# ECal Code Overview & Ideas

N. Baltzell

HPS Software Meeting

August 6, 2015

# Ecal Reconstruction Overview

1) org.hps.evio.EvioReader [Sho]

[current "expert"]

- **converts raw EVIO format to LCIO format**
- no change of information, just format

2) org.hps.recon.ecal.EcalRawConverter (&Driver) [Nathan]

- **converts raw FADC format to energy/time**
  - e.g. pedestal subtraction and gain scaling
- knows how to deal with Mode-1/3/7 raw formats differently
  - e.g. intermediate extraction of FADC pulse integral/time for Mode-1, after which it can deal with Mode-1 same as the others
- some say it has too many parameters and setters
- details on next page

org.hps.recon.ecal.EcalPulseFitter
- one way to extract pulse integral/time from Mode-1

org.hps.recon.ecal.EcalRunningPedestalDriver
- computes running pedestal averages from previous N readouts and puts it into LCIO

org.hps.recon.ecal.EcalTimeWalk
- performs Mode-dependent time-walk corrections

3) org.hps.recon.ecal.cluster.*Clusterer (&Driver) [Holly,Kyle]

- **groups hits into clusters, outputs cluster energy, position, time (and pointers to the clusters' hits)**
- implements time/spatial concidence between hits
- deals with energy sharing between clusters (ReconClusterer)
- hit-weighting algorithm to extract cluster positions (ReconClusterer)

4) org.hps.recon.ecal.cluster.ClusterEnergy/PositionCorrection [Holly]

- **shower loss / sampling fraction and shower depth correction**
- called downstream from org.hps.recon.particle.ReconParticleDriver
  - after attempt at track matching to get particle type ($e^-$,$e^+$,$\gamma$)

biggest candidate here for code review should probably be:

EcalRawConverter

# EcalRawConverter Details

- Basics: the driver loops over raw hits and chooses one of 3 methods to call based on raw data format (Mode-1/3/7) and dumps the converted hits (with time in ns and energy in GeV) into LCIO

- If Mode-1 raw data, extract pulse time/integral from waveform
  - outputs a hit class indistinguishable from Mode-3/7 by all code downstream from EcalRawConverter*
  - settable parameters include algorithm choice, threshold, max # pulses to find, …
  - can choose to use the same algorithms used online for Mode-3 or Mode-7 raw data
    - Integration range controlled with parameters NSA/NSB
    - Mode-3 time is threshold crossing, Mode-7 is linear interpolation on leading edge
  - or calls Pulse Fitting class that runs Minuit to extract pulse integral/time
    - settable parameters include fit range, wether to fix or limit fit parameters

- Pulse Time
  - For Mode-3/7 raw data, this is one number readout from the FADC
  - For all algorithms, there are independent time-walk corrections
  - Independent of Mode (currently) are 442 channel time offsets from DB

- Pedestal Subtraction
  - For Mode-3/7 raw data, or Mode-1 with 3/7's algorithms
    - requires knowing window size and NSA/NSB (settable parameters)
    - configurable to use static pedestals from db, or running pedestals
  - For pulse-fitting algorithm
    - pedestals can be fixed in the fit based on db or running pedestals
    - or a free parameter
    - result of fit is just analytic pulse integral (no "pedestal subtraction" required)

- Gains (conversion to FADC counts to GeV)
  - Treated identically for all Modes, and currently depends only on gains from db (no other parameters)

- There is also an option that chooses to set various parameters above on the fly (read from the DAQ configuration in EVIO). Should separate it and accompanying code from EcalRawConverter. More on this on following slides.

# Ecal Readout Overview

org.hps.readout.ecal.* (e.g. FADCEcalReadoutDriver is the "main" one)

- Primary experts here are probably Sho (he wrote most of it I think), Holly, Kyle

- Converts output of SLIC into something that should match what the DAQ "readouts" and writes to EVIO in real data. I don't know the details of the code, but some concepts it implements:
  - time latencies, pileup
  - convert SLIC's energy deposition to ADC (pedestal addition and gain scaling)
  - integrate energy deposition from SLIC in ~ns (?) samples into Ecal's FADC's 4ns samples
  - in case of Mode3/7, integrate FADCs over theshold crossing -NSB/+NSA into pulse integral

- Ideally it does all this in a way such that org.hps.recon.ecal doesn't need to know whether it was real data or simulation
  - for maintainability and simulation accuracy
  - this requires certain parameters to be set the same in readout and recon

- There's multiple readout drivers
  - maybe not well-named (?), only the experts seem to know which one to use when

# Good Ideas

- One instance of parameters that are used by both readout and recon
  - if they get out of sync, it is a problem (e.g. NSA/NSB, window size, global time shifts/latency)
  - already have this for pedestals, gains, channel time-offsets (via the conditions DB)
  - some are effectively hard coded and not obvious
    - e.g. "absolute" time cuts relative to window for both Pulse-fitting and Clustering (?).
      If we move the window, the cuts in recon must change, and readout must also shift its time window
  - surely we should do the easy ones

- Set more "proper" 2015 default values in code and remove from steering files
  - not independent of the previous bullet
  - I originally chose not to change the defaults in order to not break it for older data/simulation
  - Nice to have them in steering file to be able to see all settings at once, but clearly a problem (e.g. yesterday)
    - are people using old versions of steering files to create new ones???

- Make a separate EcalRawConverter purely for Trigger Emulation
  - to fully emulate trigger, online parameters and algorithms must be used in conversion of raw data
  - implmemented in EcalRawConverter during the Spring run to quickly/easily get trigger diagnostics working correctly and reliably (gets parameters from EVIO)
  - should be completely removed and put in its own EcalRawConverter that only emulates FADC/SSP algorithms
    - since those never change, fixed by hardware, while recon algorithms can evolve
    - some other options were discussed during the run, this is probably the best one

# Other Ideas

- Split up EcalRawConverter into subclasses
  - One possibility: one for each Mode-1/3/7, but there is much functionality they share
  - Another one: pedestal subtraction, gain scaling, pulse extraction method is each its own class
    - but again, there are some interdependencies in the algorithms, so not necessarily simple
  - Could be good, could be bad, probably largely increases overall code complexity no matter how it is done, but might seem easier after globalizing parameters (previous page)
    - and this code is pretty small anyway

- Use inheritance in EcalRawConverter to make multiple "Drivers"
  - Specialized drivers that set many of the parameters for the base class
  - Not sure about the value here, seems an unecessary complication
    - just live with doing this via steering files when necessary

- Better/more documentation
  - Won't hurt, although I really don't think this is actually a big ecal-recon problem
    - most methods that need commenting are already commented, it's not one of hps-java's 0-comment packages
    - Instead I would suggest spending more time documenting the general framework rather than ecal code
      - e.g. when do drivers' methods get called relative to all other parts of the code? (process, detectorchanged, setters, constructors)
      - a flowchart (somewhat like what Maurik generated but w/o the profiling)
  - Although ecal-readout uses some fancier concepts / data structures I haven't fully understood and may warrant some better documentation for the future

# Final Remarks

Definitely:

1. Remove many parameters from steering files (by changing defaults in code), EASY
    1. this may break old test run data and require new steering files for that, but it's time
2. Make a separate EcalRawConverter just for trigger emulation (and removing the bits then unecessary from EcalRawConverter), SLIGHTLY LESS EASY
3. Share parameters between readout & recon, "SHOULD" BE EASY
    1. Not independent of #1 above
    2. I'm opposed to doing this via the DB and conditions system in its current state
    3. Instead just a parameter class shared between packages with setters accessible via steering files

- Otherwise, I am strongly opposed to any significant ecal-recon code re-organizing/ structuring/factoring/etc without a clear new design and strong agreement that it will clearly be better, and someone committed to doing it and truly fully testing it
  - And I don't consider computer sciency reasons relevant (e.g. it will be "cleaner", or the old way was "wrong", or this is not proper java, …), the code's not really that big anyway
  - Nor trickle-down effects requiring asking others to spend a non-trivial effort on rewriting/ debugging their part of the code to fit within some new framework design
  - Could easily end up with something more complicated and something the existing experts don't have time to relearn in order to teach others

- I don't understand the readout code well enough to comment on that, except #3 above.