HPS Data Summary Tapes Tutorial

Omar Moreno

Santa Cruz Institute for Particle Physics
University of California, Santa Cruz
omoreno1@ucsc.edu

June 3, 2013

Goals

- Install the LCIO C++ API
- Install the HPS Event API
- Understand the basics required to write an analysis that uses an LCIO file
- Understand what is required to write an analysis that uses the DSTs

I will not cover any deep details instead leaving them for the afternoon session. I also wont be covering how to use analysis tools i.e. ROOT.

Preliminaries

- Need a Linux or OSX (assume it works) machine $\Rightarrow$ Haven't tried using Windows/Cygwin
- Installation of the LCIO C++ API and the HPS Event API will require the following packages and their dependencies:
  - ROOT
    - Make sure the ROOT environment is setup correctly
  - CMake
  - git
  - gcc
- This tutorial assumes that all packages have been installed
- It is also assumed that a user has a work directory where they will install everything $\Rightarrow$ "workdir/"

Installing the LCIO C++ API

- Writing analyses which read LCIO files directly requires installation of the LCIO C++ API
- LCIO is built as follows

```
cd work/dir
svn co svn://svn.freehep.org/lcio/trunk lcio/trunk
cd lcio/trunk
mkdir build; cd build
make -D BUILD_ROOTDICT=1 -D ROOT_DIR=$ROOTSYS ..
make install; cd ../
```

- Once LCIO has been installed the LCIO environment is set up as follows

```
export LCIO=path/to/lcio/trunk/
export LD_LIBRARY_PATH=$LCIO/lib:$ROOTSYS/lib
```

- More details can be found on Confluence: Loading LCIO Files into ROOT

Installing the HPS Event API

- The DST writer along with the HPS Event API is available through github and can be obtained by issuing the following command from a terminal

```
cd workdir/
git clone https://github.com/omar-moreno/hps_dst/
```

This will clone (copy) the following directories to your local machine

- `examples` ⇒ contains example analyses
- `include` ⇒ HPS API header files
- `src` ⇒ HPS API implementations
- `utils` ⇒ contains a range of utility classes and DST writer
- `sandbox` ⇒ Junk!

- The binaries, the ROOT dictionaries along with the shared library (`lib/HpsEvent.so`) can then be built by issuing the following command

```
cd hps_dst/; make
```

Getting Started Analysing Data

- Analyses can be written to make use of either LCIO files or DSTs
  - Reading LCIO files directly will give you access to a bit more lower level information
  - ROOT can be used in conjunction with both
- Data can be obtained from the following sources
  - Reconstructed LCIO files can be found here: `http://www.slac.stanford.edu/ omoreno/dst/`
  - DSTs can be found here: `http://www.slac.stanford.edu/ omoreno/dst/recon`
- An analyses directory e.g. `workdir/analysis` can be setup as follows
  - Copy the `Makefile` provided in the directory `hps_dst/examples` to `workdir/analysis` $\Rightarrow$ This will be used to build the analyses
  - Copy the file `min_lcio_analysis.cxx` to the analyses folder and run the following commands

```
export HPS_DST_HOME=path/to/hps_dst; make
./bin/min_lcio_analysis -i <input LCIO file>
```

- If everything is setup correctly, a message "Analysis Complete!" will appear

Example Analysis Using an LCIO File

- The file `min_lcio_analysis.cxx` also illustrates some of the basic routines required to do an analysis using an LCIO file
- Reading of an LCIO file and looping through all events is done using the class `LCReader`

```
// Create the LCIO reader and open the LCIO file specified by the user.
// If the file doesn't exist or can't be opened, notify the user and exit
IO::LCReader *lc_reader = IOIMPL::LCFactory::getInstance()->createLCReader();
try{
    lc_reader->open(lcio_file_name.c_str());
} catch(IO::IOException exception){
    cout << "File " << lcio_file_name << " cannot be opened!" << endl;
    return(2);
}


// Loop over all events in the file until the end of file is reached.
EVENT::LCEvent *event = 0;
IMPL::LCCollectionVec* tracks = 0;
while( (event = lc_reader->readNextEvent()) ){
```

- The method `readNextEvent()` continues to read `LCEvents` from the LCIO file until it reaches the end of the file

Example Analysis Using an LCIO File

- Getting a collection from an `LCEvent` can be done as follows

```
while( (event = lc_reader->readNextEvent()) ){

    // Get the collection of tracks from the event. If the event doesn't
    // have the specified collection, skip the rest of the event
    try{
        tracks = (IMPL::LCCollectionVec*) event->getCollection(track_col_name);
    } catch(EVENT::DataNotAvailableException exception){
        cout << "Collection " << track_col_name << " was not found. "
             << "Skipping event ..." << endl;
        continue;
    }
}
```

- The container used to store the collection is an `LCCollectionVec` i.e. a vector of `LCObjects` which needs to be accessed to retrieve physics objects which can be operated on

Example Analysis Using an LCIO File

- A detailed example illustrating how this is done can be found within the file `example/TwoTrackAnalysis_Example.cxx`

```
// Loop over all tracks in the event
for(int track_n = 0; track_n < tracks->getNumberOfElements(); ++track_n){

    // Get a track from the LCIO collection
    track = (IMPL::TrackImpl*) tracks->getElementAt(track_n);

    // Calculate the transverse momentum of the track
    pt = abs((1/track->getOmega())*b_field*param);

    // Calculate the momentum components
    px = pt*cos(track->getPhi());
    py = pt*sin(track->getPhi());
    pz = pt*track->getTanLambda();

    // Calculate the momentum of the track
    p = sqrt(px*px + py*py + pz*pz);
```

- The same can be done using clusters (`IMPL::ClusterImpl`), reconstructed particles (`IMPL::ReconstructedParticleImpl`) etc.
- More details can be found in the LCIO C++ API manual

What Collections Are Persisted?

- The `dump_hps_event` utility can be used to determine which collections were persisted as follows

```
./bin/dump_hps_event -i <input LCIO file>
```

- The utility will produce the following printout

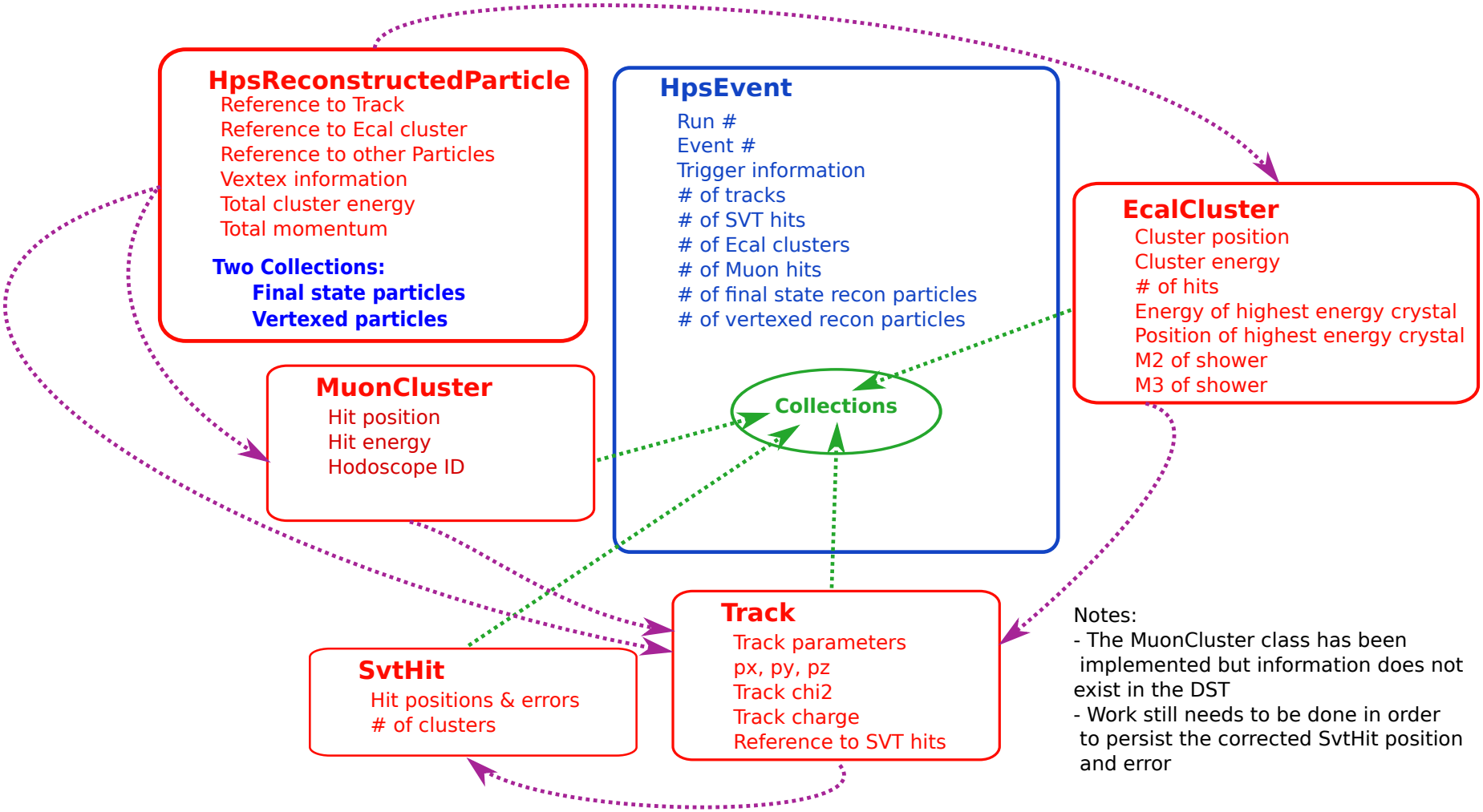
```
////////////////////////////////////
EVENT: 1
RUN: 1351
DETECTOR: HPS-TestRun-v3
COLLECTIONS: (see below)
////////////////////////////////////

-------------------------------------------------------------------------
COLLECTION NAME                 COLLECTION TYPE          NUMBER OF ELEMENTS
=========================================================================
ConfirmedMCParticles            MCParticle                        0
EcalCalHitPositions             LCGenericObject                   2
EcalCalHits                     CalorimeterHit                    2
EcalCalHitsRelation             LCRelation                        2
EcalClusters                    Cluster                           1
FinalStateParticles             ReconstructedParticle             0
HelicalTrackHitRelations        LCRelation                        0
HelicalTrackHits                TrackerHit                        0
HelicalTrackMCRelations         LCRelation                        0
MatchedTracks                   Track                             0
RotatedHelicalTrackHitRelations LCRelation                        0
RotatedHelicalTrackHits         TrackerHit                        0
RotatedHelicalTrackMCRelations  LCRelation                        0
SeededMCParticles               MCParticle                        0
StripClusterer_SiTrackerHitStrip1D TrackerHit                     0
TriggerBank                     LCGenericObject                   1
VertexedReconParticles          ReconstructedParticle             0
-------------------------------------------------------------------------
```

HPS Event Structure

- The ROOT based DST is composed of HpsEvent objects which are used to encapsulate collections (`TClonesArray`) of the following objects
  - EcalCluster
  - SvtTrack
  - SvtHit $\Rightarrow$ A class used to describe a 3D hit (stereo hit)
  - MuonCluster
  - HpsReconstructedParticle $\Rightarrow$ A class used to describe a particle i.e. a track associated with a cluster
    - An HpsReconstructedParticle can either be a single particle or be a composite of several particles $\Rightarrow$ vertexing information is provided for particles composed of daughther particles
    - Two collections: Final state particles, and particles which have been vertexed
    - Additional event information is also contained within HpsEvent
- Corrected SVT hit information and muon cluster data is still not available

Contents of an HPS Event



Notes:
- The MuonCluster class has been implemented but information does not exist in the DST
- Work still needs to be done in order to persist the corrected SvtHit position and error

Example Analysis Using the HPS DSTs

- The file `min_dst_analysis.C` illustrates the basic routines required to do an analysis using a DST
- It can be run as follows

```
root -l
.L min_dst_analysis.C
runMinDstAnalysis(<input DST file>)
```

- Reading a DST will require loading the HPS Event API as follows

```
std::string hps_dst_path(getenv("HPS_DST_HOME"));
hps_dst_path += "/lib/libHpsEvent.so";
// Check if the classes (HpsEvent, Track, EcalCluster, ...) are in
// the dictionary.  If not, load their definitions from libHpsEvent.so
if(!TClassTable::GetDict("HpsEvent")){
    std::cout << "Class definitions were not found! Loading libHpsEvent.so"
              << std::endl;
    gSystem->Load(hps_dst_path.c_str());
}
```

Example Analysis Using the HPS DSTs

- Once the DST has been loaded, the `TTree` ("HPS_Event") is obtained in turn allowing the address of the "Event" branch to be set to an instance of `HpsEvent`

```
// Get the TTree "HPS_EVENT" containing the HpsEvent branch and all
// other collections
TTree *tree = (TTree*) file->Get("HPS_Event");

// Create a pointer to an HpsEvent object in order to read the TClonesArrays
// collections
HpsEvent *hps_event = new HpsEvent();

// Get the HpsEvent branch from the TTree and set the branch address to
// the pointer created above
TBranch *b_hps_event = tree->GetBranch("Event");
b_hps_event->SetAddress(&hps_event);
```

- An event can then be loaded as follows

```
// Loop over all events
for(int entry = 0; entry < tree->GetEntries(); ++entry){
    // Print the event number every 500 events
    if((entry+1)%500 == 0){
        std::cout << "Event: " << entry+1 << endl;
    }

    // Read the ith entry from the tree.  This "fills" HpsEvent and allows
    // access to all collections
    tree->GetEntry(entry);
```

Example Analysis Using the HPS DSTs

- Once the event has been loaded, the various methods available through the HPS Event API can be used to access the collections
- A detailed example illustrating how this is done can be found within the file `example/TwoTrackAnalysis_Example.C`

```
// Loop over all tracks in the event
for(int track_n = 0; track_n < hps_event->getNumberOfTracks(); ++track_n){

    // Get the track from the event
    track = hps_event->getTrack(track_n);

    // Calculate momentum
    px = track->getPx();
    py = track->getPy();
    pz = track->getPz();
    p = sqrt(px*px + py*py + pz*pz);

    // Fill the plots
    h_p->Fill(p);
    h_px->Fill(px);
    h_py->Fill(py);
    h_pz->Fill(pz);
}
```

- The various methods used to access each of the collections can be found in the folder `include` $\Rightarrow$ Doxygen documentation is coming soon

What's Next?

- Corrected SVT hit information needs to persisted $\Rightarrow$ easy just need time
- Look at data integrity
  - Do the LCIO file and DST agree?
- Documentation for the HPS Event API needs to be written
- Will also begin adding unit test
- Final DST format is close to finish so production will begin shortly after