INO Data Structure - V2.0

Gobinda Majumder

July 19, 2021

1 Introduction

This data structure is constructed keeping the INO-ICAL in mind. In an event in ICAL, not all 30000 RPCs are going to have latch signals. So this data structure thus intent to store the information of only the RPCs which has signals. Again, not all TDC channels in a DAQ is going to contain hits. So only the information of the TDC channels with hits are going to be stored in the structure.

Filling an empty vector in a TBranch occupy a little disk-space. It might be insignificant for a single event, but it gets accumulate with each events and becomes hugely significant.

Same is applicable to CAU channels. At each CAU-cycle, CAU selects some of the DAQs to calculate the round-trip-delay. Hence only the TDCs with information of CAU-delay are stores.

Similarly, in each mon-cycle, a few strips are monitored in each DAQ. Only those information from all (or few) the DAQs will be sent to event builder and get stored in each cycle.

One important aspect of this structure is that each DAQ must be identified by the physical location, i.e. in terms of module, x-row, y-row and zlayer. In this way, the user does not need to have any mapping between IDs. The **rpcID** is constructed as per the Figure 1.

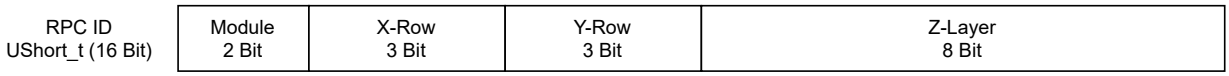

Figure 1: RPC-ID Structure.

2 User Interface

The Figure 2 shows a general format of the user interface of the Even-Builder. More fields may be added if necessary.

For some of the fields, the values can be retained from the last run. This needs to be carefully considered. Unintentional mistakes can lead to serious consequence for some of the fields. So for important fields, user must enter the values before every run (i.e. the fields with deep colour).

For safety, the **TEST** box is checked by default.

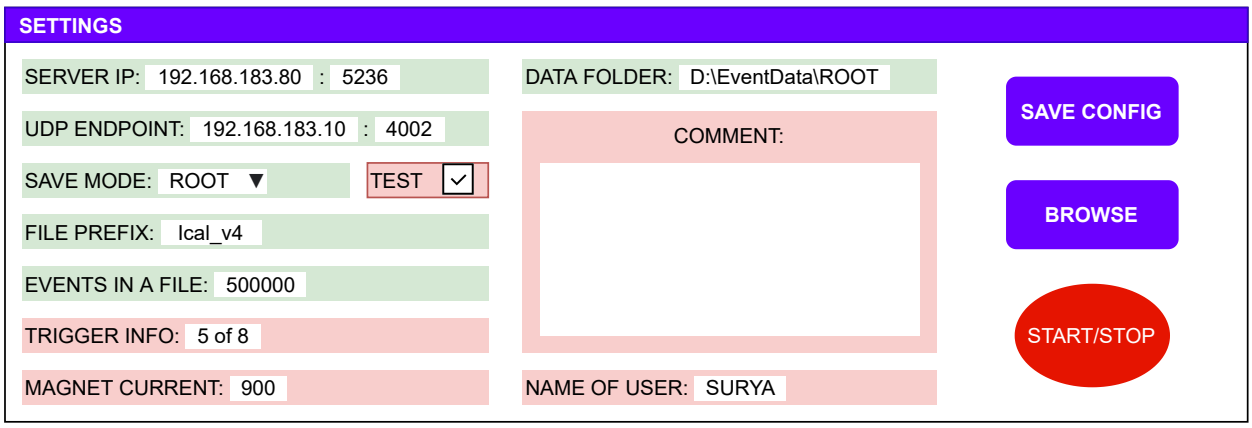

Figure 2: User Interface of Even-Builder.

The scheme of the name of the files written is shown in the bellow.

$$\overbrace{\text{Ical\_v4}}^{\text{file prefix}}\underbrace{\text{\_yyyyMMdd\_hhmmss}}_{\text{timestamp}}\overbrace{\text{\_B/N}}^{\text{B if magnet run}}\underbrace{\text{\_T/P}}_{\text{T if test run}}\overbrace{\text{\_0001}}^{\text{File Count of Run}}\underbrace{\text{.root}}_{\text{file extension}}$$

A new file should be created when the desired count of events is reached. The name of the new file should be constructed from the timestamp of the first event in the new file or the system time.

3 Run Info

This object is written once in the file and holds the inforamtion of the run details. The member variables of this class, namely **RunInfo.h**, are,

```
UInt_t      RunNo;                  /* Unique incremental /*
Bool_t      isTest;                 /* true if test */
UShort_t    MagnetCurrent;          /* in Amp */
TTimeStamp  StartTime, StopTime;    /*  */
TString     TriggerInfo;            /*  */
TString     Creator;                /* User  */
TTimeStamp  CreatedOn;              /* UTC of File Creation */
TString     Comment;                /*  */
```

The **RunNo** is a unique number for each run. This mumber should increase with each run without users' intervention. Some of the information are acquired from the User Interface and some are from the evets. Hence this object is written at the end of the run.

4 Data Structure

Till now there are three sub-structures,

- **Event Data** which includes the latch and TDC infomation.
- **CAU Data** which contains CAU-TDC information.
- **Mon Data** which contains temperature, humidity, pressure, RPC HV & Current, Strip Noise rate and anything else.

Each of the structures is filled in seperate trees and are independent of each other. Here the trees are like bin/containers. The EB tosses the data into respective bins whenever it is made available to it. When the bin being filled by event data reaches a desired limit, all the bins are dumped into a root file. The bins are then cleared and the process starts again.

Any more sub-structures can be added if nessesary. The current structures are discussed in details in the following.

4.1 Event Data

One entry to this tree corresponds to one event data (latch and TDC). The member variables of this class, namely **DigiStore.h**, are,

```
TTimeStamp* EventTime=0;     // Event Timestamp
UInt_t ENum;                 // Event Numbers
int NDigiOpt1;               // Number of DigiStore not sets!
TClonesArray* fDigiOpt1;     // array with all DigiStore
```

One element of this array **fDigiOpt1** corresponds to one RPC which has signal in it. The member variable of these elements' class, namely **DigiOpt1.h**, are

```
UShort_t rpcId;              // module + xrow + yrow + zlay
ULong64_t strips[2];         // 64(X) + 64(Y) strip latch
vector<UInt_t> tdc;          // TDC_REF + TDC channel info
```

First two elements of the **tdc** vector are TDC references from the leading and trailing edges. Next, all the TDC hits in that layer are formatted as per the Figure 3 and pushed in the **tdc** one by one.

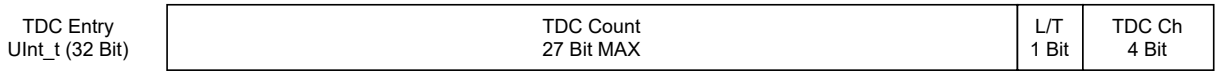

Figure 3: TDC Structure.

4.2 CAU Data

One entry of this tree contains the round trip delays for a set of DAQs. The member variables of this class, namely **CauStore.h**, are,

```
TTimeStamp        *Cautime = 0;  // CAU event timestamp
Int_t              CauENum;      // Event Number
vector<ULong64_t> cau_tdc;       // Round-trip-delay
```

In this case, a group of DAQs are tested for the round-trip-delay by setting the **select-line** in each CAU-cycle. In that one cycle, all the CAU-TDC values for the selected DAQs are formatted as per the Figure 4 and pushed in the **cau_tdc** one by one.

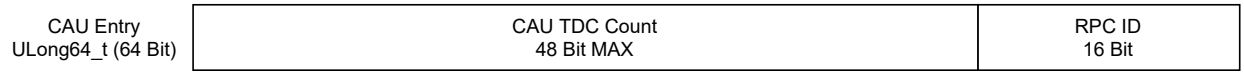

Figure 4: CAU-TDC Structure.

4.3 Mon Data

One entry in this tree contains the information from one mon cycle. The member variables of this class, namely **HealthStore.h**, are,

```
TTimeStamp                *HealthTime = 0;    // Mon timestamp
vector<UShort_t>           rpcId;
vector<UInt_t>             TPHdata;
vector<UShort_t>           xHVdata, yHVdata; // in Volt
vector<UShort_t>           xHCdata, yHCdata; // in nano-Amp
vector<vector<UInt_t>>     NOISEdata;        //
```

The **TPHdata** and **NOISEdata** is formatted as per the Figure 5.

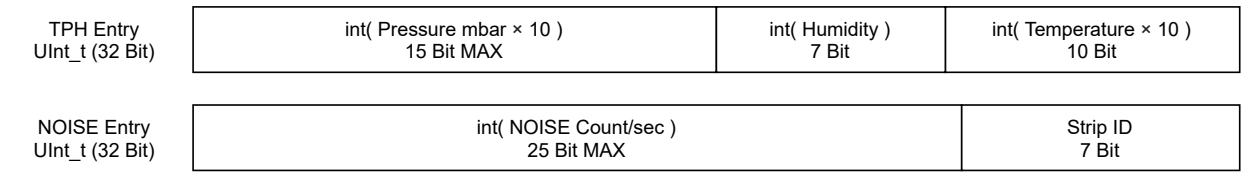

Figure 5: Mon-Data Structures.

4.4 Note

RPC Ch: 0-7 for X-side and 8-15 for Y-side.
Strip ID: 0-63 for X-side and 64-127 for Y-side.

5 Important Files

RunInfo.h, CauStore.h, HealthStore.h DigiStore.h, DigiOpt1.h, DigiStore.cc, ino_digi_set1.C, ino_digi_read1.C