CalRecon Data

Diagram of Proposed CalRecon TDS Architecture

- ... can be found here

Prototype Definition of the CalParams Class

- Prototype class found here
- Intended to provide a single interface to all the basic energy reconstruction parameters:
  - Energy
  - The reconstructed energy centroid (cluster position)
  - The axis of the energy flow (cluster axis)
  - And the errors assigned to these
    - For energy: a number
    - For centroid and axis: two 3x3 error matrices
- Is a subclass of the main recon output classes

Prototype Definition of the CalCluster Class

- Prototype class found here
- This replaces the current CalCluster class, splitting out the corrected energy information that was formerly contained in CalCluster
- Otherwise, basically a reformatting of what was there:
  - The layer by layer information formerly stored in three std::vectors is now replaced by a subclass (CalClusterLayerData). CalCluster now derives from a std::vector of these objects making them directly accessible externally without having to provide access methods
    - Example:

```
CalClusterLayerData lyrData = calCluster[layer];
```

  - The energy, cluster centroid and cluster axis information is stored in an internal CalParams object.
    - Now includes errors on these quantities
    - But not yet filled with "real" values
- New things include:
  - CalCluster is a "ContainedObject"
    - typedef's are provided to defined the TDS Object container (CalClusterCol) which will store the clusters in the TDS
  - A relational table will relate CalClusters to the CalXtalRecObjects which comprise them
    - typedef's are provided to define this table
- A comment on the layer by layer data
  - Attempted to preserve the "old" system of an entry for each layer in the Cal, whether data existed or not. Unfortunately, one downside is the need to #define a default value of 8 so the constructor will be sure to create a vector of size 8 for the CalClusterLayerData objects...
  - A possible future solution would be to include a Cal identifier in the CalClusterLayerData object and then only keep non-zero entries
  - Other possibilities?
- Something forgotten: a status bit word...

Prototype Definition of the CalCorToolResult Class

- Prototype class found here
- **DISCLAIMER:** bare bones version, no attempt yet to add mechanism for storing correction tool dependent information
- CalCorToolResult will store the output of any given energy correction tool that runs during event reconstruction
  - There will be one CalCorToolResult output for **each** correction tool run (they will be stored in CalEventEnergy, see below)
- Vary basic minimum output includes:
  - CalParams to give energy, position, direction and errors
  - "ChiSquare" to give some measure of the "goodness" of the correction
  - A status bit word to describe:
    - The correction tool which provided this output
    - Other useful things that we will think of...
- CalCorToolResult is a Gaudi "ContainedObject" and typedef's are provided to define the container to hold the collection of them.
- typedef's are also provided to define a the table which will relate each CalCorToolResult to the cluster(s) from which the result was derived.

Prototype Definition of the CalEventEnergy Class

- Prototype class found here
- CalEventEnergy is the "final" output of the Calorimeter energy reconstruction.
- The basic makeup of the class:
  - The class derives from an ObjectVector of CalCorToolResult objects, thereby acting as their "Gaudi Container" for the TDS.
    - Note that this introduces some interesting problems which cause a couple of issues with ownership and deletion, etc.
      - Just need to be aware of this
  - The class contains a CalParams object to contain the "final" corrected energy for use outside CalRecon
    - It is envisaged that the CalEnergyAlg which creates CalEventEnergy will provide a mechanism (via a Tool or something) which will select the "best" energy solution amongst those returned by the various correction tools
  - A status bit word which can

        - Keep track of what stage of recon the contained output came from
        - Identify which correction tool provided the "final" energy
        - etc.

Status as of May 4, 2005

- Above classes implemented with necessary modifications made to:
    - CalRecon v5r24p1
        - **Note:** this is already several revisions behind the current CalRecon tag so the modifications made to CalRecon for this demonstration most likely cannot be checked into cvs
    - Event v10r3p1
    - HepRepSvc v0r13
    - RootIO v14r7
        - No attempt made to make PDS versions, just mods to reconRootWrigerAlg and reconRootReaderAlg to compile and run
    - TkrRecon v10r3p10
- Add one new Algorithm to CalRecon:
    - CalEnergyAlg
        - This assumes responsibility for driving the energy correction tools
        - Provides user definable correcton tool result selection for the "final" answer
- Add one new Tool to CalRecon
    - CalRawEnergyTool
        - Runs at the end of the first pass iteration of energy recon and serves to sum the energy over all available clusters and re-calculate a "grand" cluster centroid and axis
        - Serves as the "super" cluster result
- Modify the job options flow:

```
Reconstruction.Members={
    "Sequencer/Cal1",
    "Sequencer/Tkr",
    "Sequencer/Cal2",
    "Sequencer/TkrIter",
    "Sequencer/Acd"
    };
Cal1.Members = {
    "CalXtalRecAlg",
    "CalClustersAlg/first",
    "CalEnergyAlg/RawEnergy"
    };
Tkr.Members = {
    "TkrReconAlg/FirstPass"
    };
Cal2.Members = {
    "CalEnergyAlg/second"
    };
TkrIter.Members = {
    "TkrReconAlg/Iteration"
    };
```

    - Assumes clustering only need be called in the first pass recon
    - Sets up call to CalEnergyAlg which will run only the CalRawEnergyTool in the first pass
    - Sets up call to CalEnergyAlg in the second pass which will call all remaining correction tools
- The above structure appears to work in test Gleam jobs (10k 100 MeV gammas and 10k 1 GeV muons). So... on the right track?