An XML generic detector description system and geometry editor for the ATLAS detector at the LHC

Laurent Chevalier[1], Andrea Dell'Acqua[2], Jochen Meyer[3] on behalf of the ATLAS collaboration

[1] CEA Saclay (Commissariat à l'Energie Atomique), Gif-sur-Yvette, France
[2] CERN, Geneva, Switzerland
[3] Julius-Maximilians-Universität, Würzburg, Germany

E-mail: laurent.chevalier@cea.fr, Andrea.Dellacqua@cern.ch, Jochen.Meyer@physik.uni-wuerzburg.de

Abstract. In this paper we describe a software package which was developed to describe the ATLAS muon spectrometer. The package is based on a generic `XML` detector description (ATLAS Generic Detector Description, `AGDD`), and is used in the *PERSINT* visualization program and in a series of parsers, or converters which build a generic, transient geometry model which can be translated into commonly used geometry descriptions like Geant4, the ATLAS `GeoModel`, ROOT `TGeo` or others. The system presented allows for an easy, self descriptive approach to the detector description problem, for intuitive visualization and rapid turn-around: indeed, the results of the description process can be immediately fed into e.g. a Geant4 simulation for rapid prototyping. Examples of the current usage for the ATLAS detector description will be given and further developments needed to meet future requirements.

ATL-SOFT-PROC-2012-053
21/06/2012

1. Introduction

Accurate and detailed descriptions of the HEP detectors are turning out to be crucial elements of the software chains used for simulation, visualization and reconstruction programs. For this reason, it is of paramount importance to dispose of and to deploy generic detector description tools which allow for precise modeling, visualization, visual debugging and interactivity. In addition they should be capable of feeding information in e.g. Geant4 [1, 2] based simulation programs and in reconstruction-oriented geometry models. At the same time, these tools must allow for different levels of descriptions, ranging from very accurate geometries aimed at very precise Geant simulation to more generic descriptions of scattering centers in a track reconstruction program.

Such a system was developed a few years ago and is currently in use for the dead material like support structures and services of the ATLAS muon spectrometer. In spite of other parts of the ATLAS experiment [3] such as the inner detectors, calorimeters or the actual detectors of the muon spectrometer, the dead material is profoundly irregular, asymmetric and inhomogeneous. Apart from that the dimensions of the structures entering the description extend from millimeters up to several meters. This fact makes the implementation further more difficult since it requires accurate positioning down to the millimeter level. Of course common simulation programs are able to fit such demands and can display their explicit geometry representation as well, but in most cases only if the underlying detector description implementation is free of any conflicts.

In the following we describe the main functionality and benefit of the ATLAS Generic Detector Description (AGDD) project [4] and especially focus on its use case in the ATLAS common software framework Athena [5] by the package AGDD2Geo which is used since a while. After that we partially explain the syntax of the input file that follows the makeup language XML. We point out the advantage of starting with a generic format that already sticks to basic programming conventions. Along with other visualization options we finally introduce the program "Perspective Interactive" (*PERSINT*) [6, 7] while here we highlight dedicated features that are of tremendous importance for debugging the XML input file. A brief section on derived geometrical representations for fast tracking and simulation will follow. In the conclusions we give apart from a short summary a rather detailed outlook for further developments and future fields of application.

2. ATLAS Generic Detector Description

To understand the basic mechanisms and various benefits coming along with an ATLAS Generic Detector Description (AGDD) we firstly give a short description of the current situation. The common framework Athena provides amongst others the capability of Geant4 based simulation which is used by ATLAS for Monte Carlo studies. The detector simulation comprises all sub detectors and is very accurate. To supply a uniform format of the detector descriptions to the Geant4 program the Athena package GeoModel [8] is used. Each of the subdetectors' description is done using GeoModel classes for volume, material and position representatives which are rather close to the Geant4 classes. On the intermediate level of GeoModel some optimization mainly regarding memory is performed.

Before migrating to Athena and GeoModel the ATLAS muon spectrometer detector description was implemented [4, 9] in bare Geant4, but already using kind of generic representatives. At this stage and later by using GeoModel all structures are implemented in C++ and therefore hard coded utilizing the corresponding classes. Only variables like dimensions and conditional positions are manipulable after the compiling step since they are retrieved from an external source, in this case from a database. The format in which these quantities are stored follows the ATLAS Muon DataBase (AMDB) [10] convention. Solving the detector description problem hereby has the advantage that especially the very regular active parts can be enveloped by bigger volumes so that e.g. not every single tube of the monitored drift tubes has to be described.

On the other hand it is very tough to describe complex structures like the dead material of the muon spectrometer via this mechanism since it is not uncommon that shapes have to be redesigned in more detail or structures have to be added to reproduce reality more accurately. Therefore it is necessary to implement the new structures using GeoModel classes and to add new columns or rows in the database and fill them. In terms of duty cycles and maintenance this obviously is a time consuming effort. It turns out to be more useful following an approach that neither requires hard coded structure building nor fixed paths to values. With the generic detector description using XML a first step is done due to a reliable syntax of the input. The next steps are parsers and converters to produce a representation that can be supplied to GeoModel in our case or to any other visitor program in general. This is done by the package AGDD2Geo which is currently available as part of the Athena framework in terms of a service. The regarding tools were already developed since a while and became more established during the last years.

The TinyXML [11] parser is used for reading the XML ASCII code. The location of the input file can be arbitrary because the path is specified in the python configuration of the service on the Athena initialization level. Besides a database, it is also possible to point to a file in the local working area. The following handler holds for each of the XML elements a class that is used to build a new object according to the read element. Aspects like memory fragmentation might become an issue if the system grows bigger, but at the current status it is as good as

possible. In the next most essential step the generic detector description representation is built by using volume and positioner classes of AGDD. Here volumes get already equipped with material properties. This operation is greatly flexible and configured during the initialization process as well. It is possible to specify single volumes, structures, full sections or everything available to be built. To save memory the volume hierarchy does not strictly match a tree structure since branches can be reclaimed or merged. The building process though follows a tree like flow.

With this generic detector representation in memory it is possible to satisfy any needs of visitor programs and provide them a convenient description in terms of their classes, respectively volumes and positioners. In the case of AGDD2Geo a AGDD2GeoModelBuilder class creates the according GeoModel representation that enters the flow foreseen by Athena. Apart from that a AGDD2G4ModelBuilder is available that can produce decent input for direct use in Geant4. Some results for those two builders are shown in section 4.

It should be remarked that in principle there is no restriction to use only one input file. It is possible to select in the configuration step volumes or sections of various files to enter the final model. In addition a (back) translation of already existing code of a visitor program into XML is feasible as well. From foregoing explanations it should become clear that this approach of constructing a generic detector description has many advantages as there are various visitor programs possible, the input file and the final representation can be changed very easily without touching code that has to be compiled and finally new elements can be added in a convenient way by creating AGDD classes accordingly.

3. Input based on XML

The idea of a generic detector description available for various applications is not novel [4] in ATLAS as mentioned. In the early days the XML format was just considered to give the ASCII code a convenient structure while most of the features provided by this markup language were secondary. Amongst others the existence of powerful open source parser like TinyXML or Xerces [12] which also include efficient syntax debuggers led to the decision to use the full capability of XML. In the realized attempt each element in the XML corresponds to either a basic single volume with dimensions and a specific material attributed (see Appendix A.1) or a composite volume (see Appendix A.2) built out of basic single ones and/or other composite ones, respectively. These grouped volumes can on the one hand steer a simple repositioning of the involved objects or on the other hand initiate a boolean operation to create a new single volume. The composite volume definition follows in some sense the concept of mother volumes known from Geant4.

To allow a clearer structure of the file and easier adjustments of dimensions or positions there is an element `<var name="variable" value="10."/>` that holds a concrete value (here `10.`) or gives a formula to calculate a value using other `var` elements. Apart form the basic arithmetic operations also trigonometric operations are supported by the AGDD parser and *PERSINT* program. For iteration operations there is an element that holds several values `<array name="Array" values="0.; 3000.; 15000."/>`. The following example demonstrates a repeated placement of a volume `Box` using a loop iteration.

```
<foreach  index="Inum"  begin="0"  loops="3" >
  <posXYZ  volume="Box"  X_Y_Z=" 0.; 0. ; Array[Inum]" />
</foreach>
```

After the introduction of basic elements we proceed with explanations on the structure, respectively organization of an XML holding the generic detector description. As usual for XML, the ASCII file starts with the XML declaration followed by the tag which indicates the type of the content (in our case is AGDD).

```
<?xml version="1.0"?>
<AGDD>

...

</AGDD>
```

Within the main tag there are sub-tags, called `section`, to separate major structures like for the ATLAS muon spectrometer the barrel toroid, the endcap toroids, the feet and so forth. The attribute `name` serves to address the element. For the building process in the `AGDD` package the attribute `top_volume` is meaningful. If it does not determine any volume all top most volumes are built, i.e. all volumes that are not entering any other element. The remaining attributes have no other purpose than documentation.

```
<section name       = "ExamplesForVolumes"
         version    = "1.0"
         date       = "27 July 2010"
         author     = "somebody"
         top_volume = "MergedVolume">

...

</section>
```

It should be mentioned that the `XML` file will be read chronologically by the `AGDD` parser or *PERSINT*. As a consequence of this chronological reading, once an element is created, it remains known later on even if the `section` is closed and a new one is started. Simply put this means a global name-space can be defined to hold frequently used constants (e.g. π, $\sqrt{2}$, $\sin 15^\circ$, ...). However the danger of overwriting is given if the elements are not named uniquely. To detect such bugs the mentioned open source tools can be used. The Document Type Definition (DTD) file which gives the grammar of the `XML` helps for debugging as well. Though it is a general features of `XML` it should be stressed that the flexibility to extend the number of elements to anybody's need is given. In case albeit the subsequent parsers and converters have to be adjusted.

To give an idea of the final size of such an `XML ASCII` file we present in Table 1 the current amount of objects which are actually used to describe the dead material of the ATLAS muon spectrometer. Including also prototype structures and comments the current version consists of approximately 10 thousand lines of code.

4. Visualization and visual debugging

At least of the same importance than the debugging of the bare syntax is to test the eventually built description for its correctness. Two aspects play a crucial role in this context: Firstly it has to be made sure that the chain of parsers and converters which construct the final geometrical

Table 1. Number of elements used to describe the ATLAS muon spectrometer in `XML` disentangled in different categories.

Category	amount
variables	1442
basic single volumes	498
simple composite volumes	265
composite boolean volumes	92
sections	14

representation do not misinterpret any of the elements or their attributes. Secondly it has to be checked that there are no inconsistencies like overlapping volumes or touching surfaces in the final model since such conflicts affect substantially the stability of simulation programs like Geant4. Either of these two issues can be addressed most comfortably by actually looking at the visualized model of the detector description.

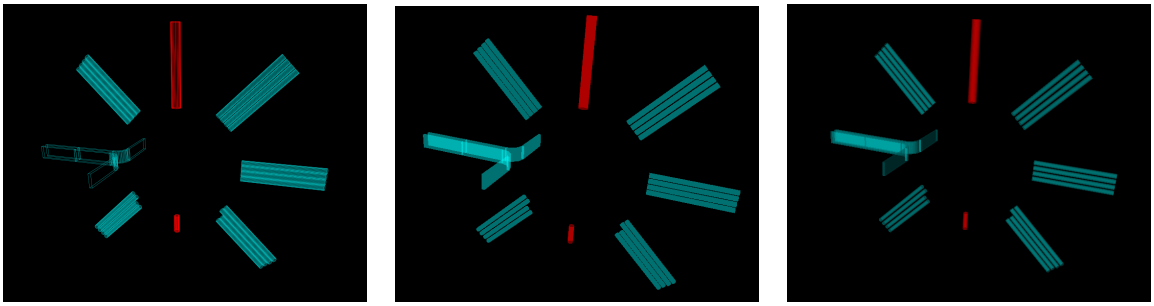
To display the description there are several options that all together give a perfect combination for a comprehensive debugging. The most handy application to run is Perspective Interactive (*PERSINT*). This completely stand alone program works on all common operating systems and can be installed rather easily. The handling is very intuitive and there is a manifold of features starting from the mere visualization of the ATLAS generic detector description which we are highlighting in this paper, displaying full events and even performing fast simulation. More information is available in references [6, 7].

The `XML ASCII` file is interpreted by an internal parser of *PERSINT* and converted to a QtRoot model whose three dimensional representation is displayed. The user is able to interactively change the perspective by e.g. rotating and zooming and can choose between an illustration of solid volumes or just wired envelopes. By selecting the volumes in a recursive volume tree it can be decided which structures to visualize. In addition it is possible to arbitrarily colorize and to move volumes on the fly to get a better view at potentially critical regions. Apart from manually searching for such regions *PERSINT* holds a function to automatically check for overlaps and report them. Depending on the complexity of the implemented structures it at times becomes helpful to disable the boolean processor for those volumes that make use of it. This can be done by checking a single switch in the GUI.

All operations including the read-in of the `XML` file proceed without delays for e.g. loading. Therefore *PERSINT* is the first tool to make visual debugging even before inserting the file in `AGDD2Geo`, respectively the Athena framework. The immediate visual feedback of this program is tremendously important during the implementation of new components. A picture of the example service structure given in Appendix B directly exported from *PERSINT* is given in figure 1(a).

Inside Athena there is "virtual point one" [13] (*VP1*) which is another visualization tool using Qt libraries to display a geometrical representation. Of course the functionality of this package extends as well the bare visualization and includes many other features not listed here, e.g. full event display. As mentioned before Athena uses the `GeoModel` package to describe the full ATLAS detector while the `XML` description is processed by the `AGDD2Geo` package. Therefore *VP1* can be used to debug the `AGDD2Geo` converter in addition to the visual search for conflicting volumes. The second item though is more time consuming in *VP1* in the sense that with every change in `XML` the full Athena initialization process has to be restarted while in *PERSINT* only the input file has to be reloaded via the GUI without any restart. Therefore the profit of using *PERSINT* is a reduction of human time rather than a gain in software performance since there is hardly a difference. In addition *VP1* provides less options to manipulate the displayed volumes. In figure 1(b) we present a picture of the example structure exported from *VP1*.

For a comprehensive validation of the generic detector description used in the ATLAS software both tools, *PERSINT* and *VP1*, are mandatory since they cover slightly different aspects. Regarding the application of `AGDD` outside any ATLAS software framework we present in figure 1(c) a picture of the example service structure produced by the Geant4 visualization package. For this a `AGDD2G4` handler was used which works on the `AGDD` classes. With figure 1 we summarize that it is possible to produce approximately equivalent displays with all mentioned visualization packages. On the one hand this means that `AGDD2Geo` works properly inside Athena and on the other hand this result emphasizes the high flexibility of `AGDD` in general and its viability in many applications.



(a) *Persint* (b) *VP1* (c) Geant4

Figure 1. Displays of the sample `XML` of Appendix B produced with various visualization programs.

5. Implications for derived geometrical representations

For tracking or fast simulation purposes the demand on the geometrical representation is in contrast to full simulation not high accuracy but speed. This can be achieved by reducing the variety of different materials and the number of volume boundaries and group volumes in bigger envelopes which approximate the amount of material in terms of scattering centers. Due to the high complexity of the ATLAS muon spectrometer and its inhomogeneously distributed structures the common cylinder based approach in use in the inner detector, is not viable. For fast tracking in the muon spectrometer two independent tools had been developed to produce such a light detector description.

Firstly there is the project Muonbox [14] which is also part of the *PERSINT* program. Here the geometrical information for dead material are directly derived from the `XML` file while information on the active part is converted from `AMDB` [10]. Via internal algorithms a simple geometrical representation is produced. The stand alone character of these tools allows separate tuning especially on the material properties. This is especially useful to compensate for the missing structures if an incomplete underlying description is assumed.

On the other hand there is a tracking geometry package [15] available in Athena. Here also internal mechanisms generate a fast detector representation in this case based on the `GeoModel` description. The advantage is a very good agreement to the material distribution used in the ATLAS Monte Carlo simulation. Once the detailed detector description is very close to reality the fast description produced by this package is also very good since no special tuning is needed. In the current ATLAS muon reconstruction both fast representations are use to have the best muon reconstruction performance and good validation.

6. Conclusions

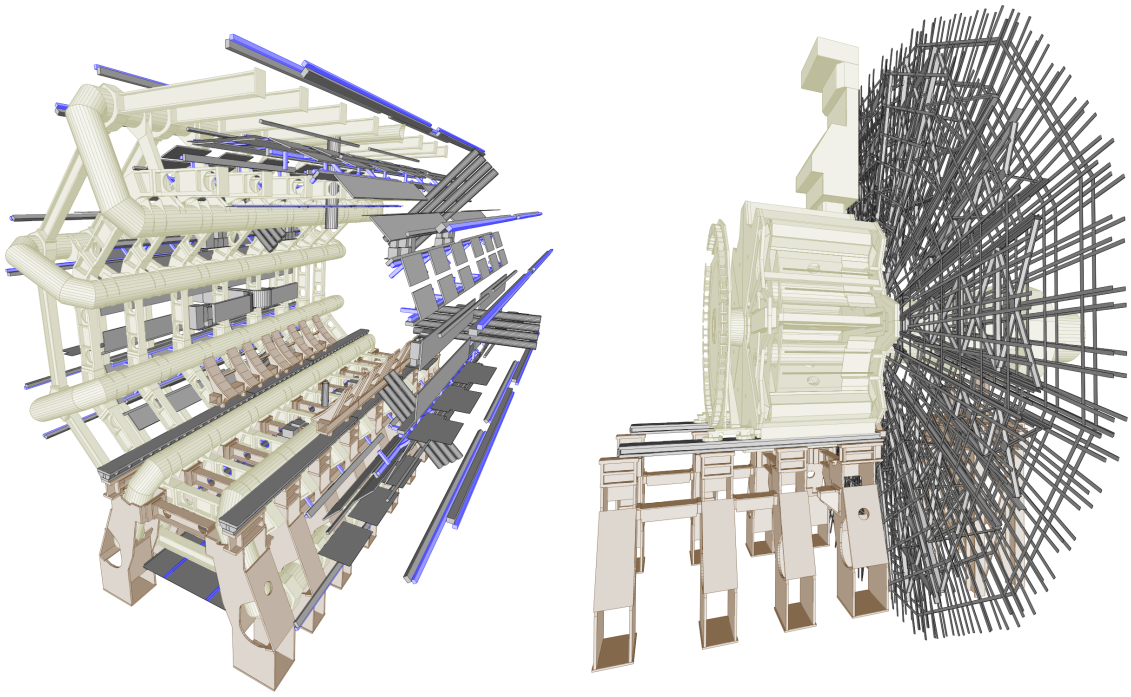
In this paper we pointed out the benefits of the ATLAS Generic Detector Description. We introduced the structure of the `XML` based input file and explained in details and how the description is made available for the software framework Athena by `AGDD2Geo` or for other programs. It was stressed that apart from syntax debugging a visual debugging serves several crucial intents. Therefore we presented various options to display the geometric representation. Finally two packages to generate a less detailed but faster description were mentioned.

The plans for future developments of `AGDD` and also its application within the ATLAS software are quite extensive. In a next step the implementation of additional builder classes is foreseen to provide the `XML` based description also to other programs like ROOT [16]. The capability for back translation will be increased, too. Going along with that the number of available volume types, respectively elements will be enlarged to have at some point all shapes of Geant4 available

as well. Possibly some efforts will be made to decouple the AGDD parser chain from Athena to enhance the amount of use cases.

For the ATLAS detector description it is foreseen to migrate more structures to the XML based attempt. Two main domains to start with are the cavern hall representation and the active parts of the ATLAS muon spectrometer. Since for the latter one also handling of sensitive detectors becomes mandatory corresponding classes have to be constructed. These have to combine the high number of volumes with their very regular disposal.

As the closing remark we present in figure 2 all structures of the ATLAS muon spectrometer currently described by the AGDD.



(a) barrel structures (b) endcap structures

Figure 2. Structures of the ATLAS muon spectrometer described via AGDD displayed by *PERSINT*. To give a better view and display the complexity of the description some volumes had been removed.

7. Acknowledgements

Jochen Meyer acknowledges the support of BMBF, Germany.

Appendix A. Elements in XML

In this section we give an overview on elements, respectively kinds of volumes, that are currently supported by the handler of the Athena package AGDD2Geo and also by *PERSINT*. They are functional if put in a XML ASCII and read by the either the applications. Firstly we list in Appendix A.1 elements which describe basic single volumes and do a priori not have any dependence on other elements than possibly var. The opposite is the case for the elements of the part Appendix A.2 which have relations to other volume like elements which have to be defined in advance. All elements have the attribute name so that they can be addressed properly. The attribute material is reserved the basic single volumes.

Appendix A.1. Basic single volumes

cube : A box like volume is defined by the lengths of the edges. This is done via the three values of `X_Y_Z`. The first value is the dilation along x, the second one along y and the third one along z axis. The origin of the coordinate system is in the center of the box. The following example is illustrated in the figure A.1(a).

```
<box name="Box" material="Aluminium" X_Y_Z="2500.; 2000.; 1500."/>
```

tube: The dimensions of the tube are put in `Rio_Z`. Here the first value is the inner while the second one is the outer radius. The third value is the dilation along the z axis. Again the origin of the coordinate system is in the center of the object. The first value of the optional attribute `profile` is the angle where the tube starts while the second one indicates the angluar dilation in degrees. If `profile` is missing a closed tube around the z axis is built. The final attribute `nbPhi` is important for *PERSINT* only and determines the number of surfaces used to approximate the round shape. The following example is illustrated in figure A.1(b).

```
<tubs  name="Tube" material="Iron" Rio_Z="1000.; 1250.; 1500." profile="45.;225." nbPhi="20"/>
```

pyramid: The shape itself is defined by the five values of `Xmp_Ymp_Z`. These values determine the following lengths: The first one is the dilation of the edge along x axis in negative (minus) z while the second one is the similar but for positive (plus) z. The next two values are the dilation along y axis for negative and positive z respectively. The last value is the complete length along z axis. Since the smaller quadrangle in xy plane is always center with respect to bigger one the z axis is in the centered of the quadrangles. The pyramid's dilation in negative/positive is the half of the complete z length. So the origin of the coordinate system is in the center of the object. figure A.1(c) is the illustration of the following example.

```
<trd   name="Pyramid" material="Aluminium"  Xmp_Ymp_Z="2000.; 1000.; 2500.; 1250.; 1500.0" />
```

cylinder: Since a cylinder is a round object the already introduced attribute `nbPhi` has to be declared as well. After that one defines several `polyplans` determined by the three values `Rio_Z`. The first values gives the inner radius while the second one is the outer radius. The third value is the z position of this ring. The final cylinder is the connection of all orbital planes. The thickness is adjusted to fulfill the conditions given by the rings. In figure A.1(d) the following example is illustrated.

```
<pcon name="Cone" material="PolyBoron5percent" nbPhi="20">
  <polyplane Rio_Z="1000.; 1250.;  750."/>
  <polyplane Rio_Z=" 750.; 1000.;    0."/>
  <polyplane Rio_Z=" 250.;  500.; -750."/>
</pcon>
```

chain: A chain is nothing else than a tube with one or more kinks that don't need to be 90 degrees like in the following example which is illustrated in figure A.1(e).

```
<snake  name="Chain" material="Brass" radius="300." nbPhi="20">
  <snake_point  X_Y_Z="-750.; -750.; -750." />
  <snake_point  X_Y_Z="-750.; -750.;  500." />
  <snake_point  X_Y_Z="-750.;  750.;  500." />
  <snake_point  X_Y_Z=" 750.;  750.;  500." />
</snake>
```

In addition to `nbPhi` one has to declare the radius at the beginning. For this object there is only a outer radius. Once this is done one gives several points in `X_Y_Z` in ordinary coordinates. The final structure is the connection of all these points with a tube of given radius. The kinks are described the way that half of the angle is part of the incoming while the other half is part of the outgoing tube.

arbitrary polygon: Firstly one determines the length along the z axis via the variable `dZ`. Once this is done it is possible to define arbitrary many points in the xy plane. The volume is built by connecting the points one after the other and then pulling the plane in half `dZ` in positive and half `dZ` in negative z direction. The following example is displayed in figure A.1(f).

```
<gvxy name="Polygon" material="Copper" dZ="1500.">
  <gvxy_point X_Y="-1250.;-1000."/>
  <gvxy_point X_Y="0.;1000."/>
  <gvxy_point X_Y="1250.;250."/>
  <gvxy_point X_Y="250.;-750."/>
</gvxy>
```

symmetric polygon: This type is defined like the arbitrary polygon with the exception that all points should be either negative or positive in the x coordinate. The volume is built using not only the defined points for the xy plane, but also their mirror image with respect to zy plane, i.e. also with x coordinate of opposite sign. After that it is stretched in z direction like the arbitrary polygon. The example in displayed in figure A.1(g).

```
<gvxysx name="SymmetricPolygon" material="Carbon" dZ="1500.">
  <gvxy_point X_Y="-1250.;1000."/>
  <gvxy_point X_Y=" -500.;-750."/>
</gvxysx>
```

double symmetric polygon: This basic single volume is in principle like the symmetric polygon, but the defined points are not only mirrored with respect to zy plane but also with respect to zx plane. So the volume is completely determined with points in one sector. Figure A.1(h) shows the following example.

```
<gvxysxy name="DoubleSymmetricPolygon" material="Lead" dZ="1500.">
  <gvxy_point X_Y="-1250.; 250."/>
  <gvxy_point X_Y=" -500.;1000."/>
</gvxysxy>
```

Appendix A.2. Composite volumes

These elements are used to relocate or manipulate already defined volumes. It is not mandatory that the used elements have to be basic single volumes. In all cases the origin of the volume is translated and rotations are also done with respect to the axis through the origin of the volume. This should especially be kept in mind if one works with polygons or already composite volumes.

combined: The most simple way to reorganize volumes is their combination. The following lines give an example (see figure A.1(i)):

```
<composition name="Combined">
  <posXYZ  volume="Pyramid" X_Y_Z=" 0.; -800.; 0."  rot="270.; 0.;0."/>
  <posXYZ  volume="Pyramid" X_Y_Z=" 0.;  800.; 0."  rot=" 90.; 0.;0."/>
</composition>
```

It is possible to group arbitrary many volumes. Thereby one volume can be used more than once and be relocated in different ways. The volume allocates memory only once while different positioners are applied and stored in memory as well. If `X_Y_Z` what is the translation in x, y and z direction is not defined the origin of the volume will not be moved. The same is valid for `rot` what is the rotation of the volume with respect to x, y and z axis. These attributes have the same meaning for the following volumes, respectively tags, too. A priori this combination of volumes allows overlaps, so one has to take care by oneself to avoid these conflicts.

merged: The merged volume is in most aspects similar to the combined one. The most crucial difference is that overlapping volumes are merged and become one. This merging initiates a recalculation of the surface of the arising new volume and makes use of the boolean processor. Therefore this volume type is more CPU extensive during the building of the model but consumes less memory. Obviously the attributed material of the entering elements has to be the same. In figure A.1(j) the following example is displayed as wired model. This shows that the surface of the pyramid is merged with the one of the box.

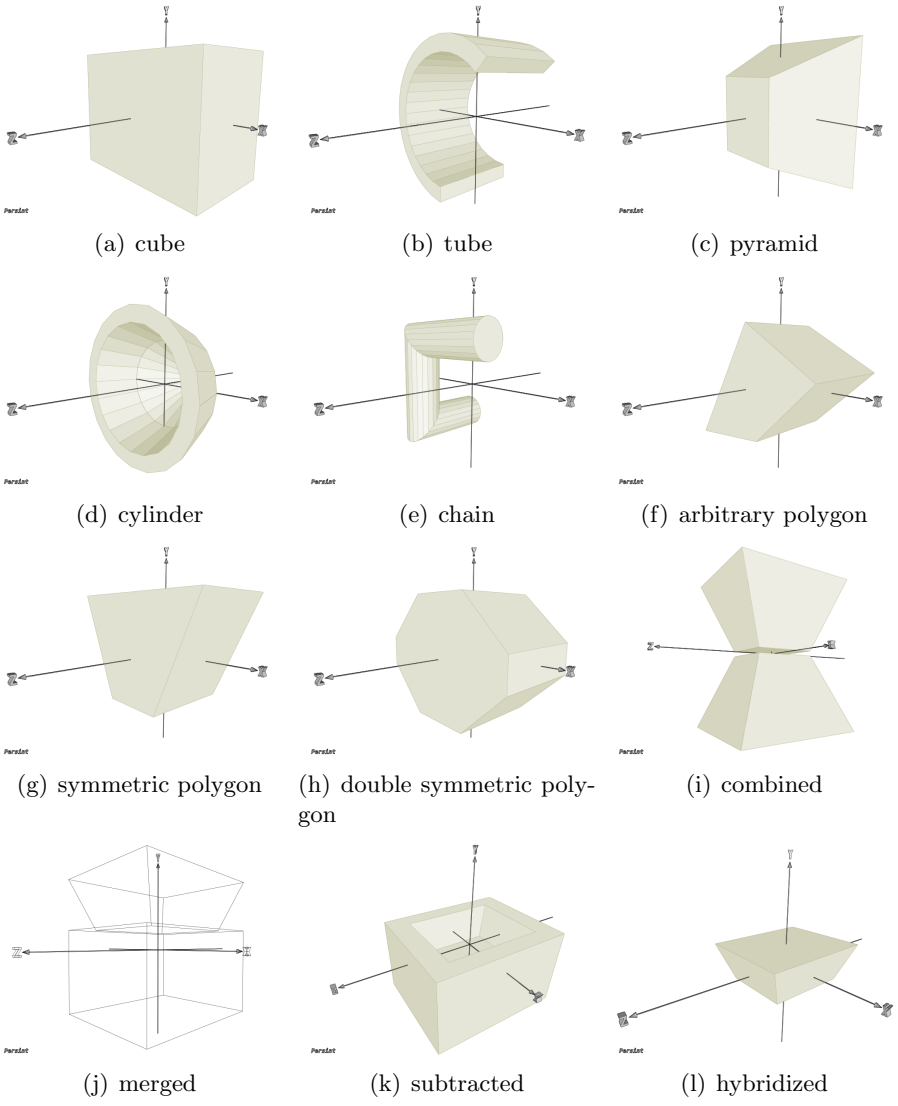
```
<union name="Merged" >
  <posXYZ volume="Pyramid" X_Y_Z=" 0.;  500.; 0." rot=" 90.;  0.;0."/>
  <posXYZ volume="Box"     X_Y_Z=" 0.; -400.; 0." rot=" 90.; 90.;0."/>
</union>
```

subtracted: This combination takes the first volume listed and subtracts all the following ones. For this operation the boolean processor is needed. See figure A.1(k) for the example.

```
<subtraction name="Subtraction" >
  <posXYZ volume="Box"     X_Y_Z=" 0.; -400.; 0." rot=" 90.; 90.;0."/>
  <posXYZ volume="Pyramid" X_Y_Z=" 0.;  500.; 0." rot=" 90.;  0.;0."/>
</subtraction>
```

hybridized: The hybridized volume is the most complex one and requires the boolean processor. As a matter of principle it combines only two volumes. It is built such that all corner points that are placed inside a volume as well as all the corner points of the cutting plane of the volumes are used. Figure A.1(l) shows the example.

```
<intersection name="Hybrid" >
  <posXYZ volume="Box"     X_Y_Z=" 0.; -400.; 0." rot=" 90.; 90.;0."/>
  <posXYZ volume="Pyramid" X_Y_Z=" 0.;  500.; 0." rot=" 90.;  0.;0."/>
</intersection>
```



(a) cube (b) tube (c) pyramid

(d) cylinder (e) chain (f) arbitrary polygon

(g) symmetric polygon (h) double symmetric polygon (i) combined

(j) merged (k) subtracted (l) hybridized

Figure A1. Available elements. See the text for explanations.

Appendix B. Example in XML

Following we present the XML code of the of the components displayed in figure 1

```
<section name       = "Eta0Services"
         version    = "7.0"
         date       = "29 Mar 2010"
         author     = "Laurent Chevalier"
         top_volume = "servicesAtZ0">

<tubs  name="EMCaloPipeI"   material="Iron"       Rio_Z=" 310.; 320.; 5500."    nbPhi=" 20." />
<tubs  name="EMCaloPipeO"   material="Iron"       Rio_Z=" 190.; 200.; 1400."    nbPhi=" 20." />
<tubs  name="CurvedCable"   material="Aluminium"  Rio_Z="   0.; 170.; 5500."    nbPhi=" 20." />
<tubs  name="CurvedCabl0"   material="Aluminium"  Rio_Z="   0.; 170.; 5000."    nbPhi=" 20." />

<composition  name="services0" >
    <posXYZ  volume="CurvedCable"    X_Y_Z=" 0.;  601.; -100."  rot=" 0.; 90.; 0." />
    <posXYZ  volume="CurvedCable"    X_Y_Z=" 0.;  201.; -100."  rot=" 0.; 90.; 0." />
    <posXYZ  volume="CurvedCable"    X_Y_Z=" 0.; -201.; -100."  rot=" 0.; 90.; 0." />
    <posXYZ  volume="CurvedCable"    X_Y_Z=" 0.; -601.; -100."  rot=" 0.; 90.; 0." />
</composition>

<composition  name="services1" >
    <posRPhiZ  volume="services0"    R_Phi_Z=" 7500.; 0. ; 0." />
</composition>

<composition  name="services2" >
    <posXYZ  volume="CurvedCabl0"    X_Y_Z="  200;  601.; -100."  rot=" 0.; 90.; 0." />
    <posXYZ  volume="CurvedCabl0"    X_Y_Z="  200;  201.; -100."  rot=" 0.; 90.; 0." />
    <posXYZ  volume="CurvedCable"    X_Y_Z="    0; -201.; -100."  rot=" 0.; 90.; 0." />
    <posXYZ  volume="CurvedCable"    X_Y_Z="    0; -601.; -100."  rot=" 0.; 90.; 0." />
</composition>

<composition  name="services3" >
    <posRPhiZ  volume="services2"    R_Phi_Z=" 7500.; 0. ; 0." />
</composition>

<box  name="Flexible" material="Aluminium"  X_Y_Z=" 200.; 1000.; 3200." />
<tubs name="Curving"  material="Aluminium"  Rio_Z=" 800.; 1000.; 1000."  profile=" 0.; 90." />

<composition name="Chain">
   <posXYZ volume="Flexible"  X_Y_Z="   900.; 0.;    0."  />
   <posXYZ volume="Flexible"  X_Y_Z=" -1600.; 0.; 2500." rot="  0.; 90.; 0." />
   <posXYZ volume="Flexible"  X_Y_Z=" -4800.; 0.; 2500." rot="  0.; 90.; 0." />
   <posXYZ volume="Curving"   X_Y_Z="     0.; 0.; 1600." rot=" 90.;  0.; 0." />
</composition>

<composition  name="servicesAtZ0" >
   <posXYZ    volume="EMCaloPipeI"  X_Y_Z=" 0.;  7000.;   0."   rot="  90.; 0.; 0." />
   <posXYZ    volume="EMCaloPipeO"  X_Y_Z=" 0.; -5000.; 100."   rot="  90.; 0.; 0." />
   <posXYZ    volume="services1"    X_Y_Z=" 0.;     0.;   0."   rot="   0.; 0.; 0." />
   <posRPhiZ  volume="services1"    R_Phi_Z=" 0.;  45.;   0."   rot="   0.; 0.; 0." />
   <posRPhiZ  volume="services1"    R_Phi_Z=" 0.; 135.;   0."   rot="   0.; 0.; 0." />
   <posRPhiZ  volume="services3"    R_Phi_Z=" 0.; 225.;   0."   rot="   0.; 0.; 0." />
   <posRPhiZ  volume="services3"    R_Phi_Z=" 0.; 315.;   0."   rot=" 180.; 0.; 0." />
   <posXYZ    volume="Chain"        X_Y_Z=" -5440.; 0.; -2750." rot="   0.; 0.; 0." />
   <posXYZ    volume="Chain"        X_Y_Z=" -5440.; 0.;  2750." rot=" 180.; 0.; 0." />
</composition>
</section>
```

References

[1] Allison J et al *"Geant4 developments and applications"* IEEE Transactions on Nuclear Science **53** 1 (2006) 270 doi:10.1109/TNS.2006.869826

[2] Agostinelli S et al *"Geant4 - simulation toolkit"* Nuclear Instruments and Methods in Physics Research **506** 3 (2003) doi:25010.1016/S0168-9002(03)01368-8

[3] ATLAS Collaboration et al *"The ATLAS Experiment at the CERN Large Hadron Collider"* JINST **3** S08003 (2008) doi:10.1088/1748-0221/3/08/S08003

[4] Laporte JF, Chevalier L, Guyot C and Virchaux M *"G4AGDD an AGDD based G4 application"* ATL-SOFT-2001-002

[5] Duckeck G et al [ATLAS Collaboration] *"Atlas Computing: technical design report"* CERN-LHCC-2005-022

[6] Pomarède D and Virchaux M *"The Persint visualization program for the ATLAS experiment"* Conference for Computing in High Energy and Nuclear Physics 2003 (La Jolla, California, USA), http://arxiv.org/abs/cs.GR/0305057

[7] Virchaux M and Pomarède D *"The PERSINT Manual"* ATL-SOFT-2001-003, updated version available at *http://twiki.cern.ch/twiki/pub/Atlas/Persint2Wiki/PERSINT2_Manual.pdf*

[8] Rimoldi A, Dell'Acqua A, Gallas M, Nairz A, Boudreau J, Tsulaia V and D Costanzo *"The Simulation of the ATLAS Experiment: Present Status and Outlook"* Computing in High Energy Physics and Nuclear Physics 2004 (Interlaken, Switzerland), CERN-ATL-COM-SOFT-2004-006

[9] Dell'Acqua A, Rimoldi A, Chevalier L, Laporte JF and Virchaux M *"The Atlas Muon Spectrometer Simulation using Geant4"* ATL-MUON-2000-020

[10] Chevalier L *"AMDB_SIMREC: A Structured data base for the ATLAS Spectrometer Simulation Program"* ATL-MUON-97-148

[11] Thomason L *http://www.grinninglizard.com/tinyxml/*

[12] The Apache Software Foundation *http://xerces.apache.org/*

[13] Kittelmann T et al *http://atlas-vp1.web.cern.ch/atlas-vp1/*

[14] Virchaux M *"Muonbox: a full 3D tracking programme for Muon reconstruction in the ATLAS"* ATL-MUON-97-198

[15] Salzburger A, Todorova S and Wolter M *"The ATLAS Tracking Geometry Description"* ATL-SOFT-PUB-2007-004

[16] I. Antcheva et al *"ROOT - A C++ framework for petabyte data storage, statistical analysis and visualization"* Comp Phys Comm **180** 12 (2009) 2499-2512 doi:10.1016/j.cpc.2009.08.005