Discussion of the histogramming package for psana

Abstract

We discuss a `PSHist` - a histogramming package for PSANA project.

Objectives

In PSANA framework we need in package which allows to accumulate data in form of histograms and tuples and save them in file for further analysis. Though it might be based on well-known underlying packages like ROOT, HBOOK, or HippoTuple, we prefer to use an uniform interface with abstract base class, substituting the direct interaction with underlying methods. This intermediate abstract layer provides a flexibility in implementation of new things, for example multithreding in analysis, which algorithm is not yet defined.

Interface

An example of the program interface for this histogram-tuple-management package can be presented as follows. First, the underlying (derived) package needs to be choisen and instantiated. For example for ROOT

```
HManager *hMan = new RootHist("my-data-file.root"); // for ROOT
```

Then later in the program we may create ntuple(s) without knowing what is the underlying histogram-tuple-storage management system

```
Tuple *nt = hMan->ntuple("EXP Data");
```

and histograms

```
H1 *pHis1i = hMan->his1i("His1 int    title",100,0.,1.);
H1 *pHis1f = hMan->his1f("His1 float  title",100,0.,1.);
H1 *pHis1d = hMan->his1d("His1 double title",100,0.,1.);
H2 *pHis2d = hMan->his2d("His2 double title",100,0.,1.,100,0.,1.);
```

When the ntuple is created, its parameters need to be defined once

```
TuplePar *pBeamEnergy  = nt->parameter("beamEnergy");  // minValue, maxValue, dtype ?
TuplePar *pBeamCurrent = nt->parameter("beamCurrent");
```

Then, in data processing stage, for example for each event, we may accumulate data in created histogram and ntuple objects using pointers

```
pBeamEnergy ->fill(E);
pBeamCurrent->fill(I);
nt          ->addRow();
```

or parameter names

```
nt         ->fill("beamEnergy", E);
nt         ->fill("beamCurrent", I);
nt         ->addRow();
```

Histograms can be accumulated using their pointers

```
pHis1i     ->fill(x,[weight]);
...
pHis2      ->fill(x,y,[weight]);
```

where we assume that all parameters like `x`, `y`, `E`, and `I` and optional `weight` were earlier defined.

When you are all done, write the data into a file:

```
hMan->write();
delete hMan;
```

Structure and content of the package

Package PSHist contains the base abstract class and methods, which have to be re-implemented in derived classes for HBOOK, ROOT, HippoTuple, etc., i.e. in packages like HBookHist, RootHist, HippoHist, etc.

QUESTIONS

- What kind of interface is preferable for tuple parameters: (1) name-based, (2) pointer-based, (3) index-based?
- Do we really have to care about different types of data for histograms and tuple parameters (int, float, double)?
- Arrays in tuples?
- What should be the right mapping between PSHist and Root classes?

```
HManager  <-> TFile
H1        <-> TH1D
Tuple     <-> TTree or TBranch
Parameter <-> TBranch or TBranch-item
```

APPENDIX

Histogramming in Root

```
// Base Class Headers --
#include "root/TROOT.h"
#include "root/TFile.h"
#include "root/TH1D.h"
#include "root/TTree.h"
#include "root/TBranch.h"
#include "root/TRandom.h"
```

Histograms

```
// Initialization
if(!TROOT::Initialized()) {static TROOT root( "RootManager", "RootManager ROOT God" );}

// Open output file
TFile* pfile = new TFile("file.root", "RECREATE", "Created for you by RootManager" );

// Create histograms
TH1D *pHis1 = new TH1D("pHis1","My comment to TH1D", 100, 0.5, 100+0.5);
TH2D *pHis2 = new TH2D("pHis2","My comment to TH1D", 100, 0.5, 100+0.5,
                                                     200, 0.5, 200+0.5);

// Fill histograms in each event
pHis1 -> Fill( value, [weight] );
pHis2 -> Fill( x, y, [weight] );

// Write histograms in file in the very end
pHis1 -> Write();
pHis2 -> Write();

//Close file
pfile -> Close();
```

NTuple

Assuming that file is already open,

```
// Define some structures for TTree
typedef struct {float x,y,z;} POINT;
static POINT point;
float new_v;

// Create TTree
TTree* ptree = new TTree("ptree", "My comment to TTree");

// Create branches
TBranch *pbranch = ptree->Branch("new_v", &new_v, "new_v/F");
                   ptree->Branch("point",&point,"x:y:z");

// Fill data structures for each event
      new_v   = gRandom->Gaus(0, 1);
      point.x = gRandom->Gaus(1, 1);
      point.y = gRandom->Gaus(2, 1);
      point.z = gRandom->Gaus(3, 1);

// Add an event record to the tree
      ptree->Fill();

// Write the tree to the file
ptree -> Write();
```

Histogramming in BABAR

All histograms in Workbook examples are based on plane Root...

Histograms

```
    HepHistogram myPlot("The Title",100,0.,1.);
    float x, weight;
      ...
    myPlot.accumulate( x, weight );
```

NTuple

```
#include "HepTuple/Histogram.h"
#include "HepTuple/TupleManager.h"

    HepTupleManager* manager = gblEnv->getGen()->ntupleManager();

    HepTuple *_ntuple = manager->ntuple("file-name.root");

   _ntuple->column("run",   eventID->run(), -99, "Event" );
   _ntuple->column("event", eventCounter,   -99, "Event" );

   _ntuple->dumpData(); // for each event
```

Histogramming in CLEO

Histograms

Declare Histogram in your .h file

```
#include "HistogramInterface/HistogramPackage.h''
...
HIHist1D *m_trackMom;
```

Define Histogram in your Processor

```
m_trackMom = iHistoManager.histogram("track momentum (GeV)",50, 0, 1.5);
```

Fill Histogram in event() method of your Processor

```
m_trackMom->fill(momentum);
```

Load Conversion Module in .tcl script

```
module sel RootHistogramModule
root file root_suez_style_example_data31.root
root init
```

NTuple

Declare Ntuple in your Processor's .h file

```
#include "HistogramInterface/HistogramPackage.h''
#include ''HistogramInterface/HINtupleVarNames.h''
...
HINtuple *m_trackTuple;
```

Make an enum of variable names

```
enum{kpx, kpy, kpz, kd0, kz0, knum_vars};
```

Define Ntuple in hist_book()
// make an object that holds the names of the variables in your ntuple

```
HINtupleVarNames knames(knum_vars);
knames.addVar(kpx,"px");
knames.addVar(kpy,"py");
knames.addVar(kpz,"pz");
knames.addVar(kd0,"d0");
knames.addVar(kz0,"z0");
m_trackTuple = iHistoManager.ntuple(1,"tracktuple",knum_vars,50000,knames.names());
```

Fill Ntuple in event() method

```
trackTuple[kpx] = px;
trackTuple[kpy] = py;
trackTuple[kpz] = pz;
trackTuple[kd0] = d0;
trackTuple[kz0] = z0;
m_trackTuple->fill(trackTuple);
```