LCIO
Persistency and Data Model for LC Simulation and Reconstruction

LCWS 2004, Paris
Simulation, April 20th 2004
Frank Gaede  DESY -IT-
Outline

• Introduction/Motivation
• Implementation
• Data model
• Status
• Examples
• Summary
Introduction

• at 3rd ECFA workshop in Prague decided to have Data format/persistency task force:
  “Define an abstract object persistency layer and a data model for linear collider simulation studies until the Amsterdam workshop.”

• -> **LCIO** – Linear Collider Input/Output
  • DESY/SLAC/LLR joined project
  • design of data model and software first introduced at the 4th ECFA workshop in Amsterdam
  • since Montpellier production version 1.0 (simulation only)
  • now v1.1beta released (incl. reconstruction)
Motivation for LCIO

Generator

Simulation

Reconstruction

Analysis

LCIO Persistency Framework

Java, C++, Fortran

Geant3, Geant4

geometry
Requirements

- need Java, C++ and f77 (!) implementation
- extendable data model for current and future simulation studies
- user code separated from concrete data format
  - want to be flexible for future decisions on persistency
- needed a.s.a.p.
  - keep it simple (lightweight)
- no dependence on other frameworks
LCIO persistency framework

LCIO

- data model
- data handling
- data format

contents
API
implementation
persistency
Data Format (persistency): SIO

- 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 😊
  - no direct access 😞
    -> use fast skip 😐
LCIO SW-Architecture

*.slcio files (SIO)

- common API
  - generated from one source using AID

- C++ API
  - LCIO C++ implementation

- Java API
  - LCIO Java implementation

- f77 API
  - JAS/AIDA

- root

- hbook

compressed records, pointer retrieval
C++ and f77 example code

```c
// 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 << " evt: \" << evtNum << std::endl;
    std::cout << " det: \" << detName << std::endl;
}

// end event loop
```

```c
// event loop
int runnum = levtgetrunnumber( event)
evtNum = levtgeteventnumber( event)
detname = levtgetdetectorname( event)

write(*,*) " run: ", runnum
write(*,*) " evt: ", evtNum
write(*,*) " det: ", detname
```

plus additional methods in f77 for user convenience, e.g. HEPEvt <-> LCIO conversion
Data Model II

Monte Carlo

**MCParticle**
- Kinematics (4Vector)
- Parents/Daughters
- Generator Status
- PID
- Vertex
- ...
- -> all of HEPEVT
- + Simulator Status
- + Endpoint

**SimCalorimeterHit**
- CellID
- Energy/Amplitude
- Position (opt.)
- MCParticle Contributions

**SimTrackerHit**
- Position
- dEdx
- MCParticle Contribution
Data Model III

RawData and Digitization

CalorimeterHit
- CellID
- Energy/Amplitude
- Position (opt.)

TPCHit
- CellID
- Time
- Charge

TrackerHit
- Position
- Covariance
- dEdx
- raw hit

SimHits

Serve as interface classes to reconstruction

Add more specialized classes as needed -> user request!
Data Model IV

Reconstruction & Analysis

Cluster
- Energy
- Position
- Direction
- Shape
- Hits
- Clusters

Track
- \((p, \theta, \phi, d_0, z_0)\)
- \(dE/dx\)
- Errors
- Chi2
- Hits
- Tracks

ReconstructedParticle
- Kinematics (4Vector)
- Charge, Mass, ...
- Reference point
- errors
- ParticleIDs
- Clusters
- Tracks
- ReconstructedParticles

ParticleID
- TypeID (PDG)
- Probability
- Algorithm
- Parameters

ReconstructedParticles can be simple particles and compound objects like jets, vertices, ...
Data Model V

Run and Event

**LCEvent**
- Event number
- Collections

**LCRun**
- Run number
- description

**LCCollection**
- Type
- Name
- Elements

**LCOBJECT**

**MCParticle**

**SimCalorimeterHit**

**SimTrackerHit**

**TPCHit**

**CalorimeterHit**

**TrackerHit**

**TrackerHit**

**Track**

**Cluster**

**Reconstructed Particle**

**LCIntVec**

**LCFloatVec**

The LCEvent serves as a container of named collections of the various data types in LCIO (LCOBJECT subclasses)

Generic integer and float vectors for user extensions


Frank Gaede, DESY -IT-
LCIO for Transient Data

- The LCEvent can be used as container for transient data in an application, e.g. reconstruction
- Application will call list of modules that read existing collections from the LCEvent and add resulting new Collections
- LCIO has (Event/Run)-Listener classes that can serve as base classes for modules
- LCIO defines a simple application framework
  -> see example lcioframe provided in release
Example LCEvent
LCIO Status I

- production version 1.0 (09/2003)
  - C++, Java, f77 complete for simulation data
    - and generator data (HEPEvt<->LCIO)
  - simple example code for all languages
  - ‘real world’ examples (JAS3, root, hbook)
  - documentation
    - users manual
    - API documentation HTML (javadoc, doxygen)
  - available for download via CVS
    - linux (gcc), windows (cygwin)
- schema evolution from now on (reading old files)
LCIO Status II

- new beta release v1.1beta:
  - includes C++ implementation for reconstruction data (Track, Cluster, ReconstructedParticle)
  - Java and f77 soon to come
  - added optional use of CLHEP four vectors:
    - Handler classes for MCParticle/ReconstructedParticles that can be used as 4vectors and LCIO Objects
  - reading of a list of files (chain)
  - simplified API structure (one level of inheritance less)
  - not for production use yet!
    - -> user feedback welcome
LCIO on the web

- LCIO homepage: http://lcio.desy.de
  - downloads and documentation

- LCIO forum at: http://forum.linearcollider.org
  - user/developer questions and comments
  - discussions on new developments

- LCIO bug reports at: http://bugs.freehep.org
  - bug report and new feature requests
Javadoc example

```
Method Summary

Int close()
   Closes the output file/stream etc.

Int open(String filename)
   Opens a file for reading (read-only)

LCEvent readEvent(int runNumber, int evtNumber)
   Reads the specified event from file.

LCEvent readNextEvent()
   Reads the next event from the file.

LCEvent readNextEvent(int accessMode)
   Same as above allowing to set the access mode (LCIO.READ_ONLY is default)

LCRunHeader readNextRunHeader()
   Reads the next run header from the file.

Int readStream()
   Reads the input stream and notifies registered listeners according to the object type found in the stream.

Void registerLCEventListener(LCEventListener lis)
   Registers a listener for reading LCEvents from a stream.

Void registerLCRunListener(LCRunListener lis)
   Registers a listener for reading LCEventsLCRunHeaders from a stream.

Void removeLCEventListener(LCEventListener lis)
   Remove a listener for reading LCEvents from a stream.

Void removeLCRunListener(LCRunListener lis)
   Remove a listener for reading LCRunHeaders from a stream.
```

Doxygen example

DATA::LCOBJECT Class Reference

The generic object that is held in an LCCollectionData.

#include <LCOBJECT.h>

Inheritance diagram for DATA::LCOBJECT:

```
DATA::LCOBJECT

DATA::CalorimeterHitData  DATA::MCParticleData  DATA::SimCalorimeterHitData  DATA::SimTrackerHitData

EVENT::CalorimeterHit  EVENT::MCParticle  EVENT::SimCalorimeterHit  EVENT::SimTrackerHit

IMPL::CalorimeterHitImpl  IMPL::MCParticleImpl  IMPL::SimCalorimeterHitImpl  IMPL::SimTrackerHitImpl

IOIMPL::CalorimeterHitIOImpl  IOIMPL::MCParticleIOImpl  IOIMPL::SimCalorimeterHitIOImpl  IOIMPL::SimTrackerHitIOImpl
```

List of all members
LCIO Customers/Users

- Mokka simulation (see talk)
- Brahms reconstruction (see talk)
- JAS3
  - provides convenient file browser
  - will have LCIO-WIRED plugin -> generic event display!
- Calorimeter group (DESY)
  - has MiniCal raw data converted to LCIO files
  - to be used also for Hcal physics prototype
- TPC groups (DESY & LBNL?)
  - will use LCIO for prototype
- Lelaps fast Monte Carlo
- hep.lcd reconstruction
- other groups looking into using LCIO
JAS3 – LCIO file browser

LCIO Future developments

- reconstruction data model
  - refine some details
  - Java and f77 implementation soon
- add convenient methods and tools
  - analyzing the MCParticle tree (graph)
    - iterators for MCParticles, e.g. all stable/final particles
    - analyzing parent/daughter relationships
  - adding convenience iterators for LCIO types
    - simplify complex C++ syntax with dynamic_casts etc.
- add possibility to store generic user data
  - calibration constants etc.
  - discussion ongoing about implementation (not so easy)
- respond to user requests
Summary

- LCIO, persistency and datamodel for the LC:
  - Java, C++ and f77 user interface
  - Java and C++ implementation
  - data model for simulation, (prototype) data and reconstruction
    - persistent and transient
- production version v1.0
- new beta version released (v1.1beta)
- used by several groups and tools
  - others invited to join!
- see LCIO homepage for more:
  - http://lcio.desy.de
Appendix

- Extension slides, details, examples
Data entities UML
Example code: reading events

```cpp
LCReader* lcReader = 
    LCFactory::getInstance()->createLCReader();

try{
    lcReader->open( "myFile" ) ;
    LCEvent *myEvt ;
    while( 1){
        try{
            myEvt = lcReader->readNextEvent() ;
        }catch(EndOfDataException) { break ;}
        cout << "Evt : " << myEvt->getEventNumber()
            << " - "  << myEvt->getRunNumber()
            << " : ":" " << myEvt->getDetectorName() << endl ;
    }
    lcReader->close() ;
}catch(IOException& e){ cout << e.what() << endl ; }
```
Example code: reading collections

```cpp
...LCEvent *evt;
while( 1 ){
    try{
        evt = lcReader->readNextEvent();
    } catch(EndOfDataException) { break ;}
    const LCCollection* col;
    try{
        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();
        } catch(DataNotAvailableException){
            cout << "collection not found: EcalHits " << endl ;
        }
    } catch(DataNotAvailableException){
        cout << "collection not found: EcalHits " << endl ;
    }
}...
```
Example code: writing data (events)

```cpp
LCWriter* lcWrt = LCFactory::getInstance()->createLCWriter();
try{
    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();
} catch(IOException& e) { cout << e.what() << endl; }
```
Data model - Design

Prototype Data

user extensions, e.g. calibration constants