Development of tools for event data decoding and quality analysis

Ilnur Gabdrakhmanov

VBLHEP, JINR

Dubna April 19, 2021

7th Collaboration Meeting of the BM@N Experiment at the NICA Facility

Primary tasks

- Provide tools for seamless transition between two tracking approaches. Particularly develop DST and also MC digits' converter.
- Create building blocks for the unified online/offline QA system. Making them as flexible as possible is of great importance.

DST conversion. Main points

CBM/L1 DST structure:

- flat structure: 1 array for all tracks with links to corresponding hits
- separate array for clusters
- each found Primary Vertex contain track indices involved

BM@N DST structure:

- Global tracks contain links to subtracks

$$\textit{Global track} \rightarrow \left\{ \begin{array}{c} \textit{Silicon track} \rightarrow \\ \textit{GEM track} \rightarrow \\ \textit{CSC hit} \\ \ldots \end{array} \right. \quad \ldots$$

- each subtrack contains links to hits
- each hit contains cluster information
- Primary Vertex format generally the same as in L1

DST converter general logic

- Recreate tracks/subtracks structure
- Remap hit array
- Gather cluster information and rewrite it into hit data

Usage examples

As a macro:

```
void DSTConv(
        TString inFile = "~/filesbmn/4649-cbm-full.root",
        TString outFile = "$VMCWORKDIR/macro/run/dst-bmn-4649.root",
        Int_t nStartEvent = 0,
        Int_t nEvents = 100) {
```

As a task in an analysis workflow:

```
BmnTrackConv * conv = new BmnTrackConv(periodId, runId, kBMNSETUP);
fRunAna->AddTask(conv);
```

DST converter completeness

- ☐ Inner tracker content (including CSC) conversion is implemented
- ☐ Cluster info is fully converted
- ☐ Indices of tracks leading to primary vertices are also being copied
- ▷ ToF detectors as well as DCH are in progress
- ▷ MC digits are also needed to be converted

Pad generator for QA. Motivation

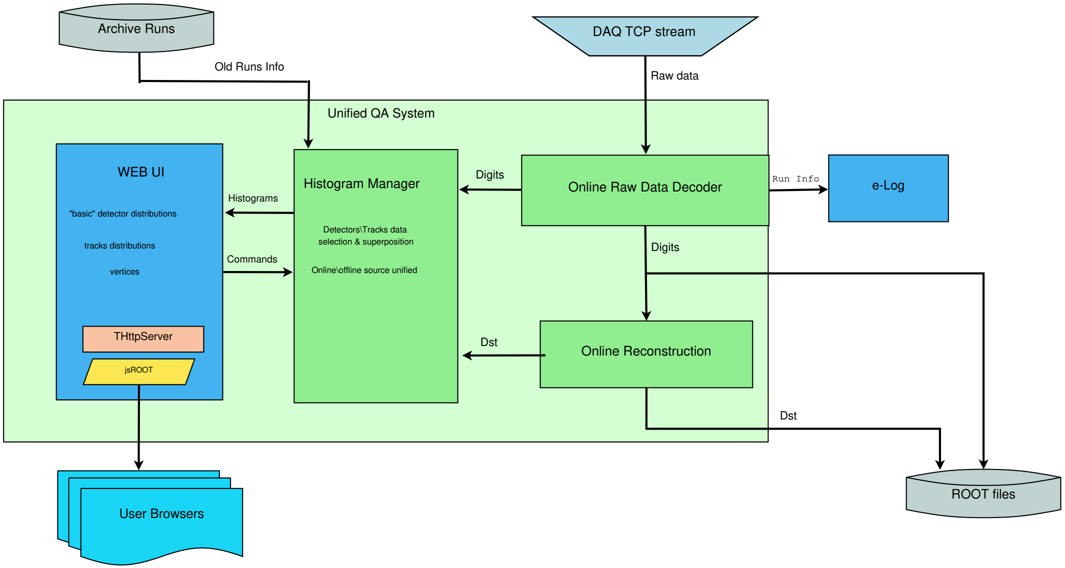
Why?

Experiment upgrade as well as conduction of two experimental setups require distribution of work on the development of the online QA system.
Namely each detector team should be able to extend system's functionality easily.

Main objectives:

- Move monitoring configuration outside of the code
- Make addition of histogram simple and flexible (It should not require code rebuild)
- If possible make the same for filling logic

Scheme of the future QA system



Pad generator for QA. Workflow

Implementation and usage pattern:

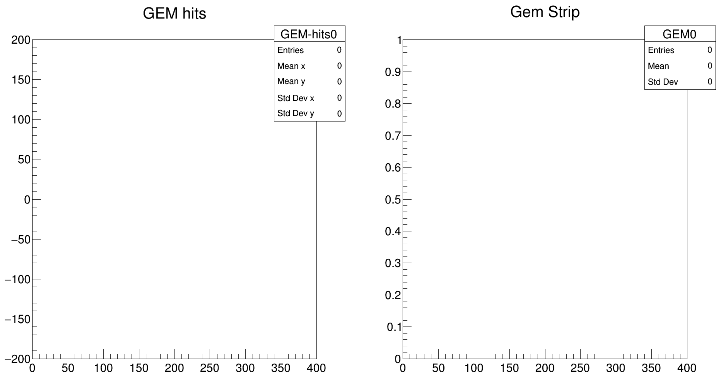
- User writes histograms configuration in the json file
- BmnPadGenerator converts json structure to the recursively nested BmnPadBranch objects
- BmnPadBranch tree can be mapped on any canvas
- BmnPadBranch tree can be filled or drawn [on the corresponding canvas] afterwards

Simple configuration

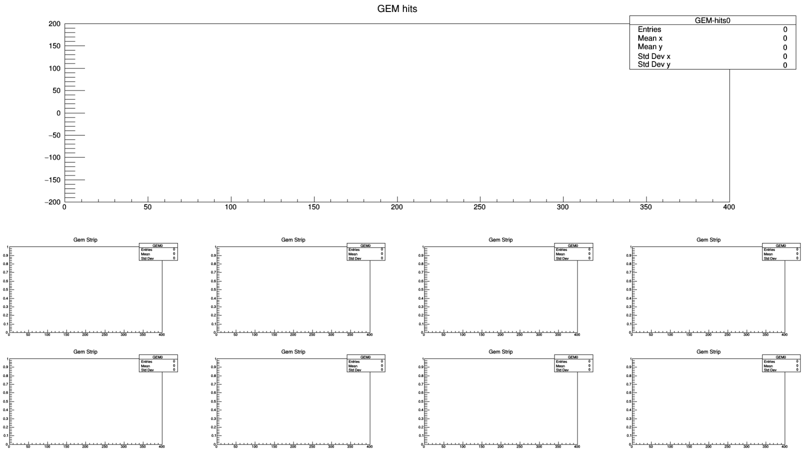
JSON scheme:

```
{
  "Name": "GEMS",
  "Title": "GEM Canvas",
  "DivX": "2",
  "DivY": "1",
  "Pads": [
    {
      "Class": "TH2I",
      "Name": "GEM-hits0",
      "Title": "GEM hits",
      "Options": "colz",
      "Dimensions": [
        200,
        0,
        400,
        400,
        -200,
        200
      ]
    },
    {
      "Class": "TH1F",
      "Name": "GEM0",
      "Title": "Gem Strip",
      "Dimensions": [
        200,
        0,
        400
      ]
    }
  ]
}
```

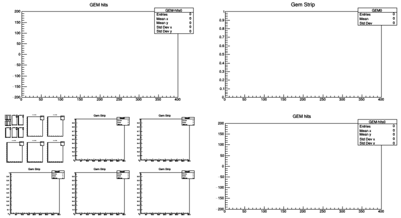
Canvas structure:

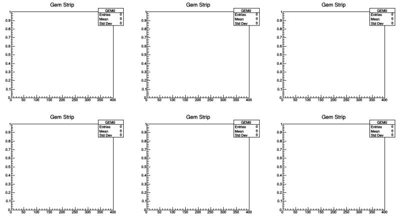


More complex configuration



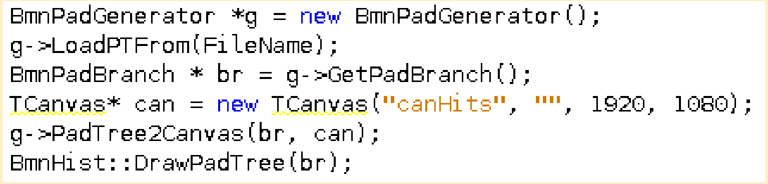
More complex configuration





Code example

Typical example



```
BmnPadGenerator *g = new BmnPadGenerator();
g->LoadPTFrom(FileName);
BmnPadBranch * br = g->GetPadBranch();
TCanvas* can = new TCanvas("canHits", "", 1920, 1080);
g->PadTree2Canvas(br, can);
BmnHist::DrawPadTree(br);
```

Next steps

- Implement filling procedures
- [optional] Add support for style options (or use one style for all)
- Try to implement configuration scheme for histogram filling logic

Thank you!