

How to perform event building in a MINIBALL experiment

R. Lutter

May 20, 2005

Abstract

This document describes how to perform event building and to generate `ROOT` trees from med data in a Miniball experiment.

Contents

1	Online data acquisition	2
2	Filling hitBuffers from raw data	3
2.1	Extracting hits from DGF-4C data	3
2.2	Extracting hits from CAEN ADC/TDC data	3
3	Event building	4
3.1	Inserting time stamps in ADC/TDC data	5
3.2	Event building algorithm	6
4	Replay session	8

1 Online data acquisition

Any data in a MINIBALL experiment will be produced by connecting to the MBS subsystem and reading raw data from the hardware modules in VME and CAMAC crates [1]. Online data are then written to disk using the MED format (MBS event data) [2].

Fig. 1 shows data flow as well as software components used to acquire experimental data.

Following files are needed for this step:

- **DgfReadout.c**
user's readout function given by hardware definitions in **Config.C**, auto-generated during config step
- **DgfAnalyze.cxx**
methods and functions to setup the ROOT system and to decode raw data from MBS, auto-generated during the config step
- **udef/ProcessEvent.cxx**
consistency checks and accumulation of diagnostic histograms, has to be provided by user

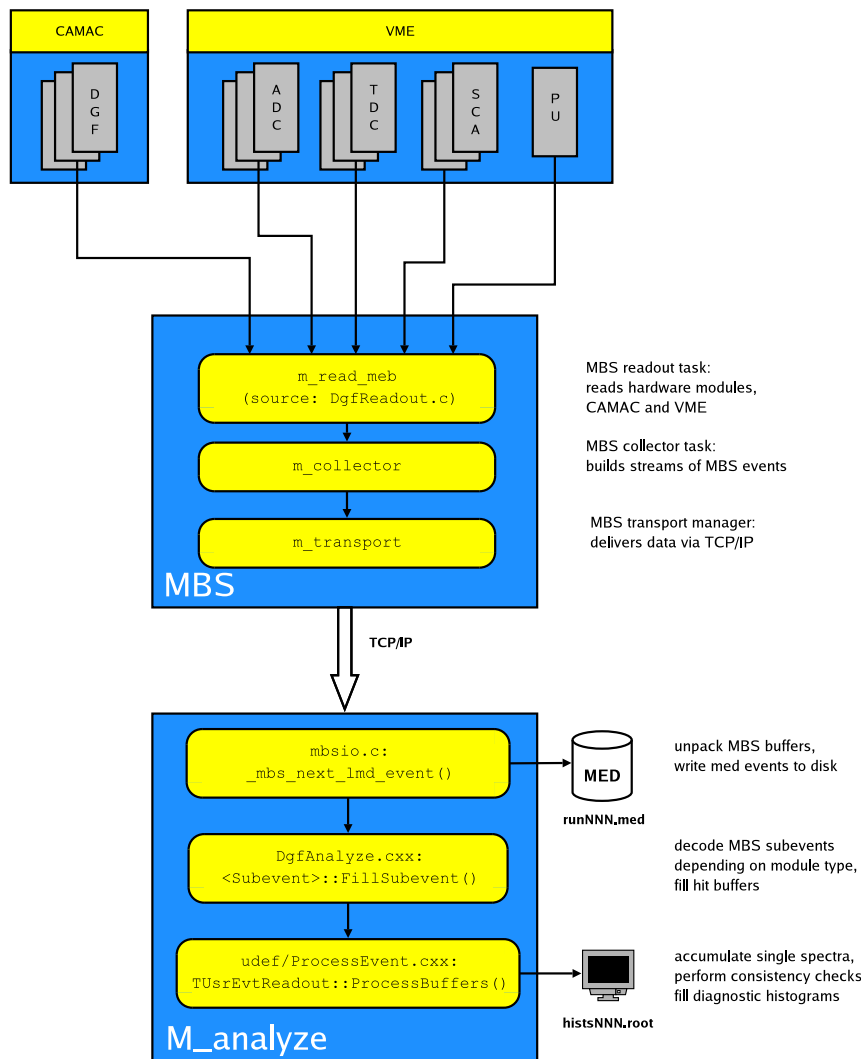


Figure 1: Online data acquisition

2 Filling hitBuffers from raw data

2.1 Extracting hits from DGF-4C data

2.2 Extracting hits from CAEN ADC/TDC data

3 Event building

In a second step one may create meaningful events from raw data read from MED files. These events may then be written to ROOT trees and used in replay sessions afterwards. Fig. 2 shows how to do the event building.

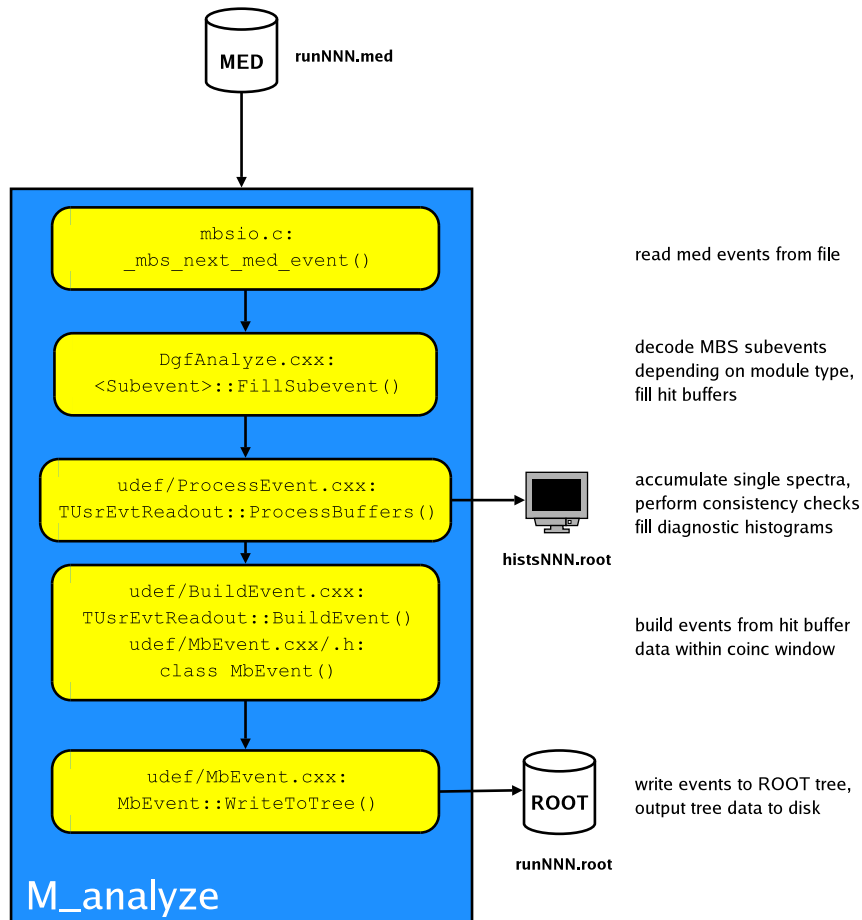


Figure 2: Event building

Steps to be performed for the event building process:

- in **SetCppIfdefs.C**: activate event building


```
dgf->MakeDefined("__EVENT_BUILDING_ON_", kTRUE, "Start event building");
```
- in **Config.C**: define an event class to be used for event building


```
dgf->IncludeUserClass("udef", "MbEvent.cxx", kTRUE);
```

 when doing the config step for the first time this will generate a template class `MbEvent`; you then have to modify this class according to your needs
- provide an event building algorithm: `TUsrEvtReadout::BuildEvent()` in **udef/BuildEvent.cxx** (table 2, fig. 3)
- in **.rootrc** file: add an entry `TMrbAnalyze.CoincWindow: N`
- perform config, compile and link the whole stuff, run `C_analyze`

3.1 Inserting time stamps in ADC/TDC data

```
Bool_t TUsrEvtReadout::InsertTimeStamp(Int_t TsIdx, Int_t TsChannel, Int_t StartIdx, Int_t NofHbx) {
//-----[C++ METHOD]-----
// Name:          TUsrEvtReadout::InsertTimeStamp
// Purpose:       Insert time stamps into adc data
// Arguments:     Int_t TsIdx      -- index of timestamping buffer
//               Int_t TsChannel  -- number of timestamping channel
//               Int_t StartIdx   -- index of adc buffer to start with
//               Int_t NofHbx     -- number of buffers to process
// Results:       kTRUE/kFALSE
// Description:   Inserts time stamps from timestamping dgf into adc buffers.
//               Time stamps in dgf buffer are sorted by time,
//               adc/tdc buffers contain single channel data grouped by event number.
//               This algorithm inserts time stamps one by one in adc channel data;
//               channels belonging to one event (i.e. having the same event number)
//               will be marked with same time stamp.
// Keywords:
//-----

    TUsrHBX *tsHbx = this->GetHBX(TsIdx);          // connect to hitbuffer of time stamping dgf

    for (Int_t i = StartIdx; i < StartIdx + NofHbx; i++) { // loop over adcs/tdcs
        tsHbx->ResetIndex();                          // reset to first item in ts-dgf buffer
        TUsrHit * tsHit = tsHbx->FindHit(TsChannel);  // fetch first item (= time stamp)
        if(!tsHit) break;                             // no time stamps in this buffer (should never be)
        TUsrHBX * cHbx = this->GetHBX(i);            // connect to current adc/tdc buffer
        cHbx->ResetIndex();                            // reset to first item in buffer
        TUsrHit * cHit = cHbx->NextHit();             // fetch first adc/tdc hit (= channel data)
        while(tsHit && cHit) {                         // step thru buffers
            Int_t evtNo = cHit->GetEventNumber();     // save current event number from adc/tdc hit
            while(cHit->GetEventNumber() == evtNo) { // as long as event number doesn't change:
                cHit->SetChannelTime(tsHit);          // insert current time stamp in adc hit
                cHit = cHbx->NextHit();
                if (!cHit) break;                     // end of adc/tdc buffer reached
            }
            tsHit = tsHbx->FindHit(TsChannel);        // event number in adc/tdc buffer has changed:
                                                    // get next time stamp from ts-dgf buffer
        }
    }
    return(kTRUE);
}
```

Table 1: Algorithm used for time stamp insertion

3.2 Event building algorithm

```

Bool_t TUsrEvtReadout::BuildEvent() {
//-----[C++ METHOD]-----
// Name:          TUsrEvtReadout::BuildEvent
// Purpose:       Event building
// Arguments:     --
// Results:       kTRUE/kFALSE
// Exceptions:
// Description:   Loops over all hit buffers,
//               collects hits within window 'coincWindow'
//               stores hits in events of type MbEvent
//               then calls method MbEvent::WriteToTree() for each event
// Strategy:
//               a binary tree TTree is used to compare events
//               method TTree::FindObject() calls MbEvent::Compare()
//               MbEvent::IsEqual() then tests if time stamps are 'equal' within 'coincWindow'
//               hits belonging to an event are stored in a hit buffer of type TClonesArray,
//               to get the benefits of TClonesArray these hit buffers have to be static
//               they will be allocated only once and reused afterwards.
// Keywords:
//-----

    TTree evtsS;
    TTree * evts = &evtsS;

// reset hit buffers
    TObjArrayIter epI(&poolOfMbEvents);
    while(MbEvent * evt = (MbEvent *) epI.Next()) evt->Reset();

// fill binary tree, create events if necessary
    Int_t evtCount = 0;
    for (Int_t i = kIdxCluster1; i < kIdxCluster1 + kNofSevts; i++) { // loop over all hit buffers
        if (i == kIdxTSCluster) continue; // discard time stamper
        TUsrHBX *hbx = this->GetHBX(i); // pointer to current hit buffer
        if (hbx) {
            Int_t hitNo = 0;
            hbx->ResetIndex(); // reset buffer index to first hit
            while (TUsrHit *hit = hbx->NextHit()) { // loop over hits of this buffer
                if (hit->GetChannelTime() == 0) continue; // time stamp = 0 -> junk data
                tmpEvent.SetEventTime(hit->GetChannelTime()); // insert time stamp into temp event
                MbEvent * evt = (MbEvent *) evts->FindObject(&tmpEvent); // compare it with events already there
                if (evt == NULL) { // no match
                    if (evtCount < poolOfMbEvents.GetEntriesFast()) { // try to get a event from pool
                        evt = (MbEvent *) poolOfMbEvents[evtCount];
                    } else {
                        evt = new MbEvent(); // create a new one
                        poolOfMbEvents.Add(evt); // add it to pool
                    }
                    evt->Reset(); // reset event time, hit buffer etc
                    evt->SetEventTime(hit->GetChannelTime()); // insert time stamp
                    evt->SetCoincWindow(coincWdw); // should know about time window
                    evts->Add(evt); // add new event to binary tree
                    evtCount++;
                }
                evt->AddHit(hit); // append current hit to event
                hitNo++;
            }
        }
    }

// call method MbEvent::WriteToTree() for all events
    TTreeIter evI(evts);
    while(MbEvent * evt = (MbEvent *) evI.Next()) evt->WriteToTree(); // write event data to tree
    evtsS.Clear(); // clear entries in binary tree
    return(kTRUE);
}

```

Table 2: Algorithm used for the event building process

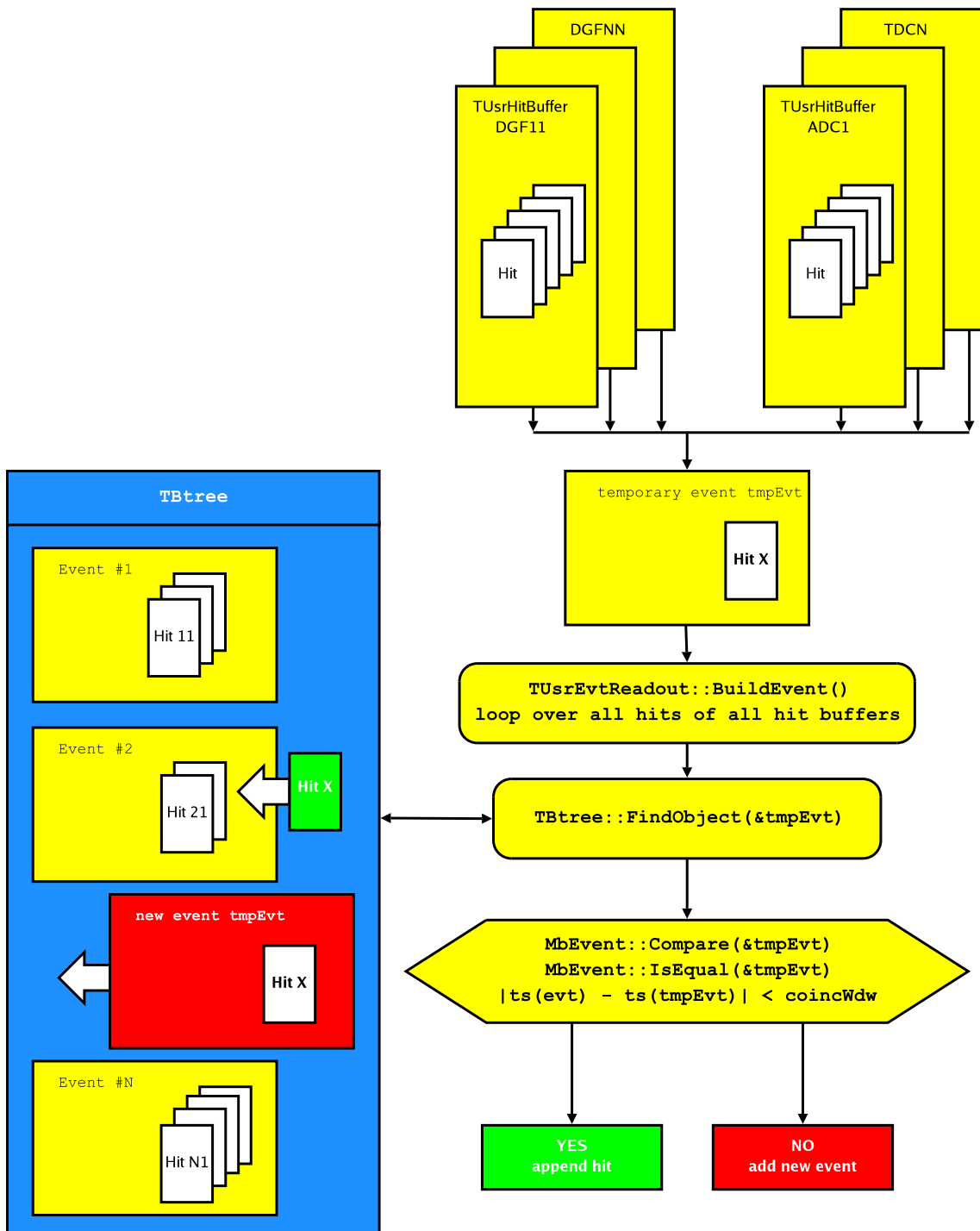


Figure 3: Event builder: schematic diagram

4 Replay session

Once events have been built and tree data have been written to ROOT file one may start a replay session by reading these pre-sorted data. Fig. 4 shows the data flow for such a replay session. The gain in speed is about a factor of 10 as compared to replaying raw data from MED files.

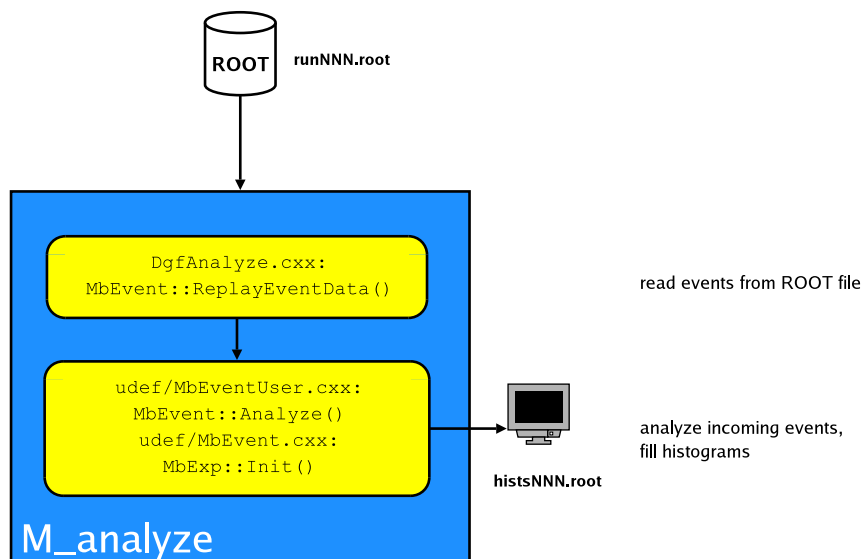


Figure 4: Offline replay session

User has to provide a method `MbEvent::Analyze()` containing his analysis. A wrapper class `MbExp` may be used to establish pointer arrays pointing to different event components such as cores, segments, channels, etc (table 3).

```

//-----[C++ CLASS DEFINITION]
//-----
// Name:          MbExp
// Purpose:       Wrapper class to access MbEvent data
// Description:   Organizes MbEvent data in a more "experiment-oriented" way.
//               Data may be accessed via cluster, core, and segment numbers (dgf),
//               via adc and channel numbers, resp.
//               Performs boundary check if requested.
//               Indices start with 0.
// Keywords:
//-----

class MbExp {

public:
    MbExp(MbEvent * Event = NULL) { Init(Event); }; // default ctor
    virtual ~MbExp() {}; // default dtor

    TUsrHit * Core(Int_t Cluster, Int_t Core, Bool_t BoundaryCheck = kFALSE);
    TUsrHit * Segment(Int_t Cluster, Int_t Core, Int_t Seg, Bool_t BoundaryCheck = kFALSE);
    TUsrHit * Particle(Int_t Dgf, Int_t Channel, Bool_t BoundaryCheck = kFALSE);
    TUsrHit * Time(Int_t Tdc, Int_t Channel, Bool_t BoundaryCheck = kFALSE);

    void Init(MbEvent * Event = NULL); // initialize ptr arrays with data from MbEvent

protected:
    TUsrHit * fCores[kNofDgfClusters][kNofDgfCoresPerCluster];
    TUsrHit * fSegs[kNofDgfClusters][kNofDgfCoresPerCluster][kNofDgfSegsPerCore];
    TUsrHit * fPart[kNofAdcs][kNofChannelsPerAdc];
    TUsrHit * fTime[kNofTdcs][kNofChannelsPerTdc];

protected:
    TUsrHit fEmptyHit;
};

```

Table 3: Wrapper class to access event components

References

- [1] See also: "Instructions how to use the DAQ in a MINIBALL experiment",
<http://www.bl.physik.uni-muenchen.de/marabou/html/doc/DaqInstructions.pdf>
- [2] For details see: "MED Data Structure",
<http://www.bl.physik.uni-muenchen.de/marabou/html/doc/MedStructure.pdf>