QA

Online/Offline Quality assurance

Requirements

- Check the correctness of the data produced at various steps of the data processing (Raws, SDigits, Digits, RecPoints, RecParticles)
- Make use of the same algorithms online (DAQ, HLT) than offline
- Provide the users with information flags and give them access to the data being monitored
- Stay simple (data format, quality check, quality check result)

Architecture 1.

- The QA object (result of the QA check):
  - one bit map per detector
    - 5 processing levels (Raw, Sim, Rec, ESD, Analysis)
    - 4 severity levels (Info, Warning, Error, Fatal)
    - Stored in a single root file (one for all detectors; online requires merging!)
  - provides naming

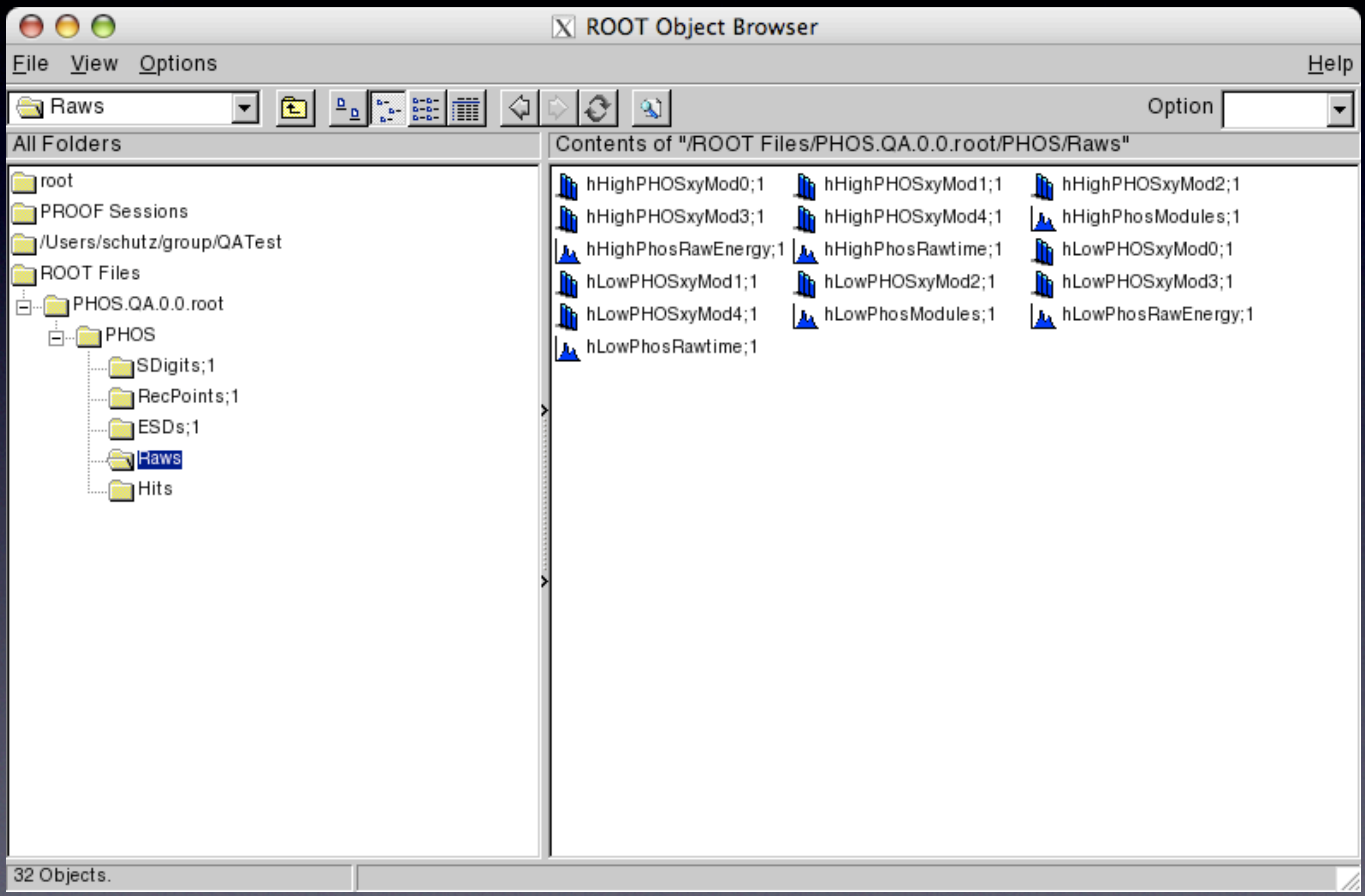
The QA object

```
root [0] TFile f("QA.root")
root [1] AliQA * qa = dynamic_cast<AliQA *>(f.Get("QA"))
root [2] qa->ShowAll()
I-AliQA::ShowStatus: QA Status for ITS raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for TPC raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for TRD raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for TOF raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for PHOS raw =0x2, sim=0x90, rec=0x400, esd=0x2000, ana=0x0

I-AliQA::ShowStatus: QA Status for HMPID raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for EMCAL raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for MUON raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for FMD raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for ZDC raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for PMD raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for T0 raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for VZERO raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for ACORDE raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

I-AliQA::ShowStatus: QA Status for HLT raw =0x0, sim=0x0, rec=0x0, esd=0x0, ana=0x0

root [3] qa->IsSet(AliQA::kPHOS, AliQA::kSIM, AliQA::kFATAL)
(const Bool_t)(1)
root [4] qa->IsSet(AliQA::kPHOS, AliQA::kSIM, AliQA::kWARNING)
(const Bool_t)(0)
root [5] cap
```

Architecture 2.

- Creation and accumulation of data
  - QA data are TH1
  - All QA data are stored in a single root file making use of a directory structure (online requires to split in detetector files PHOS.QA.<run>.<cycle>.root)
  - Data are accumulated during cycles (# of events)
  - Detectors implement only booking and filling

The QA data File



Architecture 3.

- Checking
  - Checking is done by comparison with reference data
  - By inspection of the content of the root data file
  - Detectors implement the checking algorithms and set the severity flag

Implementation: booking and filling cycle

```
    void QAforOn(char * fileName )
{
   AliPHOSQADataMaker * phosQAdm = new AliPHOSQADataMaker() ;
   AliRawReaderRoot * rawReader = new AliRawReaderRoot(fileName) ;
   rawReader->NextEvent() ;
    Int_t cycleLength = 10 ;
//------> SOR
   TList * phosQADataList = phosQAdm->Init(AliQA::kRAWS, rawReader->GetRunNumber(), cycleLength) ;
   //------> SOC
      phosQAdm->StartOfCycle(AliQA::kRAWS) ;
      while (rawReader->NextEvent()) {
         if ( phosQAdm->IsCycleDone() ) {
            phosQAdm->EndOfCycle(AliQA::kRAWS) ;
            phosQAdm->StartOfCycle(AliQA::kRAWS) ;
         }
      //----> Fill one event
         phosQAdm->Exec(AliQA::kRAWS, rawReader) ;
         phosQAdm->Increment() ;
          }
   //------> EOC
      phosQAdm->EndOfCycle(AliQA::kRAWS) ;
//------> EOR
   phosQAdm->Finish(AliQA::kRAWS) ;
}
```

Implementation: checking

```
  void QA() {
 AliQAChecker qac ;
 qac.SetRefDirName("local://home/group/alice/schutz/QATest/") ;
 TStopwatch timer;
 qac.SetQAResultDirName("local://home/group/alice/schutz/QATest/") ;
 timer.Start();

 qac.Run() ;

 timer.Stop();
 timer.Print();
}
```

Implementation: detector view

- `Ali<Det>QADataMaker: new class`
- `Ali<Det>QAChecker: new class`
- `Ali<Det>SDigtizer: ctor, Exec()`
- `Ali<Det>Digitizer: ctor, Exec()`

Implementation: detector view

```
  class AliPHOSQADataMaker: public AliQADataMaker {

public:
  AliPHOSQADataMaker() ;          // ctor
  AliPHOSQADataMaker(const AliPHOSQADataMaker& qadm) ;
  AliPHOSQADataMaker& operator = (const AliPHOSQADataMaker& qadm) ;
  virtual ~AliPHOSQADataMaker() {;} // dtor

private:
  virtual void   EndOfDetectorCycle() ;
  virtual void   InitHits() ;
  virtual void   InitESDs() ;
  virtual void   InitDigits() ;
  virtual void   InitRecPoints() ;
  virtual void   InitRaws() ;
  virtual void   InitSDigits() ;
  virtual void   MakeESDs(AliESDEvent * esd) ;
  virtual void   MakeHits(TClonesArray * hits) ;
  virtual void   MakeDigits(TClonesArray * digits) ;
  virtual void   MakeRecPoints(TTree * recpo) ;
  virtual void   MakeRaws(AliRawReader* rawReader) ;
  virtual void   MakeSDigits(TClonesArray * sigits) ;
  virtual void   StartOfDetectorCycle() ;

  ClassDef(AliPHOSQADataMaker,1)  // description

};
```

Implementation: detector view

```
    //
    ____________________________________________________________________________
void AliPHOSQADataMaker::InitHits()
{
  // create Hits histograms in Hits subdir
  TH1F * h0 = new TH1F("hPhosHits",    "Hits energy distribution in PHOS",
100, 0., 100.) ;
  h0->Sumw2() ;
  Add2HitsList(h0, 0) ;
  TH1I * h1  = new TH1I("hPhosHitsMul", "Hits multiplicity distribution in
PHOS", 500, 0., 10000) ;
  h1->Sumw2() ;
  Add2HitsList(h1, 1) ;
}
//____________________________________________________________________________
void AliPHOSQADataMaker::MakeHits(TClonesArray * hits)
{
  //make QA data from Hits

    GetHitsData(1)->Fill(hits->GetEntriesFast()) ;
    TIter next(hits) ;
    AliPHOSHit * hit ;
    while ( (hit = dynamic_cast<AliPHOSHit *>(next())) ) {
      GetHitsData(0)->Fill( hit->GetEnergy()) ;
    }
}
```

Implementation: detector view

```
    //
    ____________________________________________________________________________
void AliPHOSQADataMaker::EndOfDetectorCycle(AliQA::TASKINDEX task, TList *
list)
{
  //Detector specific actions at end of cycle
  // do the QA checking
  AliQAChecker::Instance()->Run(AliQA::kPHOS, task, list) ;
}
```

Implementation: detector view

```
    //_____________________________________________________________________________
AliPHOSSDigitizer::AliPHOSSDigitizer(const char * alirunFileName,
                 const char * eventFolderName)
{
  // ctor
  //FIXME: get the run number
  Int_t run = 0 ;
  //EMXIF
 fQADM = new AliPHOSQADataMaker() ;
 fQADM->Init(AliQA::kHITS, run, fgkCycles) ;
 fQADM->StartOfCycle(AliQA::kHITS) ;
 fQADM->Init(AliQA::kSDIGITS, run) ;
 fQADM->StartOfCycle(AliQA::kSDIGITS, "same") ;
}
//____________________________________________________________________________
void AliPHOSSDigitizer::Exec(Option_t *option)
{
  for (ievent = fFirstEvent; ievent <= fLastEvent; ievent++) {
   // make Quality Assurance data

    if (fQADM->IsCycleDone() ) {
      fQADM->EndOfCycle(AliQA::kHITS) ;
     fQADM->EndOfCycle(AliQA::kSDIGITS) ;
      fQADM->StartOfCycle(AliQA::kHITS) ;
     fQADM->StartOfCycle(AliQA::kSDIGITS, "same") ;
   }
    fQADM->Exec(AliQA::kHITS, hits) ;
    fQADM->Exec(AliQA::kSDIGITS, sdigits) ;
    fQADM->Increment() ;
  }
  //Write the quality assurance data
  fQADM->EndOfCycle(AliQA::kHITS) ;
  fQADM->EndOfCycle(AliQA::kSDIGITS) ;
  fQADM->Finish(AliQA::kHITS) ;
  fQADM->Finish(AliQA::kSDIGITS) ;
}
```

To be done

- Detectors adopt the framework and implement their algorithms
- DAQ (done) and HLT use these algorithms
- QA data archiving
    - Offline to OCDB
    - Online to FXS; shuttle to OCDB
- Reference data archiving and retrieving (online)
- QA results
    - Archive in OCDB: root file or meta data
    - Usage of QA results