LCDD: A Complete Detector Description Package

Norman Graf^{a,*}, Jeremy McCormick^{a,*}

^aSLAC National Accelerator Laboratory, 2575 Sand Hill Road, Menlo Park, CA, 94025, USA

Abstract

LCDD has been developed to provide a complete detector description package for physics detector simulations using Geant4. All aspects of the experimental setup, such as the physical geometry, magnetic fields, and sensitive detector readouts, as well as control of the physics simulations, such as physics processes, interaction models and kinematic limits, are defined at runtime. Users are therefore able to concentrate on the design of the detector system without having to master the intricacies of C++ programming or being proficient in setting up their own Geant4 application. We describe both the XML-based file format as well as the processors which communicate this information to the underlying Geant4 simulation toolkit.

Keywords: Simulation software, Detector modeling, XML, Geometry, GDML, LCDD, Geant4

1. Introduction

As the size, complexity and cost of modern physics detectors increase, the need for detailed simulations of the experimental setup plays an increasingly important role. Designing detector systems composed of many disparate subsystems requires efficient tools to simulate the detector response. Comparisons of different technology options, or geometric layouts, are facilitated if the results can be obtained with a flexible, easy-to-use simulation framework.

Geant4 [1] is a software framework for simulating the detailed interactions of particles with matter and fields. It has become the standard tool for detector response simulations in high energy physics (HEP) and is increasingly being used in other fields, such as medical physics and the aerospace industry. Distributed as a set of source files and examples, the toolkit can be used to assemble a domain-specific application based upon experimental requirements. This requires a considerable amount of expertise in the C++ language and the details of configuring the framework. Typically, the most complex task is modeling the geometry and detectors, which for complex setups may take hundreds or even thousands of lines of code. This task can be daunting, and providing a flexible application which can meet the

needs of many different users can alleviate these technical barriers.

The size of the code base will increase over time as detector models are added with their own unique C++ code definitions. Each major detector variant will usually require its own new set of classes or a customization of existing ones, sometimes leading to severe maintenance issues. These problems may include a great amount of code duplication between different detector models, the treatment of geometries as “black boxes” without externally accessible data descriptions, and a lack of separation between procedural code and data. Ideally, the detector description would be provided by a data format rather than a set of compiled classes. GDML[2] provides a language for describing detector geometries, but completely describing an experimental apparatus requires much more than just the geometry.

Providing a comprehensive and flexible solution to these problems has been the goal of the Linear Collider Detector Description (LCDD) project. LCDD was introduced to simulate detector models for International Linear Collider (ILC) design studies. A large number of different detector designs were simulated without having to write customized C++ code for each model. By providing a clear separation between code and detector description data, researchers are freed from needing to know the complex details of the Geant4 APIs. They may instead focus on defining the detector setup for their particular experiment using a structured data language. The framework was designed to support

*Corresponding author

Email addresses: Norman.Graf@slac.stanford.edu
(Norman Graf), Jeremy.McCormick@slac.stanford.edu
(Jeremy McCormick)

the requirements of HEP collider detector studies and does not, at present, provide data bindings of all related Geant4 classes. However extensions can be added if and when the need arises.

An overview of the LCDD language and framework will be provided, with a schematic of the full document structure. The extensions to GDML will be explained and described along with simple examples. Each primary XML element type will be explained in detail along with an example of its usage. Its application to two different HEP experiments, one a collider detector, the other a fixed-target detector, will be briefly described.

2. Complete Detector Description

LCDD was designed to provide a complete description of complex experimental setups using an XML-based textual description. In addition to the physical geometry, information such as detector identifiers, definitions of sensitive detectors, and descriptions of electromagnetic fields, if any, is required to describe a full detector. For the Geant4 detector response simulation, one also needs to be able to define attributes such as regions and physics limits and cuts. Visualization attributes are also included to facilitate viewing the detector.

Various types of detectors, from simple test beams to full “4 π ” HEP detectors, can be modeled to an arbitrary level of detail using the structured XML of the LCDD document.

2.1. GDML

The Geometry Description Markup Language (GDML) was developed to describe detector geometries using materials, mathematical variables and definitions, solids such as boxes and tubes, and a hierarchical structure of logical and physical volumes. Originally released as a stand-alone application, the GDML project has evolved to represent just the data schema, while a parsing engine for I/O persistency has become part of the standard Geant4 distribution. Therefore, it serves as an ideal starting point for a complete detector description language. The syntax and usage of GDML is fully described in the *GDML User's Guide* [3] but a brief overview is provided here for completeness. Every valid GDML file has the following basic structure.

```
<gdml>
  <define/>
  <materials/>
```

```
<solids/>
<structure>
  <volume>
    <materialref/>
    <physvol>
      <volumeref/>
    </physvol>
  </volume>
</structure>
<setup/>
</gdml>
```

GDML's `define` block contains expressions and definitions that are composed of double precision numbers, simple arithmetic operators (* / + -), trigonometric functions, and units. The CLHEP [4] expression evaluator is used to parse and evaluate these mathematical statements. The processor predefines standard units for distance, weight, etc., as well as a set of constants such as the speed of light. Rotations and positions that will be referenced later to create physical volumes may also be included in this section.

The `materials` section has a list of `material` and `element` tags that bind to the Geant4 `G4Material` and `G4Element` classes for materials and atomic elements. Materials are defined by atomic or mass composition and density. Material parameter sheets may be attached to provide pre-computed values for dEdx calculations and optical properties.

The `solids` section contains a collection of shape definitions that are used within the geometry. GDML has bindings to a large and nearly complete subset of the Constructive Solid Geometry (CSG) solids defined by the Geant4 geometry module, including tubes, boxes, trapezoids, tori, twisted tubes and boxes, polyhedra, and faceted shapes. Boolean subtraction, addition and intersection can be used with these primitives to define arbitrarily complex geometries.

The `structure` section contains all of the geometry's logical volume definitions. A logical volume is composed of a shape and material, plus any number of “child” sub-volumes, defined by `physvol` tags. The child volumes must contain a reference to a previously defined logical volume and an in-lined or referenced position and rotation. The top-level volume or “world volume”, typically but not necessarily a large box containing the detector envelope volume, is defined in the `setup` block using the `world` element. The full hierarchy of volumes defines the complete geometric structure for the detector simulation.

2.2. LCDD Document Structure

LCDD uses GDML to provide the complete physical geometry for the simulation. The `gdml` root node is embedded as part of the LCDD document. The GDML element is essentially left intact with only minor additions. The main point of extension is the volume element, which is redefined so that it may contain additional references to LCDD elements. The child volumes are also redefined so that they may have one or more physical volume identifiers attached to them. Using this approach, a GDML volume can be associated with important supplementary information such as readout parameters. The following snippet outlines the structure of an `lcdd` document.

```
<lcdd>
  <header/>
  <iddict>
    <idspec>
      <idfield/>
    </idspec>
  </iddict>
  <sensitive_detectors>
    <calorimeter/>
    <tracker/>
    <scorer/>
  </sensitive_detectors>
  <limits>
    <limitset>
      </limit>
    </limitset>
  </limits>
  <regions>
    <region/>
  </regions>
  <display>
    <vis/>
  </display>
  <gdml>
    <define/>
    <materials/>
    <solids/>
    <structure>
      <volume>
        <sdref/>
        <limitsetref/>
        <regionref/>
        <visref/>
      </volume>
    </structure>
    <setup/>
  </gdml>
```

```
<fields>
  <solenoid/>
  <dipole/>
  <box_dipole/>
  <rz_field_map/>
  <field_map_3d/>
</fields>
</lcdd>
```

LCDD is built upon the GDML data format and code infrastructure. GDML itself uses the Xerces C++ [5] XML toolkit for parsing and schema validation. LCDD extends GDML by using built-in facilities of the XML Schema (XSD) language such as the `xs:redefine` and `xs:extension` elements. The GDML parser is also used by registering additional element handlers for LCDD types. This extended parser is then able to read both plain GDML and LCDD documents. One can include valid GDML documents directly into the `lcdd` document as long as no extensions (such as sensitive detectors or visualization attributes) are required.

Every LCDD file has the same basic structure that is enforced through XML schema checking at runtime against the document contents.

The LCDD parser checks the input document for correctness against the XML Schema, which is located at a standard URL accessible via the `http` protocol, viz. www.lcsim.org/schemas/lcdd/1.0/lcdd.xsd.

The `lcdd` root element has a number of sections, most of which contain a list of elements with a specific type. Each of these elements has a name assigned to it, which can be used as a unique reference within the document. The GDML volumes may reference these LCDD elements by their names. In this way, the information in LCDD is linked to specific volumes defined by GDML.

The header has basic meta data about the document, such as a name that can be used as an external tag or ID of the detector. (It is the only top-level element which has no significant level of sub-structure.) The `iddict` is a collection of identifier dictionaries that provide encodings for packed, 64-bit IDs, which contain encoded values such as a layer number or a detector ID. The `sensitive_detectors` element defines sensitive detectors that can be assigned to volumes, flagging them as readout components capable of producing hits. The `limits` are sets of physics limits that can be used to tune parameters in the physics engine, such as the maximum step size of a track. The `display` element contains visualization settings that assign colors and other visibility parameters to logical volumes. The `gdml` section contains an entire, embedded GDML doc-

ument, which must conform to that format's XML syntax. It may also include the additional elements defined in the LCDD schema as GDML extensions. Finally, the `fields` element contains definitions for magnetic field maps that apply to the world volume.

2.3. Header Element

Every document begins with a header that provides basic meta data about the file and the detector.

```
<header>
  <detector name="sidloi3"/>
  <generator name="GeomConverter"
    version="1.0"
    file="./sidloi3/compact.xml"
    checksum="2152839912"/>
  <author name="Jeremy McCormick"
    email="jeremym@slac.stanford.edu">
  <comment>The SiD detector</comment>
</header>
```

The `detector` tag provides a name attribute that can be used as an external ID. For instance, the name can be written into the run headers of output data files to identify which detector was used to generate those events. An `author` element gives the names of the authors of the file, as well as an email contact. The `generator` provides information about an external program that was used to generate the file, if applicable, including a source file name, a version of that program, and a checksum that could be created with an MD5 algorithm. Finally, there is a comment block that may be used to provide additional descriptive text about the detector.

2.4. Sensitive Detectors

A sensitive detector is assigned to a logical volume to designate it as a readout component that is capable of producing hits. When a particle deposits energy into that volume, hits are created, which may later be written into an output file for analysis. (Providing these output data formats is not one of LCDD's features.) The sensitive detectors accumulate position, time and energy measurements from particle interactions within the material. Hits are grouped by event and added to collections based on their volume's assigned sensitive detector.

Two primary types of detectors are modeled by the framework. Trackers store output from the simulation that corresponds closely to individual steps within a volume. This information can be used later to reconstruct in detail the exact particle momentum at the hit location. Calorimeters record the total accumulation of energy

by event and typically have much less granular position information. Calorimeter volumes may be virtually segmented into cells using a `segmentation` element. There is also a third type of detector called a `scorer` which is essentially a simplified tracker. It can be used to insert scoring planes to derive flux counts.

Sensitive detectors extend a common element which defines basic settings, including those shown in Table 1.

<code>name</code>	unique sub-detector identifier
<code>endcap_flag</code>	barrel or endcap flag
<code>ecut</code>	minimum energy cut for hits
<code>eunit</code>	energy unit for cut
<code>verbose</code>	verbosity setting

Table 1: Sensitive Detectors

Each sensitive detector has a name uniquely identifying it within the document. The `sdref` element of a volume references this name to associate the volume with a specific sensitive detector. There is a flag which indicates whether or not the detector is an end cap. (This is primarily a concept relevant for HEP collider-detectors.) An energy cut can be used to discard low-energy hits that do not reach a certain threshold. There is a verbosity setting to control print screen output while the simulation is running e.g. for debugging. The name of the sub-detector is required, and the other settings are optional.

Each sensitive detector has one or more hits collection containing objects which are implementations of Geant4's virtual hit class, `G4VHit`. The `CalorimeterHit` class is used to store hits in the calorimeters, and a list of `HitContribution` objects keeps the information about each energy deposition made in a cell. The `TrackerHit` class is used to store information from trackers. (There is no specific output data binding provided by LCDD itself to persist this information.) It is assumed that applications which include LCDD as a dependency would translate from the provided hit objects into an output format such as LCIO [6].

2.4.1. Scorers

The `scorer` type is the simplest of the sensitive detector implementations. It records the passage of particles through a volume. The main difference between the tracker and the scorer is that the latter will only record one hit for each unique `G4Track` that passes through it, whereas the tracker class records all separate steps as individual hits. The scorer simply provides a way to determine if a given track passed through the volume.

2.4.2. Trackers

Trackers record information from each step as a track propagates through the material of a volume. The stored information includes the mid-point position of the G4Step, momentum of the track at that point, length of the step, global time, and the energy deposited along that path. A `TrackerHit` object is created and stored into a hit collection to persist this information event by event. The Tracker is most commonly used to model non-destructive, low-mass, high-granularity readout technologies used to measure the trajectories of charged particles, such as TPCs or silicon pixel and microstrip detectors. Algorithms for digitizing the hits within the simulation are not provided, as it is assumed this would be done later in an external analysis environment using the stored hit data.

This is an example of a simple tracking detector:

```
<tracker name="BarrelTracker"
  hits_collection="BarrelTrackerHits">
  <idspecref ref="BarrelTrackerHits"/>
</tracker>
```

Essentially the tracker is a relatively simple implementation of a sensitive detector that records the information from individual steps. The persisted information can then be used later as input to more sophisticated digitization algorithms.

2.4.3. Calorimeters

The calorimeter is used to record the energy depositions of particle showers in homogeneous or sampling detectors. Typically these showers are composed of many individual steps, the details of which are normally not of interest. The energy depositions from all these steps are accumulated to determine the total energy deposited in the volume for that event. The energies may be summed for an entire volume, as in a physical crystal or pad, or assigned to bins within an array of virtual cells. When volumes are artificially segmented, there is at most one `CalorimeterHit` object created per virtual cell for the entire event.

The following XML defines a calorimeter with uniform sized cells created by a virtual segmentation class. The `grid_xyz` element will divide the detector's sensor layers into a grid of 3.5mm x 3.5mm cells.

```
<calorimeter name="EcalBarrel"
  hits_collection="EcalBarrelHits">
  <idspecref ref="EcalBarrelHits"/>
  <grid_xyz
    grid_size_x="3.5*mm"
```

```
    grid_size_y="3.5*mm"
    grid_size_z="0.0"/>
</calorimeter>
```

The calorimeter hits each have a list of individual energy contributions that correspond to single steps in the simulation, recording the PDG code of the particle and the positional information about that step, e.g. its endpoints. Depending on the user requirements, this low-level step data can be saved into the output data, as it might be useful if the readout response depends on the distance of an energy deposition from the edge of the cell, etc. The energy contributions may also be summed to obtain the total energy deposition in the cell for that event.

2.4.4. Hits Processors

Each sensitive detector has one or more `hits_processor` objects to process step information into hits, allowing flexibility in how different types of particles and physics are handled. For instance, an optical calorimeter may write out separate hits collections for the scintillation and Cherenkov energy depositions using two different hits processors. The tracker and calorimeter detectors each have a default hits processor that uses the associated segmentation, sensitive detector, and identifier objects to construct hits. It is anticipated that the hits processor could provide an extension point for future development of flexible digitization algorithms with complex requirements and input parameters.

2.4.5. Segmentation

Sensitive volumes in a calorimeter, such as planar layers, usually require virtual subdivision. The segmentation element models this division of geometric volumes into cells. The algorithmic approach has some advantages over using purely geometric constructs. Simulating millions of individual cell volumes could be expensive in terms of memory usage. There are also certain cases in which it is simpler to describe a readout system with an algorithm rather than pure geometry. For instance, projective calorimeter towers have many different shapes and sizes of cells depending on the radial distance of a layer from the origin, and this would be complicated to construct using only solids and volumes.

The segmentation element occurs as a child of the calorimeter element, and each calorimeter may have only one of these associated objects. The specific segmentation types extend an abstract XML element. The parameters which define the dimensions of the cells are

specific to a certain type. The values of the fields from the segmentation at a certain hit position can be written into the identifiers of the hits by referencing their names in the identifier specification.

2.4.6. Grid XYZ Segmentation

The `grid_xyz` segmentation divides a volume into cells along any combination of its X, Y, or Z axes, creating a regular grid of box-like virtual volumes. The indices are signed integer values, numbered about the natural origin of the volume's solid, which is its center point in local coordinates. Only the position with respect to this origin is used to compute an index value at a particular point in the grid, so no additional geometric information, such as the bounds of the current solid, is required by this algorithm to compute the bin values.

The following XML shows a `grid_xyz` segmentation that divides a volume along the X and Y axes.

```
<grid_xyz
  grid_size_x="1.0*cm"
  grid_size_y="1.0*cm" />
```

The default grid size value of zero for an axis results in that dimension being left unsegmented, so that the position is reported as zero. This actually translates to the center point along that axis in local coordinates. Typically, this segmentation type divides a plane into a rectangular grid of cells in two dimensions, such as X and Y, with the third dimension, e.g. Z, left unsegmented.

2.4.7. Projective Cylinder Segmentation

The `projective_cylinder` segmentation divides cylinders into projective towers. Unlike most other types of segmentation, this does not result in cells with uniform sizes. The size of a given cell in a projective segmentation depends on its distance from the origin. The `nphi` parameter determines how many phi bins are created within the full 360° in azimuth. Similarly, `ntheta` specifies the number of theta bins, covering the 180° in polar angle.

This is an example of a projective cylinder segmentation that divides the theta and phi regions into 32 and 32 bins, respectively.

```
projective_cylinder
  ntheta="32"
  nphi="32" />
```

This segmentation is used, for instance, in geometries where the calorimeter barrel is modeled using a series of nested tubes, rather than modules containing planar layers.

Figure 1: Cylindrical projective ϕ segmentation.

Figure 2: Cylindrical projective(top) and non-projective(bottom) z segmentation.

2.4.8. Non-projective Cylinder Segmentation

A `nonprojective_cylinder` segmentation element will divide the surface of a cylinder into cells of equal size along its length.

```
<nonprojective_cylinder
  grid_size_phi="10.0*mm"
  grid_size_z="10.0*mm" />
```

The above segmentation will divide the surface of a cylinder into 10 mm x 10 mm cells.

Figure 1 shows the expected projective ϕ segmentation, whereas Figure 2 shows examples of both the projective and non-projective types of cylindrical segmentation along the z axis.

2.4.9. Projective ZPlane Segmentation

The `projective_zplane` segmentation divides an endcap zplane into projective angular segments specified by the number of phi and theta bins, much as a `projective_cylinder` is used for a barrel.

```
<projective_zplane
  ntheta="500"
  nphi="500" />
```

Unlike most other segmentations, this algorithm uses the origin of the world volume rather than the volume's center.

2.5. Identifiers

Identifier dictionaries define formats for bit-packed integers used to associate hits with specific detector information like layer numbers and sub-detector IDs, so that these values are accessible in an external framework by decoding the IDs of the hits. The `physvolid` values or calorimeter segmentation bins may be referenced by these definitions. Each sensitive detector may have only one identifier specification, which is used to create a 64-bit integer for each of its hits at runtime. The user is ultimately responsible for ensuring that the given combination of packed field values uniquely identifies the hit in their external application.

All of the identifier specifications are contained in the `iddict`. Every ID dictionary has a corresponding element called the `idspec` with a list of `idfield` tags. Each `idfield` defines a single field of the identifier, such as a layer number or a field from the segmentation. The individual fields can be from 1 to 32 bits long and may be signed or unsigned.

Below is an example of an identifier definition for a calorimeter.

```
<idspec name="EcalBarrelHits" length="64">
<idfield signed="false"
  label="system" length="6" start="0"/>
<idfield signed="false"
  label="barrel" length="3" start="6"/>
<idfield signed="false"
  label="module" length="4" start="9"/>
<idfield signed="false"
  label="layer" length="6" start="13"/>
<idfield signed="false"
  label="slice" length="5" start="19"/>
<idfield signed="true"
  label="x" length="16" start="32"/>
<idfield signed="true"
  label="y" length="16" start="48"/>
</idspec>
```

The first five fields of the above identifier derive from the `physvolid` values. The “x” and “y” values are read from the segmentation bins at the hit position. The concatenation of these values identifies a unique read-out channel in the detector. The packed values can be subsequently decoded within an external framework to retrieve the associated detector information for a specific hit. Accessing the field information from such a

common source for both simulation and reconstruction ensures a commensurate encoding and subsequent decoding of the bit packing.

2.6. Physics Limits

Physics limits can be assigned to volumes in order to tune certain parameters within the Geant4 simulation. For example, the range cut can be increased to limit the production of low-energy secondary particles in divergent electromagnetic processes.

The following example will restrict the step lengths of all particles to 5 mm.

```
<limits>
  <limitset name="cal_limits">
    <limit name="step_length_max"
      unit="mm"
      particles="*"
      value="5.0" />
  </limitset>
</limits>
```

Each limit has a name that identifies the simulation parameter, a double value, a unit applied to that value, and a comma-delimited list of particles. The names of the particles in the list are defined within Geant4.

The available limits include the maximum step length, the maximum track length, the maximum particle lifetime, the minimum particle kinetic energy, and the minimum range (or “range cut” in Geant4 terminology). These are shown in Table 2.

<code>step_length_max</code>	maximum length of a single step
<code>track_length_max</code>	maximum length of a single track
<code>time_max</code>	maximum lifetime of a track
<code>ekin_min</code>	minimum track kinetic energy
<code>range_min</code>	range cut

Table 2: User Settable Limits

2.7. Regions

Regions are assigned to geometric volumes using the `region` element, defining collections of volumes that share similar characteristics. A flag specifies whether or not secondary particles produced in the simulation are stored into the output particle collections. The following example defines a region in which all secondary particles will be stored into the output. This is typically called a “tracking region.”

```

<regions>
  <region name="TrackingRegion"
    store_secondaries="true"
    cut="10.0" lunit="mm"
    threshold="1.0"
    eunit="MeV" />
</regions>

```

The above example also specifies that particles which would not travel more than 10 millimeters will not be produced, and it also establishes an energy cut of 1 MeV for storing particles to the output.

2.8. GDML Volume Element

LCDD extends GDML by adding additional, optional elements to volumes. The following example assigns to a volume a sensitive detector, a region, visualization attributes, and physics limits.

```

<volume name="EcalBarrel_layer0">
  <materialref ref="Silicon"/>
  <solidref ref="EcalBarrel_layer0_box"/>
  <sdref ref="EcalBarrel"/>
  <limitsetref ref="CalLimits"/>
  <visref ref="CalVis"/>
  <regionref ref="CalRegion"/>
</volume>

```

The referenced LCDD objects are previously defined in the document. For instance, the “EcalBarrel” sensitive detector is defined prior to the volume definition, and the parser will retrieve its object from an in-memory data structure and then assign the sensitive detector to the named volume. A similar strategy is used for the other objects referenced by the extended volume element. In this way, the supplementary information defined by LCDD can be assigned to the appropriate GDML volume elements.

2.9. Physical Volume IDs

Physical volumes in GDML are also extended so that they may have one or more physical volume IDs associated with them. These values can be used in 64-bit identifiers that are attached to hits. These IDs, unlike a “touchable” within the Geant4 geometry hierarchy, can be decoded by an external analysis program to provide information such as a layer number. For instance, this is an example of assigning a layer number to a volume.

```

<volume name="DetectorEnvelope">
  <materialref ref="Air"/>
  <solidref ref="DetectorEnvelopeBox"/>

```

```

  <physvol>
    <volumeref ref="LayerVolume"/>
    <physvolid field_name="layer"
      value="1"/>
  </physvol>
</volume>

```

Any number of physvolid elements can be added to a single physvol, so the assignment of these identifier values need not be limited by the hierarchical structure of the detector.

2.10. Magnetic Fields

Geant4 can simulate in detail the interaction of particles with electromagnetic fields [7]. LCDD provides a number of different field types for describing magnetic fields commonly encountered in physics experiments, ranging from simple approximations using fixed values to full three-dimensional grids. Fields are defined globally in a list, and when their regions of validity overlap in space, the components are added at that point to compose an overlay.

2.10.1. Solenoid

The solenoid is a simplistic representation of a solenoid magnet with a constant magnetic field component aligned along the z axis. It has an inner and outer radius, and one Bz value is applied within the inner radius, and another between the inner and outer. This field type is commonly employed in collider detectors where charged particles bend in the XY plane.

The following defines a solenoid with a constant 5 Tesla field strength within the inner radius and a return field strength of -0.6 Tesla outside of that.

```

<fields>
  <solenoid name="GlobalSolenoid"
    lunit="mm"
    funit="tesla"
    outer_radius="world_side"
    inner_field="5.0"
    outer_field="-0.6"
    zmax="1000.0"
    zmin="-1000.0"
    inner_radius="2923.0" />
</fields>

```

2.10.2. Dipole

The dipole models a Bx field that varies in the Z dimension given a list of coefficients, using a simple polynomial fit with an arbitrary number of terms.

$$B_x = \sum_{i=1}^n c_i * z^i$$

Here is an example of a dipole field definition.


```
<dipole name="ExampleDipoleField"
  zmin="0.0"
  zmax="-10.0*mm" >
  <dipole_coeff value="1.0" />
  <dipole_coeff value="0.1" />
  <dipole_coeff value="0.001" />
</dipole>
```

2.10.3. Box Dipole

The `box_dipole` is used to define a set of fixed component values in a box-like region.

```
<box_dipole name="AnalyzingDipole"
  x="2.117*cm" y="0*cm" z="45.72*cm"
  dx="50.0*cm" dy="50.0*cm" dz="50.0*cm"
  bx="0.0" by="-1.5" bz="0.0" />
```

2.10.4. RZ Field Map

An RZ field map can be used to simulate a cylindrically symmetric field variable in the radial and Z directions. The field is input as a map of B_z and B_r on a regular grid of radius and z positions.

```
<rz_field_map name="RZFieldMap"
  lunit="cm"
  funit="kilogauss"
  num_bins_r="67"
  num_bins_z="64"
  grid_size_z="10.0"
  grid_size_r="10.0">
<rzB z="0." r="0." Bz="50.0" Br="0." />
<rzB z="10." r="0." Bz="49.9" Br="0." />
<rzB z="20." r="0." Bz="49.9" Br="0." />
<rzB z="30." r="0." Bz="49.8" Br="0." />
[etc.]
</rz_field_map>
```

The Z and R bins are computed from the global position. Then the B-field (B_z , B_r) is calculated by linear interpolation in 2 dimensions (Z, R) between these points.

2.10.5. Field Map 3D

The `field_map_3d` represents a 3D Cartesian magnetic field map as a grid of values measured on a regular cartesian grid of 3D points.

The field values are contained in an external file with each line defining a point in the map.

x, y, z, B_x, B_y, B_z

A 3D interpolation is performed to determine the B-field (B_x , B_y , B_z) at a given point (x, y, z) in the grid, based on an algorithm developed from the `PurgMagTabulatedField3D` Geant4 code example. [8] Since field maps are normally measured or calculated with respect to the physical magnet's coordinate system, we allow for offsets to position the field in global coordinates.

Here is an example that references an external data file.

```
<field_map_3d name="FieldMap3D"
  lunit="mm"
  funit="tesla"
  filename="data/3DFieldMap.dat"
  xoffset="2.117"
  yoffset="0.0"
  zoffset="45.72" />
```

2.11. Visualization

Geant4 has a number of built-in mechanisms for visualizing detectors. These range from real-time visualization using OpenGL to the generation of graphics file formats such as VRML. LCDD can assign visualization attributes to geometric volumes using the `visref` element. The supported settings include line style (broken or unbroken), drawing style (wireframe or solid), the visibility of daughter volumes, and whether the volume itself is visible. The color of volumes may also be specified with RGB values from 0.0 to 1.0, as well as the transparency or alpha. An alpha value of 0.0 would render the component invisible and 1.0 would be completely opaque.

The following is an example of visualization attributes.

```
<display>
  <vis name="CalVis"
    line_style="unbroken"
    drawing_style="solid"
    show_daughters="true"
    visible="true">
    <color R="1.0"
      G="0.0"
      B="0.0"
      alpha="1.0"/>
  </vis>
</display>
```

3. Using LCDD in an Application

The LCDD framework implements `G4VUserDetectorConstruction`, which is a required

class for Geant4 user applications. Using LCDD is simply a matter of registering its detector construction class with the Geant4 run manager, viz. `theRunManager->SetUserInitialization(new LCDDDetectorConstruction());`. This allows the detector document to be specified using a macro command, which will load a document from a remote URL, viz.

```
/lcdd/url http://www.lcsim.org/.../det.lcdd
```

Local files can also be read in by using the `file://` protocol:

```
/lcdd/url file:///local/path/to/det.lcdd
```

Arguments to this macro command without a protocol are interpreted as local files:

```
/lcdd/url /local/path/to/det.lcdd
```

The detector document must be specified in the pre-initialization phase, and then the geometry is setup when initialization occurs, either through calling the initialization method on the run manager directly or executing the `/run/initialize` macro command.

4. Examples

4.1. Linear Collider

Detectors designed to exploit the physics discovery potential of lepton collisions at the Terascale will need to perform precision measurements of complex final states. The ability to model numerous detector geometries using LCDD was a crucial element in the design of the Silicon Detector [10], one of the two concepts currently being investigated to study the physics of high energy electron-positron collisions at the International Linear Collider (ILC) [11] and Compact Linear Collider (CLIC) [12]. The optimization process for the tracker design started out with a number of simplified geometries, with the complexity of the simulations increasing as the designs became more mature. The current model is quite sophisticated, including most of the details of the engineering models for the support and assembly of the detectors, as well as the electronic readouts currently being considered. Figure 3 shows a cross section of a CAD model of the central tracker. The corresponding Geant4 model is shown in Figure 4.

A similar process was employed in the design of the calorimeters and instrumented return yoke, which functions as a muon detector. The flexibility of LCDD enabled the performance of various absorber materials, readout technologies and detector layouts to be studied all within the same simulation program. Figures 5, 6, 7 and 8 show the level of complexity supported by the LCDD package.

Figure 3: The Silicon Detector micro-strip outer tracker and pixel vertex detector as shown in a CAD model.

Figure 4: The Silicon Detector micro-strip outer tracker and pixel vertex detector as shown in the Geant4 model.

4.2. Heavy Photon Search

The Heavy Photon Search (HPS)[13] experiment is a direct Dark Matter search using a fixed target detector. The detector has a silicon microstrip tracker and a lead tungstate crystal calorimeter which have both been modeled to a high level of detail in simulation. LCDD has been used throughout the design process to study numerous detector models and variations for the 2012 Test Run as well as the full experiment, scheduled to be commissioned in the fall of 2014. In particular, LCDD has been invaluable for testing different configurations of the silicon tracker modules, including variations on the number of total layers and the layout of the sensors. Figure 9 shows the detector layout as currently constructed.

5. Conclusion

LCDD is a robust and complete system for modeling detectors using the Geant4 simulation toolkit. It has

Figure 5: The ILC Silicon Detector (SiD) YZ quadrant view.

Figure 7: A Cutaway view of the SiD Silicon Tracker

Figure 6: The ILC Silicon Detector (SiD) XY cross section.

Figure 8: A Cutaway view of the SiD complete detector

Figure 9: The Heavy Photon Search detector. The silicon microstrip trackers are shown in dark grey, the calorimeter crystals in red. The dipole is not shown. Support and dead materials have been deemphasized for illustration clarity.

been used by a variety of experimental physics collaborations to prototype various detector designs. In particular, LCDD has become part of the common set of tools used to simulate collider detector designs for CLiC and ILC, as well as the fixed-target experiment HPS. The usage of GDML allows the framework to model geometries to an arbitrary level of detail while providing all the required, extra information for a complete detector description language.

6. Acknowledgement

Work supported by the U.S. Department of Energy under contract number DE-AC02-76SF00515.

- [1] Geant4 - A Simulation Toolkit, S. Agostinelli et al., Nuclear Instruments and Methods A 506 (2003) 250-303
- [2] Geometry Description Markup Language for Physics Simulation and Analysis Applications, R. Chytrcek, J. McCormick, W. Pokorski, G. Santin IEEE Trans. Nucl. Sci., Vol. 53, Issue: 5, Part 2, 2892-2896
- [3] <http://lcgapp.cern.ch/project/simu/framework/GDML/doc/GDMLmanual.pdf>
- [4] <http://cern.ch/clhep>
- [5] <http://xerces.apache.org/xerces-c/>
- [6] LCIO - A persistency framework for linear collider simulation studies, Computing in High Energy and Nuclear Physics, 24-28 March 200
- [7] <https://cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch04s03.html>
- [8] <https://twiki.cern.ch/twiki/bin/view/Geant4/AdvancedExamplesPurgingMagnet>
- [9] <https://cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch05s07.html>
- [10] <http://silicondetector.org>
- [11] <http://www.linearcollider.org>
- [12] <http://clhc-study.org/>
- [13] <https://confluence.slac.stanford.edu/display/hpsg/Heavy+Photon+Search+Experiment>