Marlin et al - A Software Framework for ILC detector R&D

F. Gaede[*], J. Engels[*]

December 17, 2007

Abstract

We give an overview of the core software framework for ILC detector R&D that is based on LCIO, Gear, Marlin and other packages. This is a general purpose framework covering a large range of computing needs for ILC detector development. It is used in Monte Carlo based studies aiming at the optimization of the complete ILC detector layout and in various test beam programs for specific subdetector prototypes.

[*]DESY, Hamburg, Germany

1 Introduction

The International Linear Collider ILC is the next big project proposed by the high energy physics community. The detectors for this collider will be precision experiments with highly granular calorimetry and excellent tracking capabilities. A software framework has been developed to support the research and development needed to design and optimize a detector concept for the ILC. The framework provides the basis for the development of reconstruction and analysis software. While originally designed for studying data from Monte Carlo simulation of the full detector response the framework has been adapted to also be used in test beam experiments with small subdetector prototypes. Using the same core software framework both in Monte Carlo studies and for analysis of real data provides synergies for both worlds. The adaption of the framework to test beam experiments has been carried out in the context of the EUDET[1] project supported by the European Union in the 6th Framework Programme structuring the European Research Area. Here we give an overview on the framework and report on some of its applications.

2 Overview Core tools

A typical processing chain in any high energy physics software framework consists of the following steps: *Generation, Simulation, Digitization, Reconstruction and Analysis.* Fig. 1 shows a schematic overview of the software framework and the tools that are used at the various steps.

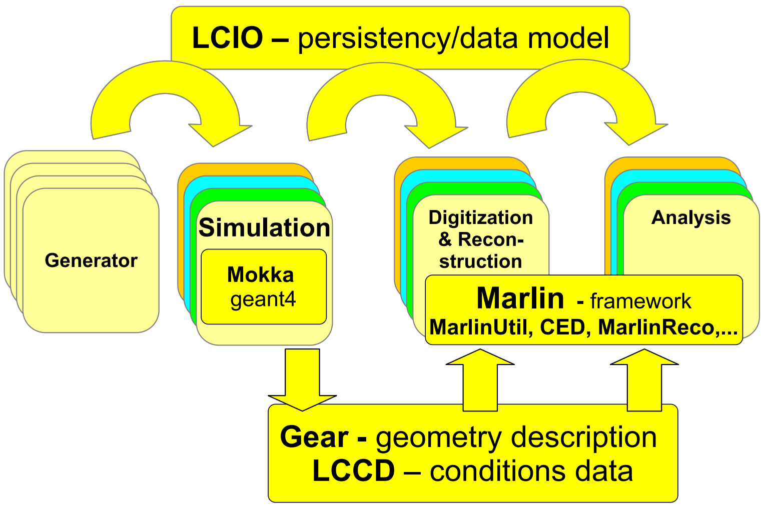


Figure 1: Schematic overview of the software framework and the tools that are used at the various levels of the processing chain.

The LCIO[3] persistency framework provides the data model that is used for the event and conditions data throughout the processing chain. The full simulation of the detector response is done in the geant4[4] based Mokka[7] application. Marlin[8] is an application framework that is used for further processing of the data at the digitization, reconstruction and analysis level. The core framework is completed by the Gear[5] tool, used to define the geometrical properties of the detector, and the LCCD[8] conditions data toolkit. In the following sections we will describe these tools in more detail.

2.1 Mokka

Mokka is an application targeted at computing the full detector response to generated events using the geant4 toolkit. Mokka has a flexible definition of the underlying detector that consists of two parts:

- dedicated C++ classes, so called drivers that define the geometrical layout, materials and the sensitive detector on a per subdetector basis
- a set of MySQL[21] databases, holding the parameters used in the corresponding geometry drivers

Mokka reads in the files produced from the generator program typically in the *stdhep* binary format and creates LCIO files holding the Monte Carlo truth particles and the simulated hits from the various tracker and calorimeter type subdetectors. Optimally, Mokka also creates Gear xml files with a subset of the geometry definitions that is needed for reconstruction (see 2.3). Additionally Mokka provides an API for querying the detector geometry, called CGA, that can be used in further processing of the simulated data.

2.2 LCIO

LCIO is a persistency framework and data model that has been adopted as a standard by the international ILC community. It defines an abstract event data model with hits, tracks, clusters etc. and a concrete file format to store the data. A Java, a C++ and a Fortran interface are provided to read and write LCIO data files. The hierarchical event data model is shown in Fig.2. It defines data entities for the different levels of the data processing where the objects hold pointers to the constituent objects from the next lower level, e.g. *Track* objects point back to the *TrackerHit* objects that are used in the track fit. Relations to Monte Carlo truth information is implemented only through indirect reference objects reflecting the good practice of having the same reconstruction and analysis code for simulated and real data. The LCIO event class *LCEvent* serves as a container of an arbitrary number of named collections of objects of the defined data types. For example, there can be several collections of type *CalorimeterHit* - one from each calorimeter subdetector - in the event. Additional user defined data objects can be stored in the LCIO event through an abstract interface called *LCGenericObject*. Implementations of which hold an arbitrary number of the basic types *int, float* and *double*. Through this mechanism virtually any data can be stored in LCIO files, a feature that is used in LCCD (see 2.4). As LCIO is also used in some test beam experiments some classes have been added to the data model, that are specifically designed to be

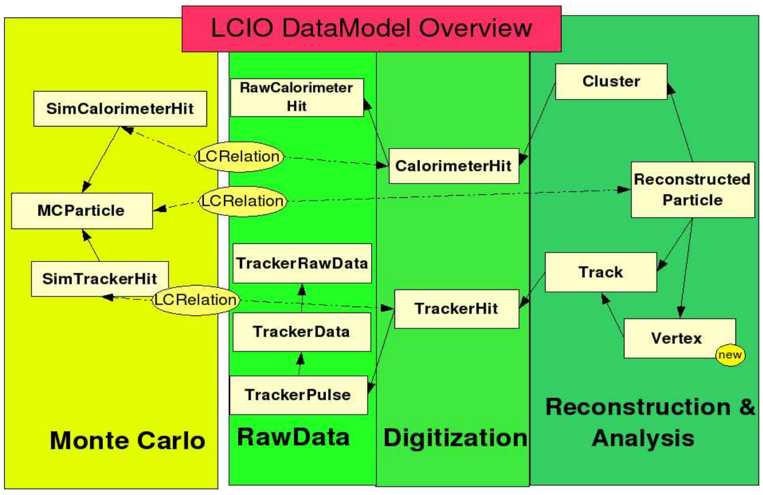


Figure 2: Schematic overview of the event data model defined by LCIO.

used in raw data files written by DAQ systems, such as *TrackerRawData*, *TrackerData* and *TrackerPulse*. Recently a new feature has been added to the C++ version that allows to define relations between arbitrary LCIO objects and to attach user defined objects to LCIO objects at runtime. This is a useful feature for developing reconstruction algorithms, that are implemented based on the LCIO data model.

2.3 Gear

Gear (**Ge**ometry **A**PI for **R**econstruction) is a geometry description toolkit for ILC reconstruction software. For the simulation of the detector response one needs a rather detailed description of the material distribution in space. For the event reconstruction however typically only a small subset of this detailed information needs to be provided in a more abstract view of the underlying sensitive detector's layout. This is particularly true if one wants to develop reconstruction algorithms that work on a variety of proposed detector concepts, as is the case for the ILC. Gear was developed to address these issues. The abstract API consists of two somewhat distinct parts:

- A set of subdetector parameter classes that define the coarse properties and general layout of certain subdetector types, such as a TPC like tracker or a (sandwich) type calorimeter. These classes additionally provide some simple navigation functionality, such as extracting the next layer of a silicon VXD detector seen in a given direction from any given point in space.

- An abstract interface that provides detailed information about the material and field properties at a given point in space (*GearPointProperties*) and integrated along a straight line connecting two space points (*GearDistanceProperties*).

The former API has been implemented by using xml files that provide the subset of geometrical information to Gear. In order to have only one well defined source of geometry,

these xml files are written out from the Mokka program used for the full simulation. This is done in the corresponding subdetector drivers, based on the used data base parameters (see 2.1).
The latter interface is implemented based on the CGA library provided by Mokka. It requires linking the reconstruction program to Mokka and geant4. On startup of the program the full geant4 detector geometry is instantiated and is then used to provide the requested material and field properties at runtime. Tab.1 shows the methods that can be used to query the material properties at any given space point.

GearPointProperties:	
long64	**getCellID (const Vector3D &pos)** The cellID of the the sensitive detector at pos.
const string&	**getMaterialName (const Vector3D &pos)** Name of material at pos.
double	**getDensity (const Vector3D &pos)** Density in kg/m^3 at pos.
double	**getTemperature (const Vector3D &pos)** Temperature in K of material at pos.
double	**getPressure (const Vector3D &pos)** Pressure in P at pos.
double	**getRadlen (const Vector3D &pos)** Radiation length of material in mm at pos.
double	**getIntlen (const Vector3D &pos)** Interaction length of material in mm at pos.
Vector3D	**getLocalPosition (const Vector3D &pos)** Position in local coordinate.
Vector3D	**getB (const Vector3D &pos)** The magnetic field vector at pos in [Tesla].
Vector3D	**getE (const Vector3D &pos)** The electric field vector at pos in [V/m].
vector<string>	**getListOfLogicalVolumes (const Vector3D &pos)** Names of (geant4) logical volumes in heirarchy starting at given pos ending with the world volume.
vector<string>	**getListOfPhysicalVolumes (const Vector3D &pos)** Names of (geant4) physical volumes in heirarchy starting at given pos ending with the world volume.
string	**getRegion (const Vector3D &pos)** Names of (geant4) region that contains the given pos.
bool	**isTracker (const Vector3D &pos)** True if region that contains pos is defined as a tracker.
bool	**isCalorimeter (const Vector3D &pos)** True if region that contains pos is defined as a calorimeter.

Table 1: Methods of the GearPointProperties interface. The GearDistanceProperties interface provides methods to query the integrated radiation and interaction lengths as well as the integrated B and E field along a straight line between two space points.

While providing all the necessary material information, using the detailed geometry also causes some run time overhead. This might turn out to be prohibitive for reconstruction code used in mass production. A future improvement of the system could be to work on a somewhat simplified version of the geometry description which in turn would allow for faster navigation during reconstruction.

2.4 LCCD

LCCD (**L**inear **C**ollider **C**onditions **D**ata Toolkit) is a toolkit for storing and retrieving conditions data. Such a package is of particular importance for test beams in order to monitor the experimental setup such as slow control readouts, electronic channel mappings, calibration constants and alignment. The key features are the possibility to tag/version certain data sets and to retrieve data sets based on a time key. Typically large conditions data bases are used to provide these features but in some cases such an approach is to heavy weight and simple flat files, holding the needed parameters, are sufficient. LCCD combines these two options in a way that is transparent to the user program by defining an abstract interface on top of the underlying storage mechanism. The software architecture of LCCD is shown in Fig. 3. LCCD uses the *LCGenericObject*

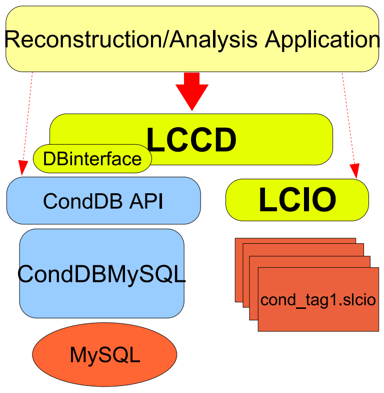


Figure 3: The API layer of LCCD provides transparent access to conditions data stored in a dedicated database or in LCIO files.

(see 2.2) of LCIO to store user defined conditions data classes, either in a database implemented with the CondDBMySQL [6] package or directly in LCIO files. Four different scenarios for reading the appropriate conditions data are implemented:

1. Reading conditions data from a CondDBMySQL database based on a given time stamp and an additional tag. The time stamp is typically taken from the LCIO event class.

2. Reading a specific set of conditions data, e.g. calibration constants from an LCIO file that has been created containing the conditions data, only.

3. Reading conditions data that occur in the event data stream of an LCIO file, e.g. when written directly from the slow control system of the DAQ interleaved with the event data.

4. Reading conditions data from a snapshot of the full database that holds the data for all time validity intervals of one given tag. LCCD provides the functionality to create such a snapshot LCIO file.

All four use cases are implemented through a common interface *IConditionsHandler* making user code independent of the actual data source. This mechanism is applied in Marlin.

2.5 Marlin

Marlin (**M**odular **A**nalysis and **R**econstruction for the **LIN**ear collider) is a modular C++ application framework for the analysis of LCIO data. It provides a platform for the distributed development of ILC detector reconstruction and analysis algorithms. Marlin uses the LCIO event data model as its transient data model, i.e. the LCIO event class serves as a container of the data that is *shipped* from processor to processor[1], as shown schematically in Fig. 4.

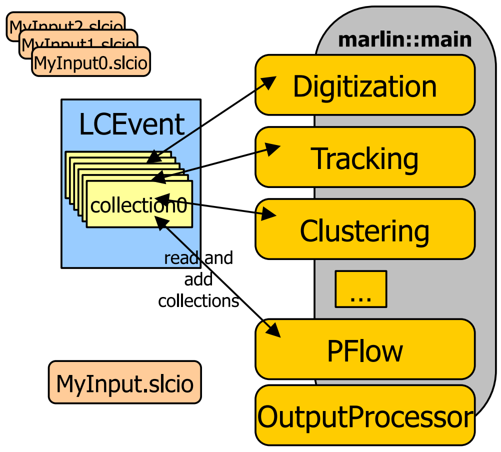


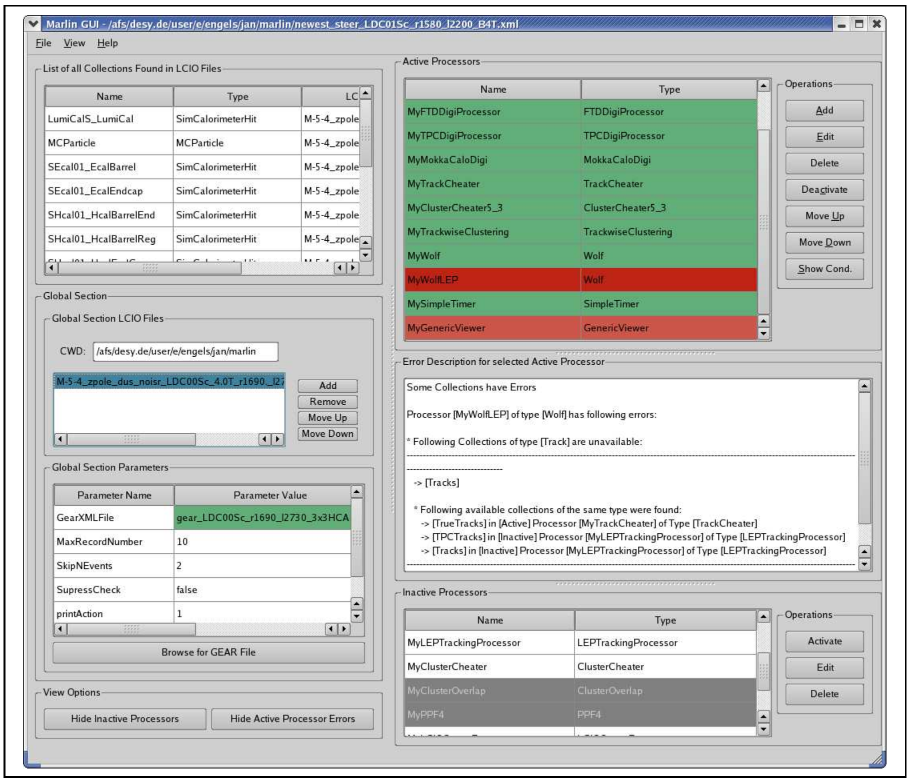
Figure 4: Schematic overview of the modular structure of Marlin. LCIO is used as the transient data model where the LCIO event serves as a container of the data seen by the *Processors*.

User code is implemented in subclasses of *Processor* which defines the relevant callbacks needed for the event loop. Marlin provides interfaces to Gear and LCCD, so that detector geometry information and conditions data are available at the processor level.
Marlin applications are entirely configured through XML steering files. The XML files hold arbitrary named parameters - defined either for a particular processor, a group of processors or globally. The steering file also defines which processors are to be called in which order, optionally allowing for logical conditions that need to be fulfilled. These conditions are evaluated at runtime and allow to skip certain computations if their prerequisites are not met.
Recently some extensions have been added to the Marlin framework in order to improve the usability for the users and developers:

2.5.1 MarlinGUI

A GUI has been developed that facilitates the creation of the appropriate xml steering file. A task that - though rather straight forward in principle - has caused some user concerns, in particular when configuring complex reconstruction jobs. A snapshot of the GUI is shown in Fig.5. In the GUI main window the following properties are displayed

[1]the Marlin modules are called *Processor*



List of all Collections Found in LCIO Files

Name	Type	LC
LumiCalS_LumiCal	SimCalorimeterHit	M-5-4_zpole
MCParticle	MCParticle	M-5-4_zpole
SEcal01_EcalBarrel	SimCalorimeterHit	M-5-4_zpole
SEcal01_EcalEndcap	SimCalorimeterHit	M-5-4_zpole
SHcal01_HcalBarrelEnd	SimCalorimeterHit	M-5-4_zpole
SHcal01_HcalBarrelReg	SimCalorimeterHit	M-5-4_zpole

Global Section

Global Section LCIO Files

CWD: /afs/desy.de/user/e/engels/jan/marlin

M-5-4_zpole_dus_noisr_LDC00Sc_4.0T_r1690._l27

Global Section Parameters

Parameter Name	Parameter Value
GearXMLFile	gear_LDC00Sc_r1690_l2730_3x3HCA
MaxRecordNumber	10
SkipNEvents	2
SupressCheck	false
printAction	1

Active Processors

Name	Type
MyFTDDigiProcessor	FTDDigiProcessor
MyTPCDigiProcessor	TPCDigiProcessor
MyMokkaCaloDigi	MokkaCaloDigi
MyTrackCheater	TrackCheater
MyClusterCheater5_3	ClusterCheater5_3
MyTrackwiseClustering	TrackwiseClustering
MyWolf	Wolf
MyWolfLEP	Wolf
MySimpleTimer	SimpleTimer
MyGenericViewer	GenericViewer

Error Description for selected Active Processor

```
Some Collections have Errors

Processor [MyWolfLEP] of type [Wolf] has following errors:

* Following Collections of type [Track] are unavailable:
--------------------------------------------------------------------------------
----------------------------
  -> [Tracks]

  * Following available collections of the same type were found:
    -> [TrueTracks] in [Active] Processor [MyTrackCheater] of Type [TrackCheater]
    -> [TPCTracks] in [Inactive] Processor [MyLEPTrackingProcessor] of Type [LEPTrackingProcessor]
    -> [Tracks] in [Inactive] Processor [MyLEPTrackingProcessor] of Type [LEPTrackingProcessor]
--------------------------------------------------------------------------------
```

Inactive Processors

Name	Type
MyLEPTrackingProcessor	LEPTrackingProcessor
MyClusterCheater	ClusterCheater
MyClusterOverlap	ClusterOverlap
MyPPF4	PPF4

Figure 5: MarlinGUI snapshot

and available for modification:

- The list of global parameters such as the LCIO input files and the Gear input file.
- A list of available LCIO collection names and their corresponding data types.
- A list of active processors, color coded with respect to the consistency of the required input collection names and types.

The GUI offers intuitive dialogs for adding and removing processors, changing their order and modifying the processor parameters.
A simplified version of the built in consistency check is also available as a command line option to the Marlin application for non-graphical environments.

2.5.2 Marlin Plugins

Another improvement in the usability of the Marlin framework is the introduction of a plugin mechanism that allows to define at startup time which plugins are to be loaded. Plugins consist of shared libraries created from a package containing one or more Marlin processors. By specifying a list of plugins users can reconfigure their Marlin application without the need to relink the binary. Creating plugins is straight forward due to the new build system introduced recently (see sec.5.1)

3 Additional packages

The software framework described in the previous sections is completed by a number of additional supporting packages:

- **MarlinUtil** a utility library providing core utilities for the development of analysis and reconstruction algorithms.
- **RAIDA** a histogram package based on AIDA and root.
- **CED** a fast client server event display based on OpenGL
- **CEDViewer** a Marlin plugin providing event display clients for CED
- **Overlay** a Marlin package providing event overlay functionality for background and pileup studies
- **MarlinReco** [9] a Marlin package aiming at providing a complete set of Reconstruction algorithms, including pattern recognition, clustering and particle flow.

Mathematical and physical utilities are provided through a small number of external packages: CLHEP[17], gsl[18], cernlib[19] and root[20].

4 Framework Applications

As mentioned earlier the framework has been designed to facilitate the distributed development of ILC reconstruction and analysis tools both for Monte Carlo based optimization of the full detector and for ongoing and planned test beam experiments. In this section we briefly describe some of the packages that are developed within the framework:

PandoraPFA [11] is a package implementing a reconstruction algorithm that follows the *particle flow* [2] paradigm. The key idea of particle flow is that every particle in a parton jet is identified and reconstructed. The particle energy is then reconstructed from the high resolution tracking detector measurement for all charged particles, from the electromagnetic calorimeter for photons and from the hadron colorimeter for neutral hadrons. It has been shown that with this procedure an unprecedented jet energy resolution of $\frac{\sigma_E}{E} \leq \frac{30\%}{\sqrt{E/GeV}}$ can be achieved for a wide range of the expected jet energies at the ILC [12].

LCFIVertex [13] is a package that provides the vertex finder algorithm ZVTOP[14], originally developed for SLD, flavor tagging as well as vertex charge determination for b- and c-jets. The provided flavor tag is obtained from a neural net approach, combining track and vertex information to distinguish b, c- and light quark jets.

EUTelescope [15] is a package targeted at the reconstruction and analysis of data from the high resolution, low material pixel telescope that is developed within the EUDET project. It provides functionality for calibration, alignment and data reduction as well as for pattern recognition and determination of the resolution of the DUT (device under test).

MarlinTPC [10] is a toolkit for analysis and reconstruction of TPC prototype data. The developed algorithms for calibration, hit finding and track fitting are applicable for different TPC endcap technologies. The main focus is the application at the planned large common TPC prototype.

4.1 Grid

While developing the software framework attention has been paid to design it in a way that is compatible with running applications on the Grid. This has been extensively used by the above mentioned applications and groups. For example during this year's calorimeter test beam at CERN, the Calice collaboration has recorded more than 200 million events[16]. These data have been processed on the Grid, using the software framework and the resulting LCIO files are made available for further analysis on the Grid.

5 Software usability

With the ever increasing functionality, complexity and number of users of the software framework it is important to provide tools that enhance and simplify the usability of the system. In this section we describe some of the measures implemented to that end:

5.1 Build Tool

The original build tool for the various software tools consisted of simple Makefiles that have proven to be more and more difficult to maintain, extend and port to different target platforms. To overcome this burden for the developers and users involved, we have recently changed to a new build system: CMake[23]. CMake generates native build environments for a number of different platforms and IDEs. The project's build parameters are defined in a simple to learn CMake-specific language in platform independent text files named CMakeLists.txt. These files are converted by CMake into the system-specific Makefiles. Build in features of CMake make managing inter package dependencies and cross platform development easier to maintain. This is why it is also used for major Open Source projects like kde[24] and others. When using CMake to build the framework packages, shared libraries are created by default. A feature that is needed for the Marlin plugin mechanism described above.

5.2 Installation Script

While it is in principle of course possible to install all of the ILC software packages manually it is a somewhat tedious and time consuming procedure, that might lead to inconsistencies in the inter package dependencies. Therefore an installation script for the ILC software framework has been developed:

ilcinstall

ilcinstall[22] is a flexible, fully configurable and automated way of downloading and installing all of the ILC software packages described above. This includes the necessary external packages that are needed. A simple to edit configuration file is used for defining the packages and their corresponding versions that are to be downloaded and installed. Already installed packages - for example external tools like CLHEP - can be linked into the installation directory. A special module, *MarlinPKG* is available in ilcinstall for user defined Marlin packages. This class may be used for installing any Marlin package that follows the structure provided in the *mymarlin* example shipped with Marlin. Released versions of ilcinstall include a configuration script that defines a set of interoperable versions of all the packages - this mechanism is used to define *ILC software releases.*
The ilcinstall script is also used for nightly builds that include the current HEAD versions of all the tools. This allows for fast feedback on possible inconsistencies to the package developers.

reference installations

The above mentioned versions of the *ILC software releases* are also installed at:

```
/afs/desy.de/group/it/ilcsoft/vXX-YY
```

as reference installations for the SL3 and SL4 operating systems. Users can build their own (updated) packages against these releases without the need to install everything at their home institute. This is supported by a special CMake configuration file: ILC-Soft.cmake which is automatically generated in the installation directory by ilcinstall.

5.3 Documentation

Proper documentation of the software is of great importance for the users of the framework. All of the tools have a rather complete API documentation which is automatically generated from the header files using doxygen[25]. Also README files and manuals are provided. In order to ease access to the documentation a software portal has been created at:

```
http://ilcsoft.desy.de
```

It serves as a common entry point for the users and provides links to:

- cvs code repositories

- API documentation
- readme files and manuals
- bug tracking and feature request tools
- discussion forums

6 Conclusion

We have given an overview of a software framework for ILC detector R&D. The framework provides the core functionality that is needed for the analysis of ILC related data, both from Monte Carlo simulation for full detector optimization and for studying data from test beam experiments. Earlier this year we have released a version “v01-00” of the software framework, marking the achievement of the deliverable “Version 1.0 of analysis framework” of the EUDET project’s task NA2-ANALYS. We will continue to improve the software tools, expecting important feedback from the planned mass production of Monte Carlo events for detector optimization as well as several upcoming test beam data taking periods.

Acknowledgement

This work is partly supported by the Commission of the European Communities under the 6th Framework Programme ”Structuring the European Research Area”, contract number RII3-026126.

References

[1] see: `http://www.eudet.org`

[2] R.D. Heuer, D. Miller, F. Richard, P. Zerwas (eds.), ”TESLA Technical Design Report, Part IV”, DESY 2001-011, ECFA 2001-209, 2001.

[3] F. Gaede, T. Behnke, N. Graf, T. Johnson *CHEP03 March24-28, 2003 La Jolla, USA*, TUKT001, *arXiv:physics/0306114.*

[4] S. Agostinelli *et al.* [GEANT4 Collaboration], Nucl. Instrum. Meth. A **506** (2003) 250.

[5] see: `http://ilcsoft.desy.de/portal/software_packages/gear`

[6] A. Amorim, J. Lima, C. Oliveira, L. Pedro and N. Barros, *CHEP03 March24-28, 2003 La Jolla, USA* , MOKT003, *arXiv:cs.db/0306006.*

[7] P. Mora de Freitas and H. Videau, LC-TOOL-2003-010 *Prepared for LCWS 2002, Jeju Island, Korea, 26-30 Aug 2002*

[8] F. Gaede, “Marlin and LCCD: Software tools for the ILC,” Nucl. Instrum. Meth. A **559** (2006) 177.

[9] O. Wendt, F. Gaede and T. Kramer, “Event reconstruction with MarlinReco at the ILC,” arXiv:physics/0702171.

[10] J. Abernathy *et al.*, “MarlinTPC: A Marlin based common TPC software framework for the LC-TPC collaboration,” arXiv:0709.0790 [physics.ins-det].

[11] see: `http://www.hep.phy.cam.ac.uk/~thomson/pandoraPFA`

[12] M. A. Thomson, “Particle flow calorimetry at the ILC,” AIP Conf. Proc. **896** (2007) 215.

[13] S. Hillert [LCFI Collaboration], “The LCFIVertex Package: vertex detector-based Reconstruction at the ILC,” arXiv:0708.2622 [physics.ins-det].

[14] D. Jackson, Nucl. Instrum. Meth. A **388** (1997) 247.

[15] A. Bulgheroni, T. Klimkovich, P. Roloff, A.F. Zarnecki, “EUTelescope: tracking software,” EUDET-Memo-2007-20,

[16] E. Garutti and F. Salvatore [CALICE Collaboration], “Summary Of The 2007 Calice Test Beam At Cern,”

[17] L. Lonnblad, “CLHEP: A project for designing a C++ class library for high-energy physics,” Comput. Phys. Commun. **84** (1994) 307.

[18] see: `http://www.gnu.org/software/gsl`

[19] see: `http://cernlib.web.cern.ch/cernlib`

[20] R. Brun, F. Rademakers and S. Panacek, “ROOT, an object oriented data analysis framework,” *Prepared for CERN School of Computing (CSC 2000), Marathon, Greece, 17-30 Sep 2000*

[21] see: `http://www.mysql.org`

[22] see: `http://ilcsoft.desy.de/portal/software_packages/ilcinstall`

[23] Martin, K., Hoffman, B. 2005, *Mastering CMake*, Kitware, ISBN 1-930934-16-5

[24] see: `http://www.kde.org`

[25] see: `http://www.stack.nl/~dimitri/doxygen`