MarlinTPC Tutorial
Introduction to ILCSoft

Martin Killenberg

University of Bonn

12. February 2009

Overview

- LCIO Data Persistency Framework
- GEAR Geometry Package
- LCCD Conditions Data Toolkit
- (R)AIDA Histogramming Package
- Marlin Analysis Framework

LCIO

Linear **C**ollider **I**nput/**O**utput

- Event Data Model
- Provides Data Classes
- Implementations for JAVA and C++

Data Class	**Description**
LCEvent	Contains collections of one event (bunch crossing)
LCCollection	Collection of data classes of a certain type e. g. TrackerRawData
MCParticle	Particle from the MC generator
SimTrackerHit	Charge deposition in the detector
TrackerRawData	ADC values from the TPC
TrackerData	Calibrated raw data
TrackerPulse	Charge and time of a pulse in one electronics channel
TrackerHit	3D hit with charge information
Track	Helix parametrisation if the fitted track
LCGenericObject	User defined data class

Interfaces and Implementations

- For every object there is an abstract, common interface for Java and C++, and an implementation class with the suffix “Impl” (e. g. TrackerHit and TrackerHitImpl).
- The interface classes live in the namespace “EVENT”, the implementations in “IMPL”. The namespace “lcio” includes all namespaces used in lcio.
- When retrieving data from a collection or a file you always get a pointer to the abstract class. With this you can access the data (get() functions), but you cannot modify them.
- When creating and modifying the data objects you have to use the implementations. Here you also have access to the set() functions.

LCCollection and LCEvent

LCCollection

Object that can store all data classes derived from LCObject

std::string &	getTypeName()	Returns the type of the objects stored.
int	getNumberOfElements()	Number of objects in the collection.
LCObject *	getElementAt(int index)	Get an element from the collection.

LCEvent

Stores LCCollections with an individual name, so you can have several collections of the same data type
(e. g. “VertexDetectorHits” and “TPCHits” both being of type TrackerHit).

int	getEventNumber()	Get the event number.
LCCollection *	getCollection (const string &name)	Get a collection with given name.

I only show the most important data members to give an idea of the functionality. Refer to the Doxygen documentation for a complete list.

Data Classes 1

TrackerRawData

Raw Data (ADC values) per electronics channel. In zero suppressed mode there can be more than one object per channel.

int	getCellID0 ()	Get the geometric cell ID 0 (PadIndex)
int	getCellID1 ()	Get the geometric cell ID 1 (ModuleID)
int	getTime()	Get the time (time bin of the first ADC sample)
ShortVec &	getADCValues ()	Vector with 16 bit ADC values (samples in one channel or pulse)

TrackerData

Similar to TrackerRawData, but uses floats instead of ints for the data values. Used for calibrated data e. g. after pedestal subtraction.

int	getCellID0()	Get the geometric cell ID 0 0(ChannelID)
int	getCellID1()	Get the geometric cell ID 1 0(ModuleID)
float	getTime()	Get the time (time bin of the first ADC sample)
FloatVec &	getChargeValues ()	Vector with calibrated ADC values (in MarlinTPC still in ADC counts)

Data Classes 2

TrackerPulse

Charge and time on a pad.

int	getCellID0()	Get the geometric cell ID 0 0(PadIndex)
int	getCellID1()	Get the geometric cell ID 1 0(ModuleID)
float	getTime()	Get the time (time in ns)
float	getCharge()	Get the charge (still in primary electrons)
TrackerData *	getTrackerData()	Back link to the TrackerData this pulse has been calculated from.

TrackerHit

3D hit with space coordinates and charge.

double *	getPosition()	The hit position (in mm)
float	getdEdx()	Returns the **charge** of the hit (in GeV) Note: This is not dE/dx! Name is due to historical reasons.
LCObjectVec &	getRawHits()	Back link to the pulses.

Data Classes 4

Track

Helix parametrisation of a track (see LC note LC-DET-2006-004.pdf)

float	getD0()	Impact paramter of the track in (r-phi).
float	getPhi()	Phi of the track at the reference point.
float	getOmega()	Signed curvature of the track in [1/mm].
float	getZ0()	Impact paramter of the track in (r-z).
float	getTanLambda()	Lambda is the dip angle in s-z.
float	getdEdx()	dEdx of the track.
TrackerHitVec &	getTrackerHits ()	The hits on this track.

Data Classes 5

LCGenericObject

Data class with arbitrary number of integer, float and double values. Used for user-defined classes, mainly conditions data.

int	getNInt()	Number of integer values stored in this object.
int	getNFloat()	Number of float values stored in this object.
int	getNDouble()	Number of double values stored in this object.
int	getIntVal(int index)	Returns the integer value for the given index.
float	getFloatVal(int index)	Returns the float value for the given index.
double	getDoubleVal(int index)	Returns the double value for the given index.
string	getTypeName()	The type name of the user class (typically the class name).
string	getDataDescription()	Description what the data is (Voltages, Pressure etc.).

LCFixedObject

Derived from LCGenericObject, but with fixed number of ints, floats and doubles. This improves the performance.

Memory Management in LCIO

- Objects are always created on the heap, i. e. dynamically using “new”.
- Objects are only accessed by pointers, only pointers to objects are stored.
- Never use copy constructors or assignment operators!
- LCEvents and LCCollections own the objects they point to. You don’t have to (and must not) delete objects you added to an event or collection.
- get() functions do not hand over the ownership. Do not delete the objects you got from a container class.

Example: Reading data

There is an “LCEvent *event” object in memory which has been read from a file. You want to get number of hits in the event and the dEdx of the first track

```
// get the collection with hits
LCCollection *hitCollection = event->getCollection("TPCHits");
int nHits = hitCollection->getNumberOfElements();

cout << "Event "<< event->getEventNumber()
     << " has " nHits "<< hits." << endl;


// get the collection with tracks
LCCollection *trackCollection = event->getCollection("TPCTracks");

// get the first track
if ( trackCollection->getNumberOfElements() > 0 )
{
   // we have to dynamic_cast beause the collection returns a pointer
   // of the LCObject base class
   Track * t = dynamic_cast<Track *> trackCollection->getElementAt( 0 );

   // If the cast succeeded the pointer is not 0
   if ( t != 0 )
     cout << "Track curvature is " << t->getdEdx() << endl;

}
```

GEAR

GEometry **A**PI for **R**econstruction

Geometry description toolkit for the whole detector. For the TPC there is an abstract interface class for row based pad layouts: **PadRowLayout2D** (see gear Doxygen documentation for details)

There are three implementations in v00-11-01:

- **RectangularPadRowLayout**: Flexible rectangular layout, all pads in a row having the same size, but changing of the size and staggering from row to row is possible.
- **FixedPadSizeDiskLayout**: All pads in all rows have the same size. Pad plane is made of complete circles. Used for LDC studies.
- **FixedPadAngleDiskLayout**: Wedge shaped module with pads all having the same angle (like the LP micromegas module).
  Version in v00-11-01 is still buggy! Use the CVS head if you want to use it.

The exact geometry (size and number of the pads etc.) is defined in an XML steering file.

All software in MarlinTPC is programmed against the abstract API and should run with all geometries!

Example XML file

Small TPC prototype with a rectangular pad plane.

```
<gear>
  <detectors>

     <detector name="TPCProtoTest" geartype="TPCParameters">

        <maxDriftLength value="260." />
        <!-- Set vDrift to 0. Take it from conditions data! -->
        <driftVelocity value="0." />
        <readoutFrequency value="10000000" />

        <!-- simple uniform row layout -->
        <!-- This is an example for a prototype TPC:
             64 pads in a row, 24 rows.
             The copper is 0.8x3.8 with 0.2 mm gap.
             64 pads on 64 mm => pad pitch = 1 mm
        <PadRowLayout2D  type="RectangularPadRowLayout"
                         xMin="-32." xMax="32." yMin="-48.">
          <row repeat="24" nPad="64" padHeight="3.8" padWidth="0.8" rowHeight="4." />
        </PadRowLayout2D>

     </detector>

  </detectors>
</gear>
```

LCCD

Linear **C**ollider **C**onditions **D**ata toolkit.
Allows to to access slow control data either from a **data base** or from an **lcio file**. The use is transparent for the user code, i. e. you don't have to care where the data comes from. Just ask LCCD for it and it ensures the data is available.

The data has to be provided as lcio::LCGenericData objects. MarlinTPC brings already brings a lot of classes in the tpcconddata directory (see next slide).

In this tutorial we will only use data from lcio files.

tpcconddata

ADCChannelMapping
Mapping of H/W channels to GEAR pad indices
- ChannelID
- PadID
- Type

ChannelCorrection
Per channel calibration
- Quality flags (broken, noisy)
- Calibration factors
- Time offset

Pedestal
per channel
- Value
- Width

TPCConditions
Calibrated TPC Parameters
- DriftVelocity
- Diffusion (trans/long)
- "Defocussing"
- Amplification

GasConditions
- Mixture
- Pressure
- Temperature
- OxygenContent
- WaterContent

FieldSettings
- Nominal drift field
- Nominal B-Field

especially for GEMs:
- GEM voltages
- Transfer fields

WeatherConditions
- float Temperature
- float Humidity
- float Pressure

TimePixPixelMode
- Mode
- Status (broken/noisy)

AIDA

Abstract **I**nterfaces for **D**ata **A**nalysis
Programming language independent histogramming package.

There are two implementations which can be used with Marlin:

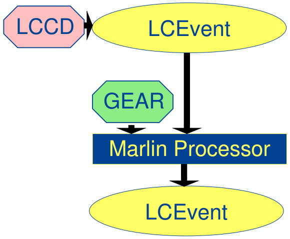
- RAIDA
  ROOT based C++ implementation, comes with ILCSoft
- JAIDA / AIDAJNI
  Java implementation, more mature than RAIDA

Marlin

Modular **A**nalysis and **R**econstruction for the **LIN**ear collider

Marlin is a C++ reconstruction framework for LCIO data.

- Controls the data flow
- Each computing task is performed by a "processor"
- Interface between the processors: LCEvent
- Programme flow is controlled with XML steering files
- Provides an interface to GEAR and LCCD



Marlin Processors

Processor parameter control the work flow of the individual processors

- Processor parameters are set in the steering file.
- Each parameter has a default value, it is not mandatory to set it.
- There are “optional parameter”, for instance for cuts.
    - If the parameter is set, the cut is executed with the given value.
    - If the parameter is not set the cut is not executed at all, no default value is applied.

Required parameters in every processor:

- Name of the input collection(s)
- Name of the output collection(s)

“Marlin -x” lists all parameters that are available for the processor. They should also be documented in the Doxygen docu.

Marlin Build-in Processors

- **AIDAProcessor:**
  Opens the output file for the histograms.
- **ConditionsProcessor:**
  Reads in the required collections from LCCD.
- **LCIOOutputProcessor:**
  Opens the outfile. Here you can define which collections are stored in the outfile.

Marlin Steering File

- In the steering file instances of the processors are defined. You can call the same processor several times with different parameters.
- **Global section:**
  - Input file
  - GEAR file
  - Verbosity level
- **Execute section:**
  List of processor instances to be executed in this order.

MarlinTPC Overview

More than 50 processors in different sections:

- Simulation
- Digitisation
- Reconstruction
- Calibration
- Analysis

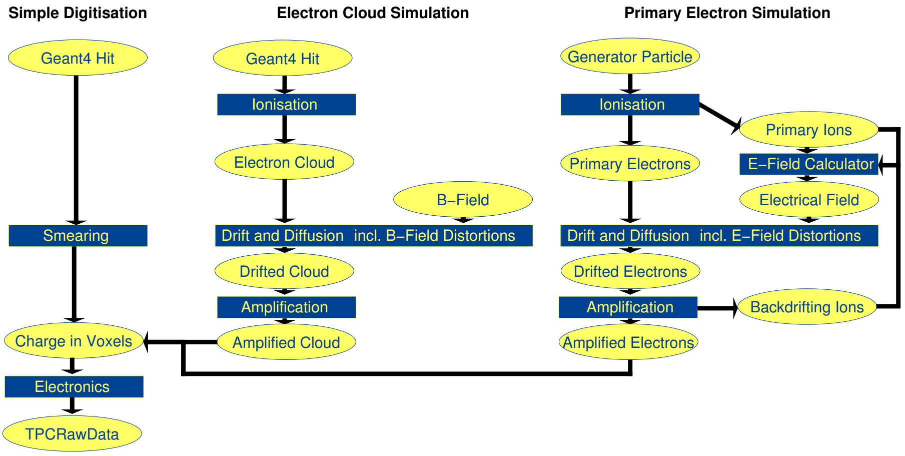
- Validation
- Tools
- Examples

- LCIO data classes for conditions data

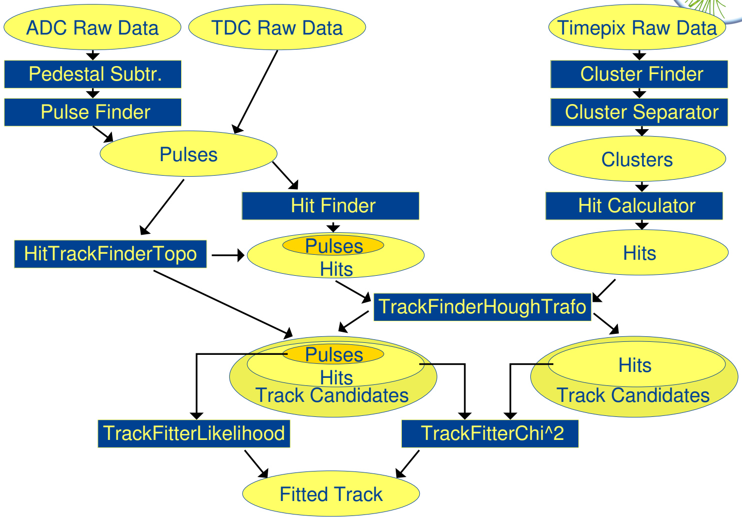
Note: MarlinTPC is not related to and does not need MarlinReco.
MarlinReco is full detector reconstrution used for simulation studies and uses old LEP tracking code for the TPC.

MarlinTPC brings its own, highly modular standalone TPC reconstruction.

MarlinTPC Simulation and Digitisation



MarlinTPC Reconstruction



MarlinTPC Basic Reconstruction

Data Class	Processor Name	Collection Name
	ConditionsProcessor	
`Pedestal`		TPCPedestal
`ADCChannelMapping`		TPCChannelMapping
`TrackerRawData`		TPCRawData
	TrackerRawDataToDataConverter	
`TrackerData`		TPCData
	PulseFinder	
	ChannelMapper[1]	
`TrackerPulse`		TPCPulses
	HitTrackFinderTopoProcessor	
`TrackerHit`		TPCHits
`Track`		TPCTrackCandidates
	TrackSeeder	
`Track`		TPCSeedTracks

[1] This processor only need an input collection name, it is able to modify an existing collection.

MarlinTPC Analysis

Look at the overview page of the Doxygen documentation:
MarlinTPC_trunk/Analysis/doc/html/index.html

Tasks for This Tutorial

- Run the ReconstructStraightTracks.xml example!
  - Compare the processors, and the input and output collection names with the overview on page 24.
  - Play with the verbosity level.
- Modify the steering file to reconstruct a curler!
  - Change the name of the input file and the gear file accordingly.
  - The pedestal and channel mapping data are in a separate file.
- Reconstruct Muons simulated with the MarlinTPC simulation/digitisation!
  - You do not need the channel mapper for these data.
- Plot the Ω distribution of the reconstructed muons using the TrackParametersDistributionProcessor!
  - To use the AIDAProcessor with RAIDA change the FileType to “root”.

You can run “Marlin -c steeringfile.xml” to check if all collections are there to run successfully.