



How to use LCIO

Frank Gaede
DESY

ILC Software Workshop
Cambridge, April 4-6, 2006

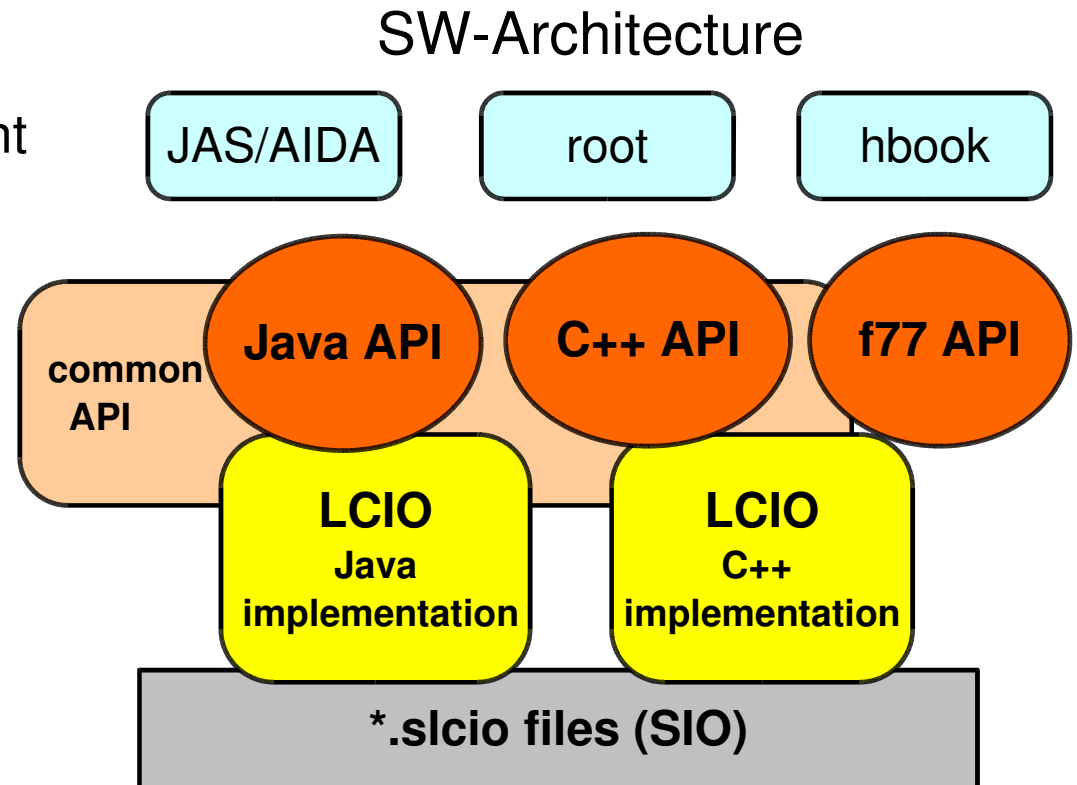
LCIO overview

- DESY and SLAC joined project:
 - provide common basis for ILC software
- Features:
 - Java, C++ and f77 (!) API
 - extensible data model for current and future simulation and testbeam studies
 - user code separated from concrete data format
 - no dependency on other frameworks

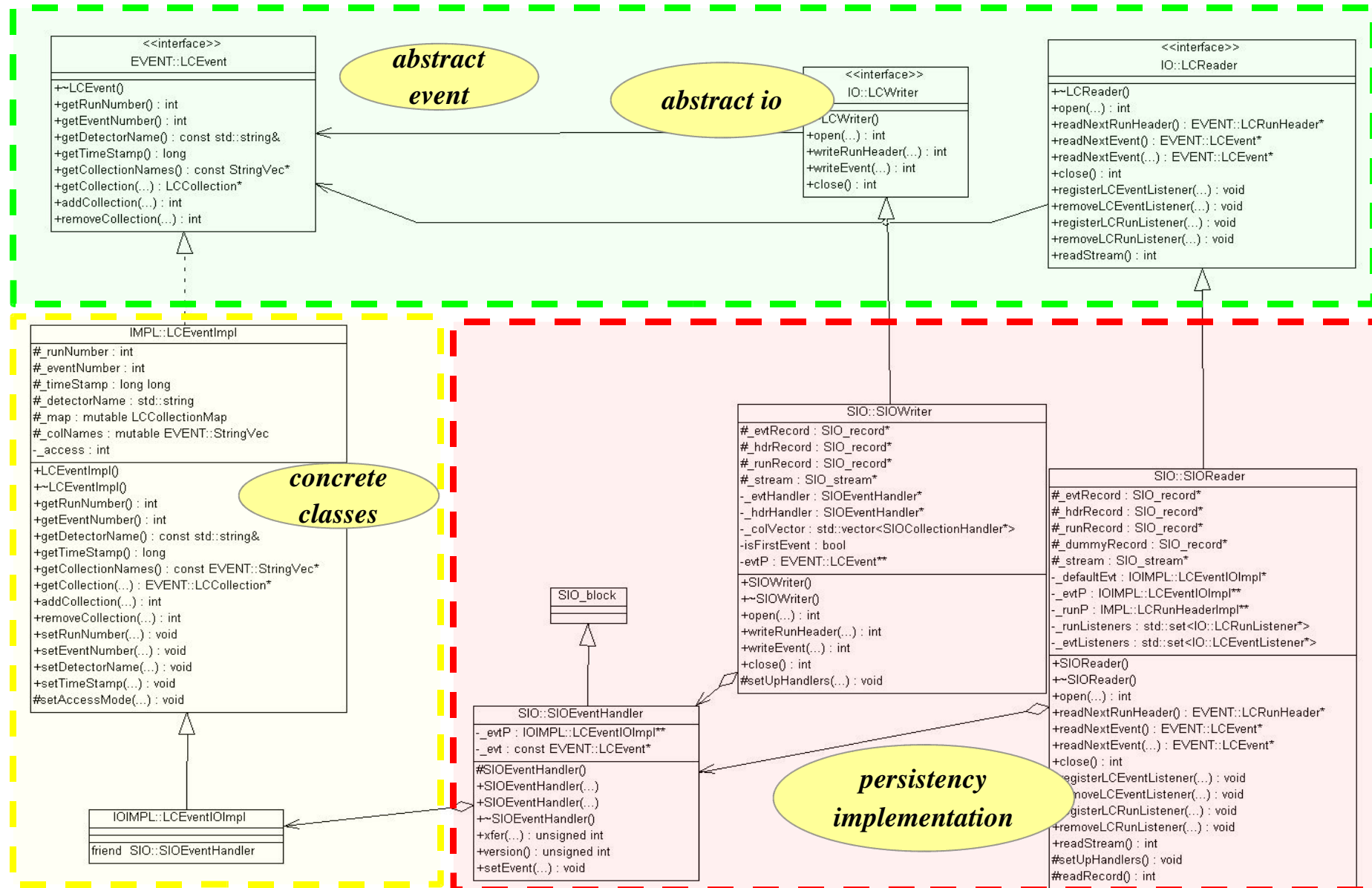
simple & lightweight

new release: **v01-06**

now de facto standard
persistency & datamodel
for ILC software



LCIO class design





Run and Event

LCEvent

- Event number
- **Collections**

LCCollection

- Type
- Name
- **Elements**

LCObject

Reconstructed Particle

MCParticle

Cluster

LCRun

- Run number
- description

SimCalorimeterHit

CalorimeterHit

Track

SimTrackerHit

TrackerHit

TPCHit

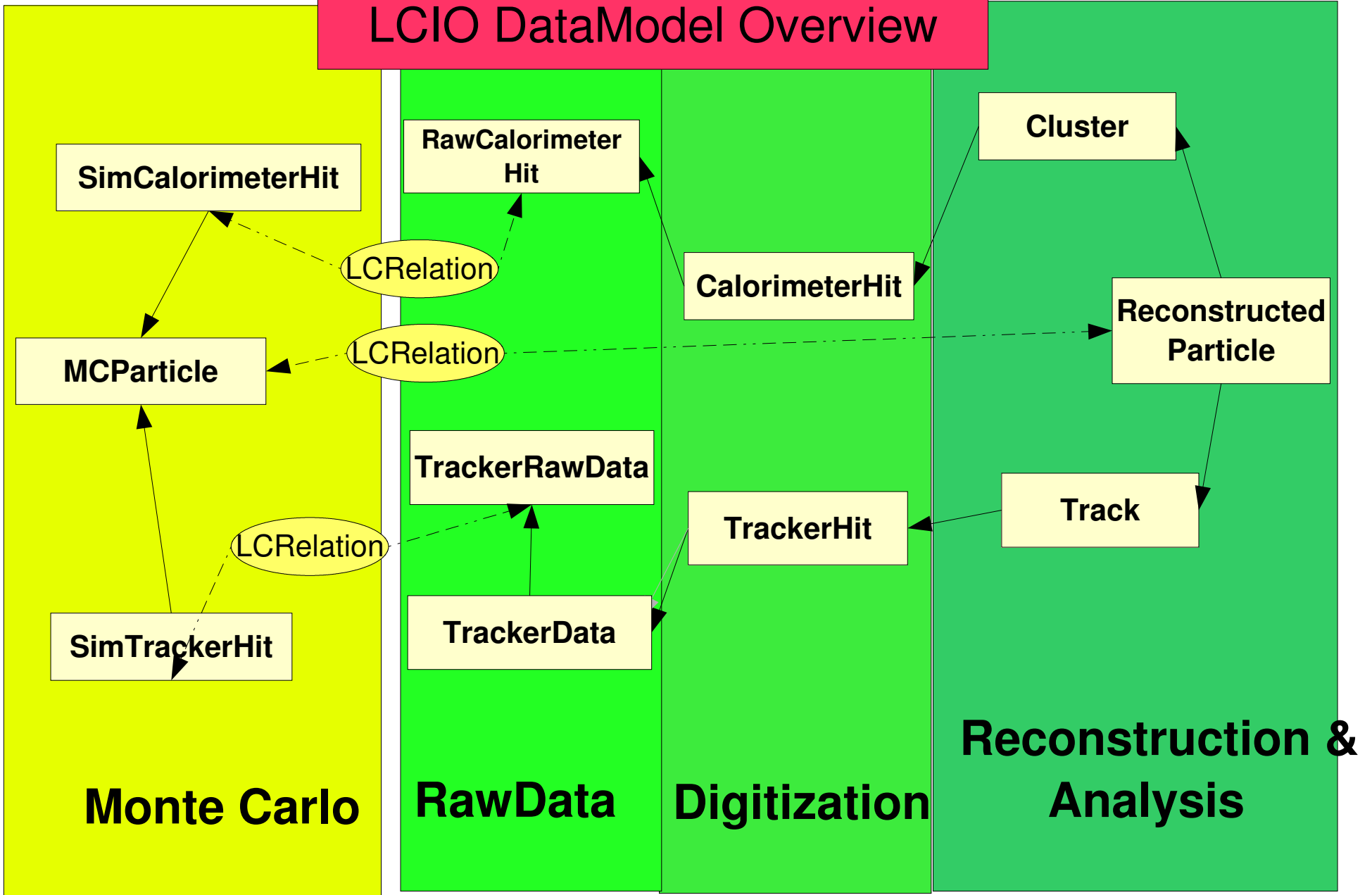
LCIntVec

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**

LCIO DataModel Overview



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**

Datamodel IV

RawData and Digitization

RawCalorimeterHit

- cellID
- amplitude
- time (opt.)

TrackerRawData

- cellID
- time
- adc[] *int*

TrackerData

- cellID
- time
- charge[] *float*

CalorimeterHit

- cellID
- energy
- time (opt.)
- position (opt.)

use abstract hit classes as input to reconstruction

TrackerHit

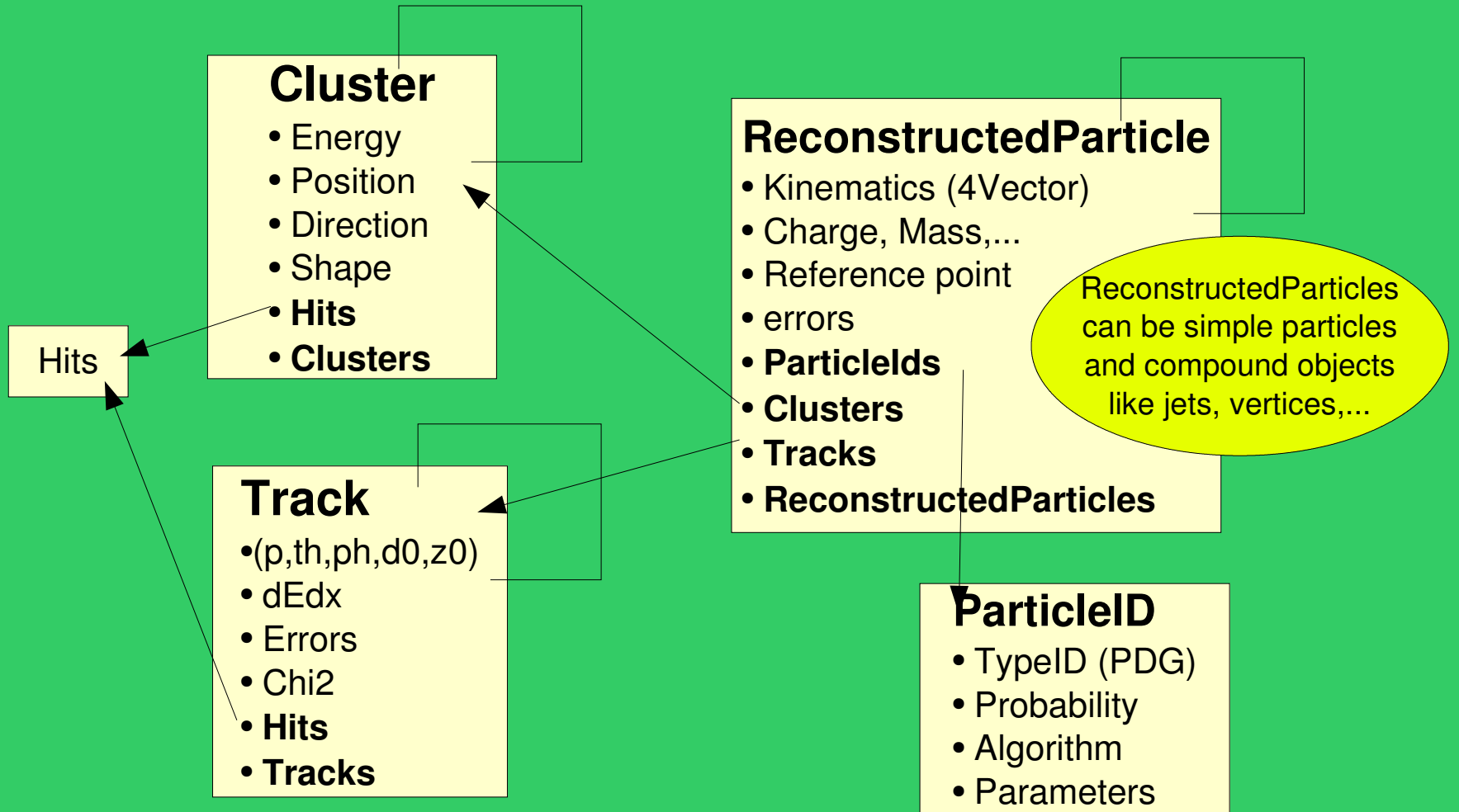
- position (x,y,z)
- covariance
- dEdx
- position (opt.)

TrackerPulse

- cellID
- time
- charge

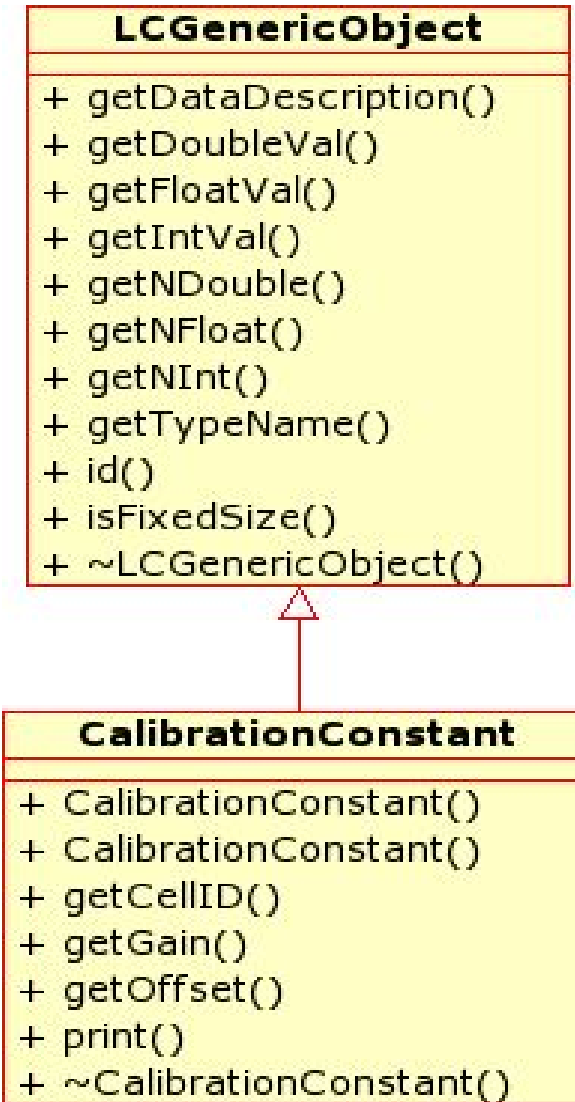
TPCHit replaced by more generic raw tracker data classes

Reconstruction & Analysis



LCIO – Generic User Information

- LCIO data model defines the object needed for ILC simulation studies, but
- users want additional information in files for specific studies
- can't create new classes within LCIO for all requests and purposes
- need generic user class:
LCGenericObject
 - almost arbitrary data objects
 - typically access provided through user subclass - but not needed:
 - has description string for reading the data without need to have access to data dictionary (library)



```

emacs@linux.site
File Edit Options Buffers Tools C Help

#include "lcio.h"
#include "UTIL/LCFixedObject.h"

#define NINT 1
#define NFLOAT 2
#define NDOUBLE 0

using namespace lcio ;

class CalibrationConstant ;

/** Example for a simple calibration class based on the LCFixedObject template.
 * <p>
 * LCFixedObject uses an instance of LCGenericObjectImpl that holds the data, thus
 * there is no overhead when the data is read from a database or file
 * for copying it to some local structure (Decorator pattern).<br>
 *
 */
class CalibrationConstant : public LCFixedObject<NINT,NFLOAT,NDOUBLE> {
public:
    /** Convenient c'tor.
     */
    CalibrationConstant(int cellID, float offset, float gain) {
        obj()->setIntVal( 0 , cellID ) ;
        obj()->setFloatVal( 0 , offset ) ;
        obj()->setFloatVal( 1 , gain ) ;
    }

    /** 'Copy constructor' needed to interpret LCCollection read from file/database.
     */
    CalibrationConstant(LCObject* obj) : LCFixedObject<NINT,NFLOAT,NDOUBLE>(obj) { }

    /** Important for memory handling*/
    virtual ~CalibrationConstant() { /* no op*/ }

    // the class interface:
    int getCellID() { return obj()->getIntVal(0) ; }
    float getOffset() { return obj()->getFloatVal( 0 ) ; }
    float getGain() { return obj()->getFloatVal( 1 ) ; }

    void print( std::ostream& os ) ;

    // ----- need to implement abstract methods from LCGenericObject

    const std::string getTypeName() const {
        return "CalibrationConstant" ;
    }

    const std::string getDataDescription() const {
        return "i:cellID,f:offset,f:gain" ;
    }
};

--- CalibrationConstant.h (C CVS-1.4 Abbrev)--L9--C17-- 4%-----

```

example for user class:
 \$LCIO/src/cpp/src/
 EXAMPLE/CalibrationConstant.h

```

emacs@linux.site
File Edit Options Buffers Tools C++ Help

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

lcReader->open( FILEN ) ;

LCEvt* evt ;
int nEvents = 0 ;

//----- the event loop -----
while( (evt = lcReader->readNextEvent()) != 0 ) {

    // loop over collections and check if we have objects of user defined
    // type "CalibrationConstant"
    // NB: this is only needed if you don't know the name of the collection
    // that holds the calibration constants

    const StringVec* colNames = evt->getCollectionNames() ;

    for(unsigned int i=0 ; i < colNames->size() ; i++ ){

        LCCollection* col = evt->getCollection( (*colNames)[i] ) ;

        if( col->getParameters().getStringVal("TypeName") == "CalibrationConstant" ) {

            // now print the calibration constants
            for(int j=0;j<col->getNumberOfElements();j++){

                CalibrationConstant cal( col->getElementAt( j ) ) ;

                std::cout << " calibration for cellid: " << cal.getCellID()
                    << " offset: " << cal.getOffset()
                    << " gain: " << cal.getGain()
                    << std::endl ;

            }

        }

    }

    nEvents ++ ;
}

--:-- readcalibration.cc (C++ CVS-1.1 Abbrev)--L33--

```

example for user class:
 \$LCIO/src/cpp/src/
 EXAMPLE/readcalibration.cc

LCIO Online documentation

Frank Gaede, ILC Software Workshop, Cambridge, Apr 4-6, 2006

Overview (LCIO API Documentation, Version v01-03) - Mozilla Firefox

LCIO API Version v01-03

All Classes

LCIO - a persistency framework for linear collider simulation studies.

See: Description

Package	Description
hep.lcio.data	The package hep.lcio.data has been removed - interfaces are now all defined in hep.lcio.event.
hep.lcio.event	Primary user interface for LCIO.
hep.lcio.example	Simple usage examples.
hep.lcio.exceptions	Exceptions thrown by LCIO.
hep.lcio.implementation.event	
hep.lcio.implementation.io	Default IO implementation.
hep.lcio.implementation.sio	SIO specific LCIO implementation.
hep.lcio.io	Interfaces for IO library.
hep.lcio.test	
hep.lcio.util	Utilities for use with LCIO.

Documentation for LCIO v01-03

- [Users manual](#) (also available as [pdf](#) and [ps](#)) Read before you get st
- [Java API documentation](#)
- [C++ API documentation](#)
- (printable version of the C++ API reference: [lciorefman.ps](#))
- XML data format description: [lcio.xml](#)

Last modified: Thu Sep 23 14:51:51 2004

LCIO - Users manual v01-03 - Mozilla Firefox

Building the library

A few variables have to be set depending on your development environment, e.g.

- Linux (and bash):

```
export LCIO=/lcio/v01-03          <-- modify as appropriate
export PATH=$LCIO/tools:$LCIO/bin:$PATH

export JDK_HOME=/usr/lib/j2sdk    <-- modify as appropriate
```
- Windows/Cygwin - DOS shell:

```
set PATH=c:/cygwin/bin;%PATH%    <-- modify as appropriate

set LCIO=c:/lcio/develop/v01-03  <-- modify as appropriate
set JDK_HOME=c:/j2sdk1.4.1_01    <-- modify as appropriate

set PATH=%LCIO%/tools;%LCIO%/bin;%PATH%
```

IMPL::ReconstructedParticleImpl Class Reference

Implementation of ReconstructedParticle. [More...](#)

#include <ReconstructedParticleImpl.h>

Inheritance diagram for IMPL::ReconstructedParticleImpl:

```
graph TD
    EVENT_LCObject[EVENT::LCObject] --> EVENT_ReconstructedParticle[EVENT::ReconstructedParticle]
    EVENT_ReconstructedParticle --> IMPL_ReconstructedParticleImpl[IMPL::ReconstructedParticleImpl]
    IMPL_ReconstructedParticleImpl --> IOIMPL_ReconstructedParticleIOImpl[IOIMPL::ReconstructedParticleIOImpl]
    IMPL_ReconstructedParticleImpl --> IMPL_AccessChecked[IMPL::AccessChecked]
```

Public Member Functions

- [ReconstructedParticleImpl\(\)](#)
Default constructor, initializes values to 0.
- [virtual ~ReconstructedParticleImpl\(\)](#)

LORE-1 PS - Overview

7.7 IMPL::ClusterImpl Class Reference

Implementation of Cluster

Public Member Functions

- [ClusterImpl\(\)](#)
Default constructor, initializes values to 1.
- [virtual ~ClusterImpl\(\)](#)
Destructor.
- [virtual void set\(\)](#)
Sets the state of the instance. (default) use in LCIO.
- [virtual void getType\(\) const](#)
Returns the type of cluster.
- [virtual float getEnergy\(\) const](#)
Energy of the cluster.
- [virtual float getNTracks\(\) const](#)
Number of tracks in the cluster.
- [virtual void setPhi\(float phi\) const](#)
Overwrite number of particles (if Parameter).
- [virtual float getPhi\(\) const](#)
Returns number of particles. (Phi).
- [virtual float getPhi\(\) const](#)
Returns number of particles. (Phi).
- [virtual void setPhi\(float phi\) const](#)
Overwrite number of particles. (Phi).
- [virtual void setPhi\(float phi\) const](#)
Overwrite number of particles. (Phi).

LCIO XML

```
<!-- end of named parameters -->
<data type="int" name="nTrack"/>
<repeat count="nTrack">
  <data type="int" name="type">Type of Track, e.g. TPC, VTX, SIT</data>
  <data type="float" name="d0">Impact Parameter in r-phi</data>
  <data type="float" name="phi">Phi of track in r-phi</data>
  <data type="float" name="omega">curvature signed with charge</data>
  <data type="float" name="z0">Impact Parameter in r-z</data>
  <data type="float" name="tanLambd">tangent of dip angle in r-z</data>
  <data type="float[15]" name="covMatrix"> Covariance matrix</data>
  <data type="float[3]" name="referencePoint">Reference point (x,y,z) </data>
  <data type="float" name="chi2">chi**2 of fit</data>
  <data type="int" name="ndf">ndf of fit</data>
  <data type="int" name="dEdx">dEdx</data>
  <data type="float" name="dEdxError">Error of dEdx</data>
  <data type="float" name="radiusOfInnermostHit">radius of innermost hit used in track</data>
  <data type="int" name="nHitNumbers">
    <repeat count="nHitNumbers">
      <data type="int" name="subdetectorHitNumbers">
        number of hits in particular sub-detectors. TODO need way to define mapping in ru
      </data>
    </repeat>
  </data>
  <data type="int" name="nTracks">
    <repeat count="nTracks">
      <data type="int" name="Track">tracks that have been combined to this track</data>
    </repeat>
  </if condition="(flag&&(1<<31))!=0">
```

LCIO Future Plans

- current data model fairly complete
 - check usability of data model, e.g. track parameters
 - ongoing through development of reconstruction code !
 - need to define conventions on how to use LCIO
 - collection names, object types, collections to be present in event
 - see talk by
- need to make LCIO more convenient (and efficient)
 - decoding of MCParticle information (parent/daughter, ...)
 - inverse relations (get all tracks for one MCParticle)
 - attach user information to LCObjects
- would like to make C++ version more C++ like, e.g. allow to use STL algorithms (templates in general)
 - planed for LCIO v02-00
- **need user input !!**

LCIO discussion I

- LCIO meeting yesterday:
 - TrackerHit: now 3d point xyz
 - proposal
 - have various types: cylindrical, xy-plane, zplane, 1d, 2d ,...
 - depending on type of tracker used
 - still use existing class TrackerHit
 - option:
 - define new classes for these specific tracker types

Note: this breaks our previous LCIO philosophy with one (simple) class TrackerHit !

keep it simple - as simple as possible but no simpler...

LCIO discussion II

- how to store multiple Track fits in LCIO
- **option 1 (current version):**
 - store one collection for every type of fit, e.g.:
 - “Tracks” - default, fitted at IP (holds pointers to hits)
 - “TracksAtInnermostHit” (points to “Tracks” tracks)
 - “TracksAtOutermostHit” (points to “Tracks” tracks)
 - “TracksAtCaloFace” (points to “Tracks” tracks)
 - pro: already implemented / easy to use collection with particular fit (calo)
 - con: maybe not so intuitive ?
- **option 2**
 - modify Track class so that it holds multiple fits
 - pro: more intuitive (?)
 - con: needs coding / not so easy to access a particular fit

LCIO discussion III

- what is your experience with LCIO ?
 - files sizes ?
 - speed reading files ?
 - direct access ?
- what feature is missing in LCIO that your need to do your work ?
- anything else you would like to discuss ?

Backup slides ...

LCIO status [v01-04]

- new features:
 - support and definition of 64bit time stamps
 - **ns since 1/1/1970 (UTC)**
 - multiple I/O streams in C++
 - LCGenericObjects in Java (user defined data objects)
 - subset collections
 - hold pointers/references to objects already existent in the event, e.g. LeptonCandidates from ReconstructedParticles
 - transient and persistent
 - if persistent, only pointers/references are stored in the file
 - files are downward compatible, LCIO 1.3 can read new files
- bug fixes
- improved documentation

LCIO status [v01-05]

- changed return values of E, P, m to double for MCParticle and ReconstructedParticle
 - stored are still floats
 - requires some trivial changes (float->double) in code where indicated by the compiler !
- new template UTIL::LCTypedVector<T> for creating std::vector<LCObjectType> from LCCollections
 - allows to use iterators and STL algorithms, e.g. std::for_each()
- added first implementations of generic tracker raw data classes (TPC, VTX, SiliconStrip,...)
 - TrackerRawData, TrackerData, TrackerPulse
- files are downward compatible with LCIO 1.4
- bug fixes
- see \$LCIO/doc/versions.readme for more

LCIO status [v01-07(06)]

- added optional **momentum[3]** and **pathLength** to SimTrackerHit (silicon digitization)
- LCReader::skipNEvents()
 - dump the n-th event in \$LCIO/bin/dumpevent
- **C++ utility code for encoding/decoding of cellids in hit classes (32bit and 64bit)**
- support for large files (>2GB) and UTIL::LCSplitWriter
- optional **python binding** (J.McCormick)
 - files are downward compatible with LCIO 1.6
 - see \$LCIO/doc/versions.readme for more