HPS Data Summary Tapes

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

June 18, 2014
Heavy Photon Search Collaboration Meeting

What's New?

- Switched to CMake build system in order to simplify cross-platform building
- DST maker was modularized
  - Several data writers (e.g. SvtDataWriter) are now used an event builder (HpsEventBuilder) to fill an HpsEvent
  - Makes it easier to add additional data to the DST
- A particle class (HpsParticle) was added and used to encapsulate reconstructed particle information e.g. mass, energy, references to daughter tracks and ecal clusters
  - Four additional particle collections have been added: Final state particles, unconstrained/target constrained/beamspot constrained particles
- MC particle information is also available, however, references to reconstructed objects have yet to be added
- Several GBL collections that can be used to run GBL were added by Pelle
  - Minor changes are still expected to occur (e.g. Add references to other DST objects) but they aren't crucial
- Doxygen can now be used to generate documentation
- Integration and unit testing of several components using gTest in conjunction with CTest is closed to being complete
  - This will be used to look check data integrity

Getting Started

Requirements

- LCIO C++ API
- ROOT data analysis framework
- Generalized Broken Lines
- CMake version > 2.8
- Doxygen (Optional. Used to build the documentation.)

Building the Source Tree

- The DST maker along with the HpsEvent source tree is available through github and can be cloned as follows

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

- Build instructions and usage can be found on confluence

```
https://confluence.slac.stanford.edu/display/hpsg/HPS+Data+Summary+Tapes
```

Generating DST's

- DST's are going to be generated as part of the reconstruction chain, however, there may be times when users may want to generate their own.
- Generating DST's from reconstructed LCIO files can be done by issuing the following command from a terminal

```
dst_maker LCIO_INPUT_FILE [additional input files] -o OUTPUT_FILE_NAME.root
```

- Multiple files can be processed at a time.
- Currently, the B field strength needs to be passed as an argument as it is used by the GBL code.
    - This is fine for now, but its probably best that this is stored in the LCIO collection somehow
- GBL collections can also be written to the DST's which can then be used by Millipede for alignment by issuing the following command

```
dst_maker LCIO_INPUT_FILE [additional input files] -o OUTPUT_FILE_NAME.root -g -b
[BField]
```


- Currently, the B field strength needs to be passed as an argument as it is used by the GBL code.
    - This is fine for now, but its probably best that this is stored in the LCIO collection somehow

- Configuration of the DST maker will be added at a later time and will be done using a CMake Cache file

* Not currently filled

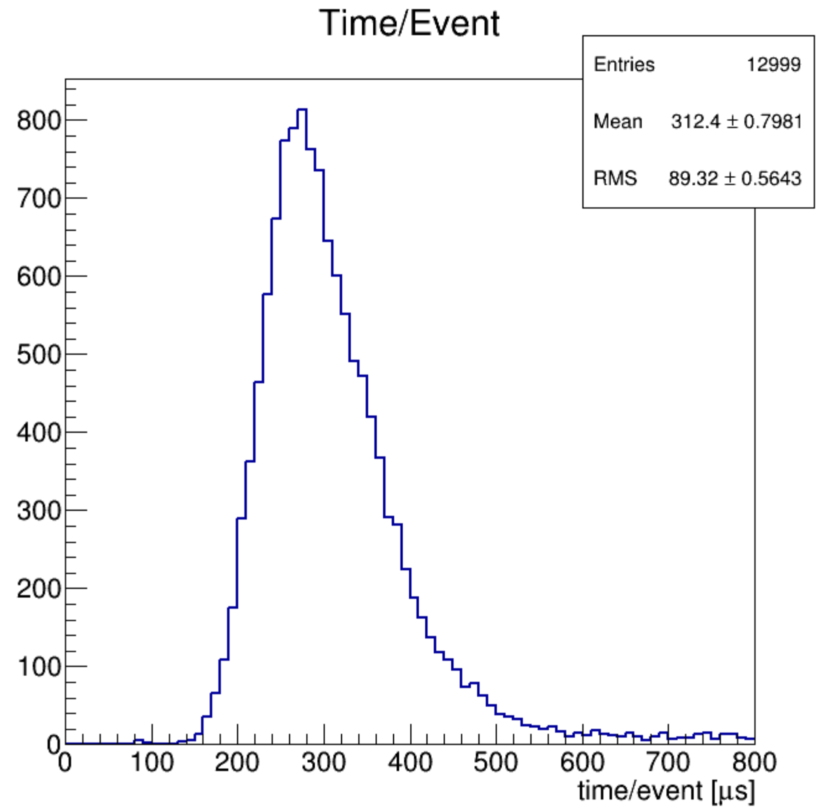
DST Structure

- HpsEvent class is used to encapsulate event variables: event #, run #, # tracks, # SVT hits, # Ecal clusters, # Ecal hits, # final state particles, # unconstrained/beam constrained/target constrained particles, # of MC particles, # GBL tracks, # GBL track data, # GBL strips data

Collection (TClonesArray)	Type	Variables (ROOT Leaves)	References (TRefArray)
tracks	SvtTrack	# hits, d0, phi, omega, tan_lambda, z0, chi^2	SvtHit
svt_hits	SvtHit	layer #, corrected position and covariance matrix, hit time	
ecal_clusters	EcalCluster	# ecal hits, position, energy, hit time*, m2*, m3*	EcalHit, SeedHit
ecal_hits	EcalHit	position*, crystal indices, energy	
fs_particles (Final State Particles)	HpsParticle	charge, momentum, energy, pdg ID	SvtTrack, EcalCluster
uc_vtx_particles (Unconstrained)	HpsParticle	# daughters, charge, fitted momentum, energy, vertex position	HpsParticle
bsc_vtx_particles (Beamspot Constrained)	HpsParticle	# daughters, charge, fitted momentum, energy, vertex position	HpsParticle
tc_vtx_particles (Target Constrained)	HpsParticle	# daughters, charge, fitted momentum, energy, vertex position	HpsParticle
mc_particles	HpsMCParticle	pdg ID, charge, generator status, energy, mass, momentum, endpoint	
gbl_tracks (optional)	GblTrack	kappa, theta, phi, d0, z0, seed_kappa, seed_theta, seed_phi, seed_d0, seed_z0, chi^2, ndf, momentum, covariance	
gbl_tacks_data (optional)	GblTrackData	# of strip hits, kappa, theta, phi, d0, z0	
gbl_strips_data (optional)	GblStripData	id, path3D, path ... tons of variables	

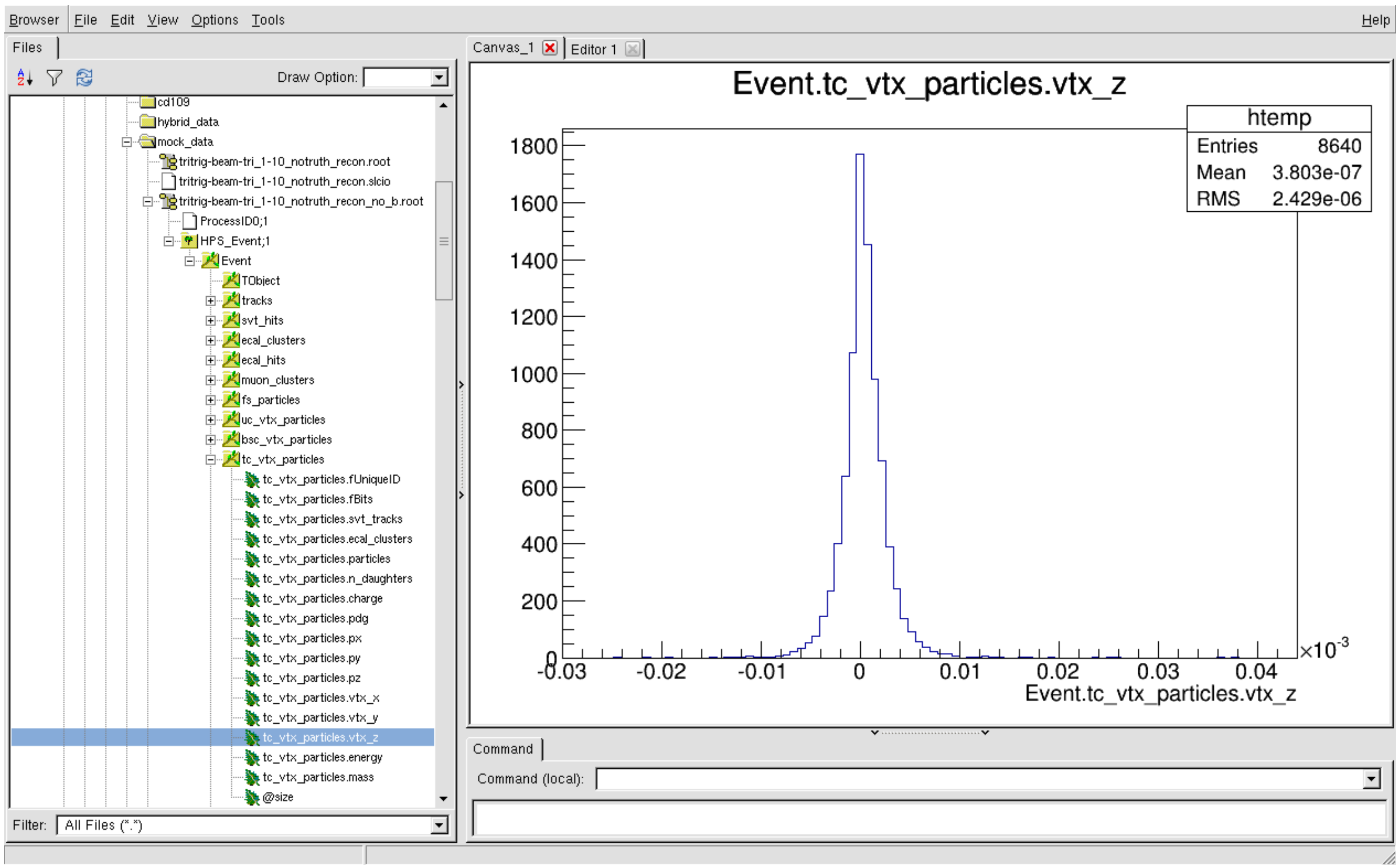
Performance

- DST is roughly the same size as the EVIO raw data and is much smaller than the recon LCIO file
  - Without the GBL collections, the DST is 6% the size of the recon file
  - With the GBL collections, the DST is 17% the size recon file
- The DST maker is also fast! It's taking ~312 us per event with GBL collections being written.



Accessing the DST's

- DST's can be viewed in a few ways
  - Using ROOT TBrowser
  - TTree::Draw
  - HpsEvent API



Using TTree::Draw to Access the DST

```
TFile file("hps_dst.root")
```

Load the DST File

```
TTree* tree = file.Get("HPS_Event")
```

Get the TTree named HPS_Event

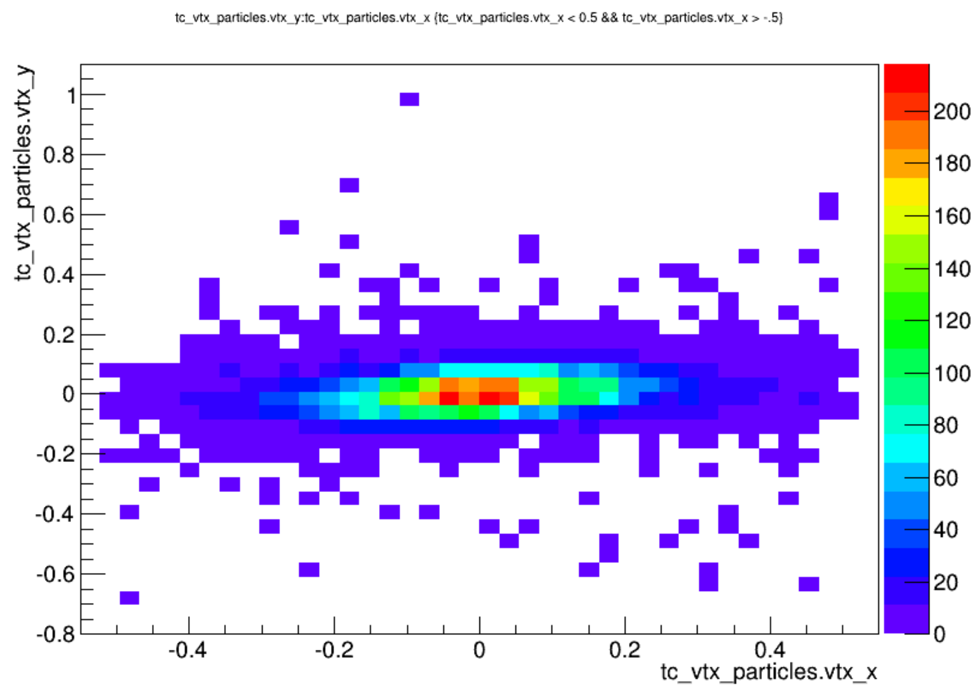
```
tree->Draw("tc_vtx_particles.vtx_y:tc_vtx_particles.vtx_x",
```

Collection name.variable_name

```
       "tc_vtx_particles.vtx_x < 0.5 && tc_vtx_particles.vtx_x > -.5", "colz")
```

Cuts

Options



Using the HpsEvent API - Setup

- A couple of examples demonstrating the usage of the DST come with the DST source code
  - `analysis_root_example.C` ← ROOT script that makes SVT, Ecal and HpsParticle plots
  - `analysis_pyroot_example.py` ← PyRoot script
- An example, `analysis_lcio_example.cxx`, demonstrating the use of the LCIO C++ API is also included
  - The example needs to be built using CMake

Set the branch address to the HpsEvent pointer address

Create the HpsEvent objects

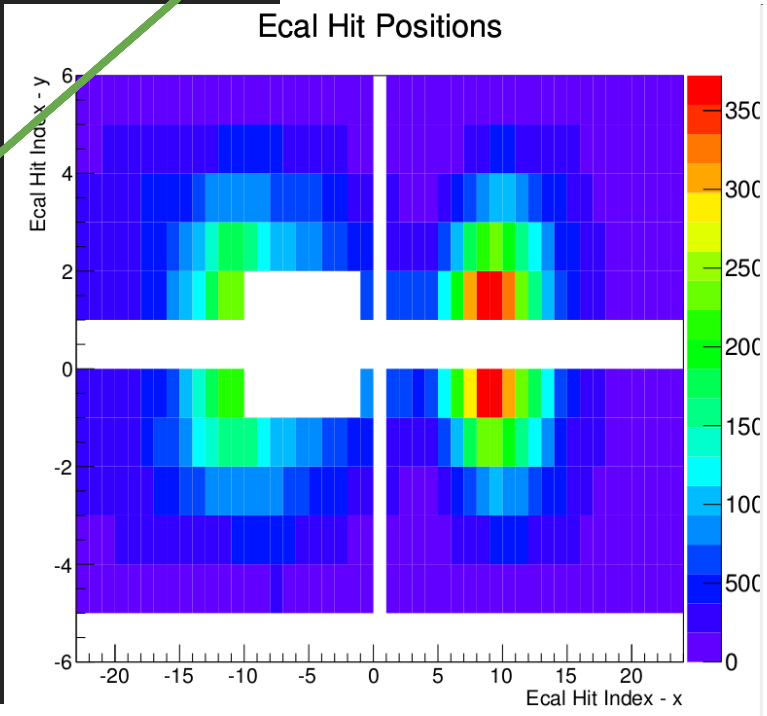
```
// Open the ROOT file
TFile *file = new TFile(root_file_name.c_str());

// 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);

int index_x, index_y;

EcalCluster* ecal_cluster = 0;
EcalHit* ecal_hit = 0;
```

Using the HpsEvent API - Analysis

```
// 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);

    // Loop over all of the Ecal clusters in the event
    for(int cluster_n = 0; cluster_n < hps_event->getNumberOfEcalClusters(); ++cluster_n){

        // Get an Ecal cluster from the event
        ecal_cluster = hps_event->getEcalCluster(cluster_n);

        // Get the Ecal cluster energy
        cluster_energy = ecal_cluster->getEnergy();

        // Fill the cluster energy plot
        h_cluster_energy->Fill(cluster_energy);

        // Get the Ecal hits used to create the cluster
        TRefArray* ecal_hits = ecal_cluster->getEcalHits();

        // Loop through all of the Ecal hits and plot their positions
        for(int hit_n = 0; hit_n < ecal_hits->GetEntries(); ++hit_n){

            // Get an Ecal hit from the cluster
            ecal_hit = (EcalHit*) ecal_hits->At(hit_n);

            // Get the crystal index of the ecal hit
            index_x = ecal_hit->getXCrystalIndex();
            index_y = ecal_hit->getYCrystalIndex();

            // Fill the Ecal hit position plot
            h_hit_pos->Fill(index_x, index_y, 1);
```

Each HpsEvent class has several getters which provide access to variables

Accessing collections related to a class (SvtTrack) is also easy



Conclusions

- Several improvements have been made to the DST's with a few more on their way
  - Integration and unit testing with googletest and CTest is almost complete and will be used to look at data quality i.e. check that conversion from LCIO to root is OK
  - Want to add a ProjectConfig.cmake which will allow enabling/disabling of DST collections
- Performance is good, and files sizes are small even with GBL
- DST for the mock data challenge is now available! Please take look at it and start exercising the API
  - Feature request, bug fixes, etc. can be submitted to the project bug tracker found here

    https://github.com/omar-moreno/hps-dst/issues?state=open
- I'll be around tomorrow if anyone needs help getting started.