TDS Class Structure for CalRecon

Current Structure
First Proposal for a New Structure
Current CalRecon TDS
As seen in Event/Recon/CalRecon (and on TV)

- **CalXtalRecData**
  - One per Crystal (with energy above threshold)
  - Derives from ContainedDataObject
  - Stored in a Gaudi ObjectVector (typedef’d in the class definition)
  - Description:
    
    The class contains reconstructed position and for both POSitive and NEGative faces of a crystal the reconstructed energy and the range number used to obtain this energy.

- **CalCluster**
  - One per event (though, technically, could be multiple/event)
  - Stored in CalCluserCol which derives from DataObject and std::vector<CalCluster*>.
  - Description:
    
    This class stores the results of the reconstruction performed in CalClustersAlg. It contains the reconstructed data for one cluster in the calorimeter.

- **CalRecon**
  - Not used?
Containers “own” objects which means, basically, that they are responsible for deleting them.

Gaudi Data Objects (and ContainedObjects) are deleted at the start of the next event (automatically)
CalRecon TDS
Why Need for Change?

• Update Old Stuff
  – Change from single cluster per event to multiple clusters per event
    • Move event summary information to a higher level
      – Would be meaningless if copied into all CalClusters (if more than one)
      – e.g. Corrected Energies
    • Relate Clusters to the CalRecXtalData objects which they contain
      – Not needed before when all crystals were included in THE cluster
  – Include more information which could be useful in later stages of Recon
    • e.g. Add a status bit word which can be used to describe which energy correction algorithms were run, were successful, etc.

• Add New Stuff
  – MIP finder
    • Will operate on subsets of CalRecXtalData objects?
    • Will be independent of Clustering?
    • Multiple MIP objects per event?
    • Other?
The following assumptions are personal prejudice
  - May well not be how this group decides to do things
  - But hopefully are flexible enough to accommodate any scheme

MIP Finding
  - Will it run post pre- or post-Clustering?
  - Ultimately operates on CalXtalRecData
    - Assume it will run post pre-Clustering for what follows

Cluster Algorithms
  - Prejudiced by “simple” cluster algorithm
  - Assume at this level it runs within single towers
  - Clusters which cross towers merged at the next higher level

There will be a “final” energy reconstruction above cluster finding
  - Will operate on input clusters for improved energy reconstruction
  - Will need a summary output class to contain the final numbers
CalRecon TDS
One Possible Proposed Structure
CalRecon TDS
One Possible Proposed Structure

• Basic Structure:
  – CalXtalRecData, CalMinIParticle, CalCluster
    • Derive from Gaudi’s ContainedObject
    • Stored in TDS in an ObjectVector with “Col” appended to their name
  – CalEventEnergy summary object
    • Derives from DataObject (only one of them per event?)
    • Relational table to get back to Clusters or MIPs?
  – Use Relational Tables to connect Xtals to Mips and Clusters
    • Quick independent lookup in either direction (see next page)

• Questions
  – Will MIP finding be before or after Clustering?
    • Before: could mark crystals to not be used by clustering?
    • Before: could identify min I events and abort recon?
    • After: Could follow a min I trail into its “explosion”
    • After: Clusterer tells MIP finder where to look
  – Is CalEventEnergy really something for the EventRecon stage of recon?
    • Still to be created package
    • Equivalent of the current AnalysisNTuple stage
Why Relational Tables

What is a Relational Table?
- See Event/RelTable/RelTable.h for the details on the GLAST implementation
- Relation: This binds (using SmartRef’s) together two Gaudi DataObjects (or ContainedObjects) in a data structure which can be inserted into a linked list
- Relational Table: This is a table of all the relations between pairs of objects of two given types
  - e.g. One could develop a set of “Relations” between CalXtalRecData objects and (if they would derive from ContainedObject) CalCluster objects and store these in a Relational Table
  - The Relational Table provides methods to then retrieve collections of relations using either related object as a key

Why would we want to use such a structure?
- Bidirectional:
  - Given a CalCluster, retrieve all the CalXtalRecData objects associated to it
  - Given a CalXtalRecData object, retrieve all CalCluster objects associated with it
    - From the same table
- Feature: if objects can be shared can easily determine who is doing the sharing

GLAST implementation as a DataObject
• Set the general structure first
  – ContainedObjects and ObjectVectors
    • Agree on naming convention
      – e.g. CalCluster will be contained in a CalClusterCol and we can define iterators as CalClusterColIter, CalClusterColConsIter, etc.
  – Using Relational Tables or not
    • Even if Cal doesn’t “share” objects, they are best because they are bidirectional
  – Need for level above CalClusters or not
    • I think it simplifies the CalCluster output making it less confusing

• Set the general content of the classes next
  – Agree on general output of each level (approximately what it is now)
  – Add StatusBit word to output detailed information
    • e.g. Which clustering algorithm was run

• But… let the exact details be somewhat flexible at this stage
  – e.g. I might find that it is useful to have my fiance’s birthday in CalCluster output so that I won’t forget it (of course, one could argue how much value keeping me alive is to the experiment as a whole)