="45*degree"/>
    <posXYZ x="1*m"/>
    <rotXYZ rotZ="90*degree"/>
    <posXYZ x=".5*m" y=".5*m"/>
    <rotXYZ rotZ="180*degree"/>
  </paramphysvol3D>
</logvol>

```

And here are the pictures obtained from these geometries :



6.2.14 Parameters

```

<!ELEMENT parameter EMPTY>
<!ATTLIST parameter name CDATA #REQUIRED
                  value CDATA #REQUIRED>

```

Parameters are described in Section 3 on page 2.

< >

parameter
name : CDATA
value : CDATA



6.2.15 References on surfaces

```
<!ELEMENT surf EMPTY>
```

```
< >
```

```
<!ATTLIST surf address CDATA #REQUIRED>
```

```
surf
```

```
address : CDATA
```

There are actually two kinds of references on surfaces. The one is called `<surfaceref>` and is a regular reference, using files and XML IDs. The other is this `<surf>` tag that uses the GAUDI transient data store to name the surface.

This tag has no child and only one attribute : the name of the pointed surface. This name is the transient store name, thus something like `"/dd/Geometry/SubDetectorName/..."`

Here is an example of the use of this tag, taken from the Rich1 description :

```
<surf
  address="/dd/Geometry/Rich1/Rich1Surfaces/Rich1GasEnclosureSurface" />
```

6.2.16 Cartesian translation

```
<!ELEMENT posXYZ EMPTY>
```

```
< >
```

```
<!ATTLIST posXYZ x CDATA #IMPLIED
                  y CDATA #IMPLIED
                  z CDATA #IMPLIED>
```

```
posXYZ
```

```
x : CDATA
```

```
y : CDATA
```

```
z : CDATA
```

This defines a translation in a cartesian coordinate system.

Here is an example of use, taken from the Rich1 description :

```
<posXYZ x = "GasDownStreamBoxX"
        y = "GasDownStreamBoxY"
        z = "GasDownStreamBoxZ" />
```

6.2.17 Cylindrical translation

```
<!ELEMENT posRPhiZ EMPTY>
```

```
< >
```

```
<!ATTLIST posRPhiZ r CDATA #IMPLIED
                   phi CDATA #IMPLIED
                   z CDATA #IMPLIED>
```

```
posRPhiZ
```

```
r : CDATA
```

```
phi : CDATA
```

```
z : CDATA
```

This defines a translation in a cylindrical coordinate system.

Here is an example of use :

```
<posRPhiZ r = "1*m"
          phi = "45*degree"
          z = "1*m" />
```



6.2.18 Spherical translation

```
<!ELEMENT posRThPhi EMPTY>
```

```
< >
```

```
<!ATTLIST posRThPhi r CDATA #IMPLIED
                theta CDATA #IMPLIED
                phi CDATA #IMPLIED>
```

```
posRThPhi
-----
r : CDATA
theta : CDATA
phi : CDATA
-----
```

This defines a translation in a spherical coordinate system.

Here is an example of use :

```
<posRThPhi r = "1*m"
           theta = "45*degree"
           phi = "90*degree" />
```

6.2.19 Cartesian rotation

```
<!ELEMENT rotXYZ EMPTY>
```

```
< >
```

```
<!ATTLIST rotXYZ rotX CDATA #IMPLIED
                rotY CDATA #IMPLIED
                rotZ CDATA #IMPLIED>
```

```
rotXYZ
-----
rotX : CDATA
rotY : CDATA
rotZ : CDATA
-----
```

This defines a rotation in a cartesian coordinate system. It may actually defines 3 rotations, one along each axis. They are applied in sequence, beginning by the one along the Z axis and ending by the one along the X axis.

Here is an example of use, taken from the Rich1 description :

```
<rotXYZ rotX ="-1.0*pi/2.0*AnnexAngle"
        rotZ ="pi/2.0*rad*TrapezAngle"/>
```

6.2.20 Axis rotation

```
<!ELEMENT rotAxis EMPTY>
```

```
< >
```

```
<!ATTLIST rotAxis axTheta CDATA #IMPLIED
                axPhi CDATA #IMPLIED
                angle CDATA #IMPLIED>
```

```
rotAxis
-----
axTheta : CDATA
axPhi : CDATA
angle : CDATA
-----
```

This defines a generic rotation along an axis. The only restriction is that the root of the coordinate system is on the axis.

The <rotAxis> tag has no child and the axis and rotation parameters are defined through the attributes : axTheta and axPhi give the axis while angle is the rotation angle along the axis.

Here is an example of use :

```
<rotAxis axTheta="45*degree"
        axPhi="45*degree">
```




```
angle="90*degree"/>
```

6.2.21 Box

```
<!ELEMENT box EMPTY>
<!ATTLIST box name ID #REQUIRED
            sizeX CDATA #REQUIRED
            sizeY CDATA #REQUIRED
            sizeZ CDATA #REQUIRED>
```

< >

box
name : ID
sizeX : CDATA
sizeY : CDATA
sizeZ : CDATA

This defines a simple box with the given dimensions, centered at position <0, 0, 0>.

Here is an example of use :

```
<box name="box_sample"
      sizeX="2*m"
      sizeY="4*m"
      sizeZ="8*m"/>
```



6.2.22 Simple trapezoids

```
<!ELEMENT trd EMPTY>
<!ATTLIST trd name ID #REQUIRED
              sizeZ CDATA #REQUIRED
              sizeX1 CDATA #REQUIRED
              sizeY1 CDATA #REQUIRED
              sizeX2 CDATA #REQUIRED
              sizeY2 CDATA #REQUIRED>
```

< >

trd
name : ID
sizeZ : CDATA
sizeX1 : CDATA
sizeY1 : CDATA
sizeX2 : CDATA
sizeY2 : CDATA

This defines a simple trapezoid. Simple means that it is far to be generic. This trapezoid is defined by two rectangular faces, orthogonal to the Z axis and centered on it, and its length along this Z axis. It is centered in <0, 0, 0>.

Here is an example of use :

```
<trd name="trd_sample"
      sizeZ="8*m"
      sizeX1="2*m"
      sizeY1="4*m"
      sizeX2="8*m"
      sizeY2="4*m"/>
```



6.2.23 General trapezoids

```
<!ELEMENT trap EMPTY>
```

```
<!ATTLIST trap name ID #REQUIRED
             sizeZ CDATA #REQUIRED
             theta CDATA #IMPLIED
             phi CDATA #IMPLIED
             sizeY1 CDATA #REQUIRED
             sizeX1 CDATA #REQUIRED
             sizeX2 CDATA #REQUIRED
             alp1 CDATA #IMPLIED
             sizeY2 CDATA #REQUIRED
             sizeX3 CDATA #REQUIRED
             sizeX4 CDATA #REQUIRED
             alp2 CDATA #IMPLIED>
```

```
< >
```

```
trap
```

```
name : ID
sizeZ : CDATA
theta : CDATA
phi : CDATA
sizeY1 : CDATA
sizeX1 : CDATA
sizeX2 : CDATA
alp1 : CDATA
sizeY2 : CDATA
sizeX3 : CDATA
sizeX4 : CDATA
alp2 : CDATA
```

Despite the title of this paragraph, this does not define a fully generic trapezoid. However, it is the most generic in this DTD.

First, the Z axis used in `<trd>` was replaced by a user defined direction, given by the two angles : *theta* and *phi*. The faces are still orthogonal to themain direction and the global length of the trapezoid along this axis is given by the *sizeZ* attribute. The trapezoid is centered in `<0, 0, 0>`.

The two faces are no more rectangles but kind of trapezes. They are defined as if *theta* and *Phi* were both equal to 0 and thus the axis of the trapezoid were the Z axis.

Each face is defined by two segments parallel to the X axis (of length *sizeX?*), the distance between them (*sizeY?*) and the angle between the Y axis and the line joining the middle of both segments (*alp?*)

Note that you can avoid defining every attribute : *theta*, *phi* and *alp?* can be omitted. Their default values are 0.

Here is a (very simple) example of use :

```
<trap name="trap_sample"
      sizeZ="12*m"
      sizeY1="2*m"
      sizeX1="2*m"
      sizeX2="4*m"
      sizeY2="4*m"
      sizeX3="4*m"
      sizeX4="8*m"/>
```



6.2.24 Tube sections

```
<!ELEMENT tubs EMPTY>
```

```
< >
```

```
<!ATTLIST tubs name ID #REQUIRED
              sizeZ CDATA #REQUIRED
              outerRadius CDATA #REQUIRED
              innerRadius CDATA #IMPLIED
              startPhiAngle CDATA #IMPLIED
              deltaPhiAngle CDATA #IMPLIED>
```

```
tubs
-----
name : ID
sizeZ : CDATA
outerRadius : CDATA
innerRadius : CDATA
startPhiAngle : CDATA
deltaPhiAngle : CDATA
-----
```

This defines a portion of tube.

The tube itself has a inner and outer radius and is oriented along the Z axis.

The portion of tube is defined by giving a length along Z (the portion will be centered in $\langle 0, 0, 0 \rangle$), and a range of value for the Phi angle.

Note that you can avoid defining every attribute. `innerRadius` can be omitted. Its default value is 0. `startPhiAngle` and `deltaPhiAngle` can be omitted too. Their default values are respectively 0 and 360° . To be short, the default builds a regular cylinder.

Here is an example of use :

```
<tubs name="tubs_sample"
      sizeZ="8*m"
      outerRadius="3.4*m"
      innerRadius="3*m"
      deltaPhiAngle="135*degree"/>
```



6.2.25 Conical sections

```
<!ELEMENT cons EMPTY>
```

```
< >
```

```
<!ATTLIST cons
              name ID #REQUIRED
              sizeZ CDATA #REQUIRED
              outerRadiusPZ CDATA #REQUIRED
              outerRadiusMZ CDATA #REQUIRED
              innerRadiusPZ CDATA #IMPLIED
              innerRadiusMZ CDATA #IMPLIED
              startPhiAngle CDATA #IMPLIED
              deltaPhiAngle CDATA #IMPLIED>
```

```
cons
-----
name : ID
sizeZ : CDATA
outerRadiusPZ : CDATA
outerRadiusMZ : CDATA
innerRadiusPZ : CDATA
innerRadiusMZ : CDATA
startPhiAngle : CDATA
deltaPhiAngle : CDATA
-----
```

This defines a conical section. It is defined by giving the height of the section, its inner and outer radius at each end and a range of values for the phi angle. The resulting solid is centered in $\langle 0, 0, 0 \rangle$ or actually would be centered if phi would go from 0 to 360° .

Note that you can avoid defining every attribute. `innerRadius*` can be omitted. Their default value is 0. `startPhiAngle` and `deltaPhiAngle` can be omitted too. Their default



values are respectively 0 and 360*degree. To be short, the default build a regular cone trunc.

Here is an example of use :

```
<cons name="cons_sample"
  sizeZ="8*m"
  outerRadiusPZ="1.4*m"
  outerRadiusMZ="3.4*m"
  innerRadiusPZ="1*m"
  innerRadiusMZ="3*m"
  deltaPhiAngle="135*degree"/>
```



6.2.26 Sphere or Bowl portions

```
<!ELEMENT sphere EMPTY>
<!ATTLIST sphere
  name ID #REQUIRED
  outerRadius CDATA #REQUIRED
  innerRadius CDATA #IMPLIED
  startPhiAngle CDATA #IMPLIED
  deltaPhiAngle CDATA #IMPLIED
  startThetaAngle CDATA #IMPLIED
  deltaThetaAngle CDATA #IMPLIED>
```

< >

sphere

name :	ID
outerRadius :	CDATA
innerRadius :	CDATA
startPhiAngle :	CDATA
deltaPhiAngle :	CDATA
startThetaAngle :	CDATA
deltaThetaAngle :	CDATA

This defines a bowl portion, or, if you prefer, a sphere portion that has a thickness. The sphere itself is centered in <0, 0, 0> and described by its inner and outer radius. The portion is defined by giving ranges for the theta and phi angles.

Note that you can avoid defining every attribute. innerRadius can be omitted. Its default value is 0. startPhiangle and deltaPhiAngle can be omitted too. Their default values are respectively 0 and 360*degree. At last, startThetaAngle and deltaThetaAngle can be omitted. Their default values are respectively 0 and 180*degree. To be short, the default builds a regular bowl.

Here is an example of use :

```
<sphere name="sphere_sample"
  outerRadius="3.4*m"
  innerRadius="3*m"
  deltaPhiAngle="135*degree"
  deltaThetaAngle="45*degree"/>
```



6.2.27 Boolean Unions

```
<!ELEMENT union (%booleanchildren;)>
<!ATTLIST union name ID #REQUIRED>
```

<<DTDElemen... union
name : ID

This defines the boolean operation union on two or more solids. The syntax of the children is described in Section 6.2.6 on page 15. Note that boolean operations can be composed.

This tag has only one attribute, its name, which is of type ID. It means that it should be unique in the whole XML file.

Here is an example of use :

```
<union name="union_sample">
  <tubs name="tubs1"
    sizeZ="1*m"
    outerRadius="2*m"/>
  <tubs name="tubs2"
    sizeZ="1*m"
    outerRadius="2*m"/>
  <posXYZ y="-2*m"/>
</union>
```



6.2.28 Boolean Subtractions

```
<!ELEMENT subtraction (%booleanchildren;)>
<!ATTLIST subtraction name ID #REQUIRED>
```

<<DTDElemen... subtraction
name : ID

This defines the boolean operation subtraction on two or more solids. The meaning of this subtraction in the case of more than two solids is that every solid (from the second one) is subtracted from the result of the previous operation. E.g., the third solid is subtracted from the result of the subtraction of the second solid from the first one. The exact syntax of the children is described in Section 6.2.6 on page 15. Note that boolean operations can be composed.

Here is an example of use :

```
<subtraction name="subtraction_sample">
  <tubs name="tubs5"
    sizeZ="1*m"
    outerRadius="2*m"/>
  <tubs name="tubs6"
    sizeZ="2*m"
    outerRadius="2*m"/>
  <posXYZ y="-2*m"/>
</subtraction>
```



6.2.29 Boolean Intersections

```
<!ELEMENT intersection (%booleanchildren;)>
<!ATTLIST intersection name ID #REQUIRED>
```

<<DTDElemen... intersection
name : ID

This defines the boolean operation intersection on two or more solids. The meaning of this intersection in the case of more than two solids is that every solid (from the second one) is intersected with the result of the previous operation. E.g., the third solid is intersected with the result of the intersection of the first solid with the second one. The exact syntax of the children is described in Section 6.2.6 on page 15. Note that boolean operations can be composed.

Here is an example of use :

```
<intersection name="intersection_sample">
  <tubs name="tubs3"
    sizeZ="1*m"
    outerRadius="2*m"/>
  <tubs name="tubs4"
    sizeZ="1*m"
    outerRadius="2*m"/>
  <posXYZ y="-2*m"/>
</intersection>
```

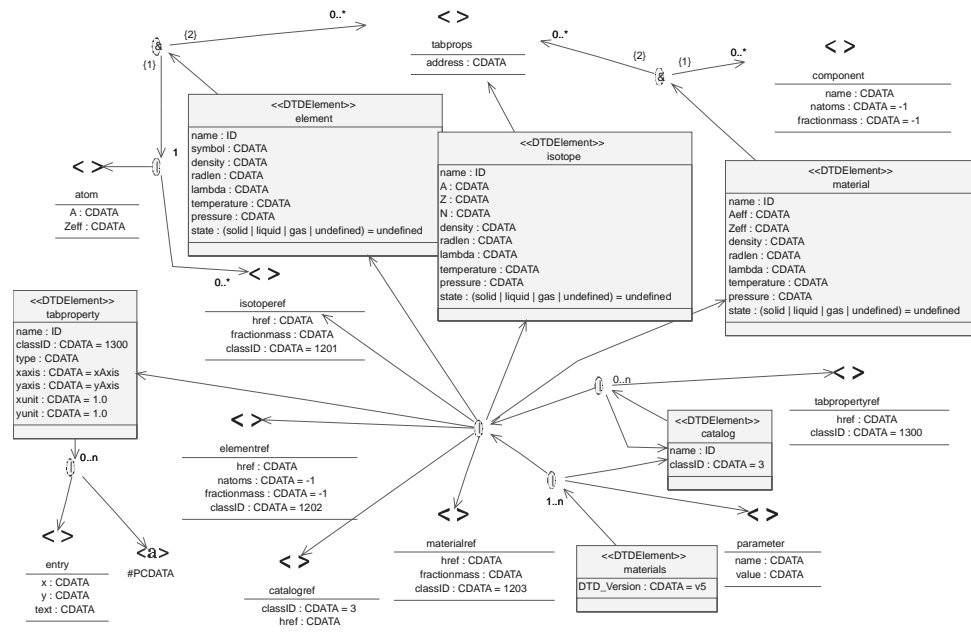


7 Materials

This part of the DTD is devoted to the description of the materials used in the detector. This includes their chemical composition, their properties and the conditions of their use (temperature, pressure, state, ...).



7.1 UML schema



7.2 The Dtd

The Dtd is in blue, examples are in green (or red for the important stuff) and comments are in black.

7.2.1 Xml header

```
<?xml version="1.0" encoding="UTF-8"?>
```

This DTD is using the default encoding for the whole LHCb XML. It uses version 1.0 of the XML specification.

7.2.2 Tabulated Properties DTD

```
<!ENTITY % dtdForTabproperty SYSTEM "tabproperty.dtd" >
% dtdForTabproperty;
```

This includes the DTD for tabulated properties. This DTD is detailed in Section 10 on page 40. It only defines two tags : <tabproperty> and <tabpropertyref>. The first one can contain a table of figures while the second one is just a reference on the first one.

7.2.3 Catalog DTD

```
<!ENTITY % dtdForCatalog SYSTEM "catalog.dtd">
% dtdForCatalog;
```



This includes the DTD for catalogs. This DTD is detailed in Section 8 on page 38. It only defines two tags : <catalog> and <catalogref>. The first one can contain any number of any other tag and the second one is just a reference on the first one.

7.2.4 Entity for conditions

```
<!ENTITY % conditions "temperature CDATA #IMPLIED
                        pressure CDATA #IMPLIED
                        state (solid | liquid | gas | undefined) 'undefined'">
```

This entity definition is intended to keep the rest of the DTD readable. It defines the concept of condition of use for a given material. These conditions are its temperature, its pressure and its state.

7.2.5 Materials

```
<!ELEMENT materials (parameter | catalog |
                    catalogref | isotope | element | material |
                    isotoperef | elementref | materialref |
                    tabproperty)+>
```

<<DTDElement>> materials
DTD_Version : CDATA = v5

```
<!ATTLIST materials DTD_Version CDATA #FIXED "v5">
```

This is supposed to be the top element of an XML file using this DTD. It has a version (currently 5) and its children are parameters, catalogs (and references on it), tabulated properties and all sorts of material definitions (namely isotopes, elements, materials and the references on these).

7.2.6 Parameters

```
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter name CDATA #REQUIRED
                    value CDATA #REQUIRED>
```

Parameters are described in Section 3 on page 2.

<>
parameter
name : CDATA
value : CDATA



7.2.7 Isotopes

```
<!ELEMENT isotope (tabprops*)>
```

```
<!ATTLIST isotope
  name ID #REQUIRED
  A CDATA #REQUIRED
  Z CDATA #REQUIRED
  N CDATA #REQUIRED
  density CDATA #IMPLIED
  radlen CDATA #IMPLIED
  lambda CDATA #IMPLIED
  %conditions;>
```

<<DTDElement>> isotope
name : ID A : CDATA Z : CDATA N : CDATA density : CDATA radlen : CDATA lambda : CDATA temperature : CDATA pressure : CDATA state : (solid liquid gas undefined) = undefined

This defines a given isotope of a given atom. This isotope has many attributes :

- name : this is of type ID and thus unique in the whole file
- A : this is the atomic mass of this isotope
- Z : this is the atomic number of the element
- N : this is the total number of hadrons for this isotope
- density : the density of this isotope, if pure
- radlen : the radiation length of this isotope
- lambda : ???
- conditions : these are described in Section 6.2.6 on page 15

On top of these attributes, an isotope can have as many children as needed. These are tabulated properties.

Here is an example of use, taken from the LHCb detector description :

```
<isotope name='Bor_10' A='10.000*g/mole' Z='5.000' N='10'
  density='2.3400*g/cm3' radlen='0.0000e+00*cm'
  lambda='0.0000e+00*cm' />
```

```
<isotope name='Bor_11' A='11.000*g/mole' Z='5.000' N='11'
  density='2.3400*g/cm3' radlen='0.0000e+00*cm'
  lambda='0.0000e+00*cm' />
```

7.2.8 References on isotopes

```
<!ELEMENT isotoperef EMPTY>
```

```
<!ATTLIST isotoperef
  href CDATA #REQUIRED
  fractionmass CDATA #IMPLIED
  classID CDATA "1201">
```

< >

isotoperef

href : CDATA
fractionmass : CDATA
classID : CDATA = 1201



The reference mechanism is explained in detail in Section 4 on page 4. This reference has a specific attribute called `fractionmass`. This is the proportion of this isotope in the mixture where the reference is placed.

Here are examples of isotope references :

```
<element name='Boron' ....>
  <isotoperef href='#Bor_10' fractionmass='0.20' />
  <isotoperef href='#Bor_11' fractionmass='0.80' />
</element>
```

7.2.9 Atoms

```
<!ELEMENT atom EMPTY>
<!ATTLIST atom A CDATA #REQUIRED
              Zeff CDATA #REQUIRED>
```

< >

atom

A : CDATA
Zeff : CDATA

This defines an atom by giving its atomic mass (A) and its atomic number (Zeff). This is not necessarily an existing atom but can also be a mixture for which the average values are given.

Here is an example of use :

```
<atom A='15.999*g/mole' Zeff='8.0000' />
```

7.2.10 Elements

```
<!ELEMENT element
  ((atom | (isotoperef*)), tabprops*)>
<!ATTLIST element
  name ID #REQUIRED
  symbol CDATA #IMPLIED
  density CDATA #IMPLIED
  radlen CDATA #IMPLIED
  lambda CDATA #IMPLIED
  %conditions;>
```

<<DTDElement>> element
name : ID symbol : CDATA density : CDATA radlen : CDATA lambda : CDATA temperature : CDATA pressure : CDATA state : (solid liquid gas undefined) = undefined

This defines an element. What we call element is either a combination of several isotopes or a single atom that constitutes an existing entity. As an example, the real Boron is not the Boron 10 nor the Boron 11 but a mixture of both.

The composition of an element is defined through its children : either a set of isotopes or a given atom. Tabulated properties may also be used if needed.

The properties of the resulting element are given by the attributes of the element :

- name : the name of the element, unique in the whole file, since it is of type ID
- symbol : the symbol of this element
- density : the density of the element



- radlen : the radiation length of the element
- lambda : ???
- conditions : this is a set of conditions for this element, including temperature and pressure. See Section 6.2.6 on page 15

Here is an example of the use of the element tag :

```
<element name='Boron'
  symbol='B'
  density='2.3400*g/cm3'
  radlen='0.00*cm'
  lambda='0.00*cm'>
  <isotoperef href='#Bor_10' fractionmass='0.20' />
  <isotoperef href='#Bor_11' fractionmass='0.80' />
</element>
```

7.2.11 References on elements

```
<!ELEMENT elementref EMPTY>
<!ATTLIST elementref
  href CDATA #REQUIRED
  natoms CDATA "-1"
  fractionmass CDATA "-1"
  classID CDATA "1202">
```

< >

elementref
href : CDATA
natoms : CDATA = -1
fractionmass : CDATA = -1
classID : CDATA = 1202

The reference mechanism is explained in detail in Section 4 on page 4. However, in the case of an element, two more attributes are defined : natoms and fractionmass. These are two ways of giving the proportion of this element in the mixture where the reference is defined. Note that the use of natoms or fractionmass implies that you use the same (either natoms or fractionmass) in every other component of the mixture where the reference appears. You can not mix both.

Here is an example of use :

```
<material name="MirrorQuartz" density="2.2*g/cm3" >
  <elementref href="#./Silicon" natoms="1" />
  <elementref href="#./Oxygen" natoms="2" />
</material>
```



7.2.12 Components

```
<!ELEMENT component EMPTY>
<!ATTLIST component
  name CDATA #REQUIRED
  natoms CDATA "-1"
  fractionmass CDATA "-1">
```

< >

component

name : CDATA
natoms : CDATA = -1
fractionmass : CDATA = -1

Components are kind of references on elements. A component points to the element whose name equals to the name attribute and defined in the GAUDI transient store under /dd/materials.

Apart from its name, a component has two attributes : natoms and fractionmass. These are two ways of giving the proportion of this component in the material where the reference is used. Note that the use of natoms or fractionmass implies that you use the same attribute in every other component of the mixture where the reference appears. You can not mix both natoms and fractionmass.

Here are some examples of use :

```
<material name='Water' density='1.0000*g/cm3' >
  <component name='Hydrogen' natoms='2' />
  <component name='Oxygen' natoms='1' />
</material>
```

```
<material name='ArgonGas' density='0.16390E-02*g/cm3' >
  <component name='Argon' natoms='1' />
</material>
```

```
<material name='Argon_CF4_CO2' density='2.14160E-3*g/cm3' >
  <component name='CF4' fractionmass='0.2' />
  <component name='CO2' fractionmass='0.5' />
  <component name='ArgonGas' fractionmass='0.3' />
</material>
```

7.2.13 Materials

```
<!ELEMENT material
  (component*,tabprops*)>
<!ATTLIST material
  name ID #REQUIRED
  Aeff CDATA #IMPLIED
  Zeff CDATA #IMPLIED
  density CDATA #IMPLIED
  radlen CDATA #IMPLIED
  lambda CDATA #IMPLIED
```

<<DTDElement>> material
name : ID
Aeff : CDATA
Zeff : CDATA
density : CDATA
radlen : CDATA
lambda : CDATA
temperature : CDATA
pressure : CDATA
state : (solid liquid gas undefined) = undefined



`%conditions;>`

This defines a real material. It is a mixture of components (given as children of it) and can have some tabulated properties. Its physical properties are defined by the attributes of the tag. Here are their meanings :

- `name` : the name of the material, unique in the whole file, since it is of type ID
- `Aeff` : the effective atomic mass of this material. This is an average value of all the components of the material.
- `Zeff` : the effective atomic number. This is an average value of all the components of the material.
- `density` : the density of the element
- `radlen` : the radiation length of the element
- `lambda` : ???
- `conditions` : this is a set of conditions for this element, including temperature and pressure. See Section 6.2.6 on page 15

Here are some examples of use :

```
<material name='Water' density='1.0000*g/cm3' >
  <component name='Hydrogen' natoms='2' />
  <component name='Oxygen' natoms='1' />
</material>
```

```
<material name='ArgonGas' density='0.16390E-02*g/cm3' >
  <component name='Argon' natoms='1' />
</material>
```

```
<material name='Argon_CF4_CO2' density='2.14160E-3*g/cm3' >
  <component name='CF4' fractionmass='0.2' />
  <component name='CO2' fractionmass='0.5' />
  <component name='ArgonGas' fractionmass='0.3' />
</material>
```

7.2.14 References on materials

```
<!ELEMENT materialref EMPTY>
```

```
<!ATTLIST materialref
```

```
  href CDATA #REQUIRED
```

```
  fractionmass CDATA #IMPLIED
```

```
  classID CDATA "1203" >
```

```
< >
```

```
materialref
```

```
href : CDATA
```

```
fractionmass : CDATA
```

```
classID : CDATA = 1203
```

The reference mechanism is explained in detail in Section 4 on page 4. However, in the case of a material, one more attribute is defined : `fractionmass`. It is used to give the proportion of this material in the mixture where the reference is defined.



Here is an example of use, taken from the description of the Rich1 :

```
<catalog name="Rich1Materials" classID = "3" >
  <materialref href="#MirrorQuartz" />
  <materialref href="#WindowQuartz" />
  <materialref href="#Aerogel" />
  <materialref href="#AerogelQuartz" />
  <materialref href="#C4F10" />
  <materialref href="#Kovar" />
</catalog>
```

8 Catalogs

This part of the DTD only defines catalogs and references on it.

8.1 Catalogs

```
<!ELEMENT catalog ANY>
<!ATTLIST catalog name ID #REQUIRED
                classID CDATA #FIXED "3">
```

<<DTDElement>> catalog
name : ID classID : CDATA = 3

This defines a catalog of elements. Since this catalog definition is used all over the dtd files, it allows any type of subelements.

A catalog has only two attributes : its name (unique in the file) and a classID that allows the converters to recognize it.

Here is an example of use, from the top XML file of the whole LHCb detector description :

```
<catalog name="dd">
  <catalogref href = "structure.xml#Structure" />
  <catalogref href = "geometry.xml#Geometry" />
  <catalogref href = "materials/materials.xml#Materials" />
</catalog>
```

8.2 References on catalogs

```
<!ELEMENT catalogref EMPTY>
<!ATTLIST catalogref classID CDATA #FIXED "3"
                    href CDATA #REQUIRED>
```

<<>>
catalogref
classID : CDATA = 3 href : CDATA

The reference mechanism is explained in detail in Section 4 on page 4.



Here is an example of a reference for a catalog, from the top XML file of the whole LHCb detector description :

```
<catalog name="dd">
  <catalogref href = "structure.xml#Structure" />
  <catalogref href = "geometry.xml#Geometry" />
  <catalogref href = "materials/materials.xml#Materials" />
</catalog>
```

9 Surfaces

This part of the DTD only defines surfaces and references on it.

9.1 Surfaces

```
<!ELEMENT surface (tabprops*)>
<!ATTLIST surface name ID #REQUIRED
                 classID CDATA #FIXED "1110"
                 model CDATA #REQUIRED
                 finish CDATA #REQUIRED
                 type CDATA #REQUIRED
                 value CDATA #REQUIRED
                 volfirst CDATA #REQUIRED
                 volsecond CDATA "\0" >
```

<<DTDElement>> surface
name : ID
classID : CDATA = 1110
model : CDATA
finish : CDATA
type : CDATA
value : CDATA
volfirst : CDATA
volsecond : CDATA = \0

This tag defines a surface. The only children of it are tabulated properties and here is the description of the attributes :

- name : the name of the surface (unique in the file).
- classID : this is for the converters.
- model : ???
- finish : ???
- type : ???
- value : ???
- volfirst : ???
- volsecond : ???

Here is an example of use, taken from the description of the Rich1 :

```
<surface name="Rich1MirrorSurfaceQuad0Sect1"
         model="glisur"
         finish="polished"
```



```

    type="dielectric_metal"
    value="0"
    volfirst="/dd/Geometry/Rich1/lvRich1Master#pvRich1GasEnclosure"
    volsecond=
        "/dd/Geometry/Rich1/lvRich1Master#pvRich1MirrorQuad0Sect1">
<tabprops address=
    "/dd/Geometry/Rich1/Rich1Surfaces/MirrorSurfaceReflectivityPT" />
<tabprops address=
    "/dd/Geometry/Rich1/Rich1Surfaces/MirrorSurfaceEfficiencyPT" />
</surface>

```

9.2 References on surfaces

```

<!ELEMENT surfaceref EMPTY>
<!ATTLIST surfaceref
    href CDATA #REQUIRED
    classID CDATA #FIXED "1110">

```

< >

surfaceref
href : CDATA
classID : CDATA = 1110

The reference mechanism is explained in detail in Section 4 on page 4.

Here is an example of a reference for a surface, taken from the description of the Rich1 :

```
<surfaceref href="#Rich1MirrorSurfaceQuad0Sect1" />
```

10 Tabulated Properties

This part of the DTD only defines tabulated properties and references on them.

10.1 Entries

```

<!ELEMENT entry EMPTY>
<!ATTLIST entry x CDATA #REQUIRED
    y CDATA #REQUIRED
    text CDATA #IMPLIED>

```

< >

entry
x : CDATA
y : CDATA
text : CDATA

This defines an entry in a tabulated property. As any entry of a tabulated property, it has 2 values : *x* and *y*. On top of it, some comment can be put in the *text* field.

Here is an example of use :

```
<entry x = "4.0" y = "0.0" />
```



10.2 Tabulated Properties

```
<!ELEMENT tabproperty (#PCDATA | entry)* >
<!ATTLIST tabproperty name ID #REQUIRED
                    classID CDATA #FIXED "1300"
                    type CDATA #REQUIRED
                    xaxis CDATA "xAxis"
                    yaxis CDATA "yAxis"
                    xunit CDATA "1.0"
                    yunit CDATA "1.0">
```

<<DTDElement>> tabproperty
name : ID
classID : CDATA = 1300
type : CDATA
xaxis : CDATA = xAxis
yaxis : CDATA = yAxis
xunit : CDATA = 1.0
yunit : CDATA = 1.0

This defines a tabulated property. This is used to describe optical properties of materials (see dedicated document "Optical Properties and Surfaces", by Ivan Belyaev, available at <http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/Documents/Optical.pdf>). It is a kind of 1D histogram.

Tabulated properties have many attributes :

- name : their name, unique in the whole file
- classId : an identifier used by converters.
- type : the kind of property defined here
- xaxis : the name of the X axis
- yaxis : the name of the Y axis
- xunit : the unit used on the X axis
- yunit : the unit used on the Y axis

There are two ways of filling a tabulated property. One is to add several entries, as child of the <tabproperty> tag, the other is to write directly the values as plain text in the XML file. These values are supposed to go by two, x first, y second. They must be separated by spaces and/or carriage return only.

Here are two examples of use, taken from the description of Rich1 :

```
<tabproperty name = "QuartzWindowSurfaceEfficiencyPT"
            type = "EFFICIENCY"
            xunit = "eV"
            xaxis = "PhotonWaveLength"
            yaxis = "QuartzWindowSurfaceEfficiency" >
```

```
1.0 0.0
2.0 0.0
3.0 0.0
4.0 0.0
5.0 0.0
6.0 0.0
7.0 0.0
```



```

      8.0 0.0
      9.0 0.0
      10.0 0.0
</tabproperty>
<tabproperty name="KovarAbsLengthPT"
             type="ABSLENGTH"
             xunit ="eV"
             yunit ="mm"
             xaxis ="PhotonMomentum"
             yaxis ="AbsortionLength" >
  <entry x = "1.5" y = "0.0" />
  <entry x = "2.0" y = "0.0" />
  <entry x = "2.5" y = "0.0" />
  <entry x = "3.0" y = "0.0" />
  <entry x = "3.5" y = "0.0" />
  <entry x = "4.0" y = "0.0" />
  <entry x = "4.5" y = "0.0" />
  <entry x = "5.0" y = "0.0" />
  <entry x = "5.5" y = "0.0" />
  <entry x = "6.0" y = "0.0" />
  <entry x = "6.5" y = "0.0" />
  <entry x = "7.0" y = "0.0" />
  <entry x = "7.5" y = "0.0" />
</tabproperty>

```

10.3 References on tabulated properties

```

<!ELEMENT tabpropertyref EMPTY>
<!ATTLIST tabpropertyref
          href CDATA #REQUIRED
          classID CDATA "1300">

```

< >

```

tabpropertyref
  href : CDATA
  classID : CDATA = 1300

```

The reference mechanism is explained in detail in Section 4 on page 4.

Here is an example of a reference for tabulated properties, taken from the description of Rich1 :

```

<catalog name="Rich1Materials" classID = "3" >
  <materialref href="#MirrorQuartz" />
  <tabpropertyref href="#MirrorQuartzAbsLengthPT" />
  <tabpropertyref href="#QuartzRIndexPT" />

```



```

<materialref href="#WindowQuartz" />
<tabpropertyref href="#WindowQuartzAbsLengthPT" />
...
</catalog>

```

10.4 Tab props

```
<!ELEMENT tabprops EMPTY>
```

```
< >
```

```
<!ATTLIST tabprops address CDATA #REQUIRED >
```

```
tabprops
```

```
address : CDATA
```

There are actually two kinds of references on tabulated properties.

The one is called `<tabpropertyref>` and is a regular reference, using files and XML IDs. The other is this `<tabprops>` tag that uses the GAUDI transient data store to name the tabulated property.

This tag has no child and only one attribute : the name of the pointed tabulated property. This name is the transient store name, thus something like `"/dd/Geometry/SubDetectorName/..."`

Here is an example of the use of this tag, taken from the Rich1 description :

```

<tabprops address=
    "/dd/Geometry/Rich1/Rich1Surfaces/MirrorSurfaceReflectivityPT" />

```



