# MarlinReco
# A Marlin based Reconstruction Package for the ILC

T. Krämer et.al., DESY

December 2, 2016

**Abstract**

This document serves as a comprehensive manual to help users getting started with the Marlin based reconstruction software MarlinReco for the international linear collider (ILC). After a short review of the underlying packages (LCIO, Gear, Marlin, MarlinUtil) and a summary of required libraries, a introduction to the features of MarlinReco is given. Furthermore a detailed description helps to install MarlinReco together with all underlying packages. It is also explained how to invoke MarlinReco and influence its behaviour using the steering file. Finally you learn how to write own processors so that many scientist from the HEP community can contribute to this Project.

# Contents

# 1 Introduction

The software package MarlinReco serves as an event reconstruction, used to support the detector development and optimisation at the proposed international linear collider (ILC) [23]. The different reconstruction algorithms i.e. track finding or cluster reconstruction are implemented in separate modules called processors. To improve the modularity, algorithms common to more processors are separated and implemented in a dedicated utility section. In the phase of the detector development as well as during data taking the detector may change. In order to adopt the reconstruction to the modified detector layout, the processors are developed to be independent of the detector geometry as far as that is possible. The actual geometry of the detector is then provided to the processors using a geometry interface. This structure not only allows the interchange of different reconstruction methods for a specific component of the detector, it also opens the possibility to use the same reconstruction for different detector concepts. A high level of modularity ensures the extendibility of the package and supports distributed development.

MarlinReco mainly depends on the analysis and reconstruction software framework Marlin [2] and on the data and persistency model LCIO [3, 4].

The Software is portable so that it can be used on a wide range of different UNIX systems. With some effort it also can be installed under MS Windows XP.

# 2 Dependency on other Packages

In order to keep the software light weight and easy to use, it is tried to keep the dependencies as moderate as possible. Nevertheless some packages have to be available on your system to run MarlinReco. This section is meant to give an overview. More detailed information can be obtained from the project homepages, where user manuals or Application Programming Interface (API) documentations are available.

## 2.1 LCIO

Linear Collider I/O (LCIO) [3,4] 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. The LCIO API has been implemented in a way that makes the user code independent of the concrete file format. So the file format could be exchanged without breaking any existing user code if the need will arises.

In principle LCIO defines the complete event data model for ILC studies. Nevertheless users often need to store additional information together with the event data. This is possible with the help of an abstract interface called *LCGenericObject* which holds arbitrary numbers of the simple types *int*, *float* and *double*. Through this interface virtually any data can be stored in LCIO files.

## 2.2 Marlin

Marlin (**M**odular **A**nalysis and **R**econstruction for the **LIN**ear collider) [2] is a simple and generic C++ application framework for the analysis of LCIO data that provides a platform for the distributed development of new reconstruction algorithms. Marlin supports the implementation of different tasks in individual modules, where tasks can be as simple as creating and filling histograms or as complex as track finding and fitting in the central tracker. As modules serve dedicated subclasses of the class *Processor*, Marlin uses LCIO also as its transient data model, i.e. the LCIO event class serves as a container of the data that is *shipped* from module to module as shown schematically in fig. 1.

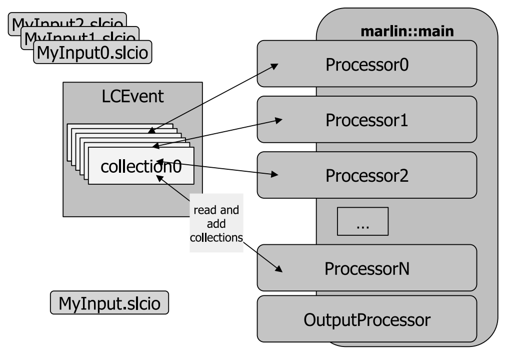


Figure 1: Schematic overview of the modular structure of Marlin. LCIO is used as the transient data model where the LCIO event serves as a container of the data seen by the *Processors*.

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 files also define the order in which the processors are called as well as optionally logical conditions assigned to some processors that are evaluated at runtime.

Marlin is released with a number of processors implementing some core functionality such as creating histograms, simple fast smearing algorithms or conditions data handler.

## 2.3 Gear

One important ingredient for reconstruction software is the geometrical description of the detector which is used to record the events. Gear [1] is a geometry description toolkit for ILC reconstruction software, defining an abstract interface for the description of an ILC detector. This description is different from the geometry description needed for the detailed simulation of the detector response, though of course related. For the simulation one needs very detailed knowledge of the material distribution in space whereas for the reconstruction mostly a higher level description of the subdetector's characteristics is needed. Depending on the particular reconstruction algorithm a hand full of parameters might suffice to describe a full subdetector, e.g. the main calorimeters. This description is certainly a lot less than what is needed for the computation of the detailed detector response with GEANT4 [6]. Of course the required parameters could be defined in the Marlin steering file. However this procedure is error prone and tedious as the parameters usually have to be extracted manually from some other source, typically the full simulation program that has been used. The aim of Gear is now to provide the interface to a global source of information for the reconstruction view of the detector geometry.

A first implementation uses XML files for the definition of the subdetector's geometry parameters in Gear. The user code however only applies the abstract interface and is thus independent from the XML-file implementation.

Currently under development is an implementation for the abstract interface to generic material properties in space, like the material's density at a given point or the integrated B-field along a (straight) path between two points.

The decoupling of the geometrical information of the detector from the reconstruction is the only way to get algorithms being independent of a specific detector model.

## 2.4 MarlinUtil

The MarlinUtil package is a library with commonly used classes and functions. All these classes and functions can be accessed from any processors in Marlin by simply binding in this package. The content of the library can be roughly divided into four sets:

1. Extension Classes for Tracker- and Calorimeter Hit classes. They expand the functionality of the corresponding, native LCIO classes. Moreover there is a class for grouping track segments.

2. Cluster Shape and Helix classes. These classes provide analysis functionality for clusters of calorimeter hits, such as calculating centre of gravity as well as fits of cluster shapes and simple helixes.

3. CED classes. These are intrinsic classes needed for the handling of the C Event Display (CED).

4. Miscellaneous classes, such as classes for printing the Monte Carlo particle tree of a LCIO file to the standard output and headers with math and physics constants.

This library is under current development, so it will evolve in the future.

### 2.5 AIDA

Using MarlinReco to reconstruct HEP events and in a later stage even to analyse real data requires a interface to histograming packages. For this task we utilise the Abstract Interfaces for Data Analysis (AIDA) [15]. It offers a standard set of interfaces for creating and manipulating histograms, n-tuples and related data analysis objects. Two versions of the AIDA interfaces exist, one for Java and one for C++. The two interfaces are as identical as the underlying languages will permit. Currently a Java implementation – JAIDA [16] – is available which also can be accessed from C++ via AIDA-JNI [17] (Java Native Interface). MarlinReco uses AIDA-JNI to enable the production of check plots. AIDA compliant applications[1] allow to access, process or analyse the information stored through the AIDA interface.

### 2.6 Other Libraries

Marlin [2] and some of the MarlinReco processors depend on the "Class Library for High Energy Physics" (CLHEP) [14]. It is recommended to use a version below 2.0. Version 2.0 and higher use a own CLHEP namespace. This has not jet been implemented in Marlin, therefore those CLHEP releases are not compatible to Marlin. This choice has been made to use the same CLHEP version for the Reconstruction and simulation (GEANT 4 [6]). In addition the "GNU Scientific Library" (GSL) [12] is required.

The TPC reconstruction algorithms are taken from former LEP experiments. they are coded in FORTRAN 77 and depend on the "CERN Program Library" (CERNLIB) [13].

[1] JAS3 [18], The PI project in LCG-AA [19] and Open Scientist [20]

# 3 MarlinReco

In this section an introduction to the MarlinReco software and the visualisation of (reconstructed) events using the CED (C Event Display) program is given. Furthermore it is explained how to run a MarlinReco job to do reconstruction. The last subsection contains a short tutorial that shows you how to write your own Marlin processor.

## 3.1 Reconstruction Algorithms

The heart of the reconstruction package are the Marlin processors which hold the different modules of algorithms to get tracks and clusters from the simulated data. This section is meant to give a short introduction to the different processors currently available. More detailed information, especially about the usage of the modules and their steering parameters can be obtained in the API documentation, which can be build together with MarlinReco using Doxygen [22].

### 3.1.1 Digitisation

There are digitisation algorithms available for the TPC and the calorimeters. The TPC digitisation converts a *SimTrackerHit* to a *TrackerHit* by smearing in $r - \phi$ and $z$ direction.

The calorimeter digitiser processor takes a *SimCalorimeterHit* and produces a *CalorimeterHit*. The simulated energy depositions in the active layers of the calorimeters are converted into physical energy. This is done by taking into account the sampling fractions of the electromagnetic and hadronic calorimeters (ECAL, HCAL).

For the vertex detector and the forward tracking system only dummy processors that do not apply any smearing exist.

**Available processors:**

- *FTDDigiProcessor*
- *TPCDigiProcessor*
- *VTXDigiProcessor*
- *MokkaCaloDigi*
- *SimpleCaloDigi*

### 3.1.2 TPC Tracking

The TPC tracking produces *Tracks* from TPC *TrackerHits* using LEP tracking algorithms taken from the Fortran based reconstruction program Brahms [10]. The LEPTracking processor now includes full unambiguous central tracking during which track strings are created through to full *Track* candidates (TK's) which have passed the Delphi Ambiguity resolver fxambi.

Currently only the TPC pattern recognition and fitting is implemented, though plans to incorporate vertex hits into tracks exist.

The final track fitting is done using a Kalman filter. Presently the VTX and SIT resolutions are hard coded in *LEPTrackingProcessor*. Both, the magnetic field and the material description of the VTX detector and SIT, as for the TPC, is hard coded in setmat. This will be improved as soon as Gear [1] becomes more evolved.

**Available processors:**

- *LEPTrackingProcessor*

### 3.1.3 Clustering Algorithm

The processor performs clustering of calorimeter hits without using information about reconstructed tracks. The dependence of the clustering procedure on the detector geometry is minimised. Only the follwoing quanteties are used:

- radius of the ECAL barrel
- n-fold symmetry of barrel (n = 8 for TESLA)
- $\phi$ offset of the barrel stave w.r.t. the x-axis (0 for TESLA detector)
- $\pm z$ coordinate of front face of ECAL endcaps

These parameters are taken from the Gear [1] XML file.

**Available processors:**

- *TrackwiseClustering*

### 3.1.4 Particle Flow Algorithm

The algorithm matches *Clusters* to *Tracks*, reconstructs particles and performs particle identification on the basis of the energy fraction deposited in the ECAL. If this fraction is greater than a predefined threshold, the particle is regarded to be a photon or electron depending on whether a track is matched to the cluster. Likewise muons and neutral and charged hadrons are identified.

The algorithm depends on the same geometrical quantities as the *TrackwiseClustering* processor. In addition it considers the magnetic field.

**Available processor:**

- *Wolf*

### 3.1.5 Cheaters

Cheater processors construct tracks or clusters by combining hits, produced by the full detector simulation with Monte Carlo information. These processors can be used to benchmark and test existing algorithms or as replacement for yet to be written parts of the reconstruction.

At the moment there are two cheaters available:

**Track Cheater:**
It constructs true tracks by comprising all the *TrackerHits* attributable to the same (Monte Carlo) particle. If a track has more than a given number of hits and if the particle producing it carries at least a certain energy it is considered. The Monte Carlo information (momentum and vertex) is used to define the track parameters $d_0, z_0, \phi, \tan\lambda$ and $\omega$. Optional a fit can be performed to the *TrackerHits* assigned to a track to extract the track parameters.

**Cluster Cheater:**
This processor constructs true clusters, where a true cluster comprises all hits attributed to the same generator particle entering the calorimeter. Hits produced by back-scattered particles are assigned to the primary which initiated this back-scattered particle. Clusters are considered only if they have at least a predefined number of hits. To avoid collecting remote hits produced by neutrons or backscattered particles, a proximity criterion is introduced in terms of a particle-to-hit distance cut. A hit is assigned to the cluster produced by a neutral particle if the following condition is fulfilled:

$$D\{\text{hit-to-vertex}\} \cdot \alpha < \text{ProximityCut}$$

$D\{\text{hit-to-vertex}\}$ is the distance of the hit to the point where the neutral particle has been produced, and $\alpha$ is the angle between the particle momentum vector and the vector connecting the particle vertex and the hit. For charged particles, the distance from the hit to the helix associated with the particle trajectory should not exceed the proximity cut.

**Available processors:**

- *TrackCheater*
- *ClusterCheater*

### 3.1.6 Event Shape Variables

There are several processors to calculate different event shape variables:

- Calculation of the thrust axis and thrust value for each event using two different algorithms: The Tasso algorithm calculates only the principle thrust value and axis. Jetset algorithm calculates the principle thrust value and axis as well as the major and minor thrust value and axis.
- Calculation of the sphericity and aplanarity.
- A universal jet finder module originally developed by Satoru Yamashita for OPAL.

**Available processor:**

- *ThrustReconstruction*
- *Sphere*
- *SatoruJetFinderProcessor*

## 3.2 Structure of a MarlinReco Steering File

Building MarlinReco with the top level makefile automatically produces a Marlin executable in the directory `$LCSoft/Marlin/bin` named `Marlin` (`$LCSoft` is the path to the directory where MarlinReco is installed as explained in section 4). It includes all processors and is linked statically against all libraries needed (Marlin, Gear, CLHEP, GSL ...) except the shared libraries provided by the linux system.

A Marlin job is fully configured by its steering file. Therefore, it is a good starting point to understand how to configure a MarlinReco application. In principle there exist two formats of the Marlin steering file: ASCII and XML. The older ASCII steering files are somewhat restricted in their functionality and are only kept for downward compatibility. For that reason we recommend to use XML steering files and only discuss those. Every Marlin executable is able to print a sample steering file to standard out. Type

```
$LCSoft/Marlin/bin/Marlin -x > exampleSteeringFile.xml
```

to get an example steering file in the XML format. You can inspect this files using your favourite text editor (emacs, pico, vi ...).

Every steering file starts with `<marlin>` and ends with `</marlin>`. (To include a comment at any point of the file use the `<!-- ... -->` tag.) Inside the file is divided into three types of sections:

**Execute Section:**
In this section the names of the processors which are to be executed are listed using the keyword `processor`. The order of the processor names in the XML-file in this section controls the order of executing the processors. A typical `execute` section could look like this:

```
<execute>
   <processor name="MyAIDAProcessor"/>
   <processor name="MyTestProcessor"/>
   <processor name="MyLCIOOutputProcessor"/>
</execute>
```

It specifies three processors which are called in the order (1) MyAIDAProcessor, (2) MyTestProcessor, (3) MyLCIOOutputProcessor.

**Global Section:**
Here all the global settings are made:

- Specification of the LCIO input files (including paths) which contain the event data
- The number of events which are to be processed
- Turn on/off check routines to produce control plots etc.

A typical `global` section could look like this:

```
<global>
   <parameter name="LCIOInputFiles"> simjob.slcio </parameter>
   <parameter name="MaxRecordNumber" value="5001" />
   <parameter name="SupressCheck" value="false" />
</global>
```

**Processor Section:**
This section contains several blocks named `processor`. Every block configures a certain processor and gives it a unique name. These names are used in the `execute` section to call the processor. Here one processor can be configured more than once using different names as identifier. In this way a processor can be called several times with different parameters. This can be useful to optimise cuts or to use the same algorithm for different input collections. Steering parameters of processors can be of type *string*, *int* or *float*. A typical `processor` section can look like this:

```
<processor name="MyClusterCheater" type="ClusterCheater">
 <!--Creates true clusters...-->
  <!--Calorimeter Collection Names-->
  <parameter name="CaloCollections" type="StringVec">ECAL HCAL  </parameter>
  <!--Magnetic Field-->
  <parameter name="MagneticField" type="float">4 </parameter>
  <!--Minimal Hits in Cluster-->
  <parameter name="MinimalHits" type="int">10 </parameter>
  <!--Hit To Cluster Proximity Cut-->
  <parameter name="ProximityCut" type="float">1200 </parameter>
  <!--SimCaloHit to CaloHit Relations Collection Name-->
  <parameter name="RelCollection" type="string">RelationCaloHit </parameter>
  <!--Trace Whole Shower Tree-->
  <parameter name="TraceWholeShower" type="int">1 </parameter>
  <!--Collection of True Clusters-->
  <parameter name="TrueClusterCollection" type="string">TrueClusters </parameter>
</processor>
```

This section describes the configuration of the processor of type *ClusterCheater*. It can be called in the `execute` section using the name `MyClusterCheater`:

```
<execute>
   . . .
   <processor name="MyClusterCheater"/>
   . . .
</execute>
```

In the definition of a parameter not only its name and value is specified but also the type of the expected input(s).

### 3.3 Running a MarlinReco Job

There are several options to run the reconstruction software. One can either use "cheater" processors to construct tracks and clusters or perform realistic tracking and clustering. In the first case tracks and clusters are considered to comprise all tracker hits or calorimeter hits attributable to the same Monte Carlo particle. In the latter case, realistic reconstruction is performed by using LEP tracking code and dedicated calorimeter clustering algorithm developed by the DESY group. Additionally, it is possible to use a combination of "cheater-" and realistic tracking- or calorimeter-clustering processors.

At this point the information from the previous subsection should be sufficient for you to edit the file `exampleSteeringFile.xml` to get a new steering file. The already modified steering file can be found at:

```
$LCSoft/packages/MarlinReco/examples/LDC/steer_ldc.xml
```

This than can be used to run MarlinReco performing realistic tracking and clustering to produce meaningful results.

In the `global` section you have to change the name of the LCIO input file. For a first try you can use the file

```
$LCSoft/packages/MarlinReco/examples/LDC/muons.slcio
```

with single muon events. Place a symbolic link to this file in the Marlin Directory:

```
cd $LCSoft/Marlin
ln -s $LCSoft/packages/MarlinReco/examples/LDC/muons.slcio
```

Add the following line to the `global` section to read the events from the file `muons.slcio`

```
<parameter name="LCIOInputFiles"> muons.slcio </parameter>
```

replacing the original line with the `LCIOInputFiles` parameter. To read the whole file set `MaxRecordNumber` to 0. In addition the reconstruction needs to know the detector geometry. For the "Large Detector Concept" (LDC) this information is stored in the file

```
$LCSoft/packages/MarlinReco/examples/LDC/gear_ldc.xml
```

Place a symbolic link to this file in the Marlin directory:

```
cd $LCSoft/Marlin
ln -s $LCSoft/packages/MarlinReco/examples/LDC/gear_ldc.xml
```

The file is made available to MarlinReco also in the `global` section by:

```
<parameter name="GearXMLFile"> gear_ldc.xml </parameter>
```

Change the `execute` section to call the appropriate processors in the right order:

```
<execute>
   <processor name="MyTPCDigiProcessor"/>
   <processor name="MyVTXDigiProcessor"/>
   <processor name="MyFTDDigiProcessor"/>
   <processor name="MySimpleCaloDigi"/>
   <processor name="MyLEPTrackingProcessor"/>
   <processor name="MyTrackwiseClustering"/>
   <processor name="MyWolf"/>
   <processor name="MyLCIOOutputProcessor"/>
</execute>
```

At this stage non of the steering parameter specific to any processor has to be changed. Now the Marlin executable can be invoked. The name of the steering file to be used is handed over as the first parameter:

```
cd $LCSoft/Marlin
bin/Marlin exampleSteeringFile.xml
```

Be sure that you are in the right directory, otherwise the relative paths to the LCIO file containing the events and the Gear XML file do not work.

After the program has finished, there is an additional file `outputfile.slcio` in the current directory, holding the reconstructed events. The contents of this file can be examined using the program

```
$LCIO/bin/dumpevent file run event
```

Here the Parameter `file` is the name of the LCIO file from which the events are read, `run` and `event` the run and event number of the event to be dumped. The `dumpevent` program prints the contents of all objects to the standard output device.

To use "cheater" processors to construct tracks and clusters, you have to modify the `execute` section of the steering file. The processors `MyLEPTrackingProcessor` and `MyTrackwiseClustering` are replaced by `MyTrackCheater` and `MyClusterCheater`:

```
<execute>
  <!--processor name="MyAIDAProcessor"/-->
  <processor name="MyVTXDigiProcessor"/>
  <processor name="MyFTDDigiProcessor"/>
  <processor name="MySimpleCaloDigi"/>
  <processor name="MyTPCDigiProcessor"/>

  <processor name="MyTrackCheater"/>
  <!--processor name="MyLEPTrackingProcessor"/-->

  <processor name="MyClusterCheater"/>
  <!--processor name="MyTrackwiseClustering"/-->

  <processor name="MyWolf"/>
  <processor name="MyLCIOOutputProcessor"/>
</execute>
```

### 3.4 C Event Display to Visualise Events

Even though the `dumpevent` program from the LCIO package prints out very detailed information about all objects of an event this is not satisfactory. Usually a more qualitative and visual approach is required, especially in the early stages of algorithm development. Here the C Event Display (CED) can be employed to get a graphical view of an event. Inside a wire-frame sketch of the detector[2], simulated hits (*SimCaloHit*, *SimTrackerHit*), reconstructed hits (*CaloHit*, *TrackerHit*), Monte Carlo objects (*TrueTrack*, *TrueCluster*) and reconstructed objects (*Track*, *Cluster*, *ReconstructedParticle*) are shown.

The CED is set up as a client server application consisting of two parts:

[2] Available models: LDC (Mokka program [9]), SiD (SLIC program [8]) and GLD (Jupiter program [7])

- the program `glced` which displays the event on the screen and
- the Marlin processor *GenericViewer*, which selects the collections to be displayed.

This structure allows to run a Marlin job with event display in three different ways: to show the events of a existing LCIO file, to display the events after the reconstruction or to visualise the different reconstruction steps done by separate processors.

To use CED in a MarlinReco job, edit the steering file `exampleSteeringFile.xml` which you created following the instructions of the last section. Add the line

```
<processor name="MyGenericViewer"/>
```

as the second last processor (before the `MyLCIOOutputProcessor`) in the `execute` section. Including cheater processors and the real reconstruction algorithms in the steering file allows a comparison between both methods. Your `execute` section then should look like this:

```
 <execute>
  <processor name="MyVTXDigiProcessor"/>
  <processor name="MyFTDDigiProcessor"/>
  <processor name="MySimpleCaloDigi"/>
  <processor name="MyTPCDigiProcessor"/>
  <processor name="MyTrackCheater"/>
  <processor name="MyLEPTrackingProcessor"/>
  <processor name="MyClusterCheater"/>
  <processor name="MyTrackwiseClustering"/>
  <processor name="MyWolf"/>
  <processor name="MyGenericViewer"/>
  <processor name="MyLCIOOutputProcessor"/>
 </execute>
```

In the section where the parameters are handed over to `MyGenericViewer` (processor of type *GenericViewer*) one has to specify the names of the collections to display. To view the example from the LCIO file (`muons.slcio`) from above, the names of two collection have to be modified. Do this by replacing the corresponding lines with:

```
<parameter name="SimCaloHitCollections" type="StringVec"> ecal02_EcalBarrel ecal02_EcalEndcap hcalFeScintillator_HcalBarrelEnd hcalFeScintillator_HcalBarrelReg hcalFeScintillator_HcalEndCaps </parameter>
<parameter name="SimTrackerHitCollections" type="StringVec">vxd00_VXD sit00_SIT tpc04_TPC </parameter>
```

To enable to toggle (on/off) displaying the contents from the specified collections, a one digit number `0, 1, ... 9` can be assigned to each collection. This can be done by replacing the corresponding lines of parameters with

```
<parameter name="LayerSimTrackerHit" type="int">1 </parameter>
<parameter name="LayerSimCaloHit" type="int">2 </parameter>
<parameter name="LayerTrackerHit" type="int">3 </parameter>
<parameter name="LayerCaloHit" type="int">4 </parameter>
<parameter name="LayerTracks" type="int">5 </parameter>
<parameter name="LayerClusters" type="int">6 </parameter>
<parameter name="LayerReco" type="int">7 </parameter>
<parameter name="LayerTrueTracks" type="int">8 </parameter>
<parameter name="LayerTrueClusters" type="int">9 </parameter>
```

in the XML steering file.

To enable the display you have to launch the CED server manually by starting the `glced` program in a separate window/shell:

```
$LCSoft/glced/glced &
```

If you now run MarlinReco by typing

```
cd $LCSoft/Marlin
bin/Marlin exampleSteeringFile.xml
```

the event is displayed after finishing the reconstruction of tracks, clusters and particle flow objects in the window panel activated by the `glced` program.

To manipulate the event display you can either use the mouse or certain key commands. Controlling will work if the mouse is placed inside the CED window.

- **Using the mouse**
  Hold a mouse button pressed while moving the mouse to manipulate the display. The following functions are available:

| | |
|---|---|
| left mouse button | rotate |
| middle mouse button | translate (pan) |
| right mouse button | zoom |

- **Using keyboard commands**

| | |
|---|---|
| `ESC` | stop and exit `glced` |
| `f` | front view. Projection of all points on the $xy$-plane. |
| `s` | side view. Projection of all points on the $yz$-plane. |
| `c` | centre view. A new origin for a rotation and zoom is set to the current position of the mouse cursor. |
| `r` | redisplay in some initial projection. |
| `0, 1, ...` | toggle layers. The assignment of the layers is done in the steering file |

After an event is displayed, the system waits until the user hits a key to continue. Please note, that since the event display is realised as a Marlin processor, the reconstruction is also stopped as long as the computer waits for the user entry to continue.

## 3.5 Writing your own Marlin Processor

Unless you want to add an processor to an existing package you should add a new package if you develop new processors to carry out reconstruction or analysis tasks. All additional packages are expected to live in the `$MARLIN/packages` directory. To decouple Marlin form the additional processors, the subdirectories holding these processors are linked using symbolic links to Marlin's `packages` directory. The actual source code is placed in an individual subdirectory - for example `MyPackage` - in `$LCSoft/packages` and have the structure:

```
$LCSoft/packages/MyPackage/include
                          ./src/
                          ./src/GNUmakefile
                          ./lib
```

The `GNUmakefile` has a target 'lib', that creates a library

```
$LCSoft/packages/MyPackage/lib/libMyPackage.a
```

To set up a new package you can execute the following commands:

```
cd $LCSoft/packages
mkdir MyPackage
cd $LCSoft/Marlin/packages
ln -s $LCSoft/packages/MyPackage
```

```
cd $LCSoft/packages/MyPackage
mkdir include
mkdir src
mkdir lib
cd src
cp $LCSoft/Marlin/examples/mymarlin/GNUmakefile .
```

Edit the `GNUmakefile` to assign the name of your package (in this case `MyPackage`) to the variable `PROGNAME`.

Lets assume we want to write a processor with the name *MyProcessor*. Each Marlin processor inherits from the class *Processor* and has to have at least the following methods implemented:

- `virtual Processor* newProcessor()`
  Returns a pointer to a object of type *MyProcessor*, needed for internal processor management.
- `MyProcessor()`
  Constructor of the class.
- `virtual void init()`
  Called at the begin of the job before anything is read. Use to initialise the processor, e.g. book histograms.
- `virtual void processRunHeader( LCRunHeader* run )`
  Called for every run.
- `virtual void processEvent( LCEvent * evt )`
  Called for every event - the working horse.
- `virtual void check( LCEvent * evt )`
  Called for each event after processing to produce checks or plots.
- `virtual void end()`
  Called after data processing for clean up.

Figures 2 and 3 show a very simple example for a processor which counts the number of runs and events in a LCIO file. In addition it prints for each event the number of Monte Carlo particles. The collection name of the Monte Carlo particles can be given to the processor via a steering parameter. The default setting is `MCParticle`. The two include statements in the header file

```
#include "marlin/Processor.h"
#include "lcio.h"
```

```
#ifndef MyProcessor_h
#define MyProcessor_h 1

#include "marlin/Processor.h"
#include "lcio.h"
#include <string>

using namespace lcio ;
using namespace marlin ;

class MyProcessor : public Processor {

 public:

  virtual Processor*  newProcessor() { return new MyProcessor ; }
  MyProcessor() ;
  virtual void init() ;
  virtual void processRunHeader( LCRunHeader* run ) ;
  virtual void processEvent( LCEvent * evt ) ;
  virtual void check( LCEvent * evt ) ;
  virtual void end() ;

 protected:

  std::string _colName ;
  int _nRun ;
  int _nEvt ;
} ;

#endif
```

Figure 2: *Header file of defining the class* MyProcessor*.*

```
#include "MyProcessor.h"
#include <iostream>
#include <EVENT/LCCollection.h>
#include <EVENT/MCParticle.h>

using namespace lcio ;
using namespace marlin ;

MyProcessor aMyProcessor ;

MyProcessor::MyProcessor() : Processor("MyProcessor") {
  _description = "MyProcessor does whatever it does ..." ;
  registerProcessorParameter( "CollectionName" ,
                              "Name of the MCParticle collection"  ,
                              _colName ,
                              std::string("MCParticle") ) ;
}
void MyProcessor::init() {
  printParameters() ;
  _nRun = 0 ;
  _nEvt = 0 ;
}
void MyProcessor::processRunHeader( LCRunHeader* run) {
  _nRun++ ;
}
void MyProcessor::processEvent( LCEvent * evt ) {
  _nEvt ++ ;
  LCCollection* inParVec = evt->getCollection(_colname) ;
  std::cout << "Event no. " << _nEvt << " has "
            << inParVec->getNumberOfElements()
            << " Monte Carlo particles" << std::endl;
}
void MyProcessor::check( LCEvent * evt ) {
  // nothing to check here - could be used to fill checkplots
  // in reconstruction processor
}
void MyProcessor::end(){
   std::cout << "MyProcessor::end()  " << name()
            << " processed " << _nEvt << " events in " << _nRun << " runs "
            << std::endl ;
}
```

Figure 3: *Implementation of the class* MyProcessor.

are mandatory because the class inherits from the Marlin class *Processor*. Furthermore Marlin uses LCIO also as the internal data model and thus every processor needs to know about it. Besides the methods already described above, there are three variables declared which are used to count the number of runs and events in the LCIO file (`_nRun`, `_nEvt`) and hold the collection name of Monte Carlo particles (`_colName`).

Of course the implementation file has to include the corresponding header file. All additional include directives include header files which are required by the user. The line

```
MyProcessor aMyProcessor ;
```

registers the processor `MyProcessor` at the Marlin framework. In the initialisation part of the constructor of `MyProcessor` the constructor of `Processor` is called with the name of the processor. A short description of the processor is stored in the string `_description`. The function call to `registerProcessorParameter()` allows to register a processor parameter which can be set in the steering file. Together with the name of the processor parameter a short description, a default value and a variable that holds the value of the parameter is specified.

In the function `init()` the processor parameters and their values as used in the program are printed to the standard output and some parameters are initialised. While `processRunHeader()` is only used to count the number of runs, `processEvent()` is used to print the number of Monte Carlo particles for each event and count the number of events found in the LCIO file. Since no checks are performed `check()` is empty. At the end of data processing `end()` prints the total number of runs and events processed.

# 4 Installation Guide

This section is supposed to help you to install all packages which are necessary to run MarlinReco. Probably some of the required software is already available on your computer. The installation guide is written in a way that you can skip the appropriate subsections.

## 4.1 System Requirements

LCIO, Gear, Marlin, MarlinUtil and MarlinReco have been developed under (SuSE) Linux and tested with gcc3.3 on Linux. As the "-ansi" switch is used during compilation it should be fairly easy to port the packages to other platforms with an ANSI compliant C++ compiler. The build scripts used also require that gmake/make is installed. In Addition GSL [12], CLHEP [14] and CERNLIB [13] are used (see also Section 2.6).

In order to build LCIO you need to have a Java VM (version 1.4 or higher) installed on your platform. This is true even if you only want to install the C++ version as the API-files are generated from an abstract description for Java and C++ using the AID [11] tool.

If histograms should be produced, the JAIDA [16] and AIDA_JNI [17] packages have to be installed on the system.

The header files are commented using the Doxygen [22] syntax. if you wish to generate a browsable API documentation Doxygen has to be available.

## 4.2 Getting the Sources

If you wish to check out the sources of Gear, Marlin, MarlinReco and MarlinUtil directly from the CVS repositories you need ccvssh [21] which works as a bridge to remote CVS pservers using SSL tunnelling. **Please check the site** http://www-zeuthen.desy.de/linear_collider **for important information on the version of ccvssh that is required**. You can also donwload precompiled binaries of ccvssh from there and access the web interface to the cvs repository that allows to download tar-balls of all the software packages. In the following we describe how to download the software through cvs.

Please make sure, that you download the recent tagged versions of the different packages (LCIO v01-05, Gear v00-02, Marlin v00-09-02, MarlinReco v00-01, MarlinUtil v00-01, CED v00-01 and CEDViewer v00-01) - both with cvs and when using the webinterface to the repository.

Create a directory in which you can download all sources and install the packages:

```
mkdir ilcsoft
export LCSoft=$PWD/ilcsoft
cd $LCSoft
```

### LCIO

If LCIO is not yet installed at your site you can get a recent copy of the sources from the CVS repository via anonymous checkout. Set the CVSROOT variable, e. g. on a Linux platform running bash:

```
export CVSROOT=:pserver:anonymous@cvs.freehep.org:/cvs/lcio
```

Then checkout a released version, e. g.:

```
mkdir $LCSoft/lcio
cd $LCSoft/lcio
cvs co -d v01-05 -r v01-05 lcio
```

### Marlin

Also for Marlin you can get a recent copy of the sources from the CVS repository via anonymous checkout. Set the `CVSROOT` and `CVS_RSH` variables, e. g. on a Linux platform:

```
export CVS_RSH=ccvssh
export CVSROOT=:ext:anonymous@cvssrv.ifh.de:/marlin
```

Start the authentification:

```
ccvssh login
```

Since “anonymous” does not have a password, just hit return when you are prompted to enter your password. Then checkout the latest version:

```
cd $LCSoft
cvs co -r v00-09-02 Marlin
```

### Gear

You can get the recent sources from the CVS repository via anonymous checkout. After setting the required environment variables checkout:

```
unset CVS_RSH
unset CVSROOT
export CVS_RSH=ccvssh
export CVSROOT=:ext:anonymous@cvssrv.ifh.de:/gear
ccvssh login
cd $LCSoft
cvs co -r v00-02 gear
```

### MarlinUtil, MarlinReco and the CEDViewer

The recent version of the source code of both packages can be obtained from the CVS repository via anonymous checkout. Set the environment variables and start the authentification:

```
unset CVS_RSH
unset CVSROOT
export CVS_RSH=ccvssh
export CVSROOT=:ext:anonymous@cvssrv.ifh.de:/marlinreco
ccvssh login
```

Then checkout the latest version:

```
mkdir $LCSoft/packages
cd $LCSoft/packages
cvs co -r v00-01 MarlinReco
cvs co -r v00-01 MarlinUtil
cvs co -r v00-01 CEDViewer
cvs co -r v00-01 CED
```

## 4.3 Building the Packages

### LCIO

A few variables have to be set depending on your development environment (modify as appropriate):

```
export LCIO=$LCSoft/lcio/v01-05
export PATH=$LCIO/tools:$LCIO/bin:$PATH
export JDK_HOME=/usr/lib/j2sdk
```

To build the Java and C++ versions you have to be in the LCIO subdirectory:

```
cd $LCSoft/lcio/v01-05
```

Use the following command to build the Java and C++ versions:

```
ant aid cpp f77
```

### Gear

Enter the Gear subdirectory:

```
cd $LCSoft/gear
```

The environment variables needed for compilation and linking are in the file `env.sh`. Open this file and adopt the line

```
export JDK_HOME=/opt/products/java/1.4.2
```

to point to the Java Development Kit of your system. Make the settings available to your shell by sourcing the script:

```
. ./env.sh
```

To build Gear as well as the API documentation issue the following commands:

```
ant aid.generate
ant cpp
ant cpp.doc
```

### MarlinReco

The top level makefile of the Marlin package is designed in a way that it builds Marlin together with MarlinReco, MarlinUtil and the CEDViewer. This works only if MarlinReco, MarlinUtil and CEDViewer are moved to the `packages` subdirectory of Marlin. To decouple Marlin form the other packages their subdirectories are linked using symbolic links to Marlin's `packages` directory:

```
cd $LCSoft/Marlin/packages
ln -s $LCSoft/packages/CEDViewer
ln -s $LCSoft/packages/MarlinReco
ln -s $LCSoft/packages/MarlinUtil
```

A few variables have to be set depending on your development environment. For user convenience there already exists a script `env.sh` in the MarlinReco subdirectory, where you can change the paths as needed. The meaning of the variables together with the default values for the DESY computing environment are given here.

- `export LCIO=/opt/products/lcio/1.5`
  Path to the LCIO package directory.
- `export MARLINDEBUG=1`
  Include debug information in output (1 = on). Comment out this line for production code.
- `export CLHEP=/opt/products/CLHEP/1.8.2`
  Path to the CLHEP package. Be sure to use a version below 2.0. There will be problems with later versions due to the useage of namespaces.
- `export GSL_HOME=/afs/desy.de/group/it/ilcsoft/gsl/1.6`
  Path to the "GNU Scientific Library" (GSL).
- `export CERNLIB_HOME=/opt/products/cernlib/pro/lib`
  Path to the "CERN programming library" (CERNLIB).
- `export MARLIN_USE_AIDA=1`
  This option enables Marlin to produce histograms using the the AIDA [15] interface.
- `export JDK_HOME=/opt/products/java/1.4.2`
  Path to the Java Development Kit.
- `export JAIDA_HOME=/opt/products/JAIDA/3.2.3`
  Java implementation of the AIDA interface.
- `export AIDAJNI_HOME=/opt/products/AIDAJNI/3.2.3`
  Java Native Interface to access JAIDA from C++ .

Edit the script as needed and copy it to the Marlin directory to replace the `env.sh` script shipped with Marlin:

```
cd $LCSoft/Marlin
cp $LCSoft/Marlin/packages/MarlinReco/env.sh env.sh
```

Source it to setup the environment:

```
. env.sh
```

In addition the "env.sh" script of MarlinReco produces the file `userlibMarlinReco.gmk` containing some additional paths to include and library directories which are necessary to build MarlinReco. Edit this file, if you need to add more include and library paths for own processors. To make the settings available to the Marlin top level makefile replace the dummy file `userlib.gmk`:

```
mv userlib.gmk.MarlinReco userlib.gmk
```

To build MarlinReco type in the Marlin directory:

```
gmake
```

This will create the MarlinReco executable `$LCSoft/Marlin/binMarlin`. Simultaneous to MarlinReco also the MarlinUtil library and the CEDviewer is build.

**Glced**

To get graphical output of CED on the screen the server program `glced` has to be build:

```
cd $LCSoft/packages/CED
gmake
```

You need X to run `glced` and the environment variable `DISPLAY` has to point to a accessible display. Before starting MarlinReco with event display type:

```
$LCSoft/packages/CED/glced
```

More information can be obtained from the README file located in the CED directory.

# 5 Further Information

We like to point both, users and developers to our software portal at http://ilcsoft.desy.de. There one can find more information about the packages mentioned in this tutorial, in particular the API documentations.

# References

[1] see: http://ilcsoft.desy.de/gear

[2] see: http://ilcsoft.desy.de/marlin

[3] see: http://lcio.desy.de

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

[5] T. Behnke *et al.*, (ed.), "Tesla TDR, Pt. 4", *DESY-01-011*

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

[7] T. Aso, K. Hoshina, K. Fujii and A. Miyamoto, *SPIRES entry Prepared for LCWS 2002, Jeju Island, Korea, 26-30 Aug 2002*

[8] J. McCormick, *SLAC-PUB-11418 Contributed to LCWS 2005, Stanford, California, 18-22 Mar 2005*

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

[10] T. Behnke *et al.*, LC-TOOL-2001-005, see: http://www-flc.desy.de/lcnotes

[11] AID Homepage:
http://java.freehep.org/aid/index.html

[12] GSL homepage:
http://www.gnu.org/software/gsl/.

[13] CERNLIB homepage:
http://cern.ch/cernlib/

[14] CLHEP homepage:
http://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/

[15] AIDA homepage:
http://aida.freehep.org

[16] JAIDA homepage:
http://java.freehep.org/jaida

[17] AIDA-JNI homepage:
http://java.freehep.org/aidajni

[18] JAS3 homepage:
http://jas.freehep.org/jas3

[19] Homepage of the PI project:
http://cern.ch/pi/

[20] Open Scientist homepage:
http://www.lal.in2p3.fr/OpenScientist/

[21] CCVSSH homepage:
http://ccvssh.sourceforge.net

[22] Doxygen homepage:
http://www.doxygen.org

[23] international linear collider (ILC) homepage:
http://www.linearcollider.org