Particle Tracking Data model to ROOT

For anyone interested primarily in analysing SuperNEMO data using ROOT, I have written a first version of a translation module to run in the pipeline (flreconstruct executable).

The idea is to run the reconstruction pipeline defined as 'official' on, for now, a simulated data source file such as the native binary output from the flsimulate executable. That will produce a data product inside the '.brio' binary output file which is labelled 'PTD', for 'Particle Track Data' and represents the final output after all reconstruction steps have finished. The translation module can read the reconstruction output file (caveat: tried it only on 'demonstrator' experiment geometries, i.e. with a tracker, a foil and calorimeters!), extract the relevant information, see below, and writes that information as simple data (numbers) into a single TTree object in a TFile on disk. The ROOT output file name is the only parameter of the module.

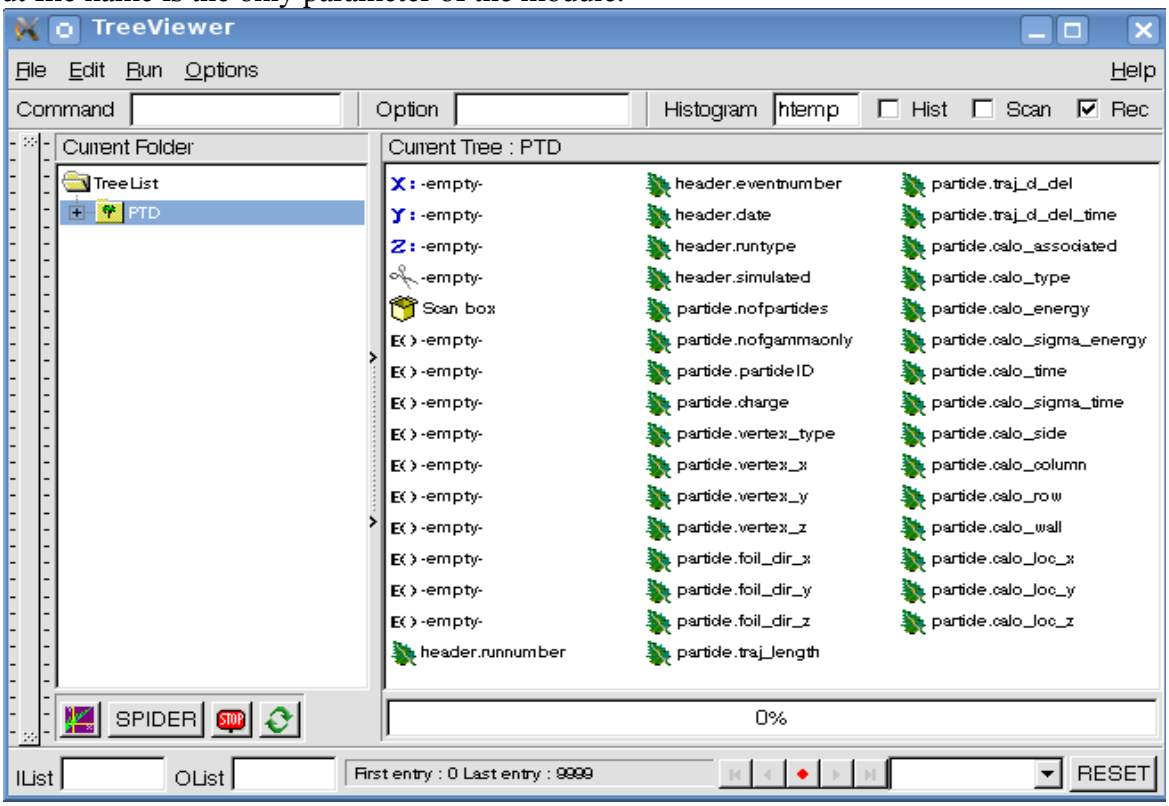

Figure 1: Screenshot of TTree with all variables available after conversion of the PTD datamodel.

The variables after conversion are shown in Figure 1. The event header data storage is quite pointless at the moment but will become more relevant. These are the variables starting with 'header'.

The extracted PTD variables start with the keyword 'particle'. Each PTD data model object is structured as a container for (a) particle track objects and (b) not associated calorimeter hits. Particle track objects are defined as patterns in the detector which contain a track in the tracker and an associated calorimeter hit which can be at any of the available calorimeters. In contrast the non associated calorimeter hits are just that, hits in calorimeters without any measurable track pointing at them.

The variable 'calo.associated' is a switch, a standard template library vector object containing integer values to say: associated yes (1) or no (0) or no information in PTD (-1). The corresponding calorimeter data is then stored in the 'calo_...' variables as listed in Figure 1. The calo_type is the geometry id integer number which might not be ideal in the long run but working with numbers in a tree is easier than with strings for applying cuts. Here one can select 1302 for the main calorimeter (is the geometry id number for the calorimeter block geometry box), 1232 for the xcalo and 1252 for the gamma veto. The energy and time variables are clear and the location variables (side, column, row, wall) characterise the individual calorimeter location which has the hit. The

calo_loc_... variables result from asking the geometry manager for the location vector3d of the geometry object with its geometry id number, here the calo sensitive detector block with a hit. Note that if location is crucial you might have the choice between a vertex location on the calorimeter and the calo_loc. The vertex location would normally be the better choice unless the error is big (and the track fitter delivers these errors which it currently doesn't).

The number of particles reconstructed is a global variable (an integer) in the particle track data container and labelled: 'particle_nofparticles'. Each of these is a particle track object. The particle track object contains information on vertices, the trajectory and associated calorimeter hits. For each particle track object that information is extracted and put into a vector container respectively. Usually, there will be two vertices, one trajectory and one associated calorimeter hit.

Explicitly extracted is: the vertex type and position coordinates, the trajectory length and the direction vector components at the foil (for the angle measurement) and all calorimeter data as listed above but now for associated hits.

The vertex type is again the geometry id of a calorimeter in case the vertex is located on a calorimeter face. Additionally, I define zero as the type number for the vertex on the foil and one for the vertex on neither the foil nor any calorimeter (internal key is 'wire' and the coordinates for those vertices are on a wire). I am aware of the enum structure called vertex_type in the particle_track object but can't get hold of it hence made this vertex type response from above for convenience. All I get easy access to is the geomtools blur_spot object (making a vertex) which is useful and can deliver a vertex type classification quite naturally.

The vertex coordinates are just that in x, y and z while the trajectory length should reflect the electron path length as measured and comes straight from the object (no calculation in the translator). The trajectory direction vector (particle.foil_dir_...) components at the foil are available from the code. These three direction variables are only filled in the translator for vertices at the foil, i.e. not for vertices at any of the calorimeters or on wires.

The next two variables also emerge from the trajectory: every trajectory is based on a cluster object and this cluster knows whether it is made from delayed hits or not. The variable 'traj_cl_del' is a flag (-1==not applicable, i.e. no trajectory, 0==not delayed, 1==delayed). In case hits are delayed, the 'traj_cl_del_time' variable holds the delay time (unit [ns]) which seems to work fine for creating Bi-214 decay time spectra.

Finally, the translator code merely grabs all available information in all categories on offer without checking. There can be missing information, for instance a trajectory between foil and a wire would still be a particle but without any calorimeter information. In order to keep the sequence of information in the ROOT tree, the translator provides padding with blank entries that can't be mistaken for realistic entries. These are typically negative numbers for particle id, calo_energy and type and so forth. Missing coordinates are set to large values well outside any programmed geometry. Only the entries in the foil_direction variable are not padded and correspond to the number of vertices on the foil, always. Something to remember when looping over that container instead of the number of particles.

What the padding allows is to loop over the number of particles, which now includes also the number of non associated calorimeter hits, i.e. gammas, or loop over three particle populations, those with associated calorimeter hits (calo_associated==1), i.e. with a vertex on at least one calorimeter, without (==0), gamma only, or padded (==-1) which are trajectories without calorimeter entry (negative energy entry - careful when simply summing energies!). Note that the padding of information is not merely convenience but necessary for a number table like a ROOT tree in order not to loose the coherence of numbers, i.e. which relates to which.

On top of presenting all the above variables to start an analysis, this module demonstrates similarly to the 'things2root' module how to access information from the Falaise software. It would therefore be straightforward to access other data model objects, other than PTD, in case additional reconstruction information is required for an analysis.

Yorck Ramachers 12/08/2015