LCIO
A persistency framework for LC detector simulation studies
Frank Gaede, DESY, IT
Linear Collider Simulation Workshop
SLAC May 19-22 2003
People

- Ties Behnke - DESY/SLAC
- Frank Gaede - DESY
- Norman Graf - SLAC
- Tony Johnson - SLAC
- Paulo Mora de Freitas - IN2P3
Outline

- Introduction
- Software design
- API
- Data model
- Status
- Summary
Motivation

LCIO Persistency Framework

Generator

Java, C++, Fortran
Geant3, Geant4

Simulation

Reconstruction

Analysis

Java, C++, Fortran

geometry
The Persistency Framework

LCIO

data model

data access

data format

contents

API

implementation

persistency
LCIO requirements

- need Java, C++ and f77 (!) implementation
- extendable data model for current and future simulation studies
- user code separated from concrete data format
  - -> need to be flexible for future decisions on persistency
- three general use cases
  - writing data (simulation)
  - reading and updating data (reconstruction)
  - read only access to data (analysis)
- needed a.s.a.p.  -> keep it simple !
API design – simulation data

Interface for
a) writing data (simulation)
b) read only access (analysis)

data entities
user extensions
tagging interface
untyped collections

Interface for
a) writing data (simulation)
b) read only access (analysis)
API & implementation

abstract event

abstract io

concrete classes

persistency implementation
Extending the base API

- Minimal interface needed for writing data
- Decorator classes add convenient methods to data objects
API definition for Java and C++

- use AID Abstract Interface Definition
  - tool from freehep.org (M. Dönzelsmann)
  - used successfully in the AIDA project
- define interfaces in Java-like language with C++ extensions
  - generates files with Java interfaces
  - generates C++ header files with pure abstract base classes
  - use javadoc for documentation
- independent implementations in Java and C++
  - keep Java “pure” i.e. machine independent
API documentation created from java implementation with javadoc
SIO::SIOObjectHandler Class Reference

Interface for all lcio object SIO-handlers, has to be implemented for all event entities (hits, tracks, clusters, ...).

#include <SIOObjectHandler.h>

Inheritance diagram for SIO::SIOObjectHandler:

SIO::SIOObjectHandler

SIO::SIO::LocalHitHandler     SIO::SIO::FloatVecHandler     SIO::SIO::ParticleHandler     SIO::SIO::TrackHitHandler

List of all members:

Public Member Functions

virtual unsigned int read (SIO_stream *stream, EVENT::LCOBJECT **obj, unsigned int flag, unsigned int vers)=0
Reads lcio objects from an SIO stream.

virtual unsigned int write (SIO_stream *stream, const EVENT::LCOBJECT *obj, unsigned int flag)=0
Writes lcio objects to an SIO stream.

documentation created from C++ implementation with doxygen
Example: reading data

```cpp
LCReader* lcReader =
    LCFactory::getInstance()->createLCReader();
lcReader->open( "myFile" );
LCEvent *myEvt;
while( ( myEvt = lcReader->readNextEvent() ) != 0 ){
    cout << " Evt : " << myEvt->getEventNumber()
         << " - " << myEvt->getRunNumber()
         << " : " << myEvt->getDetectorName()
         << endl;
}
cout << endl;
lcReader->close();
```

independent of persistency implementation!
Example: reading event collections

... LCEvent *evt;
while( ( evt = lcReader->readNextEvent() ) != 0 ){  
    const LCCollection* col = evt->getCollection( "EcalHits" );
    int nHits = col->getNumberOfElements();
    for( int i=0 ; i< nHits ; i++ ){  
        const CalorimeterHit* hit =
            dynamic_cast<const CalorimeterHit*>( col->getElementAt( i ) );
        const float* x = hit->getPosition();
    }

}...
LCReader API

- "direct access" via fast skip
- read next event/run
- callbacks for modules

Method Summary

- close(): Closes the output file/stream etc.
- open(String filename): Opens a file for reading (read-only).
- readEvent(int runNumber, int evNumber): Reads the specified event from file.
- readNextEvent(): Reads the next event from the file.
- readNextEvent(int accessMode): Same as above allowing to set the access mode (LCIO.READ_ONLY is default).
- readNextRunHeader(): Reads the next run header from the file.
- readStream(): Reads the input stream and notifies registered listeners according to the object type found in the stream.
- registerLCEventListener(LCEventListener lis): Registers a listener for reading LCEvents from a stream.
- registerLCRunListener(LCRunListener lis): Registers a listener for reading LCEvents/LCRunHeaders from a stream.
- removeLCEventListener(LCEventListener lis): Removes a listener for reading LCEvents from a stream.
- removeLCRunListener(LCRunListener lis): Removes a listener for reading LCRunHeaders from a stream.
Example: writing data (events)

```cpp
LCWriter* lcWrt = LCFactory::getInstance()->createLCWriter();
lcWriter->open("myFile");
for( int i=0; i<NEVENT; i++ ){
    LCEventImpl* evt = new LCEventImpl();
    evt->setRunNumber( rn );
    evt->setEventNumber( i );
    // add collections ...
    lcWrt->writeEvent( evt );
    delete evt; // C++ only :)
}
lcWriter->close();
```

use LCIO implementation classes or own classes for writing!
Example: writing collections

```cpp
LCCollectionVec* trkVec =
    new LCCollectionVec( LCIO::TRACKERHIT ) ;
for(int j=0; j<NHITS; j++) {
    TrackerHitImpl* hit = new TrackerHitImpl ;
    hit->setdEdx( 30e-9 ) ;
    double pos[3] = { 1., 2., 3. } ;
    hit->setPosition( pos ) ;
    trkVec->push_back( hit ) ;
}
evt->addCollection( (LCCollection*) trkVec , "TPCHits" ) ;
... // write event
```
### LCWriter API

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>close()</code></td>
<td>Closes the output file/stream etc and returns LCIO:SUCCESS if no error occurred.</td>
</tr>
<tr>
<td><code>open(String filename)</code></td>
<td>Opens a file for writing and returns LCIO:SUCCESS if no error occurred.</td>
</tr>
<tr>
<td><code>open(String filename, int writeMode)</code></td>
<td>Opens a file for writing and returns LCIO:SUCCESS if no error occurred.</td>
</tr>
<tr>
<td><code>writeEvent(LCEvent evt)</code></td>
<td>Writes the given event to file and returns LCIO:SUCCESS if no error occurred.</td>
</tr>
<tr>
<td><code>writeRunHeader(LCRunHeader hdr)</code></td>
<td>Writes the given run header to file and returns LCIO:SUCCESS if no error occurred.</td>
</tr>
</tbody>
</table>
LCIO Fortran interface

- Fortran support for
  - legacy software (e.g. BRAHMS reconstruction)
  - non OO-analyses code (“old guys”)
- not a third implementation of the library – use C++-wrapper functions and `cfortran.h` instead:
  - one function for every class member function
  - use integers to store pointers!
  - have factory and delete functions
  - -> OO-like code in fortran
LCIO f77 example:

```fortran
---- event loop -----
do 10
   event = 1rdrreadnextevent( reader )
   if ( event.eq.0 ) goto 11

   runnum = levttgetrunnumber( event )
   evtnum = levttgeteventnumber( event )
   detname = levttgetdetectorname( event )
   write(*,*), run = runnum
   write(*,*), evtn = evtnum
   write(*,*), det = detname

10 continue
11 continue
---- end event loop -----
```

```cpp
// ---- event loop -----
const LCEvent* event;
while ( (event = lcRdr->readNextEvent()) != 0 ){

   int runNum = event->getRunNumber() ;
   int evtnum = event->getEventNumber() ;
   string detName = event->getDetectorName() ;

   std::cout << " run: " << runNum << std::endl ;
   std::cout << " evtn: " << evtnum << std::endl ;
   std::cout << " det: " << detName << std::endl ;
}

//---- end event loop -----
```
Persistency Implementation

- use SIO: Simple Input Output
- developed at SLAC for NLC simulation
- already used in hep.lcd framework
- features:
  - on the fly data compression
  - some OO capabilities, e.g. pointers
  - C++ and Java implementation available
The Data Model - Overview

- RunHeader
- Event
- ReconstructedObject
- SimHeader
- MCParticle
- ReconstructedParticle
- RecoHeader
- TrackerHit
- Track
- Reco
- CalorimeterHit
- Cluster

For details see transperencies after summary
Status of LCIO

- first Java and C++ implementation (simulation data!)
  - integrated into Mokka simulation framework
    - latest release mokka-01-05 writes Hits in LCIO
- f77 prototype
  - demonstrating the design
- complete integration into simulation software chains in the next months:
  - US: hep.lcd (Java) ? -> to be discussed at this workshop
  - Europe: Mokka (C++)/BRAHMS-reco(f77)
Summary

- LCIO is a persistency framework for linear collider simulation software
- Java, C++ and f77 user interface
- LCIO is currently implemented in simulation frameworks:
  - hep.lcd
  - Mokka/BRAHMS-reco
- -> other groups are invited to join
- see LCIO homepage for more details:

Data model - LCRunHeader

- block: RunHeader
  - int: runNumber
  - string: detectorName
  - string: description
  - string[]: activeSubdetectors

=> describes the run setup
Data model - LCEventHeader

- EventHeader
  - int: runNumber
  - int: evtNumber
  - string: detectorName
  - String[]: subdetectorName
  - blockNames:
    - string: blockName
    - string: blockType
Data model – LCEvent (sim)

- **MCParticle**
  - pntr: parent
  - pntr: secondparent
  - pntr[]: daughters
  - int : pdgid:
  - int : hepevtStatus
    - (0,1,2,3 HepEvt)
    - (201, 202 sim. decay)

- **MCParticle cont.**
  - double[3]: start
    - (production vertex)
    - (at vertex)
  - float: energy
  - float: charge
Data model - LCEvent (sim)

- TrackerHit
- string: subdetector
  - int: hitFlags (detector specific: Id, key, etc.)
  - double[3]: position
  - float: dEdx
  - float: time
  - pntr: MCParticle
Data model - LCEvent (sim)

- **CalorimeterHit**
- **string: subdetector**
  - int: cellId0
  - int: cellId1
  - float: energy
  - float[3]: position – optional (file size!)
- **particle contributions:**
  - pntr: MCParticle
  - float: energyContribution
  - float: time
  - int: PDG (of secondary) - optional
Data model - LCEvent (reco)

- OutputHeader
  - int: isrFlag
  - float: colliderEnergy
  - int: flag0 (to be defined)
  - int: flag1 (to be defined)
  - int: reconstructionProgramTag
  - float: Bfield

--> could be combined with global header…
Data model - LCEvent (reco)

- **Track**
  - int: tracktype (full reconstr, TPC only, Muon only, etc.)
  - float: momentum
  - float: theta
  - float: phi
  - float: charge
  - float: d0  (Impact Parameter in r-phi)
  - float: z0  (Impact Parameter in r-z)
  - float[15]: cov.matrix
  - float: reference point (x, y, z)
  - float: chi**2 of fit
  - float[10]: dEdx (weights and probabilities)
  - TrackerHits:  - optional
    - pntr: TrackerHit
**Data model - LCEvent (reco)**

- **Cluster**
  - int: detector (type of cluster: ECAL, HCAL, combined…)
  - int: clustertype (neutral, charged, undefined cluster)
  - float: energy
  - float[3]: position (center of cluster x, y, z)
  - float[6]: errpos (cov. matrix of position)
  - float: theta (intrinsic direction: theta at position)
  - float: phi (intrinsic direction: phi at position)
  - float[3]: errdir (cov. matrix of direction)
  - float[6]: shapeParameters (definition needed)
  - float[3]: weights (compatible with em., had., muon)
  - CalorimeterHits: - optional
    - pntr: CalorimeterHit
    - float: contribution
Data model - LCEvent (reco)

- **ReconstructedParticle**
  - int: primaryFlag (0: secondary, 1: primary)
  - int: ObjectType (charged/neutral particle)
  - float[3]: 3-Vec (px, py, pz)
  - float: energy
  - float[10]: covariance matrix
  - float: charge
  - float[3]: reference position for 4-vector
  - float[5]: PID_type (hypotheses for e, g, pi, K, p, ...)

- **ReconstructedParticle cont.**
  - Tracks:
    - pntr: Track
    - float: weight
  - Clusters:
    - pntr: Cluster
    - float: weight
  - MCParticles:
    - pntr: MCParticle
    - float: weight

Have separate MC-link object?
Data model - LCEvent (reco)

- **ReconstructedObject**
  - int: ObjectType (jet, vertex, ...)
  - float[5]: 4vec (4-vector of object (px, py, pz, E, M)
  - float[3]: reference (position)
  - float[15]: covariance matrix
  - reconstructedParticle:
    - pntr: ReconstructedParticle
    - float: weight

=> generic reconstructed objects, linked to reconstructed particles