JRA1 Analysis Framework

Goal:

prepare tool for telescope data analysis starting **July 2007**

Time is short and most of us can not devote much time to development of a new framework:

⇒ try to make it as simple as possible

⇒ use existing experience (and code)

⇒ but keep it general and flexible
to allow future extensions

This meeting

To start developing the framework we have to make few decisions.

What we should probably discuss (and decide) today is:

1. programming language and (development) platform
   C++ and Linux ?
   Do we want to keep Windows option open (and how) ?

2. programming environment
   Merlin, root based, standalone code ? GUI ?

3. internal logic and data structure
   LCIO based, taken from existing code (sucimaPix), other ?
   Before we can start distributed code development, we have to define classes for coding the data, geometry etc.
   We have to think carefully about the design and objectives...

This meeting

We should also think of:

• Scheduling regular meetings/teleconferences
can we use any IP based system – we would not have to book a room each time ?

• Code repository
probably DESY would be the best place. On-site responsible person needed?

• Input data structure and options
can DAQ provide us with the format specifications? options?
can we expect any kind of test data before July 2007?

Framework data classes

Following structure could be defined:

⇒ raw data input (full frames)

⇒ hits (single pixels above threshold)

⇒ clusters

⇒ reconstructed tracks

⇒ sensor geometry and parameters

⇒ telescope geometry (list of sensors?)

⇒ run and event headers

⇒ parameters and environmental variables (if not in headers)

Framework data classes

Example of raw data classes – two approaches

LCIO: TrackerRawData

int flags, n;

n × int cellID0, cellID1, time, nADC;

nADC × short ADCValues;

sucimaPix: class TPixelMatrix

std::vector<Double_t > matrix

\+ Int_t eventNumber; Int_t noOfOTP; inherited from TEventHeader

\+ Int_t xNoOfPixel, yNoOfPixel;

Int_t noOfEntriesPerRow;

Int_t signalPolarity;

Double_t xPitch, yPitch;

Bool_t subMatrix; inherited from TDetector

Framework data classes

Example of single pixel classes

LCIO: SiliconRawHit

int flags, n;

n $\times$ int cellID0, cellID1, timeStamp, adcCounts;

sucimaPix: class TPixel

Int_t pixelID;

Double_t signal, noise; Int_t raw;

\+ Int_t xNoOfPixel, yNoOfPixel;

Int_t noOfEntriesPerRow;

Int_t signalPolarity;

Double_t xPitch, yPitch;

Bool_t subMatrix; inherited from TDetector

Framework data classes

In my opinion geometry description should be separated from the actual data – data classes should only contain detector/sensor ID

Possible geometry class layout:

Experiment (global container)

- List of geometry descriptions
  - List of detector planes (e.g. position and orientation)
    - Sensor (e.g. numbers of pixels, pitch, thickness, but also can include methods for reading the data)

Geometry descriptions could be read from files – user could view available geometries and choose the proper one.

New geometry description would also be created (written to file and added to the list) as a result of the alignment procedure.