# Simulator For The Linear Collider (SLIC): A Tool For ILC Detector Simulations

Norman Graf, Jeremy McCormick

*Stanford Linear Accelerator Laboratory, Menlo Park, CA, 94025*

**Abstract.** The Simulator for the Linear Collider (SLIC) is a detector simulation program based on the GEANT4 toolkit. It is intended to enable end users to easily model detector concepts by providing the ability to fully describe detectors using plain text files read in by a common executable at runtime. The detector geometry, typically the most complex part of a detector simulation, is described at runtime using the Linear Collider Detector Description (LCDD). This system allows end users to create complex detector geometries in a standard XML format rather than procedural code such as C++. The LCDD system is based on the Geometry Description Markup Language (GDML) from the LHC Applications Group (LCG). The geometry system facilitates the study of different full detector design and their variations. SLIC uses the StdHep format to read input created by event generators and outputs events in the Linear Collider IO (LCIO) format. The SLIC package provides a binding to GEANT4 and many additional commands and features for the end user.

**Keywords:** Geant4, Detector Simulation
**PACS:** 07.05.Tp, 07.05.Wr, 07.20.Fw, 29.40.Vj, 29.85.+c

## INTRODUCTION

Detectors for the International Linear Collider (ILC) will require precision tracking, vertexing, and calorimetry in order to meet the ambitious physics goals of the experimental program. Distinguishing between W and Z bosons decaying into quark jets requires a calorimetric energy resolution of roughly $30\%/\sqrt{E}$. The design of the full detector and its sub-detectors will need to be aggressively optimized to meet this performance goal. Monte Carlo simulations are used in this optimization process to study the performance of different proposed detector designs and their variations. In order to conduct these studies and fully explore the phase space of possible detectors, we have developed an executable based on the Geant4 toolkit which is capable of fully defining a detector using runtime input. This frees the physicist from having to worry about details of C++ coding and concentrate on the optimization of the detector performance.

The full detector designs must be optimized for maximum performance while being constrained by a reasonable cost. At the level of the full detector, different combinations of subdetector parameters need to be studied. It may therefore be true that although any particular subdetector may not be optimal, the detector as a whole may have optimal performance. Although the interface targets basic Geant4 shapes, and therefore any detector could be modeled, the focus of this particular presentation is on detectors optimized for the ILC. Although modern collider detectors have

*Contributed to 12th International Conference on Calorimetry in High Energy Physics (CALOR 06), 06/05/2006--6/9/2006, Chicago, Illinois*

Work supported by the US Department of Energy contract DE-AC02-76SF00515

converged on a common cylindrical topology, the exact composition of the trackers and calorimeters varies between concepts.

Since the number of possible full detector designs that need to be modeled by the simulation software is quite large, detector simulation software for the ILC needs to provide capabilities that facilitate an iterative development cycle allowing rapid prototyping of these detector geometries. Changes to the detector geometry must not only be easy to make but also easily to propagate to all packages that depend on this information, from the simulator through reconstruction and analysis algorithms. The detector geometry should be treated as conditions; client programs need random access to full detector designs based on the current physics event.

As an example, the following are some of the basic detector parameters which can be varied at runtime in the geometry of a sampling calorimeter:

- Inner radius of the barrel calorimeter.
- Extent along the beam axis.
- Number of layers.
- Absorber layer material.
- Absorber layer thickness.
- Sensor layer material.
- Sensor thickness.
- Sensor technology.
- Sensor segmentation ("cell size").
- Configuration of dead material / supports.

The above parameters represent only the first level of variations in a subdetector design. The nested geometry hierarchies used in detector simulation can be represented to varying levels of detail. The most complex geometries model the "as designed" or "as built" components as closely as possible. Simplified detector geometries can be used for parametric geometry studies using several full detector designs or variants thereof. For fast studies, tube or cylindrical pieces may be used to model the calorimeters.

## SLIC OVERVIEW

The Simulator for the Linear Collider (SLIC[1]) was designed around the requirements of an iterative development cycle. SLIC provides a flexible and powerful geometry system and includes all the necessary packages for a full simulations engine. The "SLIC" name includes a number of different software packages. The GEANT4[2] toolkit provides the geometry and physics simulation engine describing the interaction of particles with fields and materials. LCIO[3] is the IO subsystem for interconnecting different ILC software packages. GDML[4] is an XML-based geometry system. SLIC uses a GEANT4-based library called Linear Collider Detector Description (LCDD) for the detailed and complete description of the detector geometry at runtime. LCDD is an extension of GDML and will be discussed in more detail in the following section. Finally, SLIC is the hub package depending on all

seven of the other libraries. It provides concrete definitions of the user classes for geometry, physics, and event generation required by GEANT4. SLIC includes a complete command-line interface and many new Geant4 macro commands to facilitate both batch and interactive work.

## LCDD GEOMETRY SYSTEM

The Linear Collider Detector Description (LCDD) is an XML data format for describing detector geometries along with the associated processing infrastructure. The LCDD format is defined by an XML Schema document (XSD) at a standard URL.

```
http://www.lcsim.org/schemas/lcdd/
```

The LCDD schema uses built-in XSD features to import and extend the GDML schema. An LCDD document contains an embedded `<gdml>` block, or sub-document, that must conform to GDML's schema. Applications that need to either produce geometry data or read it in thus have a clear and complete definition of the detector description with these standardized formats.

The GDML format provides bindings to the core geometry classes of the GEANT4 toolkit. The GDML format defines expressions, materials, shapes, volumes, and volume structure. LCDD extends these basic definitions by adding information essential to the full simulation. LCDD adds a header with detector meta-information, an identifier dictionary, sensitive detectors and readouts, regions, physics limits, visualization parameters, and magnetic fields.

Unlike in HTML, where elements can be referenced regardless of order, GDML and LCDD support in-order references, only. An element must have already been defined to be referenced. For this reason, the ordering of the top-level container elements in GDML and LCDD files is important and must conform to the order specified in their respective schemas. For instance, a box solid must be defined before it can be used as the shape for a volume. The primary benefit of this approach is reduction of memory consumption during the processing phase.

The following snippet of pseudo-XML outlines the top-level structure of an LCDD file, including the embedded GDML element.

```
<lcdd>
     <header> … </header>
     <iddict> … </iddict>
     <sensitive_detectors> … </sensitive_detectors>
     <regions> … </regions>
     <limits> … </regions>
     <display> … </display>
     <gdml>
          <define> … </define>
          <materials> … </materials>
          <solids> … </solids>
```

```
            <volumes> … </volumes>
            <structure> … </structure>
            <setup> … </setup>
        </gdml>
        <fields>
</lcdd>
```

The embedded GDML block may contain additional LCDD tags in the `<volume>` element. This is the only extension to the GDML format made by LCDD. The LCDD processor can also read plain GDML files, so a file with `<gdml>` as the top element is considered valid within the LCDD processing framework.

GDML's `<define>` block contains expressions and definitions read into the CLHEP Expression Evaluator. These expressions may contain double precision numbers, simple arithmetic operators (* / + -), trigonometric functions, and units. The processor predefines a number of standard units for distance, weight, etc. Important primary constants such as the speed of light are also predefined. Rotations and positions that will be referenced later to create physical volumes are also included in this area.

The `<materials>` section has `<material>` and `<element>` elements that bind to the `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 similar algorithms. The materials defined here are referenced by `<volume>` elements in the `<structure>` area.

The `<solids>` block contains shape definitions that are also referenced by the GDML volumes. Constructive Geometry Solids (CSG) are the most common type of shapes used in GEANT4 geometries. (Boundard Represented Solids (BREPS) are also available but are not supposed by the GDML system.) GDML has bindings to a large and nearly complete subset of the CSG solids defined by the GEANT4 geometry subsystem, including tubes, boxes, trapezoids, circles, ellipses, twisted tubes and boxes, polyhedra, and facetted shapes. Boolean subtraction and addition can be used with these primitives to define arbitrarily complex geometries, such as masks and detectors containing cutouts for ingoing and outgoing beampipes.

The `<structure>` block contains a nested hierarchy of geometric volumes. A volume is composed of a shape plus its material and may contain any number of sub-volumes, defined with the `<physvol>` tag. The volumes in the `<physvol>` elements are called "child" volumes of their "parent" volume. The child volumes must contain a reference to a logical volume, plus an in-lined or referenced position and rotation. The top-level volume or "world volume", typically a large box containing the detector envelope volume, is defined in the `<setup>` block using the `<world>` element.

The `<iddict>` is an identifier dictionary that defines 32 and 64 bit identifiers to uniquely identify geometric elements in the detector. These identifiers are used to write cell identifiers into the LCIO hits. Each identifier has a corresponding `<idspec>` with fields that are defined using `<idfield>` elements. After the

dictionary is processed, a GEANT4 runtime object is created for each identifier specification. During hits processing, data is retrieved automatically from the simulation based on the identifier specification. A sample <iddict> with an <idspec> for an ECAL barrel is defined below.

```
<iddict>
      <idspec name="EcalBarrelId" length="64">
            <idfield label="layer" start="0" length="7" />
            <idfield label="sys" start="7" length="3" />
            <idfield label=”theta” start=”10” length=”8” />
            <idfield label=”phi” start=”18” />
      </idspec>
</iddict>
```

The <idspec> of 64 bits describes an ECAL calorimeter barrel. The <idfield> label sets the source of the value. Possible sources are <physvolid> elements or segmentation values from the subdetector's readout. The "layer" field is the layer number, and it is read from a <physvolid> of the GEANT4 volume that corresponds to the layer volume. The "sys" field is also taken from a <physvolid> and gives a unique identifier for the sub-detector. The "theta" and "phi" fields are not read from the volume hierarchy. These values are provided by the readout of the ECAL sensitive detector. This flexible system allows the geometric identifier information to be transferred easily from the simulation runtime to the LCIO hits. Applications requiring the identifier information at a later time can read it directly from the LCDD file.

The <sensitive_detectors> section lists sensitive detectors that are referenced from the GEANT4 volumes. When a particle passes through a sensitive volume, e.g. one that has an assigned sensitive detector, the hits processing method is called on the sensitive detector. Depending on the characteristics of the energy deposition, a hit may be created that will be transferred into the LCIO output file. The <calorimeter> and <tracker> elements are the two types of sensitive detectors supported by LCDD.

```
<sensitive_detectors>
     <calorimeter name="EcalBarrSD"
                  hits_collection="EcalBarrHits"
                  ecut="0.0"
                  eunit="MeV"
                  verbose="4">
          <idspecref ref="EcalBarrelId" />
          <projective_cylinder ntheta="840"
                               nphi="1680" />
     </calorimeter>
</sensitive_detectors>
```

The calorimeter and tracker classes have some common features. The name attribute is used for referencing the <sensitive_detector> from other LCDD

elements, such as the <volume>. The sensitive detector name must be unique. The hits_collection attribute sets the name of the hits collection that is produced and written to the LCIO file. The ecut (and the associated eunit) sets a cut on energy, below which hits will not be recorded. This allows low energy hits to be easily discarded. The verbose attribute turns on debugging output which controls how many messages from the sensitive detector are printed during the event. The <idspecref> gives the name of an <idspec> from the identifier dictionary.

The primary difference between the calorimeter and tracker types is that hits in the calorimeters are segmented into bins. Typical ILC calorimeters contain millions of cells per subdetector, and modeling these readout components as separate geometric objects would be prohibitive in terms of memory usage and simulation performance. So the layers in calorimeters are divided with virtual segmentation. An energy deposition in the calorimeter is added to the total for a single virtual cell. The energy contributions of each particle are tallied and stored into the hit should this information be needed later in reconstruction algorithms. A number of segmentation classes are provided for the end user.

GEANT4 regions are defined using <region> elements in the <regions> block. Regions are used to define the range cut, which determines the number of secondary particles produced. Trajectory storage can be tuned using the regions and physics limits may also be assigned to regions.

Some of the parameters controlling GEANT4's handling of tracks and steps can be tuned using physics limits, which are assignable to regions and volumes. These parameters are grouped into sets using the <limitset> element. Physics limits are referenced from logical volumes or regions using the <limitsetref> element.

```
<limits>
     <limitset name="TestLimitSet">
          <limit name="step_length_max"
                 particles="*"
                      value="1.0"
                 unit="mm"/>
          <limit name="track_length_max"
                 particles="p, pi+, pi-, p0"
                 value="100.0" unit="cm"/>
          <limit name="time_max"
                 particles="*"
                 value="100"
                 unit="ns"/>
          <limit name="ekin_min"
                 particles="*"
                 value="1.0"
                 unit="MeV"/>
          <limit name="range_min"
                 particles="e+,e-,gamma"
                 value="1.0"
                 unit="cm"/>
```

```
        </limitset>
    </limits>
```

The `step_length_max` sets the maximum distance of a single `G4Step`. Since SLIC uses virtual cells, the `step_length_max` can be used to minimize the occurrence of single steps crossing multiple cells. The `track_length_max` specifies the limit at which GEANT4 will kill a track. The `time_max` gives the maximum time a particle can exist after which it is killed. Tracks that are under the kinetic energy given by `ekin_min` will not be created. The `range_min` is duplicated by the region's range cut setting. Particles that will not go farther than the value in `range_min` will not be produced but the parent track's energy will be decremented by the energies of the un-produced secondary tracks (secondaries). The region's setting overrides the setting from the physics limits. Most of these settings can have adverse effects on the physics simulation. For instance, energy is not conserved when the tracks are killed. These settings are generally useful for "expert" users only but are still quite useful for tuning the simulation in certain volumes and regions.

Magnetic fields are defined in the `<fields>` block. The currently available types are simple solenoid, parameterized dipole, RZ field map and full field map.

The GDML `<volume>` is extended by LCDD to support additional reference tags. The GDML volume supports referencing of the solid and material. LCDD adds optional references to sensitive detector, region, limit set, and visualization elements (all previously described). The ref attribute gives the name of a previously defined element in the XML file, with the correct type.

```
<volume name="EcalBarrelEnvelope">
     <materialref ref="Air"/>
     <solidref ref="EcalBarrelTube"/>
     <sdref ref="EcalSD"/>
     <regionref ref="EcalBarrelRegion"/>
     <limitsetref ref="EcalBarrelLimits"/>
     <visref ref="EcalBarrelLimits"/>
     <physvol>
          <volumeref ref="EcalLayer1"/>
          <physvolid name="layer" id="1"/>
     </physol>
</volume>
```

This approach allows multiple volumes to reference the same elements, which is quite typical. For instance, each layer in a calorimeter needs to reference the sensitive detector. The other extension made to <volume> by LCDD is the addition of the `<physvolid>` that provides volume "numbering" in the context of the parent volume. Typically these values will correspond to important integer ids in the volume stack, such as layer number, sublayer number, subdetector number, and so forth. The full stack of physical volume identifiers is used to uniquely identify geometric elements using cell IDs, which are written into the hits in the LCIO file.

## GEOMETRY COMMANDS

The LCDD geometry file is a mandatory input that must be specified by the user at runtime. Other important settings, such as the physics list and event generator, are given reasonable defaults. The geometry can be read from a URL or local file path. The LCDD files for standard detectors are stored at a URL of this form.

```
http://www.lcsim.org/detectors/[detector_name]/[detector_name].lcdd
```

For instance, the LCDD file for the sid00 detector may be read as follows.

```
slic –g http://www.lcsim.org/detectors/sid00/sid00.lcdd
```

SLIC also accepts a local file url (file://), relative file path, or absolute file path.

Since the LCDD system is an extension of GDML's processing infrastructure, SLIC has the capability of loading plain GDML files.

```
slic –g mygeom.gdml
```

Conversely, the plain GDML processor is able to load a valid GDML file from LCDD by simply ignoring the LCDD elements.

## COMPACT DETECTOR DESCRIPTION

From the user's perspective, the LCDD format can be too verbose and complex for hand editing. Typically, researchers want to change a few parameters such as inner radius or layering scheme and have these changes automatically propagated to create the detailed full geometry. The Compact Detector Description, henceforth referred to as the "compact description", is a high-level format designed to facilitate this type of usage. The GeomConverter Java package converts compact descriptions to LCDD. It also supports the HepRep and GODL geometry formats and can create Java runtime objects representing the detector geometry for the org.lcsim reconstruction and analysis framework.

The compact format covers many of the same areas as LCDD, including meta-information, parameters defined by expressions and definitions, detector geometry, regions, and fields. The compact description also has an XML schema, much of which is similar in format to the LCDD schema. Each LCIO file contains a detector tag, such as "sidaug05", and the org.lcsim conditions system can locate and cache zip files containing the compact description and other geometry data sources, such as simple parameter files. The WIRED event display is given a HepRep geometry description produced from the compact description. LCDD files are produced from the compact description; then the LCDD file is used by SLIC. This is an example of the LCDD XML format being used as a "target" by an external application.

The following is a simple example of a compact description of a sampling Si/W calorimeter:

```
<detector id="2"
          name="EMBarrel"
          type="CylindricalBarrelCalorimeter"
          readout="EcalBarrHits">
     <dimensions inner_r="150.1*cm"
                 outer_z="208.0*cm" />
          <layer repeat="20">
               <slice material="Tungsten"
                      thickness="0.25*cm" />
               <slice material="Silicon"
                      thickness="0.032*cm"
                      sensitive="yes" />
          </layer>
</detector>
```

## EVENT GENERATION

SLIC has several methods for generating physics events. Built-in GEANT4 methods are available, such as the General Particle Source (GPS) and the particle gun. File-based sources are also supported. These are the StdHep binary format, supported by most physics event generators. Events can also be read from the MCParticle collection of an LCIO file.

## LCIO OUTPUT BINDING

SLIC creates LCIO files from the simulated GEANT4 events. SimCalorimeterHits, SimTrackerHits, and MCParticles are the three classes comprising SLIC's LCIO binding. A collection of MCParticle objects represents the particles created by the event generator and propagated through the simulation. Typically, an LCIO file will have one collection of MCParticles per event. Each subdetector has a single collection of hits. Hits in the tracking devices are represented by collections of SimTrackerHits. Calorimeter hits are written out to collections of SimCalorimeterHits. LCDD includes a set of "generic" hit classes, so that the sensitive detectors may create hits collections. These intermediate objects are converted to the LCIO format by utility classes in the SLIC package, primarily the `LcioHitsCollectionBuilder`. During this processing stage, the MCParticle contributions to each SimCalorimeterHit may be combined according to user settings.

## SOFTWARE DISTRIBUTION

The SLIC binary is built from 8 different software packages. Though these packages are primarily based on the GEANT4 build infrastructure, Xerces, LCIO, and GDML do not use directly the GEANT4 build system.

In order to simplify the distribution of the SLIC binary and manage the inter-package dependencies and different build systems, a build system based on Make and Autoconf is used to automate the tedious, manual configuration of each package. Instead, users need only checkout one CVS package to build SLIC.

```
> cvs –d :pserver:anonymous@cvs.freehep.org:/cvs/lcd co SimDist
> cd SimDist
> ./configure
> make
```

This system has been tested successfully on Linux, Windows, and OSX, and prebuilt binary executables are available for these platforms.

## SUMMARY

A fully-featured detector simulation package using the GEANT4 toolkit and runtime detector geometry definition has been developed. This package is being extensively used for detector design optimization for the ILC[5]. The program can be used to simulate arbitrary detectors and is freely available for other experiments.

## ACKNOWLEDGMENTS

Work supported by Department of Energy contract DE-AC03-76SF00515.

## REFERENCES

1. http://www.lcsim.org/software/slic/
2. http://geant4.web.cern.ch/geant4/
3. http://lcio.desy.de/
4. http://gdml.web.cern.ch/GDML/
5. Graf, et al. “Intelligent Detector design”, these proceedings